Teradata Factory

Course # 9038

Version 14.00.4

Student Guide

Notes

TERADATA
Raising Intelligence

Course Overview

Teradata Factory

Teradata Concepts
MPP System Architectures
Physical Design and Implementation
Application Utilities
Database Administration

Tenth Edition
April, 2012

Trademarks

The following names are registered names or trademarks and are used throughout this manual.

The product or products described in this book are licensed products of Teradata Corporation or its affiliates.

Teradata, BYNET, DBC/1012, DecisionCast, DecisionFlow, DecisionPoint, Eye logo design, InfoWise, Meta Warehouse, MyCommerce, SeeChain, SeeCommerce, SeeRisk, Teradata Decision Experts, Teradata Source Experts, WebAnalyst, and You've Never Seen Your Business Like This Before, and Raising Intelligence are trademarks or registered trademarks of Teradata Corporation or its affiliates.

Adaptec and SCSISelect are trademarks or registered trademarks of Adaptec, Inc.
AMD Opteron and Opteron are trademarks of Advanced Micro Devices, Inc.
BakBone and NetVault are trademarks or registered trademarks of BakBone Software, Inc.
EMC^2, PowerPath, SRDF, and Symmetrix are registered trademarks of EMC^2 Corporation.
GoldenGate is a trademark of GoldenGate Software, a division of Oracle Corporation.
Hewlett-Packard and HP are registered trademarks of Hewlett-Packard Company.
Intel, Pentium, and XEON are registered trademarks of Intel Corporation.
IBM, CICS, RACF, Tivoli, z/OS, and z/VM are registered trademarks of International Business Machines Corporation.
Linux is a registered trademark of Linus Torvalds.
Engenio is a registered trademarks of NetApp Corporation.
Microsoft, Active Directory, Windows, Windows NT, and Windows Server are registered trademarks of Microsoft Corporation in the United States and other countries.
Novell and SUSE are registered trademarks of Novell, Inc., in the United States and other countries.
QLogic and SANbox trademarks or registered trademarks of QLogic Corporation.
SAS and SAS/C are trademarks or registered trademarks of SAS Institute Inc.
SPARC is a registered trademark of SPARC International, Inc.
Symantec, NetBackup, and VERITAS are trademarks or registered trademarks of Symantec Corporation or its affiliates in the United States and other countries.
Unicode is a collective membership mark and a service mark of Unicode, Inc.
UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Material developed by:
Teradata Learning

Table of Contents

Trademarks....0-4
Course Materials....0-6
Options for Displaying PDF Files....0-8
Example of Left Page – Right Page Display....0-10
View and search a PDF....0-10
PDF Comment and Markup Tools....0-12
Example of Highlighter and Sticky Note Tools....0-14
Example of Typewriter Tool....0-16
Course Description....0-18
Who Should Attend....0-20
Prerequisites....0-20
Class Format....0-22
Classroom Rules....0-22
Outline of the Two Weeks....0-24
Teradata Certification Tests....0-26

Course Materials

The Teradata Factory course materials that are provided on a USB flash drive are listed on the facing page. These materials are provided at the beginning of the class.

The Teradata Factory Student Manual and the Lab Workbook have been created as PDF files which can be viewed using Adobe® Reader®.

These PDF files were created using Adobe Acrobat® and commenting has been enabled for both files. This allows you to use Adobe® Reader® Comment and Markup tools to place your own notes and comments within the files.

Course Materials

Teradata Factory course materials include:

- Paper copy of TF Lab Workbook
- Electronic copy (PDF files) of Student Manual and Lab Workbook

Contents of the flash drive include:

- Teradata Factory Class Files
  - Class Files (these PDF files allow use of Comment and Markup tools)
    - TF v1400.4 Lab Workbook.pdf
    - TF v1400.4 Student Manual.pdf
  - Miscellaneous Software
    - Acrobat Reader
    - Microsoft .NET Packages
    - Putty – use for secure shell Linux connections
    - Secure FTP – use for secure FTP to Linux servers
  - TD 14.0 Reference Manuals
  - TD 14.0 TTU – Subset of tools and utilities (numbered in order of installation)
    - 01_piom__windows_i386.14.00.00.06.zip
    - 02_TeraGSS__windows_i386.14.00.00.01.zip
    - :
  - TD Demo Lab Setup (numbered in order of installation)

Options for Displaying PDF Files

Adobe® Reader® is a tool that you can use to open and view the Teradata Factory PDF course files. You can also use the Adobe Reader to make comments or notes and save your PDF file.

Since the Teradata Factory course materials have been created in a book format (left page - right page), you may want to set options in Adobe Reader to view the materials in a book format.

- **The left page contains additional information about the right or slide page.**
- **The right page is copy of the PPT slide that is used during the presentation.**

To view the Teradata Factory Student Manual in a book format using Adobe Reader 9.2 or before, use the View Menu > Page Display and set the following options.

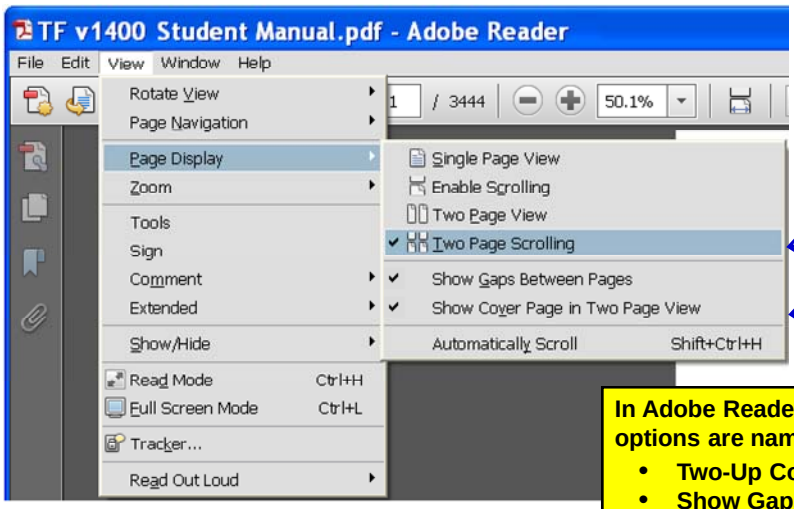
- Two-Up Continuous
- Show Gaps Between Pages (normally checked or set by default)
- Show Cover Page During Two-up

Options for Displaying PDF Files

The Teradata Factory course materials are created in a left page – right page format.

- **Left page – contains additional information about the slide page**
- **Right page – copy of the PPT slide that is used during the presentation**

To display PDF files in a book type (left page – right page) format, Adobe Reader options need to be set.



In Adobe Reader 9.2 and earlier versions, the options are named:

- **Two-Up Continuous**
- **Show Gaps Between Pages**
- **Show Cover Page During Two-Up**

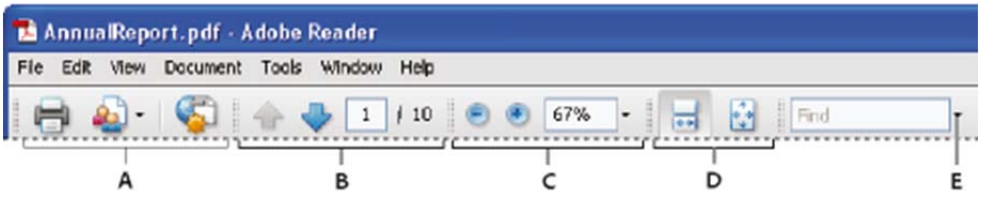
Example of Left Page – Right Page Display

The facing page illustrates an example of displaying the Teradata Factory Student Manual in a left page – right page format.

View and search a PDF

In the Adobe Reader toolbar, use the Zoom tools and the Magnification menu to enlarge or reduce the page. Use the options on the View menu to change the page display. There are various options in the Tools menu to provide you with more ways to adjust the page for better viewing (Tools > Select & Zoom).

This is an example of menus using Adobe Reader 9.2.

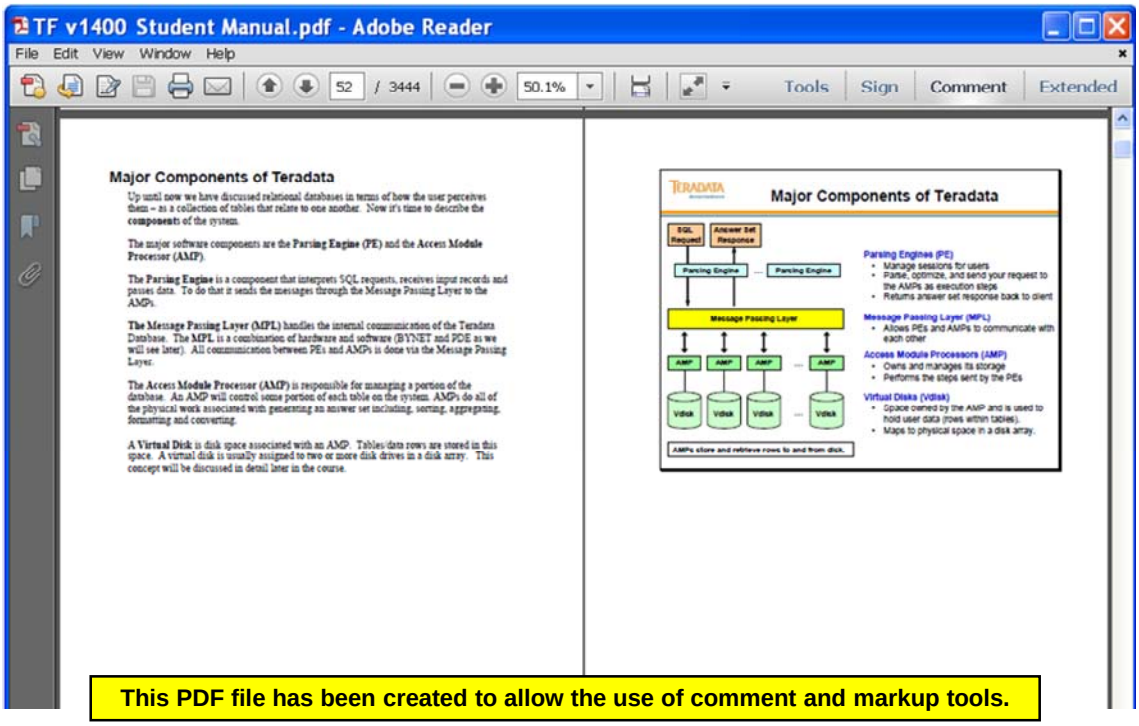


These Adobe Reader toolbars open by default:

A. File toolbar
B. Page Navigation toolbar
C. Select & Zoom toolbar
D. Page Display toolbar
E. Find toolbar

Example of Left Page – Right Page Display

After setting the Page Display options, the PDF file is displayed as below.



This PDF file has been created to allow the use of comment and markup tools.

PDF Comment and Markup Tools

The Teradata Factory course materials have "commenting" enabled. Therefore, you can make comments in these files using the commenting and markup tools. Of the many commenting and markup tools that are available, you may find it easier to use the following tools (highlighted on the facing page).

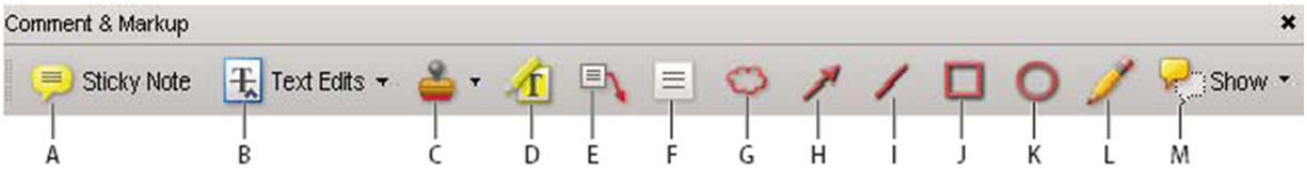
- Add Sticky Note
- Highlight Text Tool
- Typewriter

Comments can include both notes and drawings (if you have the time during class). You can enter a text message using the Sticky Note tool. You can use a drawing tool to add a line, circle, or other shape and then type a note in the associated pop-up note.

You can enable the Comment & Markup Toolbar or you can simply select the tools using the pull-down menus. The example below is for Adobe Reader 9.2.

- Enable the Comment & Markup Toolbar and select the tool to use

- Menus (View > Toolbars > Comment & Markup) to add notes or comments.



Options on the Comment & Markup toolbar:

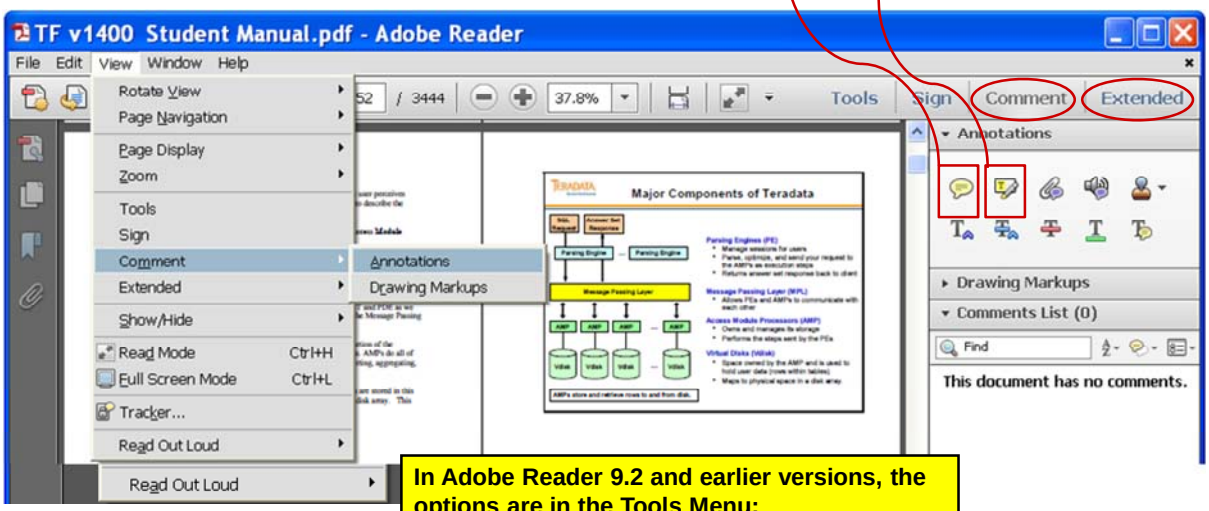
A. Sticky Note tool
B. Text Edits tool
C. Stamp tool and menu
D. Highlight Text tool
E. Callout tool
F. Text Box tool
G. Cloud tool
H. Arrow tool
I. Line tool
J. Rectangle tool
K. Oval tool
L. Pencil tool
M. Show menu

After you add a note or comment, it stays selected until you click elsewhere on the page. A selected comment is highlighted by a blue halo to help you find the markup on the page.

PDF Comment and Markup Tools

Comment and markup tools that may be useful include:

- **Sticky Note (Comment)**
- **Highlight Text Tool (Comment)**
- **Add a Text Box (Extended) or Typewriter**



In Adobe Reader 9.2 and earlier versions, the options are in the Tools Menu:

- **Comment & Markup > Sticky Note**
- **Comment & Markup > Highlight Text Tool**
- **Typewriter**

Example of Highlighter and Sticky Note Tools

The facing page illustrates an example of using the Highlighter and Sticky Note tools.

Select a commenting or markup tool.

Choose Tools > Comment & Markup > *Highlighter or* S*ticky Note (or another tool)*

Note: After you make an initial comment, the tool changes back to the Select tool so that you can move, resize, or edit your comment. (The Pencil, Highlight Text, and Line tools stay selected.)

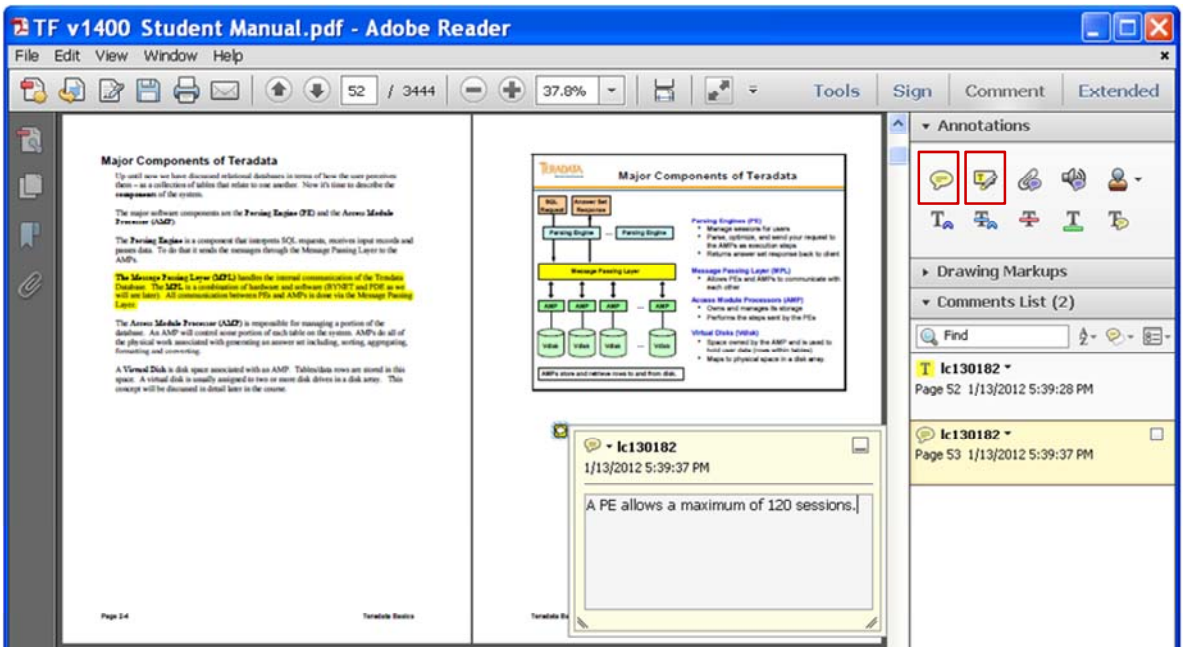
To keep a commenting tool selected so you can add multiple comments without reselecting the tool, do the following:

- Select the tool you want to use (but don't use it yet).
- Choose View > Toolbars > Properties Bar.
- Select Keep Tool Selected.

You can change the font of a text in a sticky note. Open the sticky note, choose View > Toolbars > Properties Bar, select the text in a note, and then change the font size in the Properties Bar.

Example of Highlighter and Sticky Note Tools

The left page illustrates the Highlighter tool and the right page illustrates the Sticky Note tool.



Example of Typewriter Tool

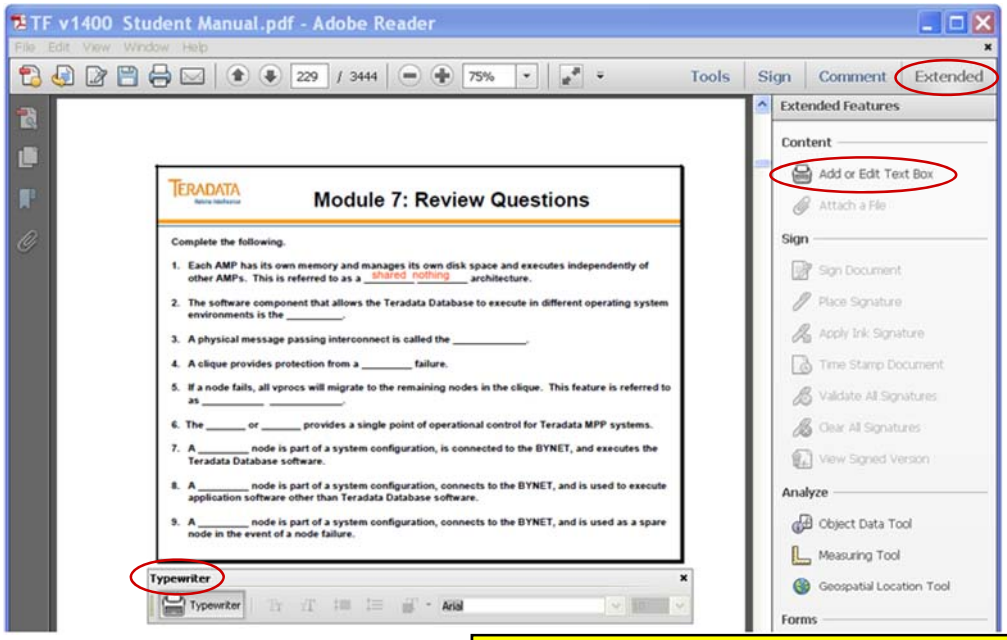
The facing page illustrates an example of using the Typewriter tool. This example also illustrates that the Typewriter Toolbar is enabled.

The Typewriter Toolbar may be useful when completing review questions as shown on the facing page. You already have the answer to one of hundreds of questions in this course.

After making notes and comments, save your changes. You may want to save your changes to a different PDF file name in order to preserve the original PDF file.

Example of Typewriter Tool

The Typewriter tool can be used to add text at any location in the PDF file.



To enable the Typewriter Toolbar in Adobe 9.2 or before:

- **Tools > Typewriter > Show Typewriter Toolbar**

Course Description

This course provides information on the following major topics:

- Teradata Concepts
- System Architectures (e.g., 2650, 2690, 6650, and 6690 Systems)
- Teradata Physical Database Design
- Teradata SQL ANSI Differences for Version 2
- Teradata Application Utilities
- Teradata Database Administration

Course Description

Description

The primary focus of this ten day course is to teach you about the design, implementation, and administration of the Teradata Database.

The major topics in this course include:

- **Teradata Database features and functions**
- **The parallelism of the Teradata Database**
- **How Teradata is implemented on MPP systems (e.g., 6690 systems)**
- **How to perform physical database design for Teradata Database**
- **Teradata SQL ANSI Differences**
- **How to load and export data using the Teradata application utilities**
- **How to perform common administrative functions for the Teradata Database**

Who Should Attend

This class is a learning event for relational database experienced individuals who need to learn the Teradata Database. This course is designed for Teradata practitioners who need to get hands-on practice with the Teradata Database in a learning environment.

- Professional Services Consultants
- Channel Partners

Prerequisites

An understanding of relational databases, SQL, and the logical data model is **necessary** before attending this course.

Experience with large systems, relational databases and SQL, and an understanding of the UNIX operating system is **useful, but not required** before attending this course.

There are Web Based Training classes that provide information about Teradata concepts and SQL.

- Overview of Teradata
- Teradata SQL

Who Should Attend and Prerequisites

Who Should Attend

This course is designed for ...

- Teradata Professional Services Consultants
- Channel Partners

Prerequisites

Required:

- An understanding of the logical data model, relational, SQL, and data processing concepts.

Useful, but not required:

- Experience with relational databases and SQL
- Experience with large systems used with Teradata

Class Format

This ten-day class will be conducted as a series of lectures with classroom discussions, review questions, and workshops.

Classroom Rules

The classroom rules are listed on the facing page.

Class Format and Rules

Class Format

This ten day class consists of ...

- Instructor presentations
- Class discussions
- Workshop exercises

Classroom Rules

The classroom rules are …

- Turn off your cellular phones.
- During lecture, only use your laptop to follow the class materials.
- Come to class on time in the morning and after breaks.
- Enjoy the two weeks.

Outline of the Two Weeks

An outline of the two weeks is described on the following page. Major topic examples are listed for each week.

Outline of the Two Weeks

1. Teradata Concepts

Teradata features and functions
Parallelism and Teradata

MPP System Architectures

Characteristics of MPP (e.g., 6690) systems – typical configurations
Disk Array subsystems and how Teradata utilizes disk arrays

Teradata Physical Database Design (continued in week #2)

Primary and secondary index selection; partitioned, NoPI, and columnar tables
How the Teradata database works
Collecting Statistics and Explains
SQL ANSI syntax & features; Teradata and ANSI transaction modes
Temporary tables, System Calendar, and Teradata System Limits

2. Teradata Application Utilities

Load utilities (e.g., BTEQ, FastLoad, MultiLoad, and TPump)
Export utilities (e.g., BTEQ and FastExport)

Teradata Database Administration

Dictionary tables and views; system hierarchy and space management
Users, Databases, Access Rights, Roles, and Profiles
Administrator and System Utilities – Teradata Administrator, Viewpoint, DBSControl
How to use the archive facility to do Archive, Restore, and Recovery procedures

Teradata Certification Tests

The facing page lists the various Teradata certification tests. Depending upon the tests that are completed, you can earn various Teradata Certified designations such as Teradata Certified Professional.

The Teradata 12 Certification tests require knowledge plus experience with Teradata. This manual will help you prepare for these Teradata 12 tests, but many of the test questions are scenario-based and Teradata experience is needed to answer these types of questions.

The Teradata V2R5 Certification tests were retired on March 31, 2010.

Teradata Certification Tests

Teradata 12.0 Certification Tests

- ✓ 1 – Teradata 12 Basics
- 2 – Teradata 12 SQL
- ✓ 3 – Teradata 12 Physical Design and Implementation
- ✓ 4 – Teradata 12 Database Administration
- 5 – Teradata 12 Solutions Development
- 6 – Teradata 12 Enterprise Architecture

- 7 – Teradata 12 Comprehensive Mastery

By passing all seven Teradata 12 certification tests, you become a Teradata 12 Certified Master.

✓ This course (along with Teradata experience) will prepare you for these tests.

Options for Teradata V2R5 Certified Masters:

- **The Teradata 12 Qualifying Exam is available as an alternative to taking tests 1 – 6.**
- **To achieve the Teradata 12 Master certification …**
  1. **Pass the Teradata 12 Qualifying Exam OR pass each of the 6 tests**
  2. **Pass the Teradata 12 Comprehensive Mastery exam**

Notes

Module 1

Teradata Overview

After completing this module, you will be able to:

- **Describe the purpose of the Teradata product**
- **Understand the history of the Teradata Corporation**
- **List major architectural features of the product**

Notes

Table of Contents

What is Teradata? 1-4
How large is a Trillion and a Quadrillion? 1-4
Teradata – A Brief History 1-6
What is a Data Warehouse? 1-8
Data Marts 1-8
Independent Data Marts 1-8
Logical Data Marts 1-8
Dependent Data Marts 1-8
What is Active Data Warehousing? 1-10
What is a Relational Database? 1-12
Primary Key 1-12
Answering Questions with a Relational Database 1-14
Foreign Key 1-14
Teradata Database Competitive Advantages 1-16
Module 1: Review Questions 1-18

What is Teradata?

Teradata is a Relational Database Management System (RDBMS) for the world's largest commercial databases. It is possible to have databases with over 100 terabytes (of data) in size. This characteristic makes Teradata an obvious choice for large data warehousing applications; however the Teradata system may also be as small as 100 gigabytes. With its parallelism and scalability, Teradata allows you to start small with a single node and grow large with many nodes through linear expandability.

Teradata is comparable to a large database server, with multiple client application making inquiries against it concurrently.

Teradata 14.0 was released on February 14, 2012.

The acronym SUSE comes from the German name "Software und System Entwicklung" which means Software and Systems Development.

The ability to manage terabytes of data is accomplished using the concept of parallelism, wherein many individual processors perform smaller tasks concurrently to accomplish an operation against a huge repository of data. To date, only parallel architectures can handle databases of this size.

Acronyms: SLES – SUSE Linux Enterprise Server
SUSE – Software und System Entwicklung (German name which means Software and Systems Development)

How large is a Trillion and a Quadrillion?

The **Teradata Database** was the first commercial database system to support a trillion bytes of data. It is hard to imagine the size of a trillion. To put it in perspective, the life span of the average person is 2.5 gigaseconds (or said differently 2,500,000,000 seconds). A trillion seconds is 31,688 years!

Teradata has customers with multiple petabytes of data. One petabyte is one quadrillion bytes of data. A petabyte is effectively 1000 terabyes.

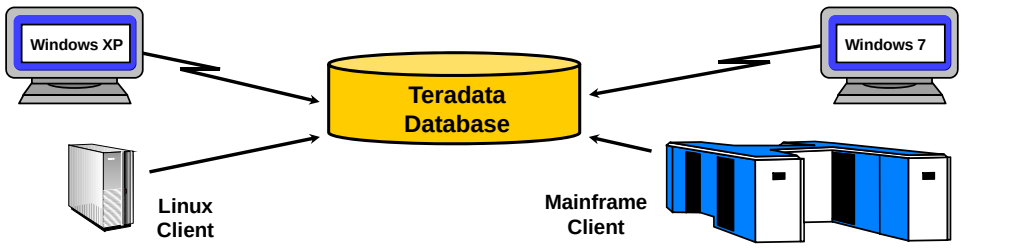
1 Kilobyte (KB)	= 1024 bytes
1 Megabyte (MB)	= 1024^2 >= 1,000,000 bytes
1 Gigabyte (GB)	= 1024^3 >= 1,000,000,000 bytes
1 Terabyte (TB)	= 1024^4 >= 1,000,000,000,000 bytes
1 Petabyte (PB)	= 1024^5 >= 1,000,000,000,000,000 bytes
1 Exabyte	= 1024^6 >= 1,000,000,000,000,000,000 bytes
1 Zetabyte	= 1024^7 >= 1,000,000,000,000,000,000,000 bytes
1 Yottabyte	= 1024^8 >= 1,000,000,000,000,000,000,000,000 bytes

What is Teradata?

The Teradata Database is a Relational Database Management System.

Designed to run the world's largest commercial databases.

- **Preferred solution for enterprise data warehousing**
- **Acts as a "database server" to client applications throughout the enterprise**
- **Uses parallelism to manage terabytes or petabytes of data**
  - **A terabyte is a trillion bytes of data – 10^{12}.**
  - **A petabyte is a quadrillion bytes of data – 10^{15}, effectively 1000 terabytes.**
- **Capable of supporting many concurrent users from various client platforms (over TCP/IP or IBM channel connections).**
- **The latest Teradata release is 14.0 and executes as a SUSE Linux application.**



Teradata – A Brief History

The **Teradata** Corporation was founded in 1979 in Los Angeles, California. The corporate goal was the creation of a "database computer" which could handle billions of rows of data, up to and beyond a terabyte of data storage. It took five years of development before a product was shipped to a first customer in 1984. In 1982, the YNET technology was patented as the enabling technology for the parallelism that was at the heart of the architecture. The YNET was the interconnect which allowed hundreds of individual processors to share the same bandwidth.

In 1987, Teradata went public with its first stock offering. In 1988, Teradata partnered with the NCR Corporation to build the next generation of database computers (e.g., 3700). Before either company could market its next generation product, NCR was purchased by AT&T Corporation at the end of 1991. AT&T purchased Teradata and folded Teradata into the NCR structure in January of 1992. The new division was named AT&T GIS (Global Information Solutions).

In 1996, AT&T spun off three separate companies, one of which was NCR which then returned to its old name. Teradata was a division of NCR from 1997 until 2001. In 1997, Teradata (as part of NCR) had become the world leader in scalable data warehouse solutions.

In 2007, NCR and Teradata separated as two corporations.

Teradata – A Brief History

1979 – Teradata Corp founded in Los Angeles, California
– Development begins on a massively parallel computer

1982 – YNET technology is patented.

1984 – Teradata markets the first database computer DBC/1012
– First system purchased by Wells Fargo Bank of California

1989 – Teradata and NCR partner on next generation of DBC.

1992 – NCR Corporation is acquired by AT&T and Teradata is merged into NCR within AT&T and named AT&T GIS (Global Information Solutions).

1996 – AT&T spins off NCR Corporation with Teradata; Teradata Version 2 is released.

1997 – The Teradata Database becomes the industry leader in data warehousing.

2000 – The first 100+ Terabyte system is put into production.

2002 – Teradata V2R5 released 12/2002; major release including features such as PPI, roles and profiles, multi-value compression, and more.

2007 – NCR and Teradata become two separate corporations. Teradata 12.0 is released.

2010 – Teradata 13.10 is released as well as 2650/4600/5600/5650 systems.

2011 – Teradata releases 6650/6680/6690 systems.
– More than 20 customers with 1 PB or larger systems

2012 – Teradata 14.0 is released on February 14, 2012.

What is a Data Warehouse?

A **data warehouse** is a central, enterprise-wide database that contains information extracted from the operational data stores. Data warehouses have become more common in corporations where enterprise-wide detail data may be used in on-line analytical processing to make strategic and tactical business decisions. Warehouses often carry many years worth of detail data so that historical trends may be analyzed using the full power of the data.

Many data warehouses get their data directly from operational systems so that the data is timely and accurate. While data warehouses may begin somewhat small in scope and purpose, they often grow quite large as their utility becomes more fully exploited by the enterprise.

Data Warehousing is a process, not a product. It is a technique to properly assemble and manage data from various sources to answer business questions not previously possible or known.

Data Marts

A **data mart** is a special purpose subset of enterprise data used by a particular department, function or application. Data marts may have both summary and detail data, however, usually the data has been pre-aggregated or transformed in some way to better handle the particular type of requests of a specific user community.

Independent Data Marts

Independent data marts are created directly from operational systems, just as is a data warehouse. In the data mart, the data is usually transformed as part of the load process. Data might be aggregated, dimensionalized or summarized historically, as the requirements of the data mart dictate.

Logical Data Marts

Logical data marts are not separate physical structures but rather are an existing part of the data warehouse. Because in theory the data warehouse contains the detail data of the entire enterprise, a logical view of the warehouse might provide the specific information for a given user community, much as a physical data mart would. Without the proper technology, a logical data mart can be a slow and frustrating experience for end users. With the proper technology, it removes the need for massive data loading and transforming, making a single data store available for all user needs.

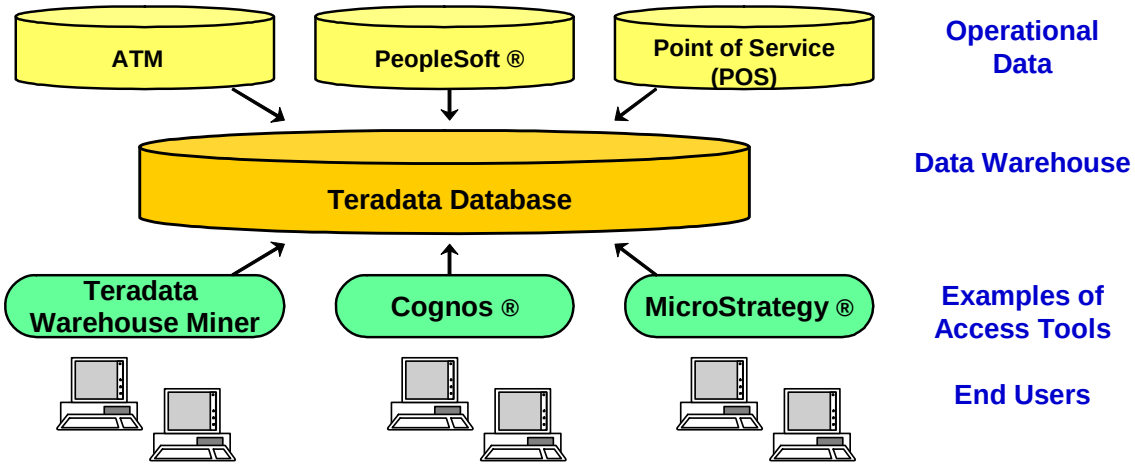
Dependent Data Marts

Dependent data marts are created from the detail data in the data warehouse. While having many of the advantages of the logical data mart, this approach still requires the movement and transformation of data but may provide a better vehicle for performance-critical user queries.

What is a Data Warehouse?

A Data Warehouse is a central, enterprise-wide database that contains information extracted from Operational Data Stores (ODS).

- Based on enterprise-wide model
- Can begin small but may grow large rapidly
- Populated by extraction/loading data from operational systems
- Responds to end-user "what if" queries
- Can store detailed as well as summary data



What is Active Data Warehousing?

The facing page provides a simple definition of Active Data Warehousing (ADW). Examples of why ADW is important (possibly mission critical applications) to different industries include:

- **Airlines** want an accurate view of customer value contribution so as to provide optimum customer service to the appropriate customer, whether or not they are frequent flyers.

- **Health care** organizations need to control costs, but not at the expense of jeopardizing quality of care. Proactive intervention programs where high-risk patients are identified and steered into case-management programs accomplish both.

- **Financial institutions** must fully understand a customer's profitability characteristics to automate appropriate and timely communications for increased revenue opportunity and/or better customer service.

- **Retailers** need to have a single, integrated view of each customer across multiple channels of opportunity - web, in-store, and catalog - to provide the right offer through the right vehicle.

- **Communications** companies must manage a constantly changing competitive environment and offer products and services to reduce customer churn rates.

One of the capabilities of ADW is to execute tactical queries in a timely fashion. Tactical queries are not the same as OLTP queries. Characteristics of a tactical query include:

- More read-oriented
- Focused on decision making
- More casual arrival rate than OLTP queries

Examples of tactical queries include determining the best offer for a customer or altering an advertising campaign based on current demand and results.

Another example of utilizing Active Data Warehousing is in the "Rental Car Business". Assume a service provider has a limited (relatively) fixed inventory of cars. The goal is to rent the maximum number of vehicles at the maximum price possible under the constraint that all prices offered exceed variable cost of the rental.

- Pricing can be determined by forecasting demand and price elasticity as it relates to demand

- Differentiated pricing is the ultimate yield management strategy

In order to do this, the business requires up to date, complete, and detailed data across the entire company.

What is Active Data Warehousing?

Data Warehousing … is the timely, integrated, logically consistent store of detailed data available for analytic business decision making.

- **Primarily batch feeds and updates**
- **Ad hoc (or decision support) queries to support strategic decisions that return in minutes and maybe hours**

Active Data Warehousing … is the timely, integrated, logically consistent store of detailed data available for strategic, tactical driven business decisions.

- **Timely updates – close to real time**
- **Short, tactical queries that return in seconds**
- **Event driven activity plus strategic queries**

Business requirements for an ADW (Active Data Warehouse)?

- **Performance – response within seconds**
- **Scalability – support for large data volumes, mixed workloads, and concurrent users**
- **Availability – 7 x 24 x 365**
- **Data Freshness – Accurate, up to the minute, data**

What is a Relational Database?

A **database** is a collection of permanently stored data that is used by an application or enterprise. A database contains **logically related data**. Basically, that means that the database was created with a purpose in mind. A database supports **shared access** by many users. A database also is **protected** to control access and **managed** to retain its value and integrity.

The key to understanding **relational databases** is the concept of the **table** made up of **rows** and **columns**.

A **column** always contains like data. In the example on the following page, the column named LAST NAME contains last name, and never anything else. The position of the column in the table is arbitrary.

A **row** is one instance of all the columns of a table. In our example, all of the information about a single employee is in one row. The sequence of the rows in a table is arbitrary.

Specifically, in a Relational Database, tables are defined as a named collection of one or more named columns by zero or more rows of related information.

Notice that each row of the table is about a person. There are no rows with data on two people, nor are there rows with information on anything other than people. This may seem obvious, but the concept underlying it is very important.

Each row represents an occurrence of an **entity** defined by the table. An **entity** is defined as a person, place or thing about which the table contains information. In this case the entity is the employee.

Primary Key

Tables, made up of rows and columns, represent **entities** or relationships. Entities are the people, places, things, or events that the Entity Tables Model. Each table holds only one kind of row, and each row is uniquely identified within a table by a **Primary Key (PK)**.

A Primary Key is **required**. A Primary Key **can be more than one column**. A Primary Key **uniquely identifies** each row in a table. **No duplicate values are allowed**. **Only one** Primary Key is allowed per table. The Primary Key for the EMPLOYEE table is the Employee number. No two employees can have the same number.

Because it is used to identify, the Primary Key **cannot be NULL**. There must be something in that field to uniquely identify each occurrence. Primary Key values **cannot be changed**. Historical information as well as relationships with other entities may be lost if a PK value is changed or re-used.

What is a Relational Database?

- A Relational Database consists of a set of logically related tables.
- A table is a two dimensional representation of data consisting of rows and columns.
- Each row is in the table uniquely identified by a Primary Key (PK) – 1 or more columns.
  - A PK cannot have duplicate values and cannot be NULL; only one per table.
  - A PK are considered "non-changing" values.
- A table may optionally have 1 or more Foreign Keys (FK).
  - A FK can be 1 or more columns, can have duplicate values, and allows NULLs
  - Each FK value must exist somewhere as a PK value

Column

Employee Table

EMPLOYEE NUMBER	MANAGER EMPLOYEE NUMBER	DEPT NUMBER	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SALARY AMOUNT
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	861015	631015	3945000
1008	1019	301	312102	Kanieski	Carol	870201	680517	3925000
1007	1005	?	432101	Villegas	Arnando	870102	470131	5970000
1003	0801	401	411100	Trader	James	860731	570619	4785000

Row →

This Employee table has 9 columns and 4 rows of sample data – one row per employee.
There is no prescribed order for the rows of the table.
There is only one row "format" for the entire table.
Missing data values are represented by "NULLs".

Answering Questions with a Relational Database

A relational database is a collection of **relational tables** stored in a single installation of a **relational database management system** (RDBMS). The words "management system" indicate that not only is this a relational database but also there is underlying software to provide additional functions that the industry expects. This includes transaction integrity, security, journaling, and other features that are expected of databases in general. The Teradata Database is a Relational Database Management System.

Relational databases do not use access paths to locate data, rather data connections are made by **data values**. In other words, data connections are made by matching values in one column with the values in a corresponding column in another table. This connection is referred to as a **JOIN** in relational terminology.

The diagram on the facing page show how the values in one table may be matched to values in another. Both tables have a column named "Department Number". That connection allows the database to answer questions like, "What is the name of the department in which an employee works?"

One reason relational databases are so powerful is that, unlike other databases, they are based on a mathematical model developed by Dr. Edgar Codd and implement a query language solidly founded in **set theory**.

To summarize, a relational database is a **collection of tables**. The data contained in the tables can be associated using data values, specifically, columns with **matching data values**.

Foreign Key

Relational Databases permit associations by data value across more than one table. **Foreign Keys (FKs)** model the relationships between entities.

On the facing page you will see that the employee table has 3 FK columns, one of which models the relationship between employees and their departments. A second one models the relationship between employees and their job codes.

A third FK column is used to model the relationship between employees and each other. This is called a "recursive" relationship.

Rules of Foreign Keys include:

- Duplicate values are allowed in a FK column.
- Missing values are allowed in a FK column.
- Values may be changed in a FK column.
- Each FK value must exist as a Primary Key.

Note that Dept_Number is the **Primary Key** for the DEPARTMENT table.

Answering Questions with a Relational Database

Employee (partial listing)

EMPLOYEE NUMBER	MANAGER EMPLOYEE NUMBER	DEPT NUMBER	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SALARY AMOUNT
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	861015	631015	3945000
1008	1019	301	312102	Kanieski	Carol	870201	680517	3925000
1005	0801	403	431100	Ryan	Loretta	861015	650910	4120000
1004	1003	401	412101	Johnson	Darlene	861015	560423	4630000
1007	1005	403	432101	Villegas	Arnando	870102	470131	5970000
1003	0801	401	411100	Trader	James	860731	570619	4785000

Department

DEPT NUMBER	DEPARTMENT NAME	BUDGET AMOUNT	MANAGER EMPLOYEE NUMBER
PK			FK
501	marketing sales	80050000	1017
301	research and development	46560000	1019
403	education	93200000	1005
402	software support	30800000	1011
401	customer support	98230000	1003

Questions:

1. **Name the department in which James Trader works.**
2. **Who manages the Education Department?**
3. **Identify by name an employee who works for James Trader.**

Teradata Database Competitive Advantages

As technology has improved, a number of aspects of the decision support environment have changed (improved). DSS systems are expected to:

- Store and efficiently process detailed data (reduces the need for summarized data).
- Process ad hoc queries in a timely fashion.
- Contain current (up-to-date) data.

Teradata meets these requirements. The facing page lists a number of the key competitive advantages that Teradata provides. This course will look at these features in detail and explain why these are competitive advantages.

Teradata provides a central, enterprise-wide database that contains information extracted from operational data stores. It provides for a single version of the business (or truth). Characteristics include:

- Based on enterprise-wide model – this type of model provides the ability to look/work across functional processes.
- Customers can begin small (right size), but may grow large rapidly
- Populated by extraction/loading of data from operational systems
- Allows end-users to submit "what if" queries

Examples of applications that Teradata enables include:

- Customer Relationship Management (CRM)
- Campaign Management
- Yield Management
- Supply Chain Management

Some of the reasons that Teradata is the leader in data warehousing include:

- Scalable – supports a small (10 GB) to a massive (Petabytes) database.
- Provides a query optimizer with approximately 30+ years of experience in large-table query planning.
- Does not require complex indexing schemes, complex data partitioning or time-consuming reorganizations (re-orgs).
- Supports ad hoc querying against the detail data in the warehouse, not just summary data in the data mart.
- Designed and built with parallelism from day one (not a parallel retrofit).

Teradata Database Competitive Advantages

- **Unlimited, Proven Scalability – amount of data and number of users; allows for an enterprise wide model of the data.**
- **Unlimited Parallelism – parallel access, sorts, and aggregations.**
- **Mature Optimizer – handles complex queries, up to 128 joins per query, ad-hoc processing.**
- **Models the Business – normalized data (usually in 3NF), robust view processing, & provides star schema capabilities.**
- **Provides a "single version of the business".**
- **Low TCO (Total Cost of Ownership) – ease of setup, maintenance, & administration; no re-orgs, lowest disk to data ratio, and robust expansion utility (reconfig).**
- **High Availability – no single point of failure.**
- **Parallel Load and Unload utilities – robust, parallel, and scalable load and unload utilities such as FastLoad, MultiLoad, TPump, and FastExport.**

Module 1: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 1: Review Questions

1. Which feature allows the Teradata Database to process enormous volumes of data quickly? ____
   a. High availability software and hardware components
   b. High performance servers from Intel
   c. Proven Scalability
   d. Parallelism

2. The Teradata Database is primarily a ____ .
   a. Client
   b. Server

3. Which choice represents a quadrillion bytes or a Petabyte (PB) of data? ____
   a. 10^9
   b. 10^{12}
   c. 10^{15}
   d. 10^{18}

4. In a relational table, the set of columns that uniquely identifies a row is the _____ _____.

Notes

Module 2

Teradata Basics

After completing this module, you will be able to:

- **List and describe the major components of the Teradata architecture.**
- **Describe how the components interact to manage incoming and outgoing data.**
- **List 5 types of Teradata database objects.**

Notes

Table of Contents

Major Components of Teradata 2-4
Teradata Storage Architecture 2-6
Teradata Retrieval Architecture 2-8
Multiple Tables on Multiple AMPs 2-10
    Here's how it works: 2-10
Linear Growth and Expandability 2-12
Teradata Objects 2-14
    Tables 2-14
    Views 2-14
    Macros 2-14
    Triggers 2-14
    Stored Procedures 2-14
The Data Dictionary Directory (DD/D) 2-16
Structure Query Language (SQL) 2-18
  Data Definition Language (DDL) 2-18
  Data Manipulation Language (DML) 2-18
  Data Control Language (DCL) 2-18
  User Assistance 2-18
CREATE TABLE – Example of DDL 2-20
Views 2-22
  Single-table View 2-22
Multi-Table Views 2-24
Macros 2-26
  Features of Macros 2-26
  Benefits of Macros 2-26
HELP Commands 2-28
SHOW Command 2-30
EXPLAIN Facility 2-32
Summary 2-34
Module 2: Review Questions 2-36

Major Components of Teradata

Up until now we have discussed relational databases in terms of how the user perceives them – as a collection of tables that relate to one another. Now it's time to describe the **components** of the system.

The major software components are the **Parsing Engine (PE)** and the **Access Module Processor (AMP)**.

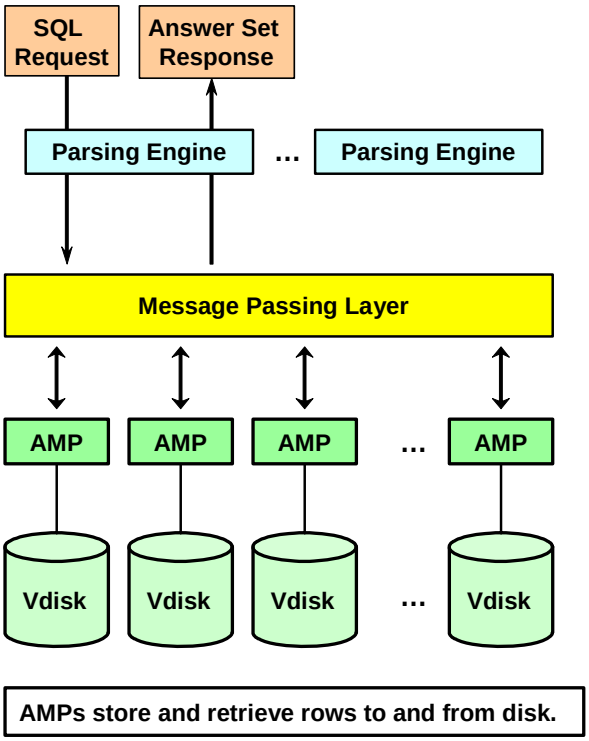
The **Parsing Engine** is a component that interprets SQL requests, receives input records and passes data. To do that it sends the messages through the Message Passing Layer to the AMPs.

The Message Passing Layer (MPL) handles the internal communication of the Teradata Database. The **MPL** is a combination of hardware and software (BYNET and PDE as we will see later). All communication between PEs and AMPs is done via the Message Passing Layer.

The **Access Module Processor (AMP)** is responsible for managing a portion of the database. An AMP will control some portion of each table on the system. AMPs do all of the physical work associated with generating an answer set including, sorting, aggregating, formatting and converting.

A **Virtual Disk** is disk space associated with an AMP. Tables/data rows are stored in this space. A virtual disk is usually assigned to two or more disk drives in a disk array. This concept will be discussed in detail later in the course.

Major Components of Teradata



AMPs store and retrieve rows to and from disk.

Parsing Engines (PE)

- Manage sessions for users
- Parse, optimize, and send your request to the AMPs as execution steps
- Returns answer set response back to client

Message Passing Layer (MPL)

- Allows PEs and AMPs to communicate with each other

Access Module Processors (AMP)

- Owns and manages its storage
- Performs the steps sent by the PEs

Virtual Disks (Vdisk)

- Space owned by the AMP and is used to hold user data (rows within tables).
- Maps to physical space in a disk array.

Teradata Storage Architecture

On the facing page you will see a simplified view of how the physical components of a Teradata database work to insert a row of data.

The PEs and AMPs are actually implemented as virtual processors (vprocs) in the system. A vproc is effectively a group of processes that represents a Teradata software component.

The **Parsing Engine** interprets the SQL command and converts the data record from the host into an AMP message.

- The Parsing Engine is a component that interprets SQL requests, receives input records and passes data. To do that it sends the messages through the Message Passing Layer to the AMPs.

The **Message Passing Layer** distributes the row to the appropriate Access Module Processor (AMP).

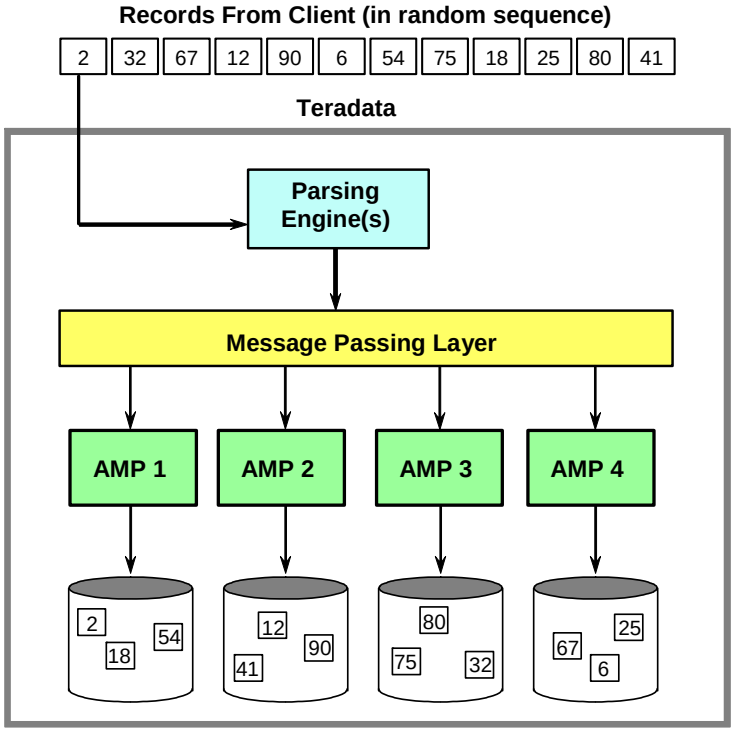
- The Message Passing Layer is implemented as hardware and/or software, depending on the platform used. It determines which vprocs should receive a message.

The **AMP** formats the row and writes it to its associated disks (Vdisks) which are assigned to physical disks in a disk array. The physical disk holds the row for subsequent access.

The **Host** or **Client system** supplies the records. These records are the raw data from which the database will be constructed.

Think of the **AMP** (**Access Module Processor)** as a independent computer designed for and dedicated to managing a portion of the entire database. It performs all the database management functions – such as sorting, aggregating, and formatting the data. It receives data from the PE, formats the rows, and distributes the rows to the disk storage units it controls. It also retrieves the rows requested by the parsing engine.

Teradata Storage Architecture



The Parsing Engine dispatches request to insert a row.

The Message Passing Layer insures that a row gets to the appropriate AMP (Access Module Processor).

The AMP stores the row on its associated (logical) disk.

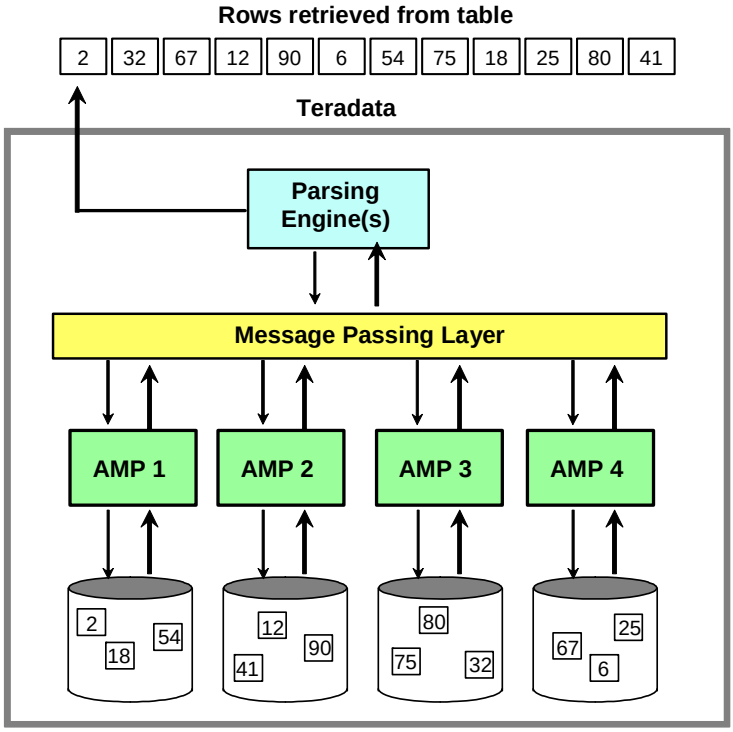
An AMP manages a logical or virtual disk which is mapped to multiple physical disks in a disk array.

Teradata Retrieval Architecture

Retrieving data from the Teradata Database simply reverses the process of the storage model. A request is made for data and is passed on to a Parsing Engine (PE). The PE optimizes the request for efficient processing and creates tasks for the AMPs to perform, which will result in the request being satisfied. These tasks are then dispatched to the AMPs via the Message Passing Layer. Often times all AMPs must participate in creating the answer set, such as in returning all rows of a table. Other times, only one or a few AMPs need participate, depending on the nature of the request. The PE will insure that only the AMPs that are needed will be assigned tasks on behalf of this request.

Once the AMPs have been given their assignments, they will retrieve the desired rows from their respective disks. If sorting, aggregating or formatting of any kind is needed, the AMPs will also take care of that. The rows are then returned to the requesting PE via the Message Passing Layer. The PE takes the returned answer set and returns it to the requesting client application.

Teradata Retrieval Architecture



The Parsing Engine dispatches a request to retrieve one or more rows.

The Message Passing Layer insures that the appropriate AMP(s) are activated.

The AMP(s) locate and retrieve desired row(s) in parallel access.

Message Passing Layer returns the retrieved rows to PE.

The PE returns row(s) to requesting client application.

Multiple Tables on Multiple AMPs

Logically, you might think that the Teradata Database would assign each table to a particular AMP, and that the AMP would put that table on a single disk. However, as you see on the diagram on the facing page, that's not what will happen. The system takes the rows that composes a table and divides those rows up among all available AMPs. TO MAKE IT PARALLEL!!!!!!!

Here's how it works:

Tables are distributed across all AMPs. This distribution of rows should be even across all AMPs. This way, a request to get the rows of a given table will result in the workload being evenly distributed across the AMPs.

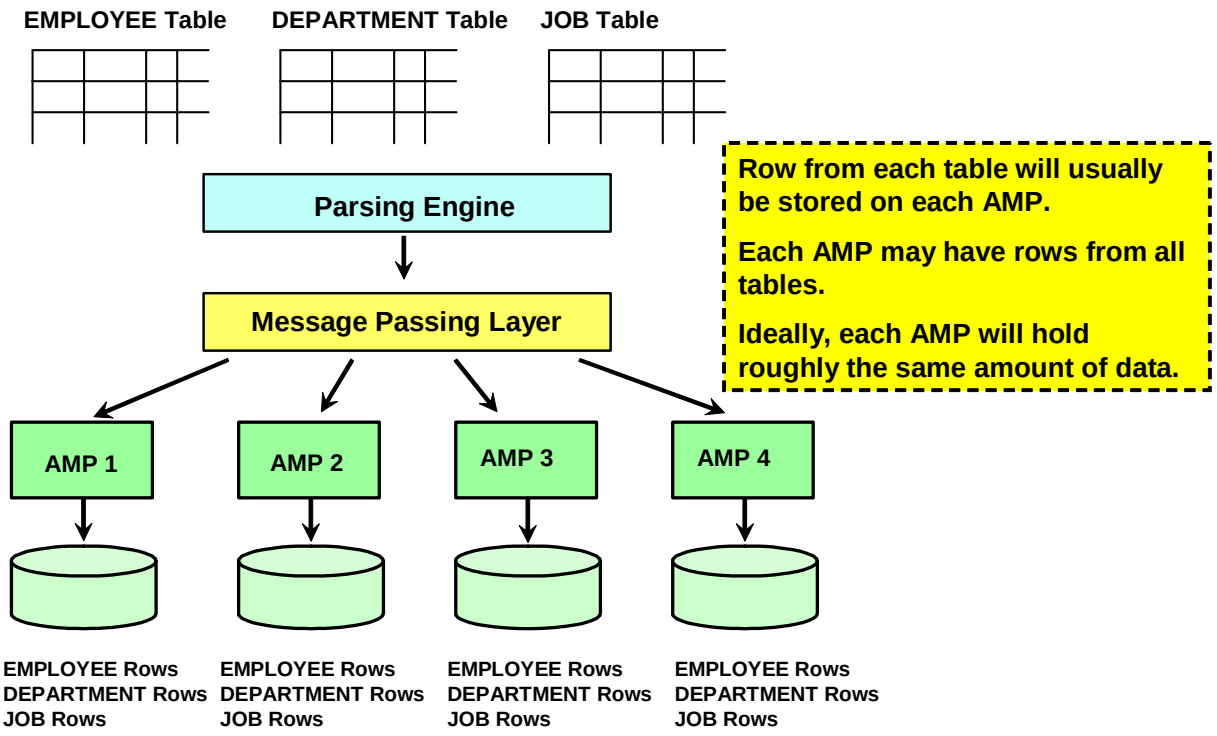
- Each table has some rows distributed to **each AMP**.
- Each AMP controls one **logical storage unit** which may consist of several physical disks VDISK
- Each AMP **places**, **maintains**, and **manages** the rows on its own disks.
- Large configurations may have **hundreds of AMPs.**
- **Full table scans**, operations that require looking at all the rows of a table, access all AMPs in parallel. That parallelism is what makes possible the accessing of enormous amounts of data. faster

Consider the following three tables: EMPLOYEE, DEPARTMENT, and JOB.

The Teradata Database takes the rows from each of the tables and divides them up among all the AMPs. The AMPs divide the rows up among their disks. Notice that **each** AMP gets part of **each** table. Dividing up the tables this way means that all the AMPs and their associated disks will be activated in a full table scan, thus speeding up requests against these tables.

In our example, if you assume four AMPs, each AMP would get approximately **25% of each table**. If, however, AMP #1 were to get 90% of the rows from the EMPLOYEE table that would be called "lumpy" data distribution. Lumpy data distribution would slow the system down because any request that required scanning all the rows of EMPLOYEE would have three AMPs sitting idle while AMP #1 finished its work. It is better to divide all the tables up evenly among all the available AMPs. You will see how this distribution is controlled in a later chapter.

Multiple Tables on Multiple AMPs



Linear Growth and Expandability

The Teradata DBS is the first commercial database system to offer true parallelism and the performance increase that goes with it.

Think back to the example of how rows are divided up among AMPs that we just discussed. Assume that our three tables, EMPLOYEE, DEPARTMENT, and JOB total 100,000 rows, with a certain number of users, say 50.

What happens if you **double the number of AMPs** and the number of users stays the same? Performance **doubles**. Each AMP can only work on half as many rows as they used to.

Now think of that system in a situation where the number of **users** is **doubled**, as well as the number of AMPs. We now have 100 users, but we also have twice as many AMPs. What happens to performance? It **stays the same**. There is no drop-off in the speed with which requests are executed.

That's because the system is **modular** and the workload is easily partitioned into independent pieces. In the last example, each AMP is still doing the same amount of work.

This feature – that the amount of time (or money) required to do a task is directly proportional to the size of the system – is unique to the Teradata Database. Traditional databases show a sharp drop in performance when the system approaches a critical size.

Look at the diagram on the facing page. As the number of **Parsing Engines** increases, the number of SQL requests that can be supported increases.

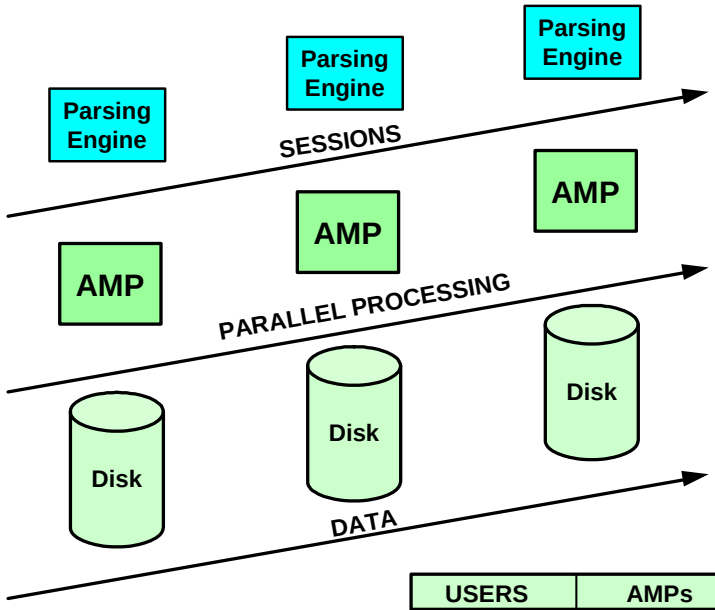
As you add **AMPs**, data is spread out more even as you add processing power to handle the data.

As you add **disks**, you add space for each AMP to store and process more information. All AMPs must have the same amount of disk storage space.

There are numerous advantages to having a system that has linear scalability. Two advantages include:

- Linear scalability allows for increased workload without decreased throughput.
- Investment protection for application development

Linear Growth and Expandability



- Teradata is a linearly expandable RDBMS.
- Components may be added as requirements grow.
- Linear scalability allows for increased workload without decreased throughput.
- Performance impact of adding components is shown below.

USERS	AMPs	DATA	Performance
Same	Same	Same	Same
Double	Double	Same	Same
Same	Double	Double	Same
Same	Double	Same	Double

Teradata Objects

A **"database"** or **"user"** in Teradata database systems is a collection of objects such as **tables**, **views**, **macros**, **triggers**, **stored procedures, user-defined functions,** or **indexes (join and hash).** Database objects are created and accessed using standard **Structured Query Language** or SQL.

All database object definitions are stored in a system database called **the Data Dictionary/Directory** (DD/D).

Databases provide a **logical grouping for information**. They are also the foundation for space allocation and access control. A description of some of the objects follows.

Tables

A table is the logical structure of data in a relational database. It is a **two-dimensional structure made up of columns and rows**. A user defines a table by giving it a table name that refers to the type of data that will be stored in the table. A **column** represents attributes of the table. Column names are given to each column of the table. All the information in a column is the same type, for example, date of birth. Each occurrence of an entity is stored in the table as a **row**. Entities are the people, things, or events that the table is about. Thus a row would represent a particular person, thing, or event.

Views

A view is a **pre-defined subset of one of more tables or other views**. It does not exist as a real table, but serves as a reference to existing tables or views. One way to think of a view is as a virtual table. Views have definitions in the data dictionary, but do not contain any physical rows. The database administrator can use views to control access to the underlying tables. Views can be used to hide columns from users, to insulate applications from database changes, and to simplify or standardize access techniques.

Macros

A macro is a predefined, stored set of one or more SQL commands and optionally, report formatting commands. Macros are used to simplify the execution of frequently used SQL commands.

Triggers

A trigger is a set of SQL statements usually associated with a column or a table and when that column changes, the trigger is fired – effectively executing the SQL statements.

Stored Procedures

A stored procedure is a program that is stored within Teradata and executes within the Teradata Database. A stored procedure uses permanent disk space.

A stored procedure is a pre-defined set of statements invoked through a single SQL CALL statement. Stored procedures may contain both Teradata SQL statements and procedural statements (in Teradata, referred to as Stored Procedure Language, or SPL).

Teradata Objects

Examples of objects within a Teradata database or user include:

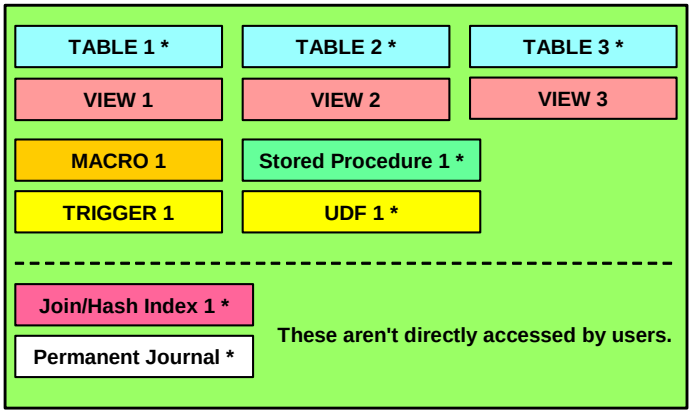
Tables – rows and columns of data
Views – predefined subsets of existing tables
Macros – predefined, stored SQL statements
Triggers – SQL statements associated with a table
Stored Procedures – program stored within Teradata
User-Defined Function – function (C or Java program) to provide additional SQL functionality
Join and Hash Indexes – separate index structures stored as objects within a database
Permanent Journals – table used to store before and/or after images for recovery

DATABASE or USER can have a mix of various objects.

* - require Permanent Space

These objects are created, maintained, and deleted using SQL.

Object definitions are stored in the DD/D.



The Data Dictionary Directory (DD/D)

The Data Dictionary/Directory is an integrated set of system tables which store database object definitions and accumulate information about **users**, **databases**, **resource usage**, **data demographics**, and **security rules**. It records specifications about **tables**, **views**, and **macros**. It also contains information about **ownership**, **space allocation**, **accounting**, and **access rights (privileges)** for these objects.

Data Dictionary/Directory information is updated automatically during the processing of Teradata SQL **data definition (DDL) statements**. It is used by the Parser to obtain information needed to process all Teradata SQL statements.

Users may access the DD/D through Teradata-supplied views, if permitted by the system administrator.

The Data Dictionary Directory (DD/D)

The DD/D ...

- is an integrated set of system tables
- contains definitions of and information about all objects in the system
- is entirely maintained by the Teradata Database
- is "data about the data" or "metadata"
- is distributed across all AMPs like all tables
- may be queried by administrators or support staff
- is normally accessed via Teradata supplied views

Examples of DD/D views:

DBC.TablesV	– information about all tables
DBC.UsersV	– information about all users
DBC.AllRightsV	– information about access rights
DBC.AllSpaceV	– information about space utilization

Structure Query Language (SQL)

Structured Query Language (SQL) is the language of relational databases. It is sometimes referred to as a "Fourth Generation Language (4GL)" to differentiate it from "Third Generation Languages" such as FORTRAN and COBOL, though it is quite different from other 4GL's. It acts as an intermediary between the user and the database.

SQL is different in some very important ways from other computer languages. Its statements resemble English-like structures. It provides powerful, set-oriented database manipulation including structural modification, data retrieval, modification, and security functions.

SQL is a non-procedural language. Because of its set orientation it does not require IF, GOTO, DO, FOR NEXT or PERFORM statements.

We'll describe three important subsets of SQL – the Data Definition Language, the Data Manipulation Language, and the Data Control Language.

Data Definition Language (DDL)

The DDL allows a user to define the **database objects and the relationships that exist among them**. Examples of DDL uses are creating or modifying tables and views.

Data Manipulation Language (DML)

The DML consists of the **statements that manipulate, change or retrieve the data rows** of the database. If the DDL defines the database, the DML lets the user change the information contained in the database. The DML is the most commonly used subset of SQL. It is used to select, update, delete, and insert rows.

Data Control Language (DCL)

The Data Control Language is used to **restrict or permit a user's access** in various ways. It can selectively limit a user's ability to retrieve, add, or modify data. It is used to grant and revoke access privileges on tables and views. An example is granting update privileges on a table, or read privileges on a view to specified users.

User Assistance

These commands allow you to list the objects in a database, or the characteristics of a table, see how a query will execute, or show you the details of your system. They vary widely from vendor to vendor.

Structured Query Language (SQL)

SQL is a query language for Relational Database Systems and is used to access Teradata.

- A fourth-generation language
- A set-oriented language
- A non-procedural language (e.g., doesn't have IF, DO, FOR NEXT, etc.)

SQL consists of:

Data Definition Language (DDL)

- Defines database structures (tables, users, views, macros, triggers, etc.)

CREATE DROP ALTER

Data Manipulation Language (DML)

- Manipulates rows and data values

SELECT INSERT UPDATE DELETE

Data Control Language (DCL)

- Grants and revokes access rights

GRANT REVOKE

Teradata SQL also includes Teradata Extensions to SQL

HELP SHOW EXPLAIN CREATE MACRO

CREATE TABLE – Example of DDL

To create and store the table structure definition in the DD/D, you can execute the CREATE TABLE DDL statement as shown on the facing page.

An example of the output from a SHOW TABLE command follows:

SHOW TABLE Employee;

```
CREATE SET TABLE Per_DB.Employee, FALLBACK,
      NO BEFORE JOURNAL,
      NO AFTER JOURNAL,
      CHECKSUM = DEFAULT,
      DEFAULT MERGEBLOCKRATIO
   (
     employee_number INTEGER NOT NULL,
     manager_emp_number INTEGER NOT NULL,
     dept_number INTEGER COMPRESS,
     job_code INTEGER COMPRESS ,
     last_name CHAR(20) NOT CASESPECIFIC NOT NULL,
     first_name VARCHAR(20) NOT CASESPECIFIC,
     hire_date DATE FORMAT 'YYYY-MM-DD'
     birth_date DATE FORMAT 'YYYY-MM-DD',
     salary_amount DECIMAL(10,2) COMPRESS 0
     )

UNIQUE PRIMARY INDEX  (employee_number)
INDEX  (dept_number);
```

You can create secondary indexes after a table has been created by executing the CREATE INDEX command. An example of creating an index for the **job_code** column is shown on the facing page.

Examples of the DROP INDEX and DROP TABLE commands are also shown on the facing page.

CREATE TABLE – Example of DDL

```
CREATE TABLE Employee
    (employee_number        INTEGER         NOT NULL
    ,manager_emp_number     INTEGER         COMPRESS
    ,dept_number            INTEGER         COMPRESS
    ,job_code               INTEGER         COMPRESS
    ,last_name              CHAR(20)        NOT NULL
    ,first_name             VARCHAR (20)
    ,hire_date              DATE            FORMAT 'YYYY-MM-DD'
    ,birth_date             DATE            FORMAT 'YYYY-MM-DD'
    ,salary_amount          DECIMAL (10,2)  COMPRESS 0
    )
UNIQUE PRIMARY INDEX (employee_number)
INDEX (dept_number);
```

Other DDL Examples

```
CREATE INDEX (job_code) ON Employee ;

DROP INDEX (job_code) ON Employee ;

DROP TABLE Employee ;
```

Views

A **view** is a pre-defined subset or filter of one or more tables. Views are used to control access to the underlying tables and simplify access to data. Authorized users may use views to read data specified in the view and/or to update data specified in the view.

Views are used to simplify query requests, to limit access to data, and to allow different users to look at the same data from different perspectives.

A view is a window that accesses selected portions of a database. Views can show parts of one table (single-table view), more than one table (multi-table view), or a combination of tables and other views. To the user, views look just like tables.

Views are **an alternate way of organizing and presenting information**. A view, like a table, has rows and columns. However, the rows and columns of a view are not stored directly but are derived from the rows and columns of tables whenever the view is referenced. A view looks like a table, but has no data of its own, and therefore takes up no storage space except for its definition. One way to think of a view is as if it was a window through which you can look at selected portions of a table or tables.

Single-table View

A **single-table view** takes specified columns and/or rows from a table and makes them available in a fashion that looks like a table. An example might be an employee table from which you select only certain columns for employees in a particular department number, for example, department 403, and present them in a view.

Example of a CREATE VIEW statement:

```
CREATE VIEW Emp403_v   AS
                SELECT      employee_number
                           ,department_number
                           ,last_name
                           ,first_name
                           ,hire_date
                FROM       Employee
                WHERE      department_number = 403;
```

It is also possible to execute SHOW VIEW viewname;

Views

Views are pre-defined filters of existing tables consisting of specified columns and/or rows from the table(s).

A single table view:

- **is a window into an underlying table**
- **allows users to read and update a subset of the underlying table**
- **has no data of its own**

EMPLOYEE (Table)

EMPLOYEE NUMBER	MANAGER EMP NUMBER	DEPT NUMBER	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SALARY AMOUNT
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	861015	631015	3945000
1008	1019	301	312102	Kanieski	Carol	870201	680517	3925000
1005	0801	403	431100	Ryan	Loretta	861015	650910	4120000
1004	1003	401	412101	Johnson	Darlene	861015	560423	4630000
1007	1005	403	432101	Villegas	Arnando	870102	470131	5970000
1003	0801	401	411100	Trader	James	860731	570619	4785000

Emp403_v (View)

EMP NO	DEPT NO	LAST NAME	FIRST NAME	HIRE DATE
1005	403	Villegas	Arnando	870102
801	403	Ryan	Loretta	861015

Multi-Table Views

A **multi-table view** combines data from more than one table into one pre-defined view. These views are also called **"join views"** because more than one table is involved.

An example might be a view that shows employees and the name of their department, information that comes from two different tables.

Note: **Multi-table Views are read only**. The user cannot update the data via the view.

One might wish to create a view containing the last name and department name for all employees.

A **Join** operation joins rows of multiple tables and creates rows in work space or spool. These are rows that contain data from more than one table but are not maintained anywhere in permanent storage. These rows in spool are created dynamically as part of a join operation. Rows are matched up based on Primary and Foreign Key relationships.

Example of SQL to create a join view:

```
CREATE VIEW  EmpDept_v AS
    SELECT    Last_Name
             ,Department_Name
    FROM      Employee  E INNER JOIN Department D
    ON        E.dept_number =  D.dept_number ;
```

An example of reading via this view is:

```
SELECT        Last_Name
             ,Department_Name
FROM          EmpDept_v;
```

This example utilizes an alias name of **E** for the **Employee** table and **D** for the **Department** table.

Multi-Table Views

A multi-table view allows users to access data from multiple tables as if it were in a single table. Multi-table views (i.e., join views) are used for reading only, not updating.

EMPLOYEE (Table)

EMPLOYEE NUMBER	MANAGER EMP NUMBER	DEPT NUMBER	JOB CODE	LAST NAME	FIRST NAME
PK	FK	FK	FK		
1006	1019	301	312101	Stein	John
1008	1019	301	312102	Kanieski	Carol
1005	0801	403	431100	Ryan	Loretta
1004	1003	401	412101	Johnson	Darlene
1007	1005	403	432101	Villegas	Arnando
1003	0801	401	411100	Trader	James

DEPARTMENT (Table)

DEPT NUMBER	DEPARTMENT NAME	BUDGET AMOUNT	MANAGER EMP NUMBER
PK			FK
501	Marketing Sales	80050000	1017
301	Research & Development	46560000	1019
302	Product Planning	22600000	1016
403	Education	93200000	1005
402	Software Support	30800000	1011
401	Customer Support	98230000	1003

Joined Together

Example of SQL to create a join view:

```
CREATE VIEW  EmpDept_v AS
SELECT       Last_Name
            ,Department_Name
FROM        Employee  E
INNER JOIN  Department D
ON          E.dept_number =  D.dept_number;
```

EmpDept_v (View)

Last_Name	Department_Name
Stein	Research & Development
Kanieski	Research & Development
Ryan	Education
Johnson	Customer Support
Villegas	Education
Trader	Customer Support

Macros

The **Macro** facility allows you to define a sequence of Teradata SQL statements (and optionally Teradata report formatting statements) so that they execute as a single transaction. Macros reduce the number of keystrokes needed to perform a complex task. This saves you time, reduces the chance of errors, reduces the communication volume to Teradata, and allows efficiencies internal to Teradata. Macros are a Teradata SQL extension.

Features of Macros

- Macros are source code stored on the DBC.
- They can be modified and executed at will.
- They are re-optimized at execution time.
- They can be executed by interactive or batch applications.
- They are executed by one **EXECUTE** command.
- They can accept user-provided parameter values.

Benefits of Macros

- Macros simplify and control access to the system.
- They enhance system security.
- They provide an easy way of installing referential integrity.
- They reduce the amount of source code transmitted from the client application.
- They are stored in the Teradata DD/D and are available to all connected hosts.

To create a macro:

```
CREATE MACRO Customer_List AS
(SELECT customer_name FROM Customer; );
```

To execute a macro:

```
EXEC Customer_List;
```

To replace a macro:

```
REPLACE MACRO Customer_List AS
(SELECT customer_name, customer_number FROM Customer; );
```

To drop a macro:

```
DROP MACRO Customer_List;
```

Macros

A MACRO is a predefined set of SQL statements which is logically stored in a database.

Macros may be created for frequently occurring queries of sets of operations.

Macros have many features and benefits:

- **Simplify end-user access**
- **Control which operations may be performed by users**
- **May accept user-provided parameter values**
- **Are stored in the Teradata Database, thus available to all clients**
- **Reduces query size, thus reduces LAN/channel traffic**
- **Are optimized at execution time**
- **May contain multiple SQL statements**

To create a macro:

```
CREATE MACRO Customer_List AS (SELECT customer_name FROM Customer;);
```

To execute a macro:

```
EXEC Customer_List;
```

To replace a macro:

```
REPLACE MACRO Customer_List AS
  (SELECT customer_name, customer_number FROM Customer;);
```

HELP Commands

HELP commands (a Teradata SQL extension) are available to display information on database objects:

- Databases and Users
- Tables
- Views
- Macros
- Triggers
- Join Indexes
- Hash Indexes
- Stored Procedures
- User-Defined Functions

The facing page contains an example of a HELP DATABASE command. This command lists the tables, views, macros, triggers, etc. in the specified database.

The **Kind** (TableKind) column codes represent the following:

T – Table
O – Table without a Primary Index
V – View
M – Macro
G – Trigger
P – Stored Procedure
F – User-defined Function
I – Join Index
N – Hash Index
J – Permanent Journal
A – Aggregate Function
B – Combined aggregate and ordered analytical function
D – JAR
E – External Stored Procedure
H – Instance or Constructor Method
Q – Queue Table
U – User-defined data type
X – Authorization

HELP Commands

Databases and Users

```
HELP DATABASE    Customer_Service;
HELP USER        Dave_Jones;
```

Tables, Views, Macros, etc.

```
HELP TABLE       Employee;
HELP VIEW        Emp_v;
HELP MACRO       Payroll_3;
HELP COLUMN      Employee.*;
                 Employee.last_name;
HELP INDEX       Employee;
HELP TRIGGER     Raise_Trigger;
HELP STATISTICS  Employee;
HELP CONSTRAINT  Employee.over_21;
HELP JOIN INDEX  Cust_Order_JI;
HELP SESSION;
```

This is not an inclusive list of HELP commands.

Example:

```
HELP DATABASE Customer_Service;

*** Help information returned. 15 rows.

*** Total elapsed time was 1 second.
```

Table/View/Macro name	Kind	Comment
Contact	T	?
Customer	T	?
Cust_Comp_Orders	V	?
Cust_Order_JI	I	?
Department	T	?
:	:	:
Orders	T	?
Orders_Temp	O	?
Orders_HI	N	?
Raise_Trigger	G	?
Set_Ansidate_on	M	?

SHOW Command

HELP commands display information about database objects (users/databases, tables, views, macros, triggers, and stored procedures) and session characteristics.

SHOW commands (another Teradata extension) display the data definition (DDL) associated with database objects (tables, views, macros, triggers, or stored procedures).

BTEQ contains a SHOW command, in addition to and separate from the SQL SHOW command. The BTEQ SHOW provides information on the formatting and display settings for the current BTEQ session, if applicable.

SHOW Command

SHOW commands display how an object was created. Examples include:

Command		Returns statement
SHOW TABLE	**table_name;**	**CREATE TABLE statement …**
SHOW VIEW	**view_name;**	**CREATE VIEW ...**
SHOW MACRO	**macro_name;**	**CREATE MACRO ...**
SHOW TRIGGER	**trigger_name;**	**CREATE TRIGGER …**
SHOW PROCEDURE	**procedure_name;**	**CREATE PROCEDURE …**
SHOW JOIN INDEX	**join_index_name;**	**CREATE JOIN INDEX …**

SHOW TABLE Employee;

```
CREATE SET TABLE PD.Employee, FALLBACK,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT,
    DEFAULT MERGEBLOCKRATIO
    (
    Employee_Number INTEGER NOT NULL,
    Emp_Mgr_Number INTEGER COMPRESS,
    Dept_Number INTEGER COMPRESS,
    Job_Code INTEGER COMPRESS,
    Last_Name CHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
    First_Name VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
    Salary_Amount DECIMAL(10,2) COMPRESS 0)
UNIQUE PRIMARY INDEX ( Employee_Number )
INDEX ( Dept_Number );
```

EXPLAIN Facility

The **EXPLAIN** facility (a very useful and robust Teradata extension) allows you to preview how Teradata will execute a query you have requested. It returns a summary of the steps the Teradata Database would perform to execute the request. **EXPLAIN** also discloses the strategy and access method to be used, how many rows will be involved, and its "cost" in minutes and seconds. You can use **EXPLAIN** to evaluate a query performance and to develop an alternative processing strategy that may be more efficient. **EXPLAIN** works on any SQL request. The request is fully parsed and optimized, but it is not run. Instead, the complete plan is returned to the user in readable English statements.

EXPLAIN also provides information about locking, sorting, row selection criteria, join strategy and conditions, access method, and parallel step processing.

There are a lot of reasons for using **EXPLAIN**. The main ones we've already pointed out – it lets you know how the system will do the job, what kind of results you will get back, and the relative cost of the query. **EXPLAIN** is also useful for performance tuning, debugging, pre-validation of requests, and for technical training.

The following is an example of an **EXPLAIN** on a very simple query doing a FTS (Full Table Scan).

EXPLAIN SELECT * FROM Employee WHERE Dept_Number = 1018;

Explanation (full)
---

1) First, we lock a distinct PD."pseudo table" for read on a RowHash to prevent global deadlock for PD.Employee.
2) Next, we lock PD.Employee for read.
3) We do an all-AMPs RETRIEVE step from PD.Employee by way of an all-rows scan with a condition of ("PD.Employee.Dept_Number = 1018") into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with high confidence to be 10 rows (730 bytes). The estimated time for this step is 0.14 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.14 seconds.

EXPLAIN Facility

The EXPLAIN modifier in front of any SQL statement generates an English translation of the Parser's plan.

The request is fully parsed and optimized, but not actually executed.

EXPLAIN returns:

- **Text showing how a statement will be processed (a plan)**
- **An estimate of how many rows will be involved**
- **A relative cost of the request (in units of time)**

This information is useful for:

- **predicting row counts**
- **predicting performance**
- **testing queries before production**
- **analyzing various approaches to a problem**

EXPLAIN SELECT * FROM Employee WHERE Dept_Number = 1018;

:

3) We do an all-AMPs RETRIEVE step from PD.Employee by way of an all-rows scan with a condition of ("PD.Employee.Dept_Number = 1018") into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with high confidence to be 10 rows (730 bytes). The estimated time for this step is 0.14 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.14 seconds.

Summary

The Teradata system is a high-performance database system that permits the processing of enormous quantities of detail data, quantities which are beyond the capability of conventional systems.

The system is **specifically designed for large relational databases**. From the beginning the Teradata system was created to do one thing: manage enormous amounts of data.

Over **one thousand terabytes of on-line storage capacity** is currently available making it an ideal solution for enterprise data warehouses or even smaller data marts.

Uniform data distribution across multiple processors facilitates **parallel processing**. The system is designed in such a way that the component parts divides the work up into approximately equal pieces. This keeps all the parts busy all the time; this enables the system to accommodate a larger number of users and/or more data.

Open architecture adapts readily to new technology. As higher-performance industry standard computer chips and disk drives are made available, they are easily incorporated into the architecture.

As the configuration grows, **performance increase is linear**.

Structured Query Language (SQL) is the industry standard for communicating with relational databases.

The Teradata Database currently runs as a **database server** on a variety of Linux, UNIX, and Windows based hardware platforms.

Summary

The major components of the Teradata Database are:

Parsing Engines (PE)

- Manage sessions for users
- Parse, optimize, and send your request to the AMPs as execution steps
- Returns answer set response back to client

Message Passing Layer (MPL)

- Allows PEs and AMPs to communicate with each other

Access Module Processors (AMP)

- Owns and manages its storage
- Performs the steps sent by the PEs

Virtual Disks (Vdisk)

- Space owned by the AMP and is used to hold user data (rows within tables).
- Maps to physical space in a disk array.

Module 2: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 2: Review Questions

1. What language is used to access a Teradata table?

2. What are five Teradata database objects?

3. What are four major components of the Teradata architecture?

4. What are views?

5. What are macros?

Notes

Module 3

Teradata Database Architecture

After completing this module, you will be able to:

- **Describe the purpose of the PE and the AMP.**
- **Describe the overall Teradata Database parallel architecture.**
- **Describe the relationship of the Teradata Database to its client side applications.**

Notes

Table of Contents

Teradata and MPP Systems....3-4
Teradata Functional Overview....3-6
Channel-Attached Client Software Overview....3-8
Network-Attached Client Software Overview....3-10
The Parsing Engine....3-12
Message Passing Layer....3-14
The Access Module Processor (AMP)....3-16
Teradata Parallelism....3-18
Module 3: Review Questions....3-20

Teradata and MPP Systems

Teradata is the software that makes a MPP system appear to be a single system to users and administrators.

The BYNET (BanYan NETwork) is the software and hardware interconnect that provides high performance networking capabilities to Teradata MPP (Massively Parallel Processing) systems.

Using communication switching techniques, the **BYNET** allows for **point-to-point, multi-cast**, and **broadcast** communications among the nodes, thus supporting a monumental increase in throughput in very large databases. This technology allows Teradata users to grow massively parallel databases without fear of a communications bottleneck for any database operations.

Although the BYNET software also supports the **multicast** protocol, Teradata software uses the **point-to-point** protocol whenever possible. When an all-AMP operation is needed, Teradata software uses the **broadcast** protocol to broadcast the request to the AMPs.

The BYNET is linearly scalable for point-to-point communications. For each new node added to the system, an additional 960 MB (with BYNET Version 4) of bandwidth is added, thus providing scalability as the system grows. Scalability comes from the fact that multiple point-to-point circuits can be established concurrently. With the addition of another node, more circuits can be established concurrently.

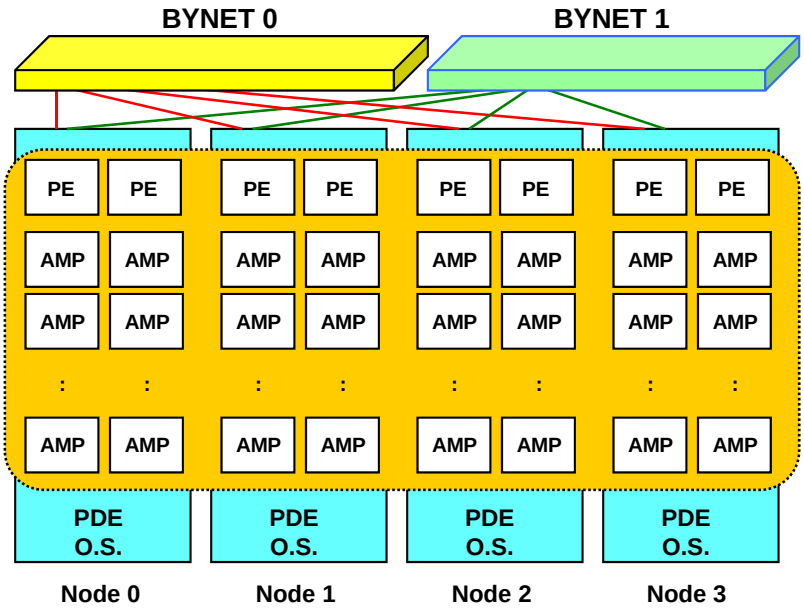
Teradata and MPP Systems

Teradata is the software that makes a MPP system appear to be a single system to users and administrators.

The major components of the Teradata Database are implemented as virtual processors (vproc).

- Parsing Engine (PE)
- Access Module Processor (AMP)

The Communication Layer or Message Passing Layer (MPL) consists of PDE and BYNET SW/HW and connects multiple nodes together.



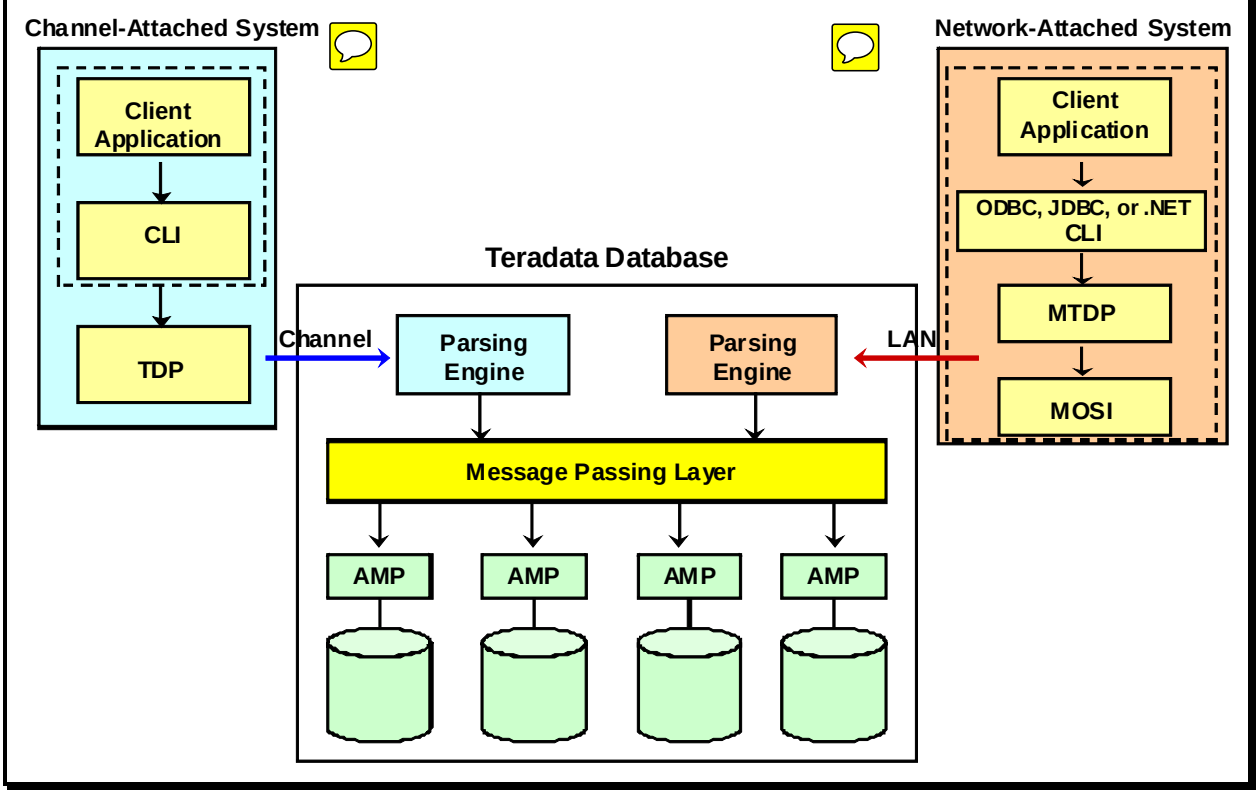
Teradata Functional Overview

The client may be a mainframe system (e.g., IBM) in which case it is **channel-attached** to the Teradata Database. Also, a client may be a PC or UNIX-based system that is **LAN or network-attached**.

The client application submits an SQL request to the Teradata Database, receives the response, and submits the response to the user.

The **Call Level Interface (CLI)** is a library of routines that resides on the client side. Client application programs use these routines to perform operations such as logging on and off, submitting SQL queries and receiving responses which contain the answer set. These routines are 98% the same in a network-attached environment as they are in a channel-attached.

Teradata Functional Overview



Channel-Attached Client Software Overview

In **channel-attached systems**, there are three major software components, which play important roles in getting the requests to and from the Teradata Database.

The **client application** is either written by a programmer or is one of Teradata's provided utility programs. Many client applications are written as "front ends" for SQL submission, but they also are written for file maintenance and report generation. Any client-supported language may be used provided it can interface to the Call Level Interface (CLI).

For example, a user could write a COBOL application with "embedded SQL". The application developer would have to use the Teradata COBOL Preprocessor and COBOL compiler programs to generate an object module and link this object module with the CLI. The CLI application interface provides maximum control over Teradata connectivity and access.

The **Call Level Interface (CLI)** is the lowest level interface to the Teradata Database. It consists of system calls which create sessions, allocate request and response buffers, create and de-block "parcels" of information, and fetch response information to the requesting client.

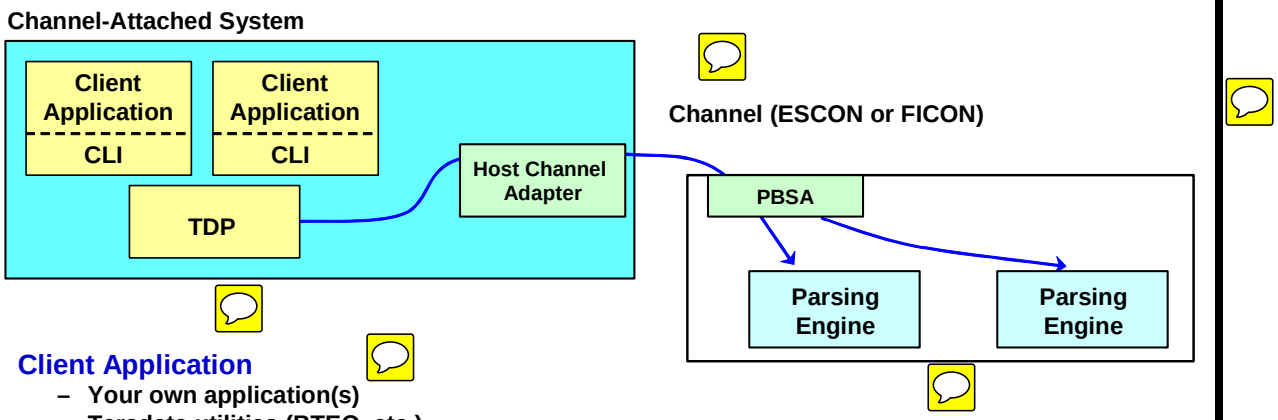
The **Teradata Director Program (TDP)** is a Teradata-supplied program that must run on any client system that will be channel-attached to the Teradata Database. The TDP manages the session traffic between the Call-Level Interface and the Database. Its functions include session initiation and termination, logging, verification, recovery, and restart, as well as physical input to and output from the PEs, (including session balancing) and the maintenance of queues. The TDP may also handle system security.

The **Host Channel Adapter** is a mainframe hardware component that allows the mainframe to connect to an ESCON or Bus/Tag channel.

The PBSA (PCI Bus ESCON Adapter) is a PCI adapter card that allows a Teradata server to connect to an ESCON channel.

The PBCA (PCI Bus Channel Adapter) is a PCI adapter card that allows a Teradata server to connect to a Bus/Tag channel.

Channel-Attached Client Software Overview



Client Application
- Your own application(s)
- Teradata utilities (BTEQ, etc.)

CLI (Call-Level Interface) Service Routines
- Request and Response Control
- Parcel creation and blocking/unblocking
- Buffer allocation and initialization

TDP (Teradata Director Program)
- Session balancing across multiple PEs
- Insures proper message routing to/from the Teradata Database
- Failure notification (application failure, Teradata restart)

`

Network-Attached Client Software Overview

In a network-attached environment, the SMPs running Teradata will typically have 1 or more Ethernet adapters that are used to connect to Teradata via a LAN connection. One of the key reasons for having multiple Ethernet adapters in a node is redundancy.

In **network-attached systems**, there are four major software components that play important roles in getting the requests to and from the Teradata Database.

The **client application** is written by the programmer using a client-supported language such as "C". The purpose of the application is usually to submit SQL statements to the Teradata Database and perform processing on the result sets. The application developer can "embed" SQL statements in the application and use the Teradata Preprocessor to interpret the embedded SQL statements.

In a networked environment, the application developer can use either the CLI interface or the ODBC driver to access Teradata.

The **Teradata CLI** application interface provides maximum control over Teradata connectivity and access. The ODBC and JDBC drivers are a much more open standard and are widely used with client applications.

The **Teradata ODBC™ (Open Database Connectivity)** or **JDBC (Java)** drivers use open standards-based ODBC or JDBC interfaces to provide client applications access to Teradata across LAN-based environments.

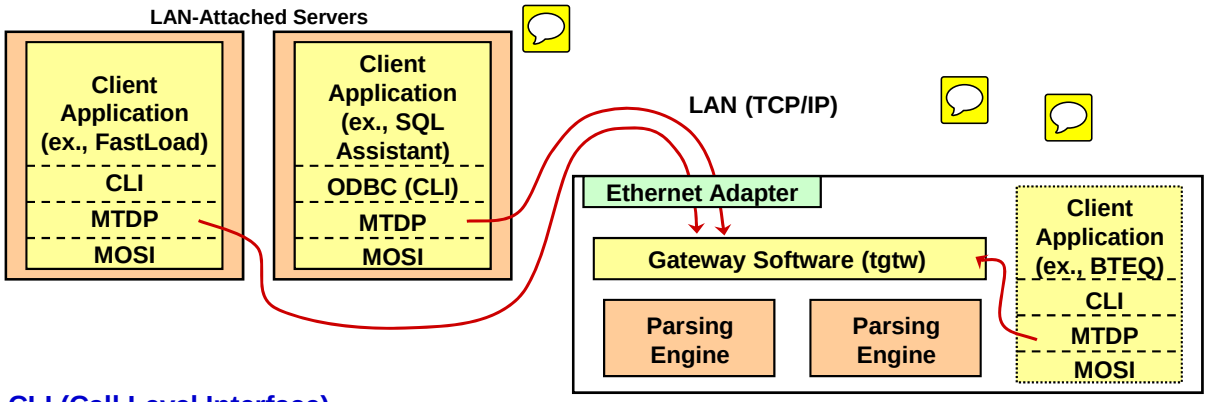
Note: ODBC 3.02.0 is the minimum certified version for Teradata V2R5.

The **Micro Teradata Director Program (MTDP)** is a Teradata-supplied program that must be linked to any application that will be network-attached to the Teradata Database. The MTDP performs many of the functions of the channel based TDP including session management. The MTDP does not control session balancing across PEs. Connect and Assign Servers that run on the Teradata system handle this activity.

The **Micro Operating System Interface (MOSI)** is a library of routines providing operating system independence for clients accessing the Teradata Database. By using MOSI, we only need one version of the MTDP to run on all network-attached platforms.

Teradata Gateway software executes on every node. Gateway software runs as a number of tasks. Two of the key tasks are called "ycgastsk" (assign task) and "ycgcntsk" (connect task). On a 4-node system with one gateway, only one node has the assign task (ycgastsk) running on it and every node will have the connect task (ycgcntsk) running on it. Initial session assignment is done by the assign task and will assign a user session to a PE and to the connect task in the same node as the PE. The connect task on a node will handle connections to the PEs on that node.

Network-Attached Client Software Overview



CLI (Call Level Interface)

- Library of routines for blocking/unblocking requests and responses to/from the Teradata Database

ODBC™ (Open Database Connectivity), JDBC™ (Java), or .NET Drivers

- Use open standards-based ODBC, JDBC, or .NET interfaces to provide client applications access to Teradata.

MTDP (Micro Teradata Director Program)

- Library of session management routines

MOSI (Micro Operating System Interface)

- Library of routines providing OS independent interface

The Parsing Engine

Parsing Engines (PEs) are made up of the following software components: session control, the Parser, the Optimizer, and the Dispatcher.

Once a valid session has been established, the PE is the component that manages the dialogue between the client application and the Teradata Database.

The major functions performed by **session control** are logon and logoff. Logon takes a textual request for session authorization, verifies it, and returns a yes or no answer. Logoff terminates any ongoing activity and deletes the session's context. When connected to an EBCDIC host the PE converts incoming data to the internal 8-bit ASCII used by the Teradata Database, thus allowing input values to be properly evaluated against the database data.

When a PE receives an SQL request from a client application, the **Parser** interprets the statement, checks it for proper SQL syntax and evaluates it semantically. The PE also must consult the Data Dictionary/Directory to ensure that all objects and columns exist and that the user has authority to access these objects.

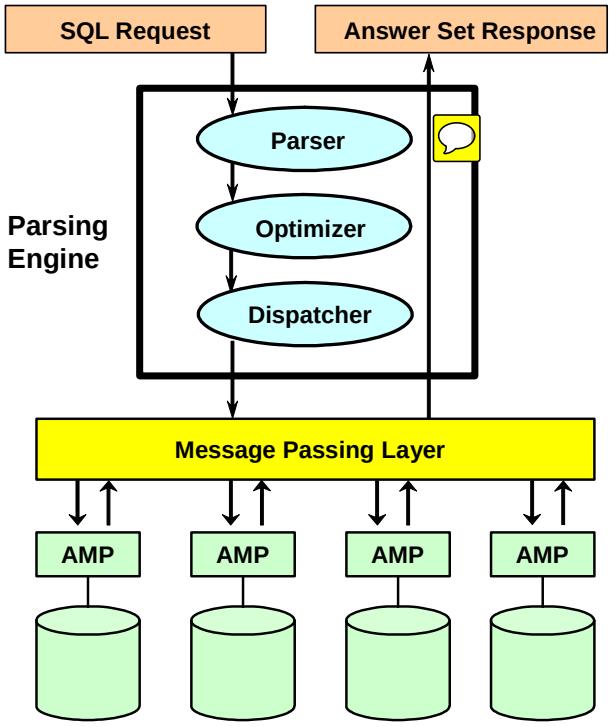
The **Optimizer's** role is to develop the least expensive plan to return the requested response set. Processing alternatives are evaluated and the fastest alternative is chosen. This alternative is converted to executable steps, to be performed by the AMPs, which are then passed to the dispatcher.

The **Dispatcher** controls the sequence in which the steps are executed and passes the steps on to the Message Passing Layer. It is composed of execution control and response control tasks. Execution control receives the step definitions from the Parser, transmits the step definitions to the appropriate AMP or AMPs for processing, receives status reports from the AMPs as they process the steps, and passes the results on to response control once the AMPs have completed processing. Response control returns the results to the user. The Dispatcher sees that all AMPs have finished a step before the next step is dispatched.

Depending on the nature of the SQL request, the step will be sent to one AMP, a few AMPs, or all AMPs.

Note: Teradata Gateway software can support up to 1200 sessions per processing node. Therefore a maximum of 10 Parsing Engines can be defined for a node using the Gateway.

The Parsing Engine



The Parsing Engine is responsible for:

- Managing individual sessions (up to 120)
- Parsing and Optimizing your SQL requests
- Dispatching the optimized plan to the AMPs
- Input conversion (EBCDIC / ASCII) - if necessary
- Sending the answer set response back to the requesting client

Message Passing Layer

The Message Passing Layer (MPL) or Communications Layer handles the internal communication of the Teradata Database. All communication between PEs and AMPs is done via the Message Passing Layer.

When the PE dispatches the steps for the AMPs to perform, they are dispatched onto the MPL. The messages are routed to the appropriate AMP(s) where results sets and status information are generated. This response information is also routed back to the requesting PE via the MPL.

The Message Passing Layer is a combination of the Teradata PDE software, the BYNET software, and the BYNET interconnect itself.

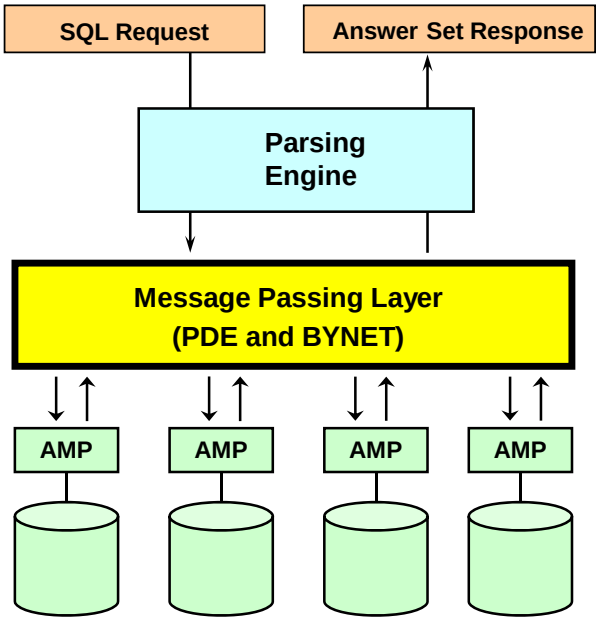
PDE and BYNET software - used for multi-node MPP systems and single-node SMP systems. With a single-node SMP, the BYNET device driver is used in conjunction with the PDE even though a physical BYNET network is not present.

Depending on the nature of the dispatch request, the communication may be a:

Broadcast - message is routed to all AMPs and PEs on the system
Multi-Cast - message is routed to a group of AMPs
Point-to-Point - message is routed to one specific AMP or PE on the system

The technology of the MPL is a key piece in the system part that makes possible the parallelism of the Teradata Database.

Message Passing Layer



The Message Passing Layer or Communications Layer is responsible for:

- Carrying messages between the AMPs and PEs
- Point-to-Point, Multi-Cast, and Broadcast communications
- Merging answer sets back to the PE
- Making Teradata parallelism possible

The Message Passing Layer or Communications Layer is a combination of:

- Parallel Database Extensions (PDE) Software
- BYNET Software
- BYNET Hardware for MPP systems

The Access Module Processor (AMP)

The **Access Module Processor (AMP)** is responsible for managing a portion of the database. An AMP will control some portion of each table on the system. AMPs do all of the physical work associated with generating an answer set including, sorting, aggregating, formatting and converting.

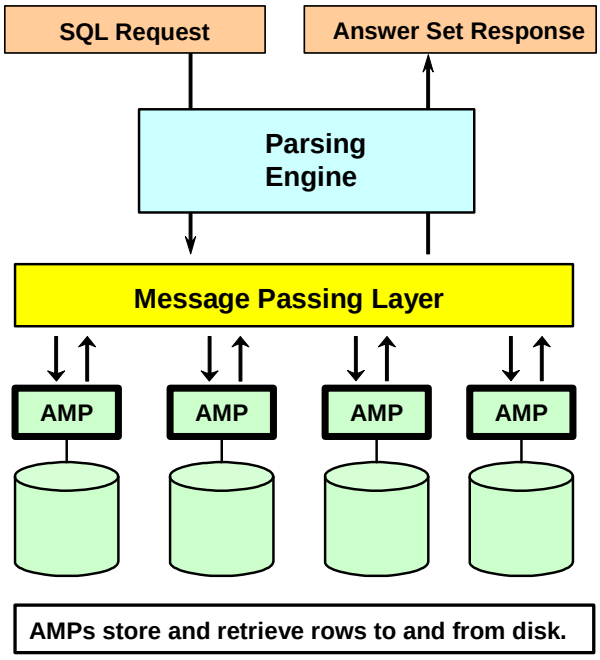
An AMP responds to Parser/Optimizer steps transmitted across the MPL by selecting data from or storing data to its disks. For some requests the AMPs may also redistribute a copy of the data to other AMPs.

The **Database Manager** subsystem resides on each AMP. It receives the steps from the Dispatcher and processes the steps. To do that it has the ability to **lock** databases and tables, to **create**, **modify**, or **delete definitions** of tables, to **insert, delete, or modify rows** within the tables, and to **retrieve information** from definitions and tables. It collects **accounting** statistics, recording accesses by session so those users can be billed appropriately. Finally, the Database manager returns responses to the Dispatcher.

Earlier in this course we discussed the **logical organization of data** into tables. The Database Manager provides a bridge between that logical organization and the **physical organization** of the data on disks. The Database Manager performs a **space management** function that controls the use and allocation of space.

AMPs also perform **output data conversion**, checking the session and changing the internal, 8-bit ASCII used by Teradata to the format of the requester. This is the reverse of the process performed by the PE when it converts the incoming data into internal ASCII.

The Access Module Processor (AMP)



The AMPs are responsible for:

- Accesses storage using Teradata's File System Software
- Lock management
- Sorting rows
- Aggregating columns
- Join processing
- Output conversion and formatting
- Creating answer set for client
- Disk space management
- Accounting
- Special utility protocols
- Recovery processing

Teradata File System Software:

- Translates DatabaseID/TableID/RowID into location on storage
- Controls a portion of physical storage
- Allocates storage space by “Cylinders”

Teradata Parallelism

Parallelism is at the very heart of the Teradata Database. There is virtually no part of the system where parallelism has not been built in. Without the parallelism of the system, managing enormous amounts of data would either not be possible or, best case, would be prohibitively expensive and inefficient.

Each PE can support up to 120 user sessions in parallel. This could be 120 distinct users, or a single user harnessing the power of all 120 sessions for a single application.

Each session may handle multiple requests concurrently. While only one request at a time may be active on behalf of a session, the session itself can manage the activities of 16 requests and their associated answer sets.

The Message Passing Layer was designed such that it can never be a bottleneck for the system. Because the MPL is implemented differently for different platforms, this means that it will always be well within the needed bandwidth for each particular platform's maximum throughput.

Each AMP can perform up to 80 tasks in parallel. This means that AMPs are not dedicated at any moment in time to the servicing of only one request, but rather are multi-threading multiple requests concurrently. The value 80 represents the number of AMP Worker Tasks and may be changed on some systems.

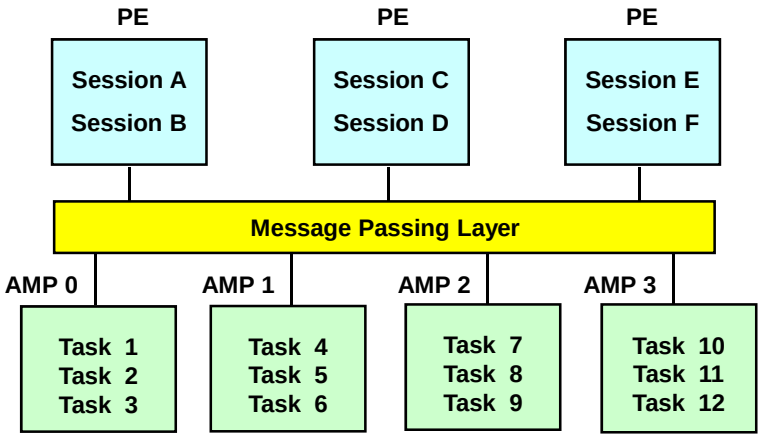
Because AMPs are designed to operate on only one portion of the database, they must operate in parallel to accomplish their intended results.

In addition to this, the optimizer may direct the AMPs to perform certain steps in parallel if there are no contingencies between the steps. This means that an AMP might be concurrently performing more than one step on behalf of the same request.

A recently added feature called Parallel CLI allows for parallelizing the client application, particularly useful for multi-session applications. This is accomplished by setting a few environmental variables and requires no changes to the application code.

In truth, parallelism is built into the Teradata Database from the ground up!

Teradata Parallelism



Parallelism is built into Teradata from the ground up!

Notes:

- Each PE can handle up to 120 sessions in parallel.
- Each Session can handle multiple REQUESTS.
- The Message Passing Layer can handle all message activity in parallel.
- Each AMP can perform up to 80 tasks in parallel.
- All AMPs can work together in parallel to service any request.
- Each AMP can work on several requests in parallel.

Module 3: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 3: Review Questions

1. What are the two software elements that accompany an application on all client side environments?

2. What is the purpose of the PE?

3. What is the purpose of the AMP?

4. How many sessions can a PE support?

Match Quiz

____ 1. CLI	a. Does Aggregating and Locking
____ 2. MTDP	b. Validates SQL syntax
____ 3. MOSI	c. Connects AMPs and PEs
____ 4. Parser	d. Balances sessions across PEs
____ 5. AMP	e. Provides Client side OS independence
____ 6. Message Passing Layer	f. Library of Session Management Routines
____ 7. TDP	g. PE S/W turns SQL into AMP steps
____ 8. Optimizer	h. PE S/W sends plan steps to AMP
____ 9. Dispatcher	i. Library of Teradata Service Routines
____10. Parallelism	j. Foundation of Teradata architecture

Notes

Module 4

Teradata Databases and Users

After completing this module, you will be able to:

- **Distinguish between a Teradata Database and Teradata User.**
- **Define Perm Space and explain how it is used.**
- **Define Spool Space and its use.**
- **Visualize the hierarchy of objects in a Teradata system.**

Notes

Table of Contents

A Teradata Database 4-4
- Tables 4-4
- Views 4-4
- Macros 4-4
- Triggers 4-4

A Teradata User 4-6
Database – User Comparison 4-8
The Hierarchy of Databases and Users 4-10
Example of a System Hierarchy 4-12
Permanent Space 4-14
Spool Space 4-16
Temporary Space 4-18
Creating Tables 4-20
Data Types 4-22
Access Rights and Privileges 4-24
Module 4: Review Questions 4-26

A Teradata Database

A Teradata database is a collection of tables, views, macros, triggers, stored procedures, join indexes, hash indexes, UDFs, access rights and space limits used for administration and security. All databases have a defined upper limit of permanent space. Permanent space is used for storing the data rows of tables. Perm space is not pre-allocated. It represents a maximum limit. All databases also have an upper limit of spool space. Spool space is temporary space used to hold intermediate query results or formatted answer sets to queries.

Databases provide a logical grouping for information. They are also the foundation for space allocation and access control. We'll review the definitions of tables, views, and macros.

Tables

A table is the logical structure of data in a database. It is a **two-dimensional structure made up of columns and rows**. A user defines a table by giving it a table name that refers to the type of data that will be stored in the table.

A **column** represents attributes of the table. Attributes identify, describe, or qualify the table. Column names are given to each column of the table. All the information in a column is the same type, for example, data of birth.

Each occurrence of an entity is stored in the table as a **row**. Entities are the people, things, or events that the table is about. Thus a row would represent a particular person, thing, or event.

Views

A view is a **pre-defined subset of one of more tables or other views**. It does not exist as a real table, but serves as a reference to existing tables or views. One way to think of a view is as a virtual table. Views have definitions in the data dictionary, but do not contain any physical rows. Views can be used by the database administrator to control access to the underlying tables. Views can be used to hide columns from users, to insulate applications from database changes, and to simplify or standardize access techniques.

Macros

A macro is a **definition containing one or more SQL commands and report formatting commands** that is stored in the Data Dictionary/Directory. Macros are used to simplify the execution of frequently-used SQL commands.

Triggers

A trigger consists of **one or more SQL statements that are associated with a table** and are executed when the trigger is “fired”.

A Teradata Database

A Teradata database is a defined logical repository for:

- Tables
- Views
- Macros
- Triggers
- Stored Procedures
- Join Indexes
- Hash Indexes
- Permanent Journals
- User-defined Functions (UDF)

Attributes that may be specified for a database:

- Perm Space – max amount of space available for tables, stored procedures, and UDFs
- Spool Space – max amount of work space available for requests
- Temp Space – max amount of temporary table space

A Teradata database is created with the CREATE DATABASE command.

Example

```
CREATE DATABASE Database_2 FROM Sysdba
        AS PERMANENT = 20E9, SPOOL = 500E6;
```

Notes:

"Database_2" is owned by "Sysdba".
A database is empty until objects are created within it.

A Teradata User

A **user** can also be thought of as a collection of tables, views, macros, triggers, stored procedures, join indexes, hash indexes, UDFs, **and** access rights.

A user is almost the same as a database except that a user can actually log on to the DBS. To accomplish this, a user must have a password. A user may or may not have perm space.

Even with no perm space, a user can access other databases depending on the privileges the user has been granted.

Users are created with the SQL statement **CREATE USER**.

A Teradata User

A Teradata user is a database with an assigned password.

A Teradata user may logon to Teradata and access objects within:

- **itself**
- **other databases for which it has access rights**

Examples of attributes that may be specified for a user:

- **Perm Space – max amount of space available for tables, stored procedures, and UDFs**
- **Spool Space – max amount of work space available for requests**
- **Temp Space – max amount of temporary table space**

A user is an active repository while a database is a passive repository.

A user is created with the CREATE USER command.

Example

```
CREATE USER User_C FROM User_A
       AS  PERMANENT = 100E6
           ,SPOOL = 500E6
           ,TEMPORARY = 150E6
           ,PASSWORD = lucky_day ;
```

"User_C" is owned by "User_A".
A user is empty until objects are created within it.

Database – User Comparison

In Teradata, a Database and a User are essentially the same. Database/User names must be unique within the entire system and represent the highest level of qualification in an SQL statement.

A User represents a logon point within the hierarchy and Access Rights apply only to Users. In many systems, end users do not have Perm space given to them. They are granted rights to access database(s) containing views and macros, which in turn are granted rights to access the corporate production tables.

At any time, another authorized User can change the Spool (workspace) limit assigned to a User.

Databases may be empty. They may or may not have any tables, views, macros, triggers, or stored procedures. They may or may not have Perm Space allocated. The same is true for **Users**. The only absolute requirement is that a User must have a password.

Once Perm Space is assigned, then and only then can tables be put into the database. Views, macros, and triggers may be added at any time, with or without Perm Space.

Remember that databases and users are both repositories for database objects. The main difference is the user ability to logon and acquire a session with the Teradata Database.

A row exists in DBC.Dbase for each User and Database.

Database – User Comparison

User	Database
Unique Name	Unique Name
Password = Value	
Define and use Perm space	Define and use Perm space
Define and use Spool space	Define Spool space
Define and use Temporary space	Define Temporary space
Set Fallback protection default	Set Fallback protection default
Set Permanent Journal defaults	Set Permanent Journal defaults
Multiple Account strings	One Account string
Logon and establish a session with a priority	
May have a startup string	
Default database, dateform, timezone, and default character set	
Collation Sequence	

- You can only LOGON as a known User to establish a session with Teradata.
- Tables, Join/Hash Indexes, Stored Procedures, and UDFs require Perm Space.
- Views, Macros, and Triggers are definitions in the DD/D and require NO Perm Space.
- A database (or user) with zero Perm Space may have views, macros, and triggers, but cannot have tables, join/hash indexes, stored procedures, or user-defined functions.

The Hierarchy of Databases and Users

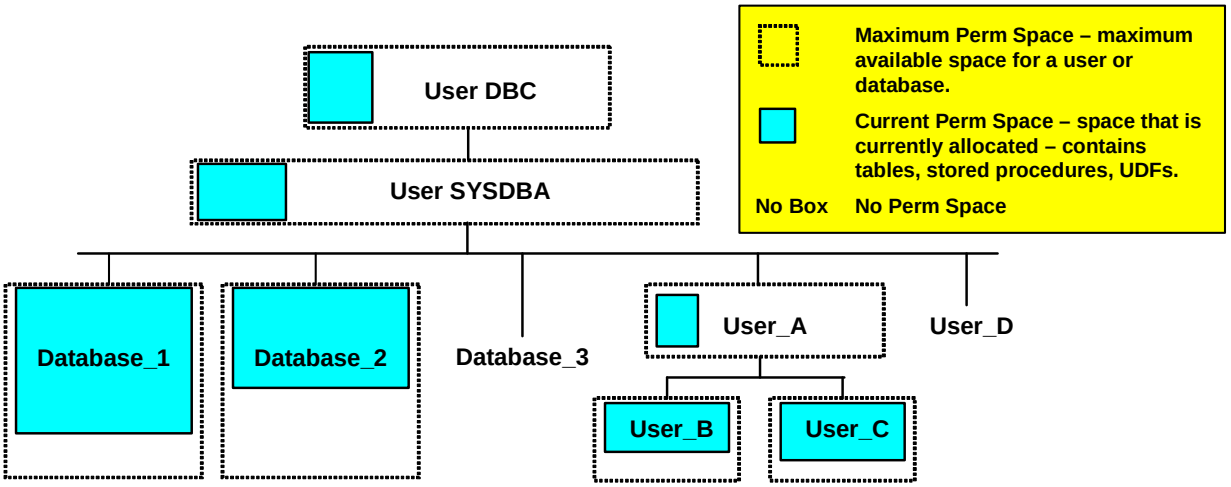
As you define users and databases, a hierarchical relationship among them will evolve.

When you create new objects, you subtract permanent space from the assigned limit of an existing database or user. A database or user that subtracts space from its own permanent space to create a new object becomes the immediate owner of that new object.

An “owner” or “parent” is any object above you in the hierarchy. (Note that you can use the terms owner and parent interchangeably.) A “child” is any object below you in the hierarchy. An owner or parent can have many children.

The term “immediate parent” is sometimes used to describe a database or user just above you in the hierarchy.

Hierarchy of Databases and Users



- A new database or user must be created from an existing database or user.
- All Perm space specifications are subtracted from the immediate owner or parent.
- Perm space is a zero sum game – the total of all Perm Space for all databases and users equals the total amount of disk space available to Teradata.
- Perm space is only used for tables, join/hash indexes, stored procedures, and UDFs.
- Perm space currently unused is available to be used as Spool or Temp space.

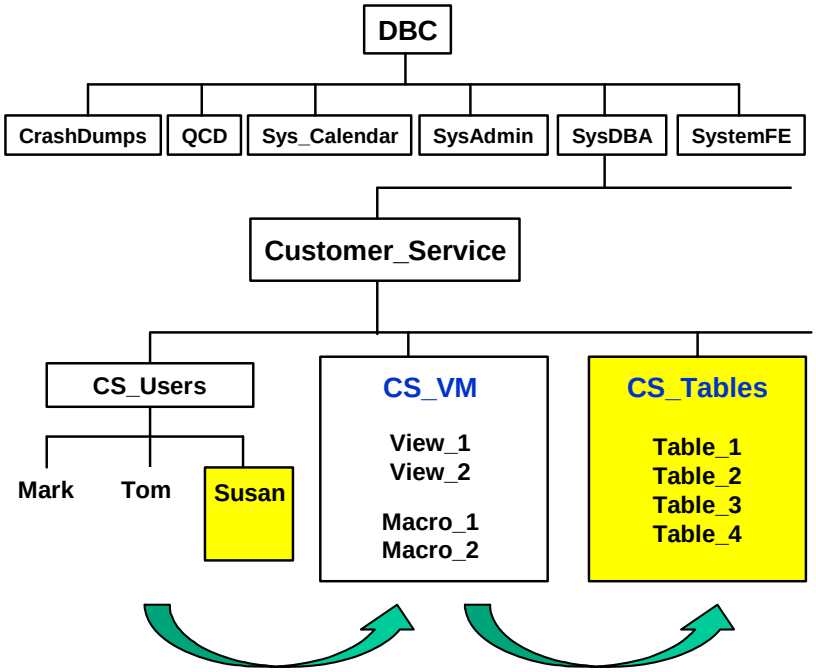
Example of a System Hierarchy

An example of a system structure for the Teradata database is shown on the facing page.

Example of a System Hierarchy

A User and/or a Database may be given PERM space.

In this example, Mark and Tom have no PERM space, but Susan does.



Users may use views and macros to access the actual tables.

Permanent Space

Permanent Space (Perm space) is the maximum amount of storage assigned to a user or database for holding table rows, Fallback tables, secondary index subtables, stored procedures, UDFs, and permanent journals.

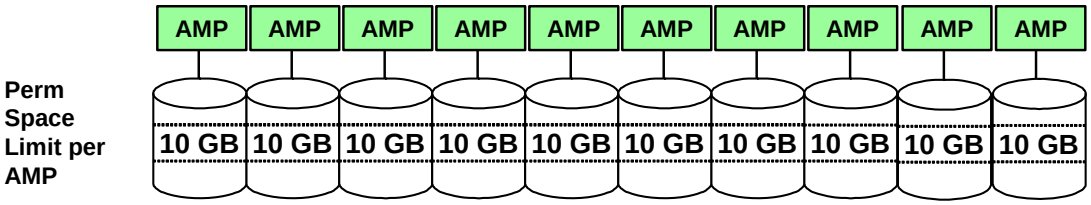
Perm space is specified in the CREATE statement as illustrated below. Perm space is not pre-allocated which means that it is available on demand, as entities are created not reserved ahead of time. Perm space is deducted from the owner's specified Perm space and is divided equally among the AMPs. Perm space can be dynamically modified.

The total amount of Perm space assigned divided by the number of AMPs equals the per-AMP limit. Whenever the per AMP limit is exceeded on any AMP, a **Database Full** message is generated.

```
CREATE DATABASE CS_Tables FROM Customer_Service AS
    PERMANENT = 100000000000 BYTES ... ;
```

Permanent Space

```
CREATE DATABASE CS_Tables FROM Customer_Service
    AS PERMANENT = 100E9 BYTES, ... ;
```



- Table rows, index subtable rows, join indexes, hash indexes, stored procedures, and UDFs use Perm space.
- Fallback protection uses twice the Perm space of No Fallback.
- Perm space is deducted from the owner's database space.
- Disk space is not reserved ahead of time, but is available on demand.
- Perm space is defined globally for a database.
- Perm space can be dynamically modified.
- The global limit divided by the number of AMPs is the per/AMP limit.
- The per/AMP limit cannot be exceeded.
- Good data distribution is crucial to space management.

Spool Space

Spool Space is work space acquired automatically by the system and used for work space and answer sets for intermediate and final results of Teradata SQL statements (e.g., SELECT statements generally use Spool space to store the SELECTed data). When the spool space is no longer needed by a query, it is released back to the system.

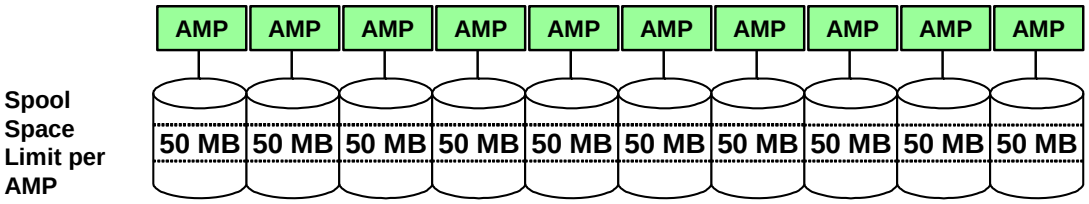
A **Spool limit** is specified in the CREATE statement shown below. This limit cannot exceed the Spool limit of the owner. However, a single user can create multiple databases or users, and each can have a Spool limit as large as the Spool limit of that owner.

The total amount of Spool space assigned divided by the number of AMPs equals the per AMP limit. Whenever the per-AMP limit is exceeded on any AMP, an **Insufficient Spool** message is generated to that client.

```
CREATE USER Susan FROM CS_Users AS
PERMANENT = 100000000 BYTES,
SPOOL = 500000000 BYTES,
PASSWORD = secret ... ;
```

Spool Space

```
CREATE USER Susan FROM CS_Users AS PERMANENT = 100E6 BYTES,
      SPOOL = 500E6 BYTES, PASSWORD = secret … ;
```



- **Spool space is work space acquired automatically by the system for intermediate query results or answer sets.**
  - **SELECT statements generally use Spool space.**
  - **Only INSERT, UPDATE, and DELETE statements affect table contents.**
- **The Spool limit cannot exceed the Spool limit of the original owner.**
- **The Spool limit is divided by the number of AMPS in the system, giving a per-AMP limit that cannot be exceeded.**
  - **"Insufficient Spool" errors often result from poorly distributed data or joins on columns with large numbers of non-unique values.**
  - **Keeping Spool rows small and few in number reduces Spool I/O.**

Temporary Space

Temporary (Temp) Space is temporary space acquired automatically by the system when Global Temporary tables are materialized and used.

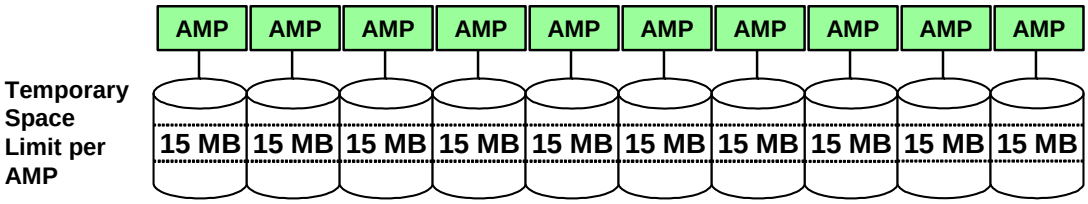
A **Temporary limit** is specified in the CREATE statement shown below. This limit cannot exceed the Temporary limit of the owner. However, a single user can create multiple databases or users, and each can have a Temporary limit as large as the Temporary limit of that owner.

The total amount of Temporary space assigned divided by the number of AMPs equals the per AMP limit. Whenever the per-AMP limit is exceeded on any AMP, an **Insufficient Temporary** message is generated to that client.

```
CREATE USER Susan FROM CS_Users AS
PERMANENT = 100000000 BYTES,
SPOOL = 500000000 BYTES,
TEMPORARY = 150000000 BYTES,
PASSWORD = secret ...
```

Temporary Space

```
CREATE USER Susan FROM CS_Users AS PERMANENT = 100E6 BYTES,
    SPOOL = 500E6 BYTES, TEMPORARY = 150E6 BYTES, PASSWORD = secret ... ;
```



- **Temporary space is space acquired automatically by the system when a "Global Temporary" table is used and materialized.**
- **The Temporary limit cannot exceed the Temporary limit of the original owner.**
- **The Temporary limit is divided by the number of AMPS in the system, giving a per-AMP limit that cannot be exceeded.**
  - **"Insufficient Temporary" errors often result from poorly distributed data or joins on columns with large numbers of non-unique values.**
- **Note: Volatile Temporary tables and derived tables utilize Spool space.**

Creating Tables

Creation of tables is done via the DDL portion of the SQL command vocabulary. The table definition, once accepted, is stored in the DD/D.

Prior to Teradata 13.0, creating tables required the definition of at least one column and the assignment of a Primary Index. With Teradata 13.0, it is possible to create tables without a primary index. Columns are assigned data types, attributes and optionally may be assigned constraints, such as a range constraint.

Tables, like views and macros, may be dropped when they are no longer needed. Dropping a table both deletes the data from the table and removes the definition of the table from the DD/D.

Secondary indexes may also optionally be assigned at table creation, or may be deferred until after the table has been built. Secondary indexes may also be dropped, if they are no longer needed. It is not uncommon to create secondary indexes to assist in the processing of a specific job sequence, then to delete the index, and its associated overhead, once the job is complete.

We will have more to say on indexes in general in future modules.

Creating Tables

Creating a table requires ...

- defining columns
- a primary index (Teradata 13.0 provides an option of a No Primary Index table)
- optional assignment of secondary indexes

```
CREATE TABLE Employee
                (Employee_Number      INTEGER NOT NULL
                ,Last_Name            CHAR(20) NOT NULL
                ,First_Name           VARCHAR(20)
                ,Salary_Amount        DECIMAL(10,2)
                ,Department_Number    SMALLINT
                ,Job_Code             CHAR(3))
Primary   →     UNIQUE PRIMARY INDEX (Employee_Number)
Secondary →     INDEX (Last_Name) ;
```

Database objects may be created or dropped as needed.

CREATE / DROP
- Tables
- Views
- Macros
- Triggers
- Procedures

Secondary indexes may be

- created at table creation
- created after table creation
- dropped after table creation

CREATE / DROP → INDEX (secondary only)

Data Types

When a table is created, a **data type** is specified for each column. Data types are divided into three classes – numeric, byte, and character. The facing page shows data types.

DATE is a 32-bit integer that represents the date as YYYYMMDD. It supports century and year 2000 and is implemented with calendar-based intelligence.

TIME WITH ZONE and **TIMESTAMP WITH ZONE** are ANSI standard data types that allow support of clock and time zone based intelligence.

DECIMAL (n, m) is a number of n digits, with m digits to the right of the decimal point.

BYTEINT is an 8-bit signed binary whole number that may vary in range from -128 to +127.

SMALLINT is a 16-bit signed binary whole number that may vary in range from -32,768 to +32,767.

INTEGER is a 32-bit signed binary whole number that may vary in size from -2,147,483,648 to +2,147,483,647.

BIGINT is a 64-bit (8 bytes) signed binary whole number that may vary in size from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 or as (-2^{63} to $2^{63} - 1$).

FLOAT, REAL, and **DOUBLE PRECISION** is a 64-bit IEEE floating point number.

BYTE (n) is a fixed-length binary string of n bytes. BYTE and VARBYTE are never converted to a different internal format. They can also be used for digitized objects.

VARBYTE (n) is a variable-length binary string of n bytes.

BINARY LARGE OBJECT (n) is similar to a VARBYTE; however it may be as large as 2 GB. A BLOB may be used to store graphics, video clips and binary files.

CHAR (n) is a fixed-length character string of n characters.

VARCHAR (n) is a variable-length character string of n characters.

LONG VARCHAR is the longest variable-length character string. It is equivalent to **VARCHAR (64000)**.

GRAPHIC, VARGRAPHIC and **LONG VARGRAPHIC** are the equivalent character types for multi-byte character sets such as Kanji.

CHARACTER LARGE OBJECT (n) is similar to a VARCHAR; however it may be as large as 2 GB. A CLOB may be used to store simple text, HTML, or XML documents.

Data Types

TYPE	Name	Bytes	Description
Date/Time	DATE	4	YYYYMMDD
	TIME (WITH ZONE)	6 / 8	HHMMSSZZ
	TIMESTAMP (WITH ZONE)	10 / 12	YYYYMMDDHHMMSSZZ
Numeric	DECIMAL or NUMERIC (n, m)	2, 4, 8 or 16	+ OR – (up to 18 digits V2R6.1 and prior) (up to 38 digits is V2R6.2 feature)
	BYTEINT	1	-128 to +127
	SMALLINT	2	-32,768 to +32,767
	INTEGER	4	-2,147,483,648 to +2,147,483,647
	BIGINT	8	-2^{63} to $+2^{63} - 1$ (+9,223,372,036,854,775,807)
	FLOAT, REAL, DOUBLE PRECISION	8	IEEE floating pt
Byte	BYTE(n)	0 – 64,000	
	VARBYTE (n)	0 – 64,000	
	BLOB	0 – 2 GB	Binary Large Object (V2R5.1)
Character	CHAR (n)	0 – 64,000	
	VARCHAR (n)	0 – 64,000	
	LONG VARCHAR		same as VARCHAR(64,000)
	GRAPHIC	0 – 32,000	
	VARGRAPHIC	0 – 32,000	
	LONG VARGRAPHIC		same as VARGRAPHIC(32,000)
	CLOB	0 – 2 GB	Character Large Object (V2R5.1)

Access Rights and Privileges

The diagram on the facing page shows access rights and privileges as they might be defined for the database administrator, a programmer, a user, a system operator, and an administrative user.

The **database administrator** has right to use all of the commands in the data definition privileges, the data manipulation privileges, and the data control privileges.

The **programmer** has all of those except the ability to GRANT privileges to others.

A typical **user** is limited to data manipulation privileges, while the **operator** is limited to data control privileges.

Finally, the **administrative user** is limited to a subset of data manipulation privileges, SELECT and EXECUTE.

Each site should carefully consider the access rules that best meet their needs.

Access Rights and Privileges

Data Definition Privileges

Command	Object
	Database and/or User
CREATE	Table and/or View
DROP	Macro and/or Trigger
	Stored Procedure
	Role and/or Profile

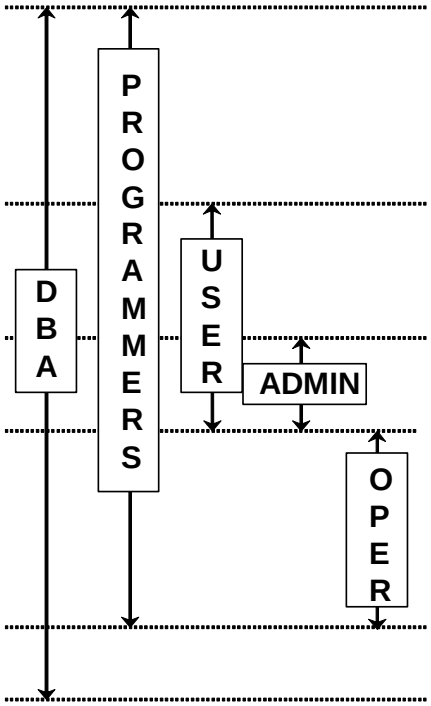
Data Manipulation Privileges

SELECT	
INSERT	Table
UPDATE	View
DELETE	
EXECUTE	Macro and/or Stored Procedure

Data Control Privileges

DUMP	Database
RESTORE	Table
CHECKPOINT	Journal
	Privileges on
GRANT	Databases
REVOKE	Users
	Objects

A Sample Scenario



Module 4: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 4: Review Questions

True or False

_____ 1. A database will always have tables.

_____ 2. A user will always have a password.

_____ 3. A user creating a subordinate user must give up some of his/her Perm Space.

_____ 4. Creating tables requires the definition of at least 1 column and a Primary Index.

_____ 5. The sum of all user and database Perm Space will equal the total space on the system.

_____ 6. The sum of all user and database Spool Space will equal the total space on the system.

_____ 7. Before a user can read a table, a database or table SELECT privilege must exist in the DD/D for that user.

_____ 8. Deleting a macro from a database reclaims Perm Space for the database.

9. Which statement is TRUE about PERM space? ____

a. PERM space cannot be dynamically modified.
b. The per/AMP limit of PERM space can be exceeded.
c. Tables, index subtables, and stored procedures use PERM space.
d. Maximum PERM space can be defined at the database or table level.

10. Which statement is TRUE about SPOOL space? ____

a. SPOOL space cannot be dynamically modified.
b. Maximum SPOOL space can be defined at the database or user level.
c. The SPOOL limit is dependent on the database limit where the table is located.
d. Maximum SPOOL space can be defined at a value greater than the immediate parent's value.

Notes

Module 5

TERADATA
Raising Intelligence

PI Access and Mechanics

After completing this module, you will be able to:

- **Explain the purpose of the Primary Index**
- **Distinguish between Primary Index and Primary Key**
- **Explain the role of the hashing algorithm and the hash map in locating a row.**
- **Explain the makeup of the Row ID and its role in row storage.**
- **Describe the sequence of events for locating a row given its PI value.**

Notes

Table of Contents

Primary Keys and Primary Indexes 5-4
Distribution of Rows 5-6
Specifying a Primary Index 5-8
Primary Index Values 5-10
Accessing Via a Unique Primary Index 5-12
Accessing Via a Non-Unique Primary Index 5-14
Row Distribution Using a Unique Primary Index (UPI) – Case 1 5-16
Row Distribution Using a Non-Unique Primary Index (NUPI) – Case 2 5-18
Row Distribution Using a Highly Non-Unique Primary Index (NUPI) – Case 3 5-20
Which AMP has the Row? 5-22
Hashing Down to the AMPs 5-24
A Hashing Example 5-26
The Hash Map 5-28
Hash Maps for Different Systems 5-30
Identifying Rows 5-32
The Row ID 5-34
Storing Rows (1 of 2) 5-36
  Storing Rows (2 of 2) 5-38
Locating a Row on an AMP Using a PI 5-40
Module 5: Review Questions 5-42

Primary Keys and Primary Indexes

While it is true that many tables use the same columns for both Primary Indexes and Primary Keys, **Indexes are conceptually different from Keys**. The table on the facing page summarizes those differences.

A Primary Key is relational data modeling term that defines, in the logical model, the columns that uniquely identify a row. A Primary Index is a physical database implementation term that defines the actual columns used to distribute and access rows in a table.

It is also true that a significant percentage of the tables in any database will use the same column(s) for both the PI and the PK. However, one should expect that in any real-world scenario there would be some tables that will not conform to this simplistic rule. Only through a careful analysis of the type of processing that will take place can the tables be properly evaluated for PI candidates. Remember, changing your mind about the columns that comprise the PI means recreating (and reloading) the table.

Primary Keys and Primary Indexes

- Indexes are conceptually different from keys.
- A PK is a relational modeling convention which allows each row to be uniquely identified.
- A PI is a Teradata convention which determines how the row will be stored and accessed.
- A significant percentage of tables may use the same columns for both the PK and the PI.
- A well-designed database will use a PI that is different from the PK for some tables.

Primary Key	Primary Index
Logical concept of data modeling	Physical mechanism for access and storage
Teradata doesn't need to recognize	Each table can have (at most) one primary index
No limit on number of columns	64 column limit
Documented in data model (Optional in CREATE TABLE)	Defined in CREATE TABLE statement
Must be unique	May be unique or non-unique
Identifies each row	Identifies 1 (UPI) or multiple rows (NUPI)
Values should not change	Values may be changed (Delete + Insert)
May not be NULL – requires a value	May be NULL
Does not imply an access path	Defines most efficient access path
Chosen for logical correctness	Chosen for physical performance

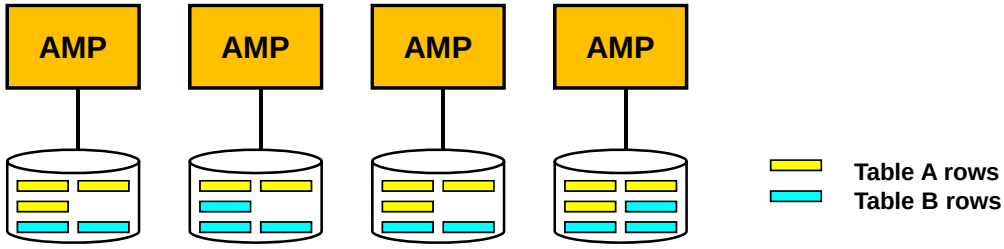
Distribution of Rows

Ideally, the rows of every table will be distributed among all of the AMPs. There may be some circumstances where this is not true. What if there are fewer rows than AMPs? Clearly in this case, at least some AMPs will hold no rows from that table. This should be considered the exceptional situation, and not the rule. Each AMP is designed to hold a portion of the rows of each table. The AMP is responsible for the storage, maintenance and retrieval of the data under its control.

More ideally, the rows of each table will be evenly distributed across all of the AMPs. This is desirable because in operations involving all rows of the table (such as a full table scan); each AMP will have an equal portion of the work to do. When workloads are not evenly distributed, the desired response will only be as fast as the slowest AMP.

Controlling the distribution of the rows of a table is done by the selection of the Primary Index. The relative uniqueness of the Primary Index will determine the uniformity of distribution of the rows of this table among the AMPs.

Distribution of Rows



- The rows of every table are distributed among all AMPs
- Each AMP is responsible for a subset of the rows of each table.
  - Ideally, each table will be evenly distributed among all AMPs.
  - Evenly distributed tables result in evenly distributed workloads.
- For tables with a Primary Index (majority of the tables), the uniformity of distribution of the rows of a table depends on the choice of the Primary Index. The actual distribution is determined by the hash value of the Primary Index.
- For tables without a Primary Index (Teradata 13.0 feature), the rows of a table are still distributed between the AMPs based on random generator code within the PE or AMP.
  - A small number of tables will typically be created as NoPI tables. Common uses for NoPI tables are as staging/intermediate tables used in load operations or as column partitioned tables.

Specifying a Primary Index

Choosing a **Primary Index** for a table is perhaps the most critical decision a database designer makes. The choice will affect the distribution of the rows of the table and, consequently, the performance of the table in a production environment. Although many tables used combined columns as the Primary Index choice, the examples used here are single column indexes, mostly for the sake of simplicity.

Unique Primary Indexes (UPI's) are desirable because they guarantee the uniform distribution of the rows of that table.

Because it is not always feasible to pick a Unique Primary Index, it is sometimes necessary to pick a column (or columns) which have non-unique values; that is there are duplicate values. This type of index is called a **Non-Unique Primary Index** or **NUPI**. While not a guarantor of uniform row distribution, the degree of uniqueness of the index will determine the degree of uniformity of the distribution. Because all rows with the same PI value end up on the same AMP, columns with a small number of distinct values which are repeated frequently typically do not make good PI candidates.

The choosing of a Primary Index is not an exact science. It requires analysis and thoughtfulness for some tables and will be completely self-evident on other tables.

The Primary Index is always designated as part of the **CREATE TABLE** statement. Once a Primary Index choice has been designated for a table, it cannot be changed to something else. If an alternate choice of column(s) is desired for the PI, it is necessary to drop and recreate the table.

Teradata, adhering to the ANSI standard, permits duplicate rows by specifying that you wish to create a **MULTISET** table. In Teradata transaction mode, the default, however, is a **SET** table that does not permit duplicate rows.

Also, if MULTISET is enabled, it will be overridden by choosing a UPI as the Primary Index or by having a unique index (e.g., unique secondary) on another column(s) on the table. Doing this effectively disables the MULTISET.

Multiset tables will be covered in more detail later in the course.

Starting with Teradata 13.0, the option of NO PRIMARY INDEX is also available.

Specifying a Primary Index

- A Primary Index is defined at table creation.
- It may consist of a single column, or a combination of columns (up to 64 columns)
  - With Teradata 13.0, an option of NO PRIMARY INDEX is available.

UPI

```
CREATE TABLE sample_1
            (col_a      INTEGER
            ,col_b      CHAR(10)
            ,col_c      DATE)
UNIQUE PRIMARY INDEX (col_b);
```

If the index choice of column(s) is unique, then this is referred to as a UPI (Unique Primary Index).

A UPI choice will result in even distribution of the rows of the table across all AMPs.

NUPI

```
CREATE TABLE sample_2
            (col_m      INTEGER
            ,col_n      CHAR(10)
            ,col_o      DATE)
PRIMARY INDEX (col_m);
```

If the index choice of column(s) isn't unique, then this is referred to as a NUPI (Non-Unique Primary Index).

The distribution of the rows of the table is proportional to the degree of uniqueness of the index.

NoPI

```
CREATE TABLE sample_3
            (col_x      INTEGER
            ,col_y      CHAR(10)
            ,col_z      DATE)
NO PRIMARY INDEX;
```

A NoPI choice will result in distribution of the data between AMPs based on random generator code.

Note: Changing the choice of Primary Index requires dropping and recreating the table.

Primary Index Values

Indexes are used to access rows from a table without having to search the entire table.

On Teradata, the **Primary Index** is the mechanism for assigning a data row to an AMP and a location on the AMP's disks. Prior to Teradata 13.0, when a table is created, a table must have a Primary Index specified (either user-assigned or Teradata assigned). This cannot be changed without dropping and creating the table.

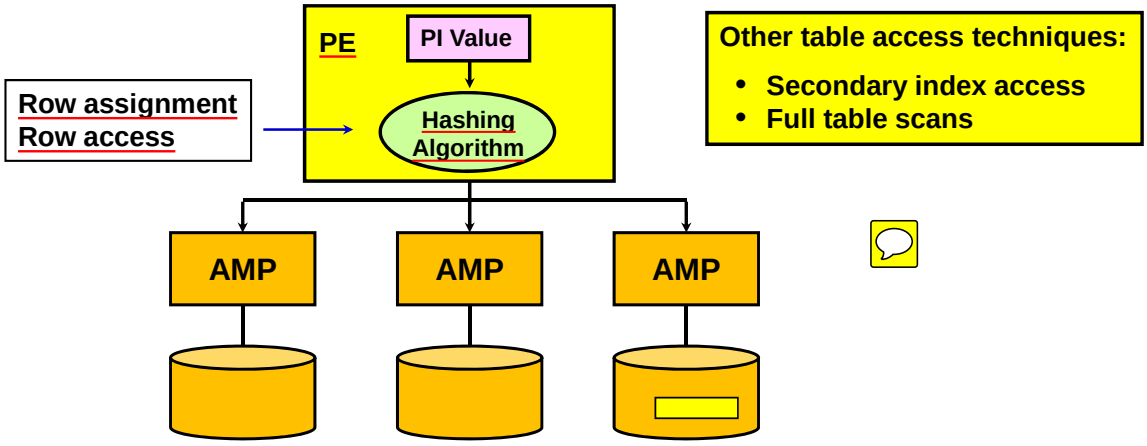
Primary Indexes are very important because they have a powerful effect on the performance of the database. The most important thing to remember is that a Primary Index is the mechanism used to assign each row to an AMP and may be used to retrieve that row from the AMP. Thus retrievals, updates and deletes that specify the Primary Index value will be much faster than those queries that do not specify the PI value. Primary Index selection is probably the most important factor in the efficiency of join processing.

Earlier we learned that the Primary Key was always unique and unchanging. This is based on the **logical model** of the data. The Primary Index may (and frequently is) be different than the Primary Key and may be non-unique; it is chosen for **the physical performance** of the database.

There are three types of primary index selection – **unique** (**UPI)**, **non-unique (NUPI**), or **NO PRIMARY INDEX**.

Primary Index Values

- The value of the Primary Index for a specific row determines the AMP assignment for that row.
- This is done using a hashing algorithm.



- Accessing the row by its Primary Index value is:
  - always a one-AMP operation
  - the most efficient way to access a row

Accessing Via a Unique Primary Index

A **Primary Index operation** is always a one-AMP operation. In the case of a UPI, the one-AMP access can return, at most, one row. In the facing example, we are looking for the row whose primary index value is 345. By specifying the PI value as part of our selection criteria, we are guaranteed that only the AMP containing the specified row will need to be searched.

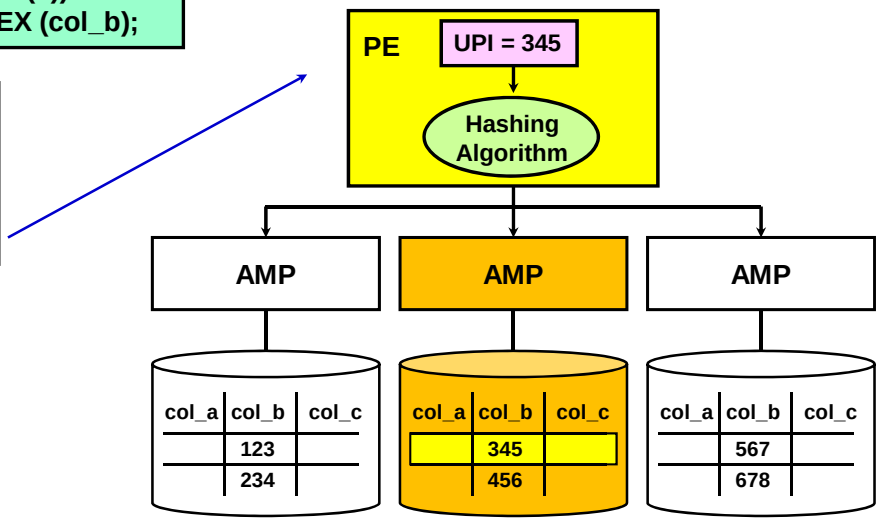
The correct AMP is located by taking the PI value and passing it through a hashing algorithm. The hashing takes place in the Parsing Engine. The output of the hashing algorithm contains information that will point the request to a specific AMP. Once it has isolated the appropriate AMP, finding the row is quick and efficient. How this happens we will see in a future module.

Accessing Via a Unique Primary Index

A UPI access is a one-AMP operation which may access at most a single row.

```
CREATE TABLE sample_1
            (col_a     INTEGER
            ,col_b     INTEGER
            ,col_c     CHAR(4))
UNIQUE PRIMARY INDEX (col_b);
```

```
SELECT   col_a
        ,col_b
        ,col_c
FROM     sample_1
WHERE    col_b = 345;
```



Accessing Via a Non-Unique Primary Index

A Non-Unique **Primary Index operation** is also a one-AMP operation. In the case of a NUPI, the one-AMP access can return zero to many rows. In the facing example, we are looking for the rows whose primary index value is 25. By specifying the PI value as part of our selection criteria, we are once again guaranteeing that only the AMP containing the required rows will need to be searched.

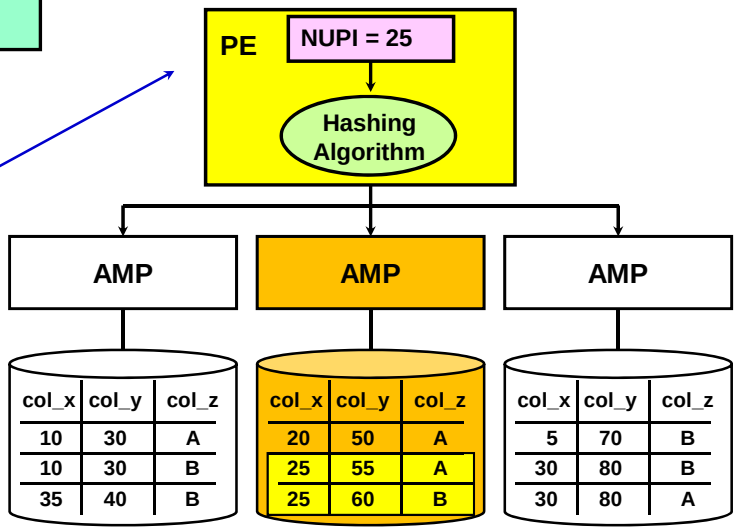
As before, the correct AMP is located by taking the PI value and passing it through a hashing algorithm executing in the Parsing Engine. The output of the hashing algorithm will once again point to a specific AMP. Once it has isolated the appropriate AMP, it must now find all rows that have the specified value. In this example, the AMP returns two rows.

Accessing Via a Non-Unique Primary Index

A NUPI access is a one-AMP operation which may access multiple rows.

```
CREATE TABLE sample_2
            (col_x     INTEGER
            ,col_y     INTEGER
            ,col_z     CHAR(4))
PRIMARY INDEX (col_x);
```

```
SELECT   col_x
        ,col_y
        ,col_z
FROM     sample_2
WHERE    col_x = 25;
```



Both UPI and NUPI accesses are one AMP operations.

Row Distribution Using a Unique Primary Index (UPI) – Case 1

At the heart of the Teradata database is a way of predictably distributing and retrieving rows across AMPs. The same value stored in the same data type will always produce the same hash value. If the Primary Index is unique, Teradata can distribute the rows evenly. If the Primary Index is slightly non-unique, that is, there are only four or five rows per index value; the table will still distribute evenly. But if there are hundreds or thousands of rows for some index values the distribution will probably be lumpy.

In this example, the Order_Number is used as a unique primary index. Since the primary index value for Order_Number is unique, the distribution of rows among AMPs is very uniform. This assures maximum efficiency because each AMP is doing approximately the same amount of work. No AMPs sit idle waiting for another AMP to finish a task.

This way of storing the data provides for maximum efficiency and makes the best use of the parallel features of the Teradata system.

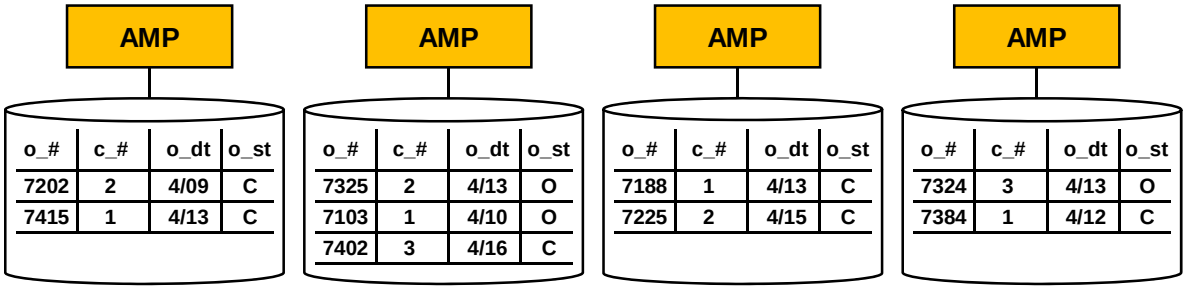
Row Distribution Using a UPI – Case 1

Orders

Order Number	Customer Number	Order Date	Order Status
PK			
UPI			
7325	2	4/13	O
7324	3	4/13	O
7415	1	4/13	C
7103	1	4/10	O
7225	2	4/15	C
7384	1	4/12	C
7402	3	4/16	C
7188	1	4/13	C
7202	2	4/09	C

Notes:

- Often, but not always, the PK column(s) will be used as a UPI.
  - Order_Number can be a UPI since all the values are unique.
- Teradata will distribute different UPI values evenly across all AMPs.
  - Resulting row distribution among AMPs is very uniform.
  - Assures maximum efficiency for parallel operations.



Row Distribution Using a Non-Unique Primary Index (NUPI) – Case 2

In the example on the facing page **Customer_Number** has been used as a non-unique Primary Index (NUPI). Note row distribution among AMPs is uneven. All rows with the same primary index value (in other words, with the same customer number) are stored on the same AMP.

Customer_Number has three possible values, so all the rows are hashed to three AMPs, leaving the fourth AMP without rows from this table. While this distribution will work, it is not as efficient as spreading all the rows among all the AMPs.

AMP 2 has a disproportionate number of rows and AMP 3 has none. In an all-AMP operation AMP 2 will take longer than the other AMPs. The operation cannot complete until AMP 2 completes its tasks. The overall operation time is increased and some of the AMPs are under-utilized.

NUPI's can create irregular distributions, called "skewed distributions". AMPs that have more than an average number or rows will take longer for full table operations than the other AMPs will. Because an operation is not complete until all AMPs have finished, this will cause the operation to finish less quickly due to being underutilized.

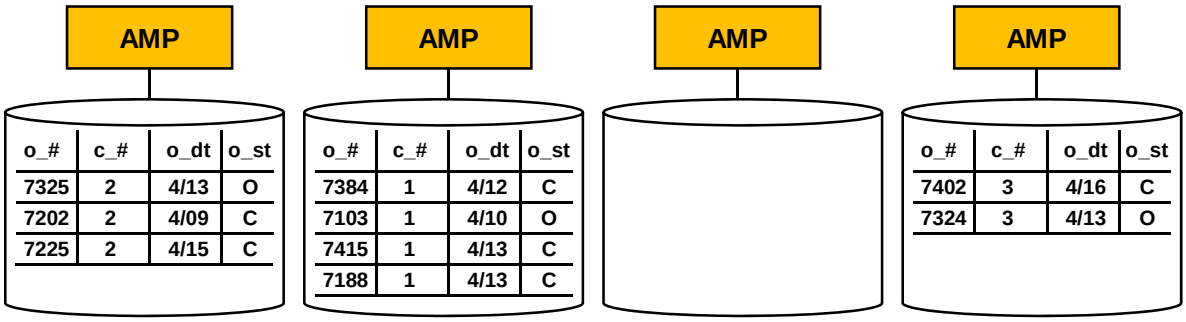
Row Distribution Using a NUPI – Case 2

Orders

Order Number	Customer Number	Order Date	Order Status
PK			
	NUPI		
7325	2	4/13	O
7324	3	4/13	O
7415	1	4/13	C
7103	1	4/10	O
7225	2	4/15	C
7384	1	4/12	C
7402	3	4/16	C
7188	1	4/13	C
7202	2	4/09	C

Notes:

- Customer_Number may be the preferred access column for this table, thus a good index candidate.
  - Since a customer can have multiple orders, Customer_Number will be a NUPI.
- Rows with the same PI value distribute to the same AMP.
  - Row distribution is less uniform or skewed.



Row Distribution Using a Highly Non-Unique Primary Index (NUPI) – Case 3

This example uses **Order_Status** as a **NUPI**. Order_Status is a poor choice, because it yields the most uneven distribution. Because there are only two possible values for Order_Status, all of the rows are placed on two AMPs.
STATUS is an example of a **highly non-unique Primary Index**.

When choosing a Primary Index, you should never choose a column with such a severely limited value set. The degree of uniqueness is critical to efficiency. Choose NUPI's that allow all AMPs to participate fairly equally.

The degree of uniqueness of a NUPI is critical to efficiency.

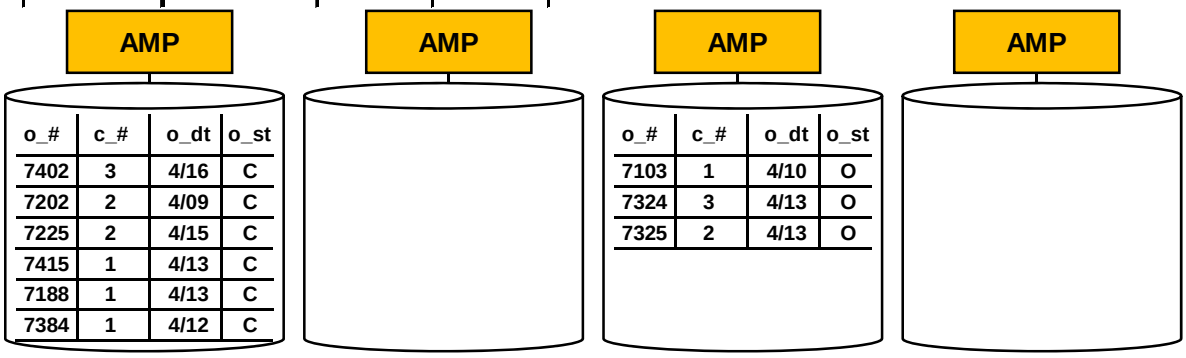
Row Distribution Using a Highly Non-Unique Primary Index (NUPI) – Case 3

Orders

Order Number	Customer Number	Order Date	Order Status
PK			
			NUPI
7325	2	4/13	O
7324	3	4/13	O
7415	1	4/13	C
7103	1	4/10	O
7225	2	4/15	C
7384	1	4/12	C
7402	3	4/16	C
7188	1	4/13	C
7202	2	4/09	C

Notes:

- Values for Order_Status are "highly" non-unique.
  - Order_Status would be a NUPI.
  - If only two values exist, then only two AMPs will be used for this table.
- Highly non-unique columns are generally poor PI choices.
  - The degree of uniqueness is critical to efficiency.



Which AMP has the Row?

This discussion (rest of this module) will assume that a table has a primary index assigned and is not using the NO PRIMARY INDEX option.

A **hashing algorithm** is a standard data processing technique that takes in a data value, like last name or order number, and systematically mixes it up so that the incoming values are converted to a number in a range from zero to the specified maximum value. A successful hashing scheme scatters the input evenly over the range of possible output values.

It is predictable in that Smith will always hash to the same value and Jones will always hash to another (and they do) different value. With a good hashing algorithm any patterns in the input data should disappear in the output data. If many names begin with "S", they should and will not all hash to the same group of hash values. If order numbers all have "00" in the hundreds and tens place or if all names are four letters long we should still see the hash values spread fairly evenly over the whole range.

Textbooks still say that this requires manually designing and tuning a hash algorithm for each new type of data values. However, the Teradata algorithm works predictably well over any data, typically loading each AMP with variations in the range of .1% to .5% between AMPs. For extremely large systems, the variation can be as low as .001% between AMPs.

Teradata also uses hashing quite differently than other data storage systems. Other hashed data storage systems equate a bucket with a physical location on disk. In Teradata, a bucket is simply an entry in a **hash map**. Each hash map entry points to a single AMP. Therefore, changing the number of AMPs does not require any adjustment to the hashing algorithm. Teradata simply adjusts the hash maps and redistributes any affected rows.

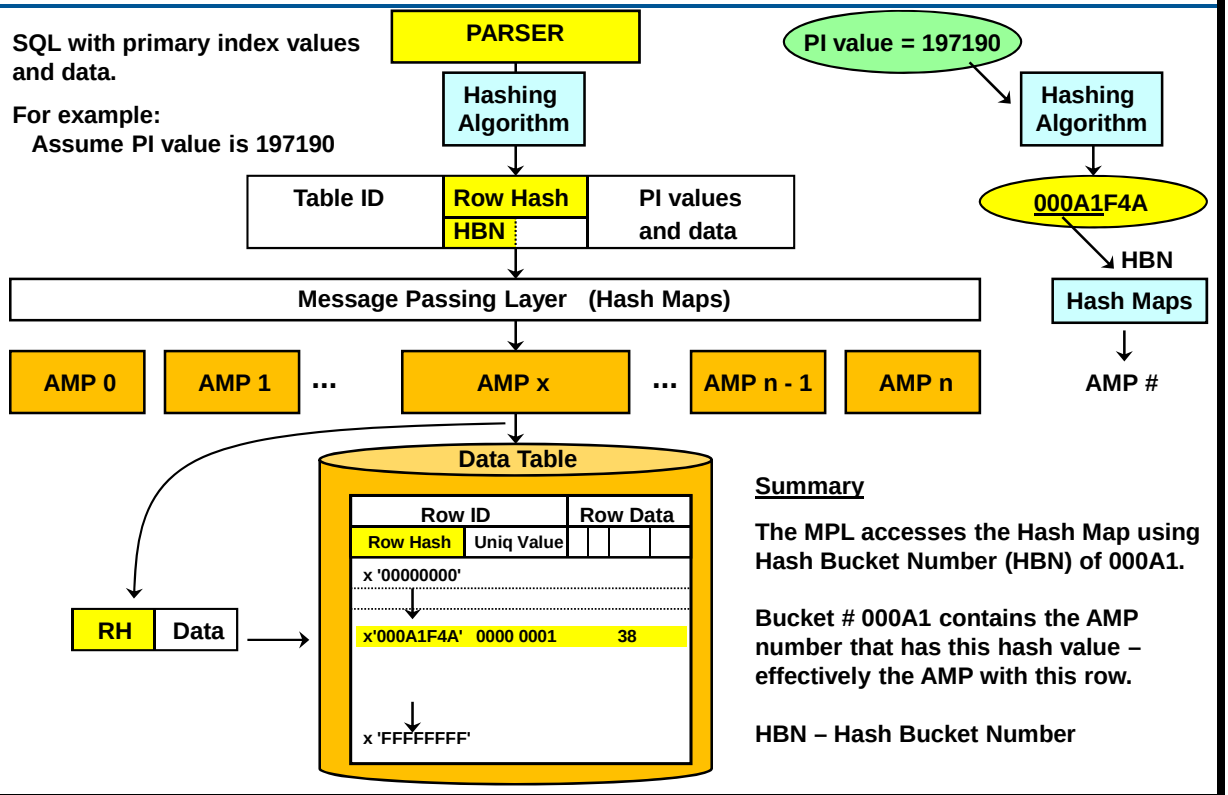
The hash maps must always be available to the Message Passing Layer. For systems using a 16-bit hash bucket number, the hash map has 65,536 entries. For systems using a 20-bit hash bucket number, the hash map has 1,048,576 entries (approximately 1 million entries). 20-bit hash bucket numbers are available starting with Teradata 12.0.

When the hash bucket has determined the destination AMP, the full 32-bit row hash plus the Table-ID is used to assign the row to a cylinder and a data block on the AMPs disk storage. The 32-bit row hash can produce over 4 billion row hash values.

Which AMP has the Row?

SQL with primary index values and data.

For example:
Assume PI value is 197190



Summary

The MPL accesses the Hash Map using Hash Bucket Number (HBN) of 000A1.

Bucket # 000A1 contains the AMP number that has this hash value – effectively the AMP with this row.

HBN – Hash Bucket Number

Hashing Down to the AMPs

The rows of all tables are distributed across the AMPs according to their **Primary Index** value. The Primary Index value goes into the hashing algorithm and the output is a 32-bit **Row Hash**. The high order bits (16 or 20) are referred to as the "bucket number" and are used to identify a hash map entry. This entry, in turn, is used to identify the AMP that will be targeted. The remaining 12 or 16 bits are not used to locate the AMP.

The entire 32-bit Row Hash is used by the selected AMP to locate the row within its disk space.

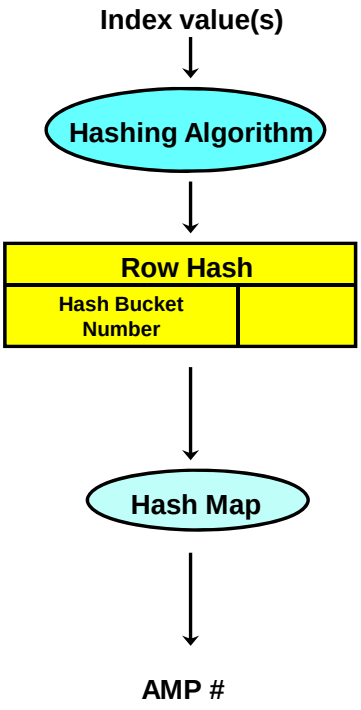
Hash maps are uniquely configured for each size of system, thus a 96 AMP system will have a hash map different from a 64 AMP system, but another 64 AMP system will have the same map (if the have the same number of bits in their HBN).

Each hash map is simply an array that associates Hash Bucket Number (HBN) values or bucket numbers with specific AMPs.

The Hash Bucket Number (prior to Teradata 12.0) has also been referred to as the DSW or Destination Selection Word.

When a system grows, new AMPs are typically added. This requires a change to the hash map to reflect the new total number of possible target AMPs.

Hashing Down to the AMPs



- The hashing algorithm is designed to insure even distribution of unique values across all AMPs.
- Different hashing algorithms are used for different international character sets.

- A Row Hash is the 32-bit result of applying a hashing algorithm to an index value.
- The Hash Bucket Number is represented by the high order bits (usually 20 on newer systems) of the Row Hash.

- A Hash Map is uniquely configured for each system.
- It is a array of entries (buckets) which associates bucket numbers with specific AMPs.

- Two systems with the same number of AMPs will have the same Hash Map (if both have the same number of bits in their HBN).
- Changing the number of AMPs in a system requires a change to the Hash Map.

A Hashing Example

The facing page shows an example of how the hashing algorithm would produce a 32-bit row hash value on the primary index value of 197190.

The hash value is divided into two parts. The first 20 bits in this example are the **Hash Bucket Number**. These bits are also simply referred to as the Hash Bucket. The hash bucket points to a particular hash map entry, which in turn points to one AMP. The entire Row Hash along with the Table ID references a particular logical location on that AMP.

A Hashing Example

Orders

Order Number	Customer Number	Order Date	Order Status
PK			
UPI			
197185	2005	2012-04-10	C
197186	3018	2012-04-10	O
197187	1035	2012-04-11	O
197188	1035	2012-04-11	C
197189	1001	2012-04-11	O
197190	2087	2012-04-11	C
197191	1012	2012-04-12	C
197192	3600	2012-04-12	C
197193	5650	2012-04-13	O
197194	1009	2012-04-13	O

```
SELECT * FROM Orders
WHERE order_number = 197190;
```

197190

↓

Hashing Algorithm

↓

000A1 F4A

↓

32 bit Row Hash	
Hash Bucket Number *	Remaining 12 bits
0000 0000 0000 1010 0001	1111 0100 1010

0 0 0 A 1

* Assumes 20-bit hash bucket numbers.

The Hash Map

A **hash map** is simply an array of entries where each entry is two bytes long. The hash map is loaded into memory and is used by Teradata software. Each entry contains an AMP number for the system on which Teradata is implemented. The hash bucket number (or bucket number) is an offset into the hash map to locate a specific entry (or AMP).

For systems using a 16-bit hash bucket number, the hash map has 65,536 entries. For systems using a 20-bit hash bucket number, the hash map has 1,048,576 entries (approximately 1 million entries).

To determine the destination AMP for a Primary Index operation, the hash map is checked by BYNET software using the row hash information. A message is placed on the BYNET to be sent to the target AMP using point-to-point communication.

In the example, the HBN entry 000A1 (hexadecimal) contains an entry that identified AMP 13. AMP 13 will be the recipient of the message from the Message Passing Layer.

The facing page identifies a portion of an actual primary hash map for a 26 AMP system.

An example of hash functions that can be used in SQL follows:

```
SELECT
 HASHROW (197190) AS "Hash Value"
,HASHBUCKET (HASHROW (197190)) AS "Bucket Num"
,HASHAMP (HASHBUCKET (HASHROW (197190)))  AS "AMP Num"
,HASHBAKAMP (HASHBUCKET (HASHROW (197190))) AS "AMP Fallback Num"
;

 *** Query completed. One row found. 4 columns returned.
 *** Total elapsed time was 1 second.

Hash Value   Bucket Num      AMP Num  AMP Fallback Num
000A1F4A            161           13                 0
```

The Hash Map

197190 → **Hashing Algorithm** → 000A1F4A

32 bit Row Hash	
Hash Bucket Number *	Remaining 12 bits
0000 0000 0000 1010 0001	1111 0100 1010

0 0 0 A 1

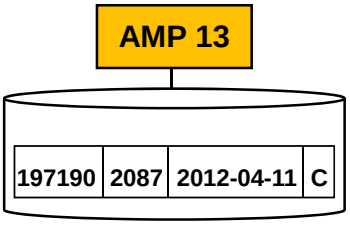
* With 20-bit hash bucket numbers, the hash map has 1,048,576 entries.

With 16-bit hash bucket numbers, the hash map only has 65,536 entries.

HASH MAP

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0007	24	25	19	12	19	19	25	23	20	20	23	20	21	24	13	20
0008	21	22	20	20	22	23	25	21	11	23	10	22	13	10	24	12
0009	21	21	22	14	21	21	22	23	22	22	24	12	25	25	11	11
000A	08	13	23	14	24	09	11	11	23	23	15	25	23	23	13	07
000B	25	06	15	08	24	24	12	24	12	09	10	24	25	05	24	24
000C	16	12	09	25	12	16	14	04	09	09	13	14	25	25	03	13

Portion of actual hash map (20-bit hash bucket numbers) for a 26 AMP system. AMPs are shown in decimal format.



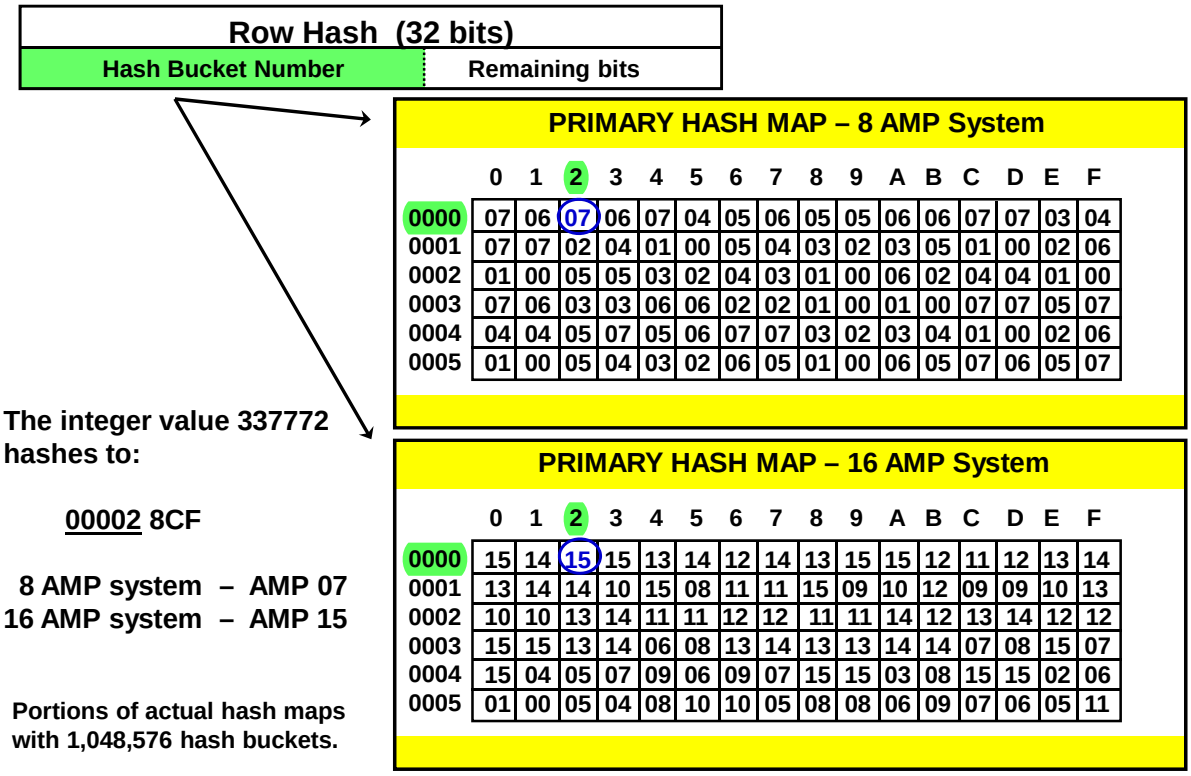
Hash Maps for Different Systems

The diagrams on the facing page show a graphical representation of a Primary Hash Map for an 8 AMP system and a Primary Hash Map for a 16 AMP system.

A data value which hashes to "000028CF" will be directed to different AMPs on different systems. For example, this hash value will be associated with AMP 7 on an 8 AMP system and AMP 15 on a 16 AMP system.

Note: These are the actual partial hash maps for 8 and 16 AMP systems.

Hash Maps for Different Systems



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	07	06	07	06	07	04	05	06	05	05	06	06	07	07	03	04
0001	07	07	02	04	01	00	05	04	03	02	03	05	01	00	02	06
0002	01	00	05	05	03	02	04	03	01	00	06	02	04	04	01	00
0003	07	06	03	03	06	06	02	02	01	00	01	00	07	07	05	07
0004	04	04	05	07	05	06	07	07	03	02	03	04	01	00	02	06
0005	01	00	05	04	03	02	06	05	01	00	06	05	07	06	05	07

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	15	14	15	15	13	14	12	14	13	15	15	12	11	12	13	14
0001	13	14	14	10	15	08	11	11	15	09	10	12	09	09	10	13
0002	10	10	13	14	11	11	12	12	11	11	14	12	13	14	12	12
0003	15	15	13	14	06	08	13	14	13	13	14	14	07	08	15	07
0004	15	04	05	07	09	06	09	07	15	15	03	08	15	15	02	06
0005	01	00	05	04	08	10	10	05	08	08	06	09	07	06	05	11

Identifying Rows

Can two different PI values come out of the hashing algorithm with the same row hash value? The answer is “Yes”. There are two ways that can happen.

First, two different primary index values may happen to hash identically. This is called a **hash synonym**.

Secondly, if a non-unique primary index is used; **duplicate NUPI values will produce the same row hash**.

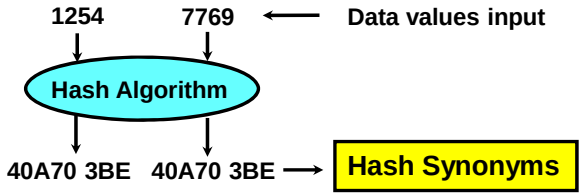
Identifying Rows

A row hash is not adequate to uniquely identify a row.

Consideration #1

A Row Hash = 32 bits = 4.2 billion possible values

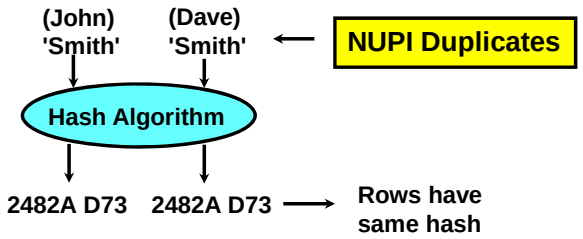
Because there is an infinite number of possible data values, some data values will have to share the same row hash.



Consideration #2

A Primary Index may be non-unique (NUPI).

Different rows will have the same PI value and thus the same row hash.



Conclusion

A row hash is not adequate to uniquely identify a row.

The Row ID

In order to differentiate each row in a table, every row is assigned a unique **Row ID**. The **Row ID** is a combination of the row hash value plus a **uniqueness value**. The AMP appends the uniqueness value to the row hash when it is inserted. The Uniqueness Value is used to differentiate between PI values that generate identical row hashes.

The first row inserted with a particular row hash value is assigned a uniqueness value of 1. Each new row with the same row hash is assigned an integer value one greater than the current largest uniqueness value for this Row ID.

If a row is deleted or the primary index is modified, the uniqueness value can be reused.

Only the Row Hash portion is used in Primary Index operations. The entire **Row ID** is used for Secondary Index support that is discussed in a later module.

In summary, Row Hash is a 32-bit value. Up to and including Teradata V2R6.2, the Message Passing Layer looks at the high-order 16 bits (previously called “DSW” - Destination Selection Word). This is used to index into the Hash Map to determine which AMP gets the row or is used to retrieve a row. Once the AMP has been determined, the entire 32-bits of the Row Hash are passed to the AMP. The AMP uses the entire 32-bit Row Hash to store/retrieve the row.

Since there are only 4 billion permutations of Row Hash, you can get duplicates. NUPI Duplicates also cause duplicate Row Hashes, therefore the Row Hash is not sufficient to uniquely identify a row in a table. Therefore, the AMP adds another 32-bit number (called a uniqueness value) to the Row Hash. This total 64-bit number (32-bit Row Hash + 32-bit Uniqueness Value) is called the Row ID. This number uniquely identifies a row in a table.

The Row ID

To uniquely identify a row, we add a 32-bit uniqueness value.

The combined row hash and uniqueness value is called a Row ID.

Row ID

Each stored row has a Row ID as a prefix.

Rows are logically maintained in Row ID sequence.

Row ID		Row Data		
Row Hash	Unique ID	Emp_No	Last_Name	First_Name
3B11 5032	0000 0001	1018	Reynolds	Jane
3B11 5032	0000 0002	1020	Davidson	Evan
3B11 5032	0000 0003	1031	Green	Jason
3B11 5033	0000 0001	1014	Jacobs	Paul
3B11 5034	0000 0001	1012	Chevas	Jose
3B11 5034	0000 0002	1021	Carnet	Jean
:	:	:	:	:

Storing Rows (1 of 2)

Rows are stored in a data block in Row ID sequence. As rows are added to a table with the same row hash, the uniqueness value is incremented by one in order to provide a unique Row ID.

Assume Last_Name is a NUPI and that all rows in this example hash to the same AMP.

The ‘John Smith’ row is assigned to AMP 3 based on the bucket number portion of the row hash. Because it is the first row with this row hash, a uniqueness id of 1 is assigned.

The ‘Sam Adams’ row has a different row hash and thus is also assigned a uniqueness value of 1. The bucket number, although different, also points to AMP 3 in the hash map.

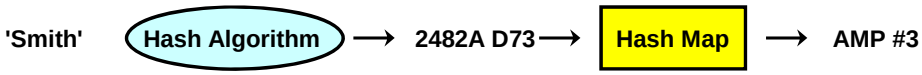
Storing Rows (1 of 2)

Assumptions:

Last_Name is defined as a NUPI.

All rows in this example hash to the same AMP.

Add a row for 'John Smith'



Row ID		Row Data		
Row Hash	Unique ID	Last_Name	First_Name	Etc.
2482A D73	0000 0001	Smith	John	

Add a row for 'Sam Adams'

Row ID		Row Data		
Row Hash	Unique ID	Last_Name	First_Name	Etc.
2482A D73	0000 0001	Smith	John	
782B7 E4D	0000 0001	Adams	Sam	

Storing Rows (2 of 2)

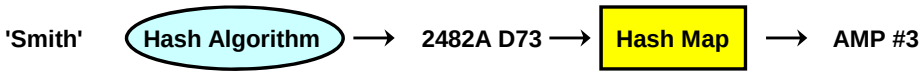
The ‘Fred Smith’ row hashes to the same row hash as ‘John Smith’ because it is a NUPI duplicate. It is therefore assigned a uniqueness id of 2.

The ‘Dan Jones’ row also hashes to the same row hash because it is a hash synonym. It is thus assigned a uniqueness id of 3.

Note: In reality, the last names of Smith and Jones DO NOT hash to the same value. This is simply an example that illustrates how the uniqueness ID is used when a hash synonym does occur.

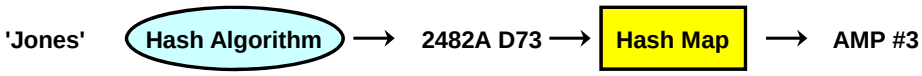
Storing Rows (2 of 2)

Add a row for 'Fred Smith' - (NUPI Duplicate)



Row ID		Row Data		
Row Hash	Unique ID	Last_Name	First_Name	Etc.
2482A D73	0000 0001	Smith	John	
2482A D73	0000 0002	Smith	Fred	
782B7 E4D	0000 0001	Adams	Sam	

Add a row for 'Dan Jones' - (Hash Synonym)



Row ID		Row Data		
Row Hash	Unique ID	Last_Name	First_Name	Etc.
2482A D73	0000 0001	Smith	John	
2482A D73	0000 0002	Smith	Fred	
2482A D73	0000 0003	Jones	Dan	
782B7 E4D	0000 0001	Adams	Sam	

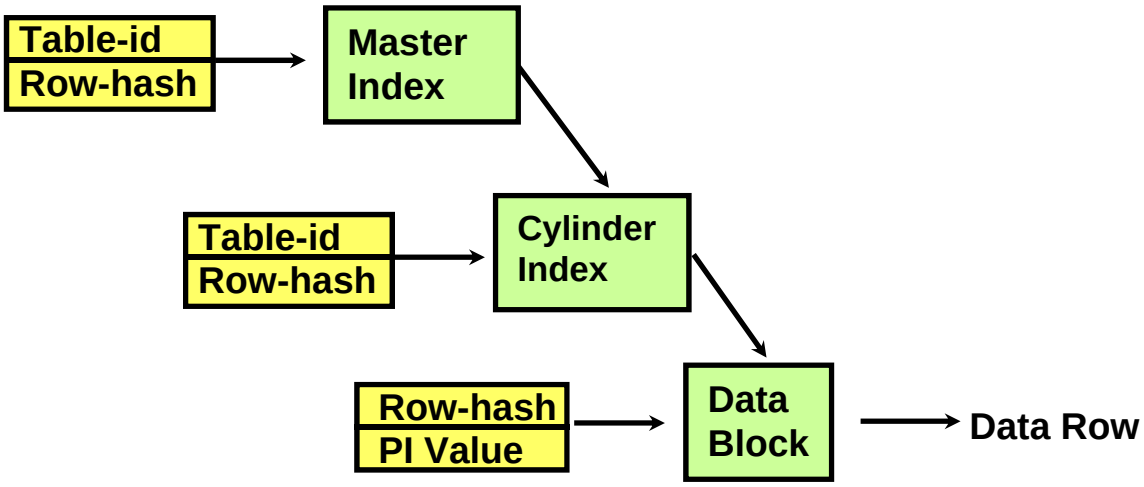
Given the row hash, what other information would be needed to find the 'Dan Jones' row? … The 'Fred Smith' row?

Locating a Row on an AMP Using a PI

To locate a row, the AMP file system searches through a memory-resident structure called the Master Index. An entry in the Master Index will indicate that if a row with this Table ID and row hash exists, then it must be on a specific disk cylinder.

The file system will use the cylinder number to locate the Cylinder Index and search through the designated Cylinder Index. There it will find an entry that indicates that if a row with this Table ID and row hash exists, it must be in one specific data block on that cylinder.

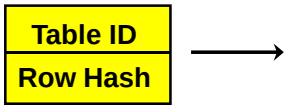
The file system then searches the data block until it locates the row(s) or returns a No Rows Found condition code.

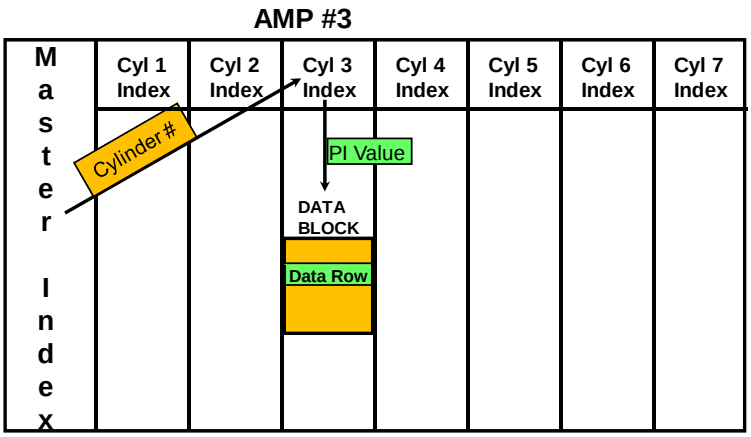


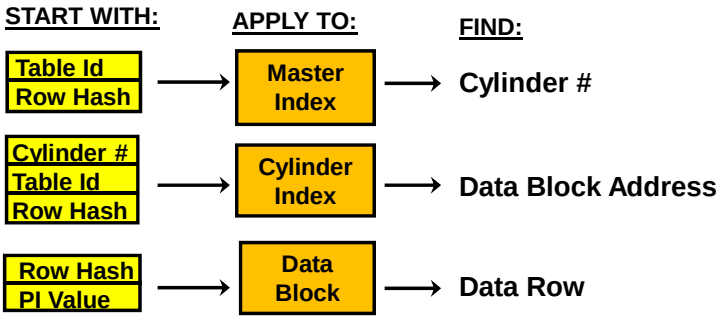
Locating a Row On An AMP Using a PI

Locating a row on an AMP requires three input elements:

1. **The Table ID**
2. **The Row Hash of the PI**
3. **The PI value itself**







Module 5: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 5: Review Questions

Answer the following either as True or False as these apply to Primary Indexes:

True or False **1. UPI and NUPI equality value accesses are always a one-AMP operation.**

True or False **2. UPI and NUPI indexes allow NULL in a primary index column.**

True or False **3. UPI, NUPI, and NOPI tables allow duplicate rows in the table.**

True or False **4. Only UPI can be used as a Primary Key implementation.**

Fill in the Blanks

5. **The output of the hashing algorithm is called the _____ _____.**
6. **To determine the target AMP, the Message Passing Layer must lookup an entry in the Hash Map based on the _____ _____ _____.**
7. **A Row ID consists of a row hash plus a _____ value.**
8. **A uniqueness value is required to produce a unique Row ID because of _____ _____ and _____ _____.**
9. **Once the target AMP has been determined for a PI search, the _____ _____ for that AMP is accessed to determine the cylinder that may hold the row.**
10. **The Cylinder Index points us to the address and length of the data _____.**

Notes

Module 6

Secondary Indexes and Table Scans

After completing this module, you will be able to:

- **Define Secondary Indexes.**
- **Distinguish between the implementation of unique and non-unique secondary indexes.**
- **Define Full Table Scans and what causes them.**
- **Describe the operation of a Full Table Scan in a parallel environment.**

Notes

Table of Contents

Secondary Indexes 6-4
Choosing a Secondary Index 6-6
Unique Secondary Index (USI) Access 6-8
Non-Unique Secondary Index (NUSI) Access 6-10
Comparison of Primary and Secondary Indexes 6-12
Full Table Scans 6-14
Module 6: Review Questions 6-16

Secondary Indexes

A **secondary index** is an alternate path to the data. Secondary Indexes are used to improve performance by allowing the user to avoid scanning the entire table. A Secondary Index is like a Primary Index in that it allows the user to locate rows. It is unlike a Primary Index in that it has no influence on the way rows are distributed among AMPs. A database designer typically chooses a secondary index because it provides faster set selection.

Primary Index requests require the services of only one AMP to access rows, while secondary indexes require at least two and possibly all AMPs, depending on the index and the type of operation. A secondary index search will typically be less expensive than a full table scan.

Secondary indexes add overhead to the table, both in terms of disk space and maintenance; however they may be dropped when not needed, and recreated whenever they would be helpful.

Secondary Indexes

There are 3 general ways to access a table:

Primary Index access	(one AMP access)
Secondary Index access	(two or all AMP access)
Full Table Scan	(all AMP access)

- A secondary index provides an alternate path to the rows of a table. 
- A secondary index can be used to maintain uniqueness within a column or set of columns.
- A table can have from 0 to 32 secondary indexes.
- Secondary Indexes:
  - Do not effect table distribution.
  - Add overhead, both in terms of disk space and maintenance.
  - May be added or dropped dynamically as needed.
  - Are chosen to improve table performance.

Choosing a Secondary Index

Just as with primary indexes, there are two types of **secondary indexes** – **unique (USI)** and **non-unique (NUSI)**.

Secondary Indexes may be specified at table creation or at any time during the life of the table. It may consist of up to 64 columns, however to get the benefit of the index, the query would have to specify a value for all 64 values.

Unique Secondary Indexes (USI) have two possible purposes. They can speed up access to a row which otherwise might require a full table scan without having to rely on the primary index. Additionally, they can be used to enforce uniqueness on a column or set of columns. This is sometimes the case with a Primary Key which is not designated as the Primary Index. Making it a USI has the effect of enforcing the uniqueness of the PK.

Non-Unique Secondary Indexes (NUSI) are usually specified in order to prevent full table scans. However, a NUSI does activate all AMPs – after all, the value being sought might well exist on many different AMPs (only Primary Indexes have same values on same AMPs). If the optimizer decides that the cost of using the secondary index is greater than a full table scan would be, it opts for the table scan.

All secondary indexes cause an AMP local subtable to be built and maintained as column values change. Secondary index **subtables** consist of rows which associate the secondary index value with one or more rows in the base table. When the index is dropped, the subtable is physically removed.

Choosing a Secondary Index

A Secondary Index may be defined ...

- at table creation (CREATE TABLE)
- following table creation (CREATE INDEX)
- may be up to 64 columns 

USI

If the index choice of column(s) is unique, it is called a USI.

Unique Secondary Index

Accessing a row via a USI is a 2 AMP operation.

```
CREATE UNIQUE INDEX
   (Employee_Number) ON Employee;
```

NUSI

If the index choice of column(s) is non-unique, it is called a NUSI.

Non-Unique Secondary Index

Accessing row(s) via a NUSI is an all AMP operation.

```
CREATE INDEX
   (Last_Name) ON Employee;
```

Notes:

- Creating a Secondary Index cause an internal sub-table to be built.
- Dropping a Secondary Index causes the sub-table to be deleted.

Unique Secondary Index (USI) Access

The facing page shows the two AMP accesses necessary to retrieve a row via a **Unique Secondary Index** access.

After the row hash of the secondary index value is calculated, the hash map points us to AMP 1 as containing the subtable row for this USI value. After locating the subtable row in AMP 1, we find the row-id of the base row we are seeking. This base row id (which includes the row hash) again allows the hash map to point us to AMP 3 which contains the base row.

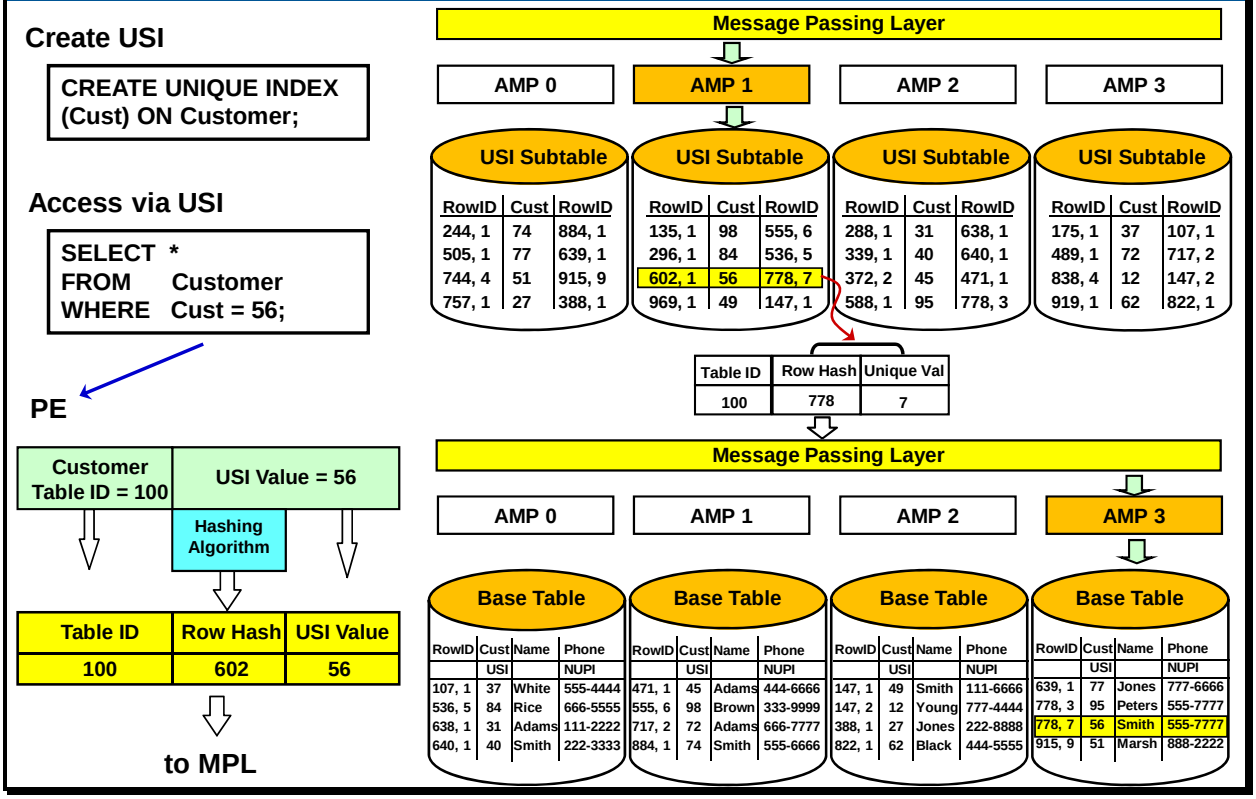
Secondary index access uses the complete row-id to locate the row, unlike primary index access, which only uses the row hash portion.

The Customer table below is the table used in the example. It is only a partial listing of the rows.

Customer Table

Cust	Name	Phone
USI		NUPI
37	White	555-4444
98	Brown	333-9999
74	Smith	555-6666
95	Peters	555-7777
27	Jones	222-8888
56	Smith	555-7777
45	Adams	444-6666

Unique Secondary Index (USI) Access



Non-Unique Secondary Index (NUSI) Access

The facing page shows an all-AMP access necessary to retrieve a row via a **Non-Unique Secondary Index** access.

After the row hash of the secondary index value is calculated, the Message Passing Layer will automatically activate all AMPs per instructions of the Parsing Engine. Each AMP locates the subtable rows containing the qualifying value and row hash. These subtable rows contain the row-id(s) for the base rows, which are guaranteed to be on the same AMP as the subtable row. This reduces activity in the MPL and essentially makes the query an AMP-local operation.

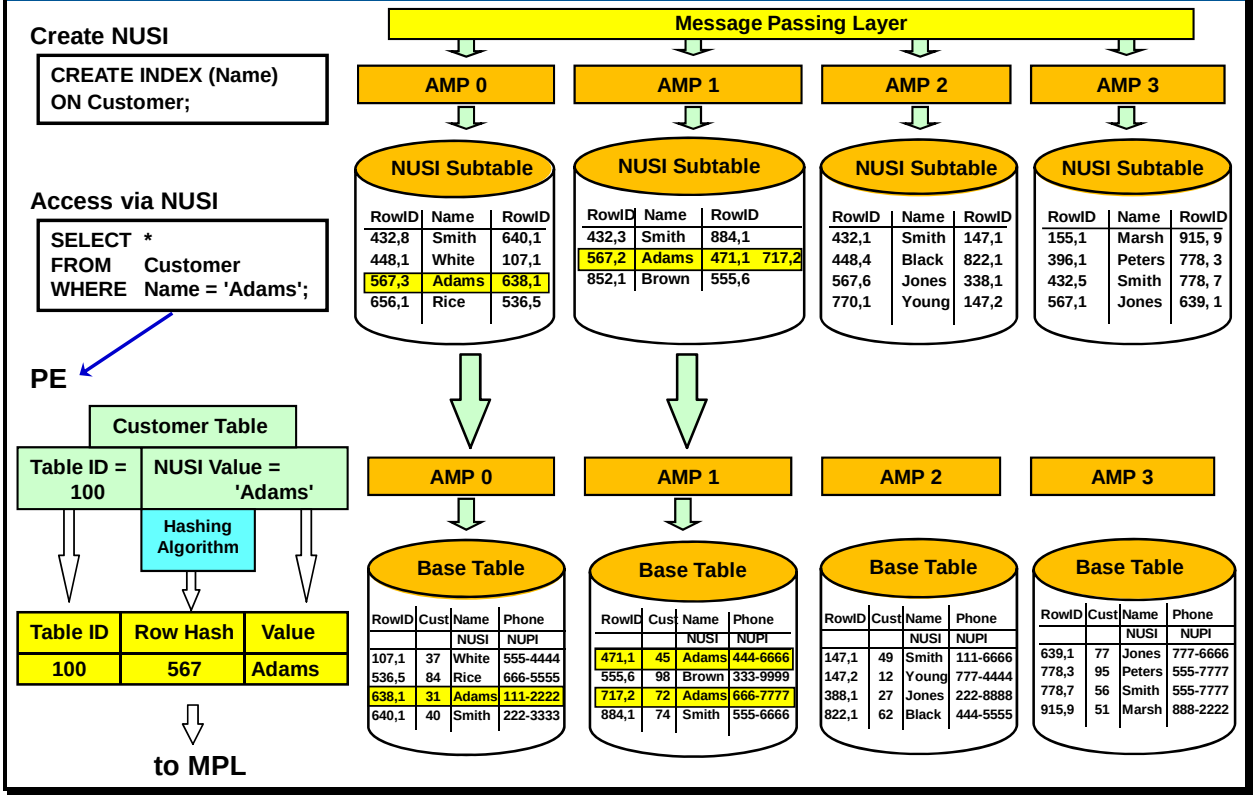
Because each AMP may have more than one qualifying row, it is possible for the subtable row to have multiple row-ids for the base table rows.

The Customer table below is the table used in the example. It is only a partial listing of the rows.

Customer Table

Cust	Name	Phone
	NUSI	NUPI
37	**White**	555-4444
98	**Brown**	333-9999
74	**Smith**	555-6666
95	**Peters**	555-7777
27	**Jones**	222-8888
56	**Smith**	555-7777
45	**Adams**	444-6666

Non-Unique Secondary Index (NUSI) Access



Comparison of Primary and Secondary Indexes

The table on the facing page compares and contrasts **primary and secondary indexes**:

Primary indexes are required; secondary indexes are optional. All tables must have a method of distributing rows among AMPs -- the Primary Index.

A table can only have one primary index, but it can have up to 32 secondary indexes.

Both primary and secondary indexes can have up to 64 columns.

Secondary indexes, like primary indexes, can be either unique (USI) or non-unique (NUSI).

The secondary index does not affect the distribution of rows. Rows are only distributed according to the Primary Index values.

Secondary indexes can be created and dropped dynamically. In other words, Secondary Indexes can be added as needed. In fact, in some cases it is a good idea to wait and see how the database is used and then add Secondary Indexes to facilitate that usage.

Both primary and secondary indexes affect system performance. However, Primary and Secondary Indexes affect performance for different reasons. A poorly-chosen PI results in "lumpy" data distribution which makes some AMPs do more work than others and slows the system.

Secondary Indexes affect performance because they require subtables. Both indexes allow rapid retrieval of specific rows.

Both primary and secondary indexes can be created using multiple data types.

Secondary indexes are stored in separate subtables; primary indexes are not.

Because secondary indexes require separate subtables, extra I/O is needed to maintain those subtables.

Comparison of Primary and Secondary Indexes

Index Feature	Primary	Secondary
Required?	Yes*	No
Number per Table	1	0 - 32
Max Number of Columns	64	64
Unique or Non-unique	Both	Both
Affects Row Distribution	Yes	No
Created/Dropped Dynamically	No	Yes
Improves Access	Yes	Yes
Multiple Data Types	Yes	Yes
Separate Physical Structure	No	Sub-table
Extra Processing Overhead	No	Yes
May be ordered by value	No	Yes (NUSI)
May be Partitioned	Yes	No

* Not required with NoPI table in Teradata 13.0

Full Table Scans

A **full table scan** is another way to access data without using any Primary or Secondary Indexes.

In evaluating an SQL request, the Parser examines all possible access methods and chooses the one it believes to be the most efficient. The coding of the SQL request along with the demographics of the table and the availability of indexes all play a role in the decision of the Parser. Some coding constructs, listed on the facing page, always cause a full table scan. In other cases, it might be chosen because it is the most efficient method. In general, if the number of physical reads exceeds the number of data blocks then the optimizer may decide that a full-table scan is faster.

With a full table scan, each data block is found using the Master and Cylinder Indexes and each data row is accessed only once.

As long as the choice of Primary Index has caused the table rows to distribute evenly across all of the AMPs, the parallel processing of the AMPs can do the full table scan quickly. The file system keeps each table on as few cylinders as practical to help reduce the cost full table scans.

While full table scans are impractical and even disallowed on some systems, the Teradata Database routinely executes ad hoc queries with full table scans.

Full Table Scans

Every row of the table must be read.

All AMPs scan their portion of the table in parallel.

- **Fast and efficient on Teradata due to parallelism.**

Full table scans typically occur when either:

- **An index is not used in the query**
- **An index is used in a non-equality test**

Customer

Cust_ID	Cust_Name	Cust_Phone
USI		NUPI

Examples of Full Table Scans:

```
SELECT  *  FROM Customer WHERE Cust_Phone LIKE '858-485-_ _ _ _ ';
SELECT  *  FROM Customer WHERE Cust_Name = 'Koehler';
SELECT  *  FROM Customer WHERE Cust_ID > 1000;
```

Module 6: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 6: Review Questions

Fill each box with either Yes, No, or the appropriate number.

	USI Access	NUSI Access	FTS
# AMPs			
# rows			
Parallel Operation			
Uses Hash Maps			
Uses Separate Sub-table			
Reads all data blocks of table			

Notes

Module 7

Teradata System Architecture

After completing this module, you will be able to:

- **Identify characteristics of various components.**
- **Specify the difference between a TPA and non-TPA node.**

Notes

Table of Contents

Teradata Database Releases 7-4
Teradata Version 1 7-4
Teradata Version 2 7-4
Teradata Database Architecture 7-6
Teradata Database – Multiple Nodes 7-8
MPP Systems 7-10
Example of 3 Node Teradata Database System 7-12
Example: 5650 and 6844 Disk Arrays 7-12
Teradata Cliques 7-14
BYNET 7-16
BYNET Communication Protocols 7-18
Vproc Inter-process Communication 7-20
Examples of Teradata Database Systems 7-22
Example of 5650 Cabinets 7-24
What makes Teradata's MPP Platforms Special? 7-26
Summary 7-28
Module 7: Review Exercises 7-30

Teradata Database Releases

The facing page identifies various Teradata releases that have been available since 1984. This page identifies some historical information about Teradata Version 1 systems.

Teradata Version 1

Teradata Database Version 1 platforms were first available to customers in 1984. This first platform was the original Database Computer, **DBC/1012** from Teradata. In 1991, the NCR Corporation introduced the **3600**. Both of these systems are older technologies and both of these systems used a proprietary 16-bit operating system known as **TOS** (Teradata Operating System). All AMPs and PEs were dedicated hardware processors that were connected together using a message-passing layer known as the **Ynet**.

Both platforms supported channel-attached (Bus and Tag) and LAN-attached host systems.

DBC/1012 Architecture – this system was a dedicated relational database management system. Two specific components of the DBC/1012 were the IFP and the COP, both of which were effectively hardware Parsing Engines. The acronyms IFP and COP still appear in the Data Dictionary even today.

Interface Processor (IFP) – the IFP was the Parsing Engine of the DBC/1012 for channel-attached systems. For current systems (e.g., 5550), PEs that are assigned to a channel are identified in the Data Dictionary with a type of IFP.

Communications Processor (COP) – the **COP** is the Parsing Engine of the DBC/1012 for network-attached systems. For current systems (e.g., 5550), PEs that are assigned to a LAN (or network) are identified in the Data Dictionary with a type of COP.

3600 Architecture – this system included hardware AMPs and PEs as well as multipurpose Application Processors executing UNIX MP-RAS. Application Processors (APs) also provided the channel and LAN connectivity. UNIX applications were executed on APs while Teradata was executed on PEs and AMPs. All processing units were connected via the Ynet.

Teradata Version 2

Starting with **Teradata Database Version 2 Release 1** (available in January 1996), the Teradata Database became an open database system. No longer was Teradata software dependent on a proprietary Operating System (TOS) and proprietary hardware. Rather, it was an application that initially executed ran under UNIX.

By porting the Teradata Database to a general-purpose operating system platform, a variety of processing options against the Teradata database became possible, all within a single system. OLTP (as well as OLCP and OLAP) applications became processing options in addition to standard DSS.

Teradata Database Releases

Teradata Releases

Version 1 Release 1
Release 2
Release 3
Release 4
Release 5

Version 1 was a combination of hardware and software.

For example, if a customer needed additional AMPs, the hardware and software components for an AMP had to be purchased, installed, and configured.

V1 Platforms	Year Available
DBC/1012	1984
3600	1991

Version 2 Release 1
Release 2
Release 3
Release 4
Release 5
Release 6

Version 2 is an implementation of Teradata PEs and AMPs as software vprocs (virtual processors).

Teradata is effectively a database application that executes under an operating system.

Platforms	Year Available
5100	1996 (UNIX MP-RAS only)
↓	↓
5650 (requires 12.0 or later)	2010
6650, 6680	2011
6690	2012

Teradata 12.0
Teradata 13.0
Teradata 13.10
Teradata 14.0

Teradata Database Architecture

Teradata is effectively an application that runs under an operating system (SUSE Linux, UNIX MP-RAS, or Windows Server 2003).

PDE software provides Teradata Database software the capability to under a specific operating system. Parallel Database Extensions (PDE) software is an interface layer on top of the operating system (Linux, UNIX MP-RAS, or Windows Server 2003). PDE provides the Teradata Database with the ability to:

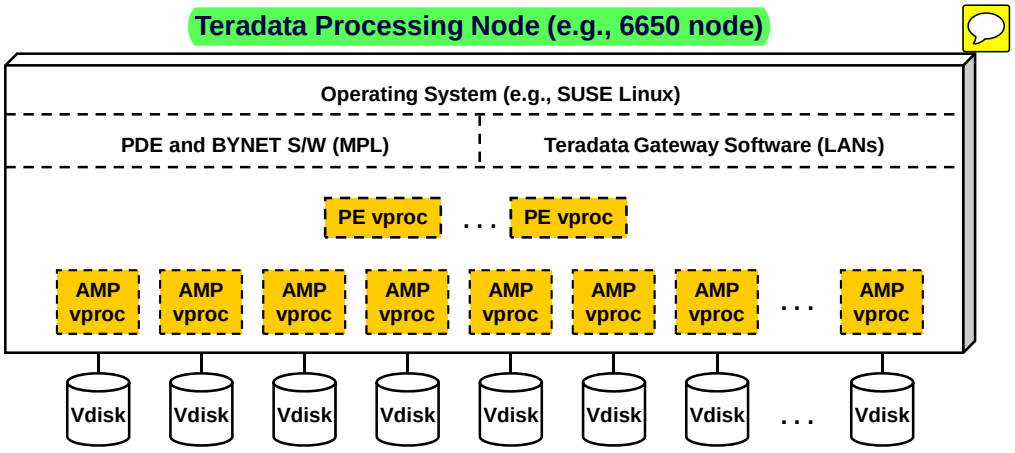
- Run the Teradata Database in a parallel environment
- Execute vprocs
- Apply a flexible priority scheduler to Teradata Database sessions
- Debug the operating system kernel and the Teradata Database using resident debugging facilities

AMPs and PEs are implemented as **"virtual processors - vprocs"**. They run under the control of PDE and their number is software configurable.

AMPs are associated with **"virtual disks – vdisks"** which are associated with logical units (LUNs) within a disk array.

The versatility of Teradata Database is based on virtual processors (vprocs) that eliminate dependency on specialized physical processors. Vprocs are a set of software processes that run on a node under Teradata Parallel Database Extensions (PDE) within the multitasking environment of the operating system.

Teradata Database Architecture



- Teradata executes on a 64-bit operating system (e.g., SUSE Linux).
  - Utilizes general purpose SMP/MPP hardware.
  - Parallel Database Extensions (PDE) is unique per OS that Teradata is supported on.
- AMPs and PEs are implemented as virtual processors (Vprocs).
- "Shared Nothing" Architecture – each AMP has its own memory, manages its own disk space, and executes independently of other AMPs.

Teradata Database – Multiple Nodes

A customer may choose to implement Teradata on a small, single node SMP system for smaller database requirements and to eventually grow incrementally to a multiple terabyte system. A single-node SMP platform may also be used as low cost development systems.

Under the **single-node (SMP)** version of Teradata, PE and AMP vproc still communicate with each other via PDE and BYNET software. All vprocs share the resources of CPUs and memory within the SMP node.

As a customer's Teradata database needs grow, additional nodes will probably be needed. A multi-node system running the Teradata Database is referred to as an MPP (Massive Parallel Processing) system.

The Teradata Database application is considered a **Trusted Parallel Application (TPA)**. The Teradata Database is the only TPA application available at this time.

Nodes in a system configuration may or may not be connected to the BYNET. Examples of nodes and their purpose include:

- TPA (Trusted Parallel Application) node – executes Teradata Database software.
- HSN (Hot Standby Node) – is a spare node in the clique (not running Teradata) used in event of a node failure.
- Non-TPA (NOTPA) node – is an application node that does not executes Teradata Database software.

Hot standby nodes allow spare nodes to be incorporated into the production environment. The Teradata Database can use spare nodes to improve availability and maintain performance levels in the event of a node failure. A hot standby node is a node that:

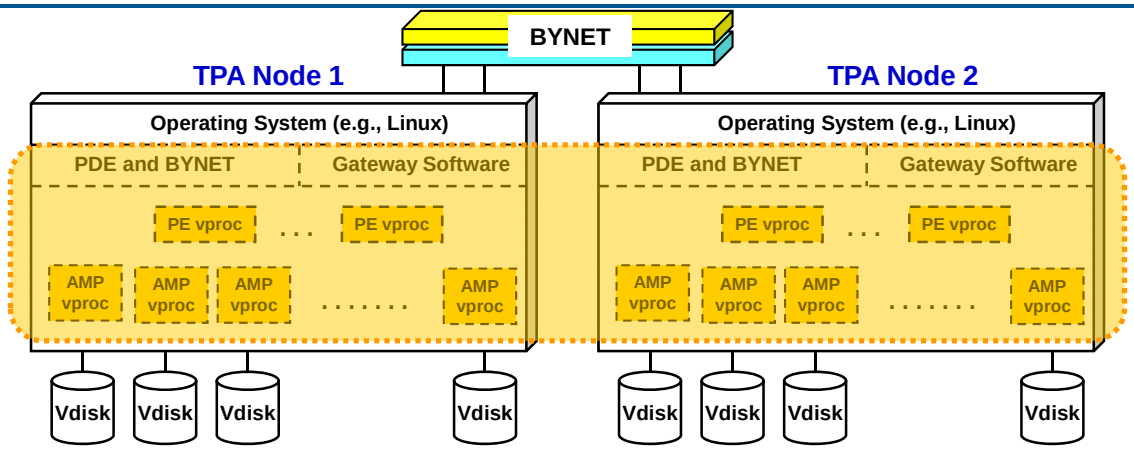
- is a member of a clique
- does not normally participate in Teradata Database operations
- can be brought in to participate in Teradata Database operations to compensate for the loss of a node in the clique

Configuring a hot standby node can eliminate the system-wide performance degradation associated with the loss of a node. A hot standby node is added to each clique in the system. When a node fails, all AMPs and all LAN-attached PEs on the failed node migrate to the node designated as the hot standby node. The hot standby node becomes a production node. When the failed node returns to service, it becomes the new hot standby node.

Configuring hot standby nodes eliminates:

- Restarts that are required to bring a failed node back into service.
- Degraded service when vprocs have migrated to other nodes in a clique.

Teradata Database – Multiple Nodes



Teradata is a linearly expandable database – as your database grows, additional nodes may be added – effectively becoming an MPP (Massive Parallel Processing) systems.

- **Teradata software makes a multi-node system look like a single-Teradata system.**

Examples of types of nodes that connect to the BYNET.

- **TPA (Trusted Parallel Application) node – executes Teradata Database software.**
- **HSN (Hot Standby Node) – spare node in the clique (not running Teradata) used in event of a node failure.**
- **Non-TPA (NOTPA) node – application node that does not executes Teradata Database software.**

MPP Systems

When multiple SMP nodes (simply referred to as nodes) are connected together to form a larger configuration, we refer to this as an **MPP** (Massively Parallel Processing) system.

The connecting layer (or system interconnect) is called the **BYNET**. The BYNET is a combination of hardware and software that allows multiple vprocs on multiple nodes to communicate with each other.

Because Teradata is a **linearly expandable** database system, as additional nodes and vprocs are added to the system, the system capacity scales in a linear fashion.

The BYNET Version 1 can support up to 128 SMP nodes. The BYNET Version 2 can support up to 512 nodes. The BYNET Version 3 can support up to 1024 nodes and BYNET Version 4 can support up to 4096 nodes.

Acronyms that may appear in diagrams throughout this course:

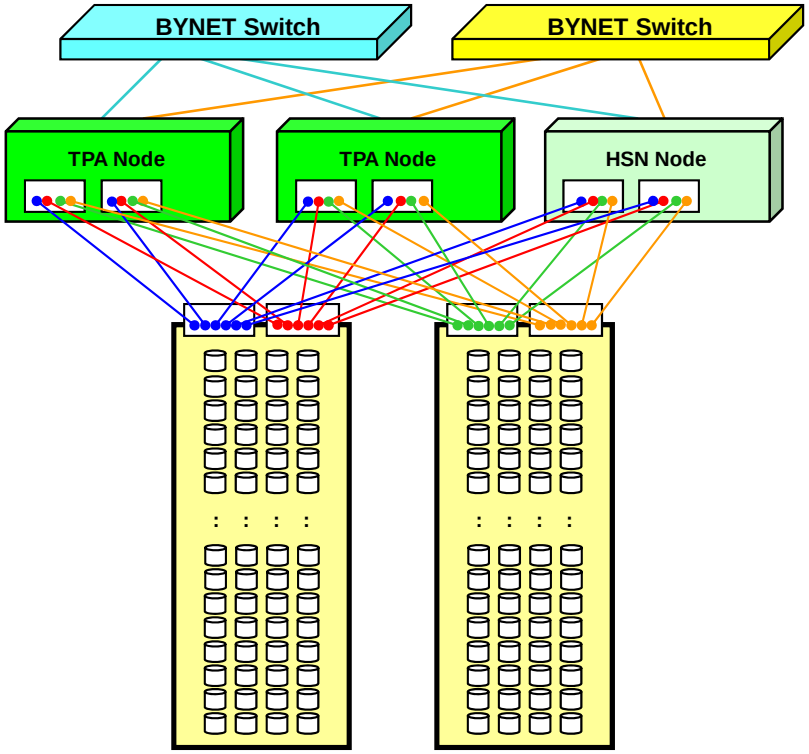
PCI – Peripheral Component Interconnect
EISA – Extended Industry Standard Architecture
PBCA – PCI Bus Channel Adapter
PBSA – PCI Bus ESCON Adapter
EBCA – EISA Bus Channel Adapter

MPP Systems

The BYNET consists of redundant switches that interconnect multiple nodes.

Multiple nodes make up Massively Parallel Processing (MPP) system.

A clique is a group of nodes connected to and sharing the same storage.



Example of 2+1 Node Teradata System

The facing page contains an illustration of a simple **three-node (2+1) Teradata Database system**. Each node has its own Vprocs to manage, while communication among the Vprocs takes place via the BYNETs. The PEs are not shown in this example.

Each node is an SMP from a configuration standpoint. Each node has its own CPUs, memory, UNIX and PDE software, Teradata Database software, BYNET software, and access to one or more disk arrays.

Nodes are the building blocks of MPP systems. A system size is typically expressed in terms of number of nodes.

AMPs provide access to user data stored within tables that are physically stored on disk arrays.

Each AMP is associated with a Vdisk. Each AMP sees its Vdisk as a single disk. Teradata (AMP software) organizes its data on its disk space (Vdisk) using a Teradata "File System" structure.

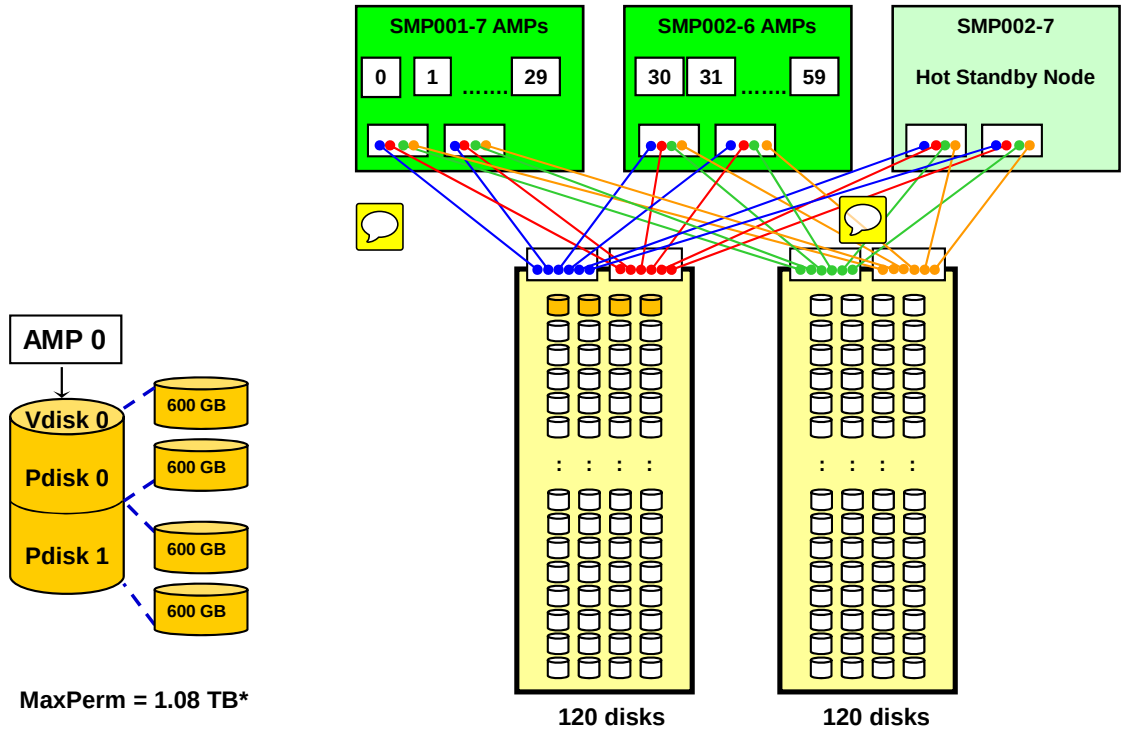
A Vdisk may be actually composed of multiple Pdisks - Physical disk. A Pdisk is assigned to physical drives in a disk array.

Example: 6650 and Internal 6844 Disk Arrays

The facing page contains an example of a 3-node (2+1) clique sharing two 6844 disk arrays.

Each node has Fibre Channel adapters and Fibre Channel cables (point-to-point connections) to connect to the disk arrays.

Example of 2+1 Node Teradata System



MaxPerm = 1.08 TB*

* Actual space is app. 90%.

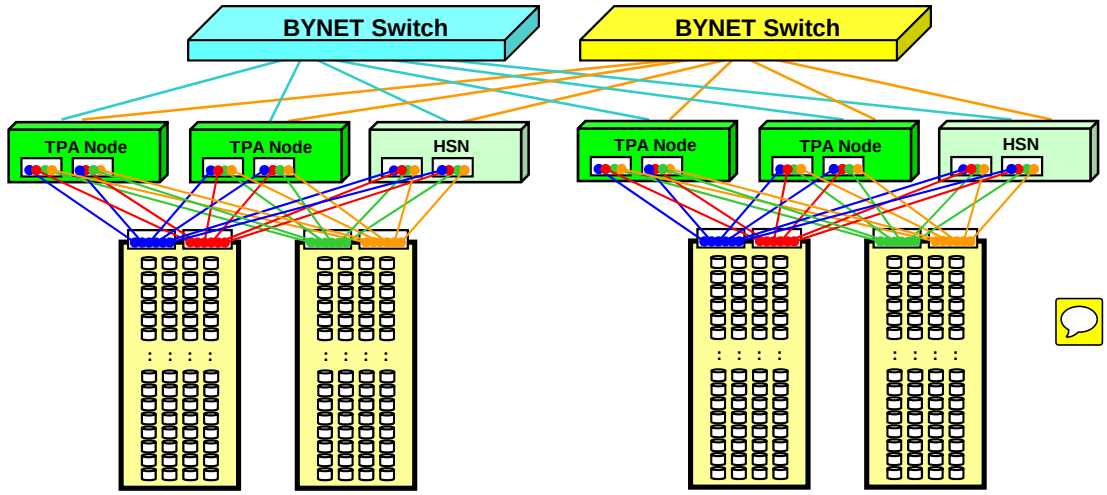
2+1 node clique sharing 240 drives; 30 AMPs/node; Linux System

Teradata Cliques

A **clique** is a set of Teradata nodes that share a common set of disk arrays. In the event of node failure, all vprocs can migrate to another available node in the clique. All nodes in the clique must have access to the same disk arrays.

The illustration on the facing page shows a 6-node system consisting of two cliques, each containing three nodes. Because all disk arrays are available to all nodes in the clique, the AMP vprocs will still have access to the rows they are responsible for.

Teradata Cliques

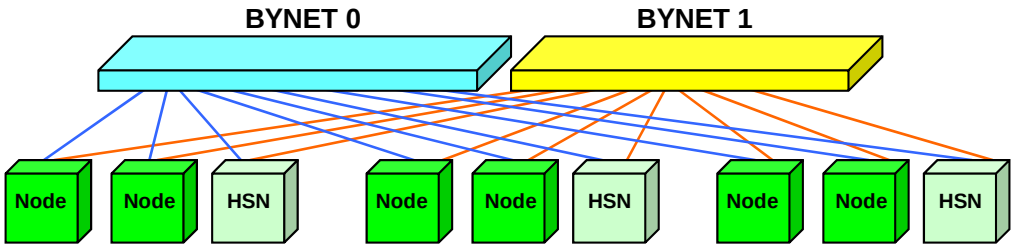


- **A clique is a defined set of nodes that share a common set of disk arrays.**
- **All nodes in a clique must be able to access all Vdisks for all AMPs in the clique.**
- **A clique provides protection from a node failure.**
- **If a node fails, all vprocs will migrate to the remaining nodes in the clique (Vproc Migration) or to a Hot Standby Node (HSN).**

BYNET

There are two physical **BYNETs,** BYNET 0 and BYNET 1. Both are fully operational and provide fault tolerance in the event of a BYNET failure. The BYNETs automatically handle load balancing and message routing. BYNET reconfiguration and message rerouting in the event of a component failure is also handled transparently to the application.

BYNET



The BYNET is a dual redundant, bi-directional interconnect network.

- All nodes are connected to both BYNETs. This example shows three (2+1) cliques.

BYNET Features:

- Enables multiple nodes to communicate with each other.
- Automatic load balancing of message traffic.
- Automatic reconfiguration after fault detection.
- Fully operational dual BYNETs provide fault tolerance.
- Scalable bandwidth as nodes are added.
- Even though there are two physical BYNETs to provide redundancy and bandwidth, the Teradata Database and TCP/IP software only see a single network.

BYNET Communication Protocols

Using communication-switching techniques, the **BYNET** allows for **point-to-point**, **multicast**, and **broadcast** communications among the nodes, thus supporting a monumental increase in throughput in very large databases. This technology allows Teradata users to grow massively parallel databases without fear of a communications bottleneck for any database operations.

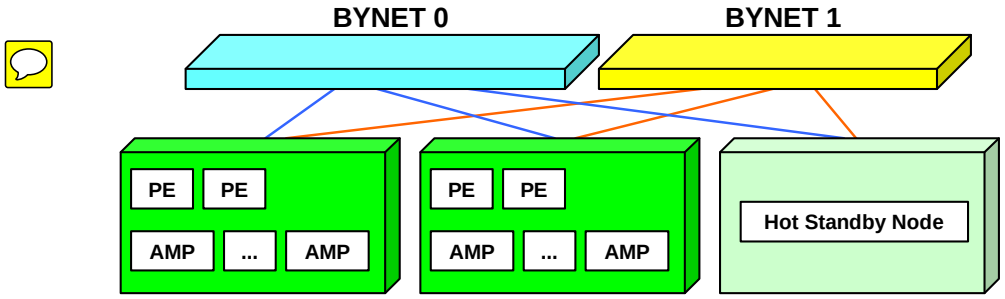
Although the BYNET software supports the **multi-cast** protocol, Teradata only uses this protocol with Group AMPs operations. This is a Teradata feature starting with release V2R5. Teradata software will use the **point-to-point** protocol whenever possible. When an all-AMP operation is needed, Teradata software uses the **broadcast** protocol to send messages to the different SMPs.

The BYNET is linearly scalable for point-to-point communications. For each new node added to a system with BYNET V4, an additional 960 MB of additional bandwidth is added to each BYNET, thus providing scalability as the system grows. Scalability comes from the fact that multiple point-to-point circuits can be established concurrently. With the addition of another node, more circuits can be established concurrently.

For broadcast and multicast operations with BYNET V4, the bandwidth is 960 MB per second per BYNET.

BYNET V1 (old implementation) had a bandwidth of 10 MB per second per direction per BYNET for a node.

BYNET Communication Protocols



Point-to-Point (one-to-one)

One vproc communicates with one vproc (e.g., 1 PE to 1 AMP). Scalable bandwidth:

- BYNET v2 – 60 MB x 2 (bi-directional) x 2 BYNETs = 240 MB per node
- BYNET v3 – 93.75 MB x 2 (bi-directional) x 2 BYNETs = 375 MB per node
- BYNET v4 – 240 MB x 2 (bi-directional) x 2 BYNETs = 960 MB per node

Multi-Cast (one-to-many)

One vproc communicates to a subset of vprocs (e.g., Group AMP operations).

Broadcast (one-to-all)

One vproc communicates to all vprocs (e.g., 1 PE to all AMPs). Not scalable.

Vproc Inter-process Communication

The “message passing layer” is a combination of two pieces of software and hardware– the PDE and the BYNET device drivers and software and the BYNET hardware.

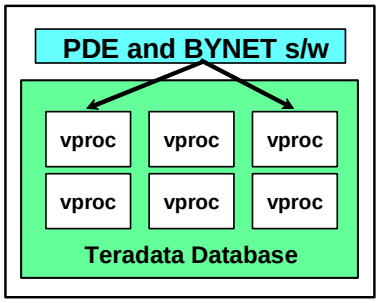
Communication among vprocs in an **MPP system** may be either inter-node or intra-node. When vprocs within the same node communicate they do not require the physical transport services of the **BYNET**. However, they do use the highest levels of the BYNET software even though the messages themselves do not leave the node.

When vprocs must communicate across nodes, they must use the physical transport services of the BYNET requiring movement of the data. Any broadcast messages, for example, will go out to the BYNET, even for the AMPs and PEs that are in the same node.

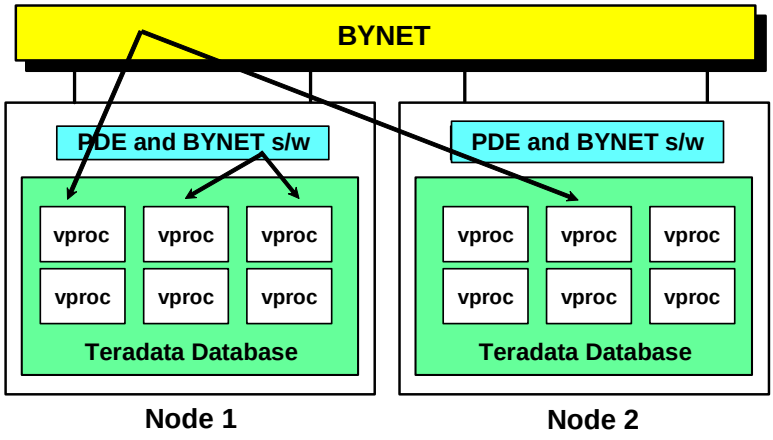
Communication among vprocs in a single SMP system occurs with the PDE and BYNET software, even though a physical BYNET does not exist in a single-node system.

Vproc Inter-process Communication

Single-Node System



MPP Systems



Examples of Teradata Database Systems

The facing page identifies various SMP servers and MPP systems that are supported for the Teradata Database.

The following dates indicate when these systems were generally available to customers (GCA – General Customer Availability).

– 5100M	January, 1996 (not described in this course)
– 4700/5150	January, 1998 (not described in this course)
– 4800/5200	April, 1999
– 4850/5250	June, 2000
– 4851/4855/5251/5255	July, 2001
– 4900/5300	March, 2002
– 4950/5350	December, 2002
– 4980/5380	August, 2003
– 5400E/5400H	March, 2005
– 5450E/5450H	April, 2006
– 5500E/5500C/5500H	March, 2007
– 2500/5550H	January, 2008
– 2550	October, 2008
– 1550	December, 2008
– 2555/5555C/H	March, 2009
– 1600/5600C/H	February, 2010
– 2650/5650C/H	July, 2010 (Internal release; Official release Oct 2010)
– 6650C/H, 6680	2011

The Teradata Database is also available on non-Teradata platforms. The Teradata Database is available on the Intel-based mid-range platforms running Microsoft Windows 2003 or Linux. For example, Dell provides processing nodes that are used in some of the Teradata appliance systems.

Examples of Teradata Database Systems

Examples of systems used with the Teradata Database include:

Active Enterprise Data Warehouse Systems

5200/525x	– up to 2 nodes/cabinet
5300/5350/5380	– up to 4 nodes/cabinet
5400/5450	– up to 10 nodes/cabinet
5500/555x/56xx	– up to 9 nodes/cabinet
6650/6680/6690	– up to 4 nodes/cabinet with associated storage

The basic building block is the SMP (Symmetric Multi-Processing) node.

Common characteristics of these systems:

- MPP systems that use the BYNET interconnect
- Single point of operational control – AWS or SWS
- Rack-based systems – each technology is encapsulated in its own chassis

Key differences:

- Speed and capacity of SMP nodes and systems
- Cabinet architecture
- BYNET interface cards, switches and speeds

*BYNET V4 – up to 4096 nodes

6650 Cabinets

The facing page contains two pictures of rack-based cabinets. This represents a two cabinet 3+1 6650 clique.

54xx, 55xx, and 56xx systems also used a rack-based cabinet. The rack was initially designed for the 54xx systems and has been improved on with later systems such as 55xx and 56xx systems.

This redesign allows for better cooling and maintenance and has a black and gray appearance. This design is also used with the LSI disk array cabinets. The 56xx cabinet is a different cabinet and is approximately 4" deeper than the 55xx cabinets.

An older style rack or cabinet is used for the 4700, 4800, 4850, 4851, 4855, 4900, 4950, 4980, 5200, 5250, 5251, 5255, 5300, 5350, and 5380 systems. This cabinet was similar in size and almond in color.

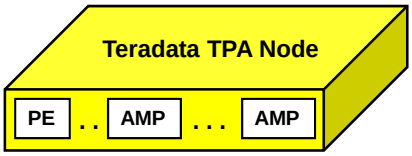
The approximate external dimensions of this rack or cabinet are:

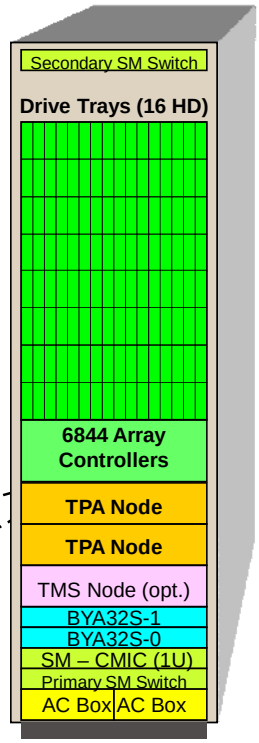
Height – 77"
Width – 24" (inside rails are 19" apart and this is often referred to as a 19" wide rack)
Depth – 40" (the 56xx/66xx cabinet is 44" deep)

This industry-standard rack is referred to as a 40U rack where a U is a unit of measure of height of 1.75" or 4.445 cm.

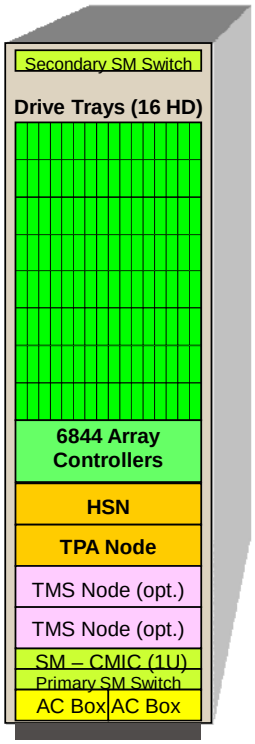
The system or processor cabinet includes a Server Management (SM) chassis which is often referred to as the CMIC (Chassis Management Interface Controller). This component is part of the server management subsystem and interfaces with the AWS or SWS.

6650 Cabinets





6650



6650

3+1 6650 Clique

Teradata uses industry standard rack-based cabinets.

HSN – Hot Standby Node

What makes Teradata's MPP Platforms Special?

The facing page lists the major features of Teradata's MPP systems.

Acronyms:

PUT – Parallel Upgrade Tool
AWS – Administration Workstation
SWS – Service Workstation – utilizes Server Management Web Services (SMWeb) for the 56xx.

What Makes Teradata's MPP Platforms Special?

Key features of Teradata's MPP systems include:

- **Teradata Database software – allows the Teradata Database to execute on multiple nodes and act as a single instance.**
- **Scalable BYNET Interconnect – as you add nodes, you add bandwidth.**
- **Operating system software (e.g., Linux) for a node is only aware of the resources within the node and only has to manage those resources.**
- **AWS/SWS – single point of operational control and scalable server management.**
- **PUT (Parallel Upgrade Tool) – simplifies installation/upgrade of software across many nodes.**
- **Redundant (availability) components. Examples include:**
  - **Hot Standby Nodes**
  - **Two BYNETs**
  - **Two Disk Array Controllers within a Disk Array**
  - **Dual AC capability for increased availability**
  - **N+1 Power Supplies within a processing node and disk arrays**

Summary

The facing page summarizes the key points and concepts discussed in this module.

Summary

- Teradata Database is a software implementation of Teradata.
  - AMPs and PEs are implemented as virtual processors (Vprocs).
- The Teradata Database utilizes a "Shared Nothing" Architecture – each AMP has its own memory and manages its own disk space.
  - Teradata is called a Trusted Parallel Application (TPA).
- Multiple nodes may be configured to provide a Massively Parallel Processing (MPP) system.
- A clique is a defined set of nodes that share a common set of disk arrays.
- The Teradata Database is a linearly expandable RDBMS – as your database grows, additional nodes may be added.

Module 7: Review Exercises

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 7: Review Questions

Complete the following.

1. **Each AMP has its own memory and manages its own disk space and executes independently of other AMPs. This is referred to as a _____ _____ architecture.**

2. **The software component that allows the Teradata Database to execute in different operating system environments is the _____.**

3. **A physical message passing interconnect is called the _____.**

4. **A clique provides protection from a _____ failure.**

5. **If a node fails, all vprocs will migrate to the remaining nodes in the clique. This feature is referred to as _____ _____.**

6. **The _____ or _____ provides a single point of operational control for Teradata MPP systems.**

7. **A _____ node is part of a system configuration, is connected to the BYNET, and executes the Teradata Database software.**

8. **A _____ node is part of a system configuration, connects to the BYNET, and is used to execute application software other than Teradata Database software.**

9. **A _____ node is part of a system configuration, connects to the BYNET, and is used as a spare node in the event of a node failure.**

Notes

Module 8

Data Protection

After completing this module, you will be able to:

- **Explain the concept of FALLBACK tables.**
- **List the types and levels of locking provided by Teradata.**
- **Describe the Recovery, Transient and Permanent Journals and their function.**
- **List the utilities available for archive and recovery.**

Notes

Table of Contents

Data Protection Features 8-4
Disk Arrays 8-6
RAID Technologies 8-8
RAID 1 – Mirroring 8-10
  RAID 10 – Striped Mirroring 8-10
RAID 1 Summary 8-12
Cliques 8-14
  Large Cliques 8-14
Teradata Vproc Migration 8-16
Hot Standby Nodes (HSN) 8-18
  Large Cliques 8-18
Performance Degradation with Node Failure 8-20
    Restarts 8-20
Fallback 8-22
Fallback Clusters 8-24
Fallback and RAID Protection 8-26
Fallback and RAID 1 Example 8-28
  Fallback and RAID 1 Example (cont.) 8-30
  Fallback and RAID 1 Example (cont.) 8-32
  Fallback and RAID 1 Example (cont.) 8-34
  Fallback and RAID 1 Example (cont.) 8-36
Fallback vs. non-Fallback Tables Summary 8-38
Clusters and Cliques 8-40
Locks 8-42
Locking Modifier 8-44
  ACCESS 8-44
  NOWAIT 8-44
Rules of Locking 8-46
Access Locks 8-48
Transient Journal 8-50
Recovery Journal for Down AMPs 8-52
Permanent Journal 8-54
Archiving and Recovering Data 8-56
Module 8: Review Questions 8-58

Data Protection Features

Disk Arrays – Disk arrays provide RAID 1, RAID 5, or RAID S data protection. If a disk drive fails, the array subsystem provides continuous access to the data. Systems with disk arrays are configured with redundant Fibre adapters, buses, and array controllers to provide highly available access to the data.

Clique – a set of Teradata nodes that share a common set of disk arrays. In the event of node failure, all vprocs can migrate to another available node in the clique. All nodes in the clique must have access to the same disk arrays.

Locks – Locking prevents multiple users who are trying to change the same data at the same time from violating the data's integrity. This concurrency control is implemented by **locking** the desired data. Locks are automatically acquired during the processing of a request and released at the termination of the request. In addition, users can specify locks. There are four types of locks: Exclusive, Write, Read, and Access.

Fallback – protects your data by storing a second copy of each row of a table on an alternative "fallback AMP". If an AMP fails, the system accesses the fallback rows to meet requests. Fallback provides AMP fault tolerance at the table level. With Fallback tables, if one AMP fails, all of the table data is still available. Users may continue to use Fallback tables without any loss of available data.

Down-AMP Recovery Journal – started automatically when the system has a failed or down AMP. Its purpose is to log any changes to rows which reside on the down AMP.

Transient Journal – exists to permit the successful rollback of a failed transaction. Transactions are not committed to the database until an End Transaction request has been received by the AMPs, either implicitly or explicitly. Until that time, there is always the possibility that the transaction may fail in which case the participating table(s) must be restored to their pre-transaction state.

Permanent Journal – provides selective or full database recovery to a specified point in time by keeping either before-image or after-images of rows in a journal. It permits recovery from unexpected hardware or software disasters.

ARC and NetVault/NetBackup – ARC command scripts provide the capability to backup and restore the Teradata database. The **NetVault** and **NetBackup** utilities provide a GUI based front-end for creation and execution of ARC command scripts.

Data Protection Features

Facilities that provide system-level protection

Disk Arrays

- RAID data protection (e.g., RAID 1)
- Redundant SCSI and/or Fibre Channel buses and array controllers

Cliques and Vproc Migration

- SMP or O.S. failures - Vprocs can migrate to other nodes within the clique.

Facilities that provide Teradata DB protection

Fallback	– provides data access with a “down” AMP
Locks	– provides data integrity
Transient Journal	– automatic rollback of aborted transactions
Down AMP Recovery Journal	– fast recovery of fallback rows for AMPs
Permanent Journal	– optional before and after-image journaling
ARC	– Archive/Restore facility
NetVault and NetBackup	– provide tape management and ARC script creation and scheduling capabilities

Disk Arrays

Disk arrays utilize a technology called RAID (**Redundant Array of Independent Disks)** Spanning the entire spectrum from personal computers to mainframes, disk arrays (utilizing RAID technology) offer significant improvements in availability, reliability and maintainability of information storage, along with higher performance. Yet the concept behind disk arrays is relatively simple.

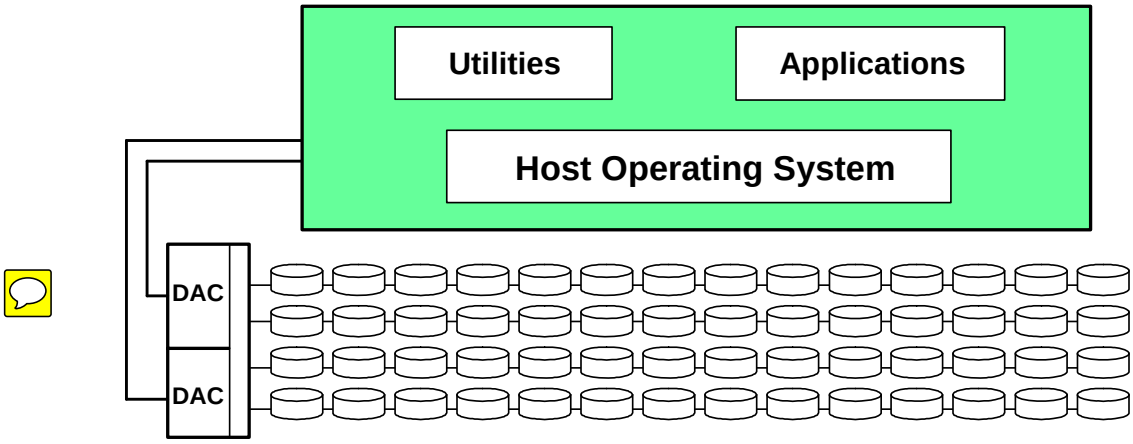
A disk array subsystem consists of controller(s) which drive a set of disks. Typically, a disk array is configured to represent a number of logical volumes (or disks), each of which appears to be a physical disk to the user. A logical volume can be configured to reside on multiple physical disks. The fact that a logical volume is located on 1 or more disks is transparent to the user.

There is one immediate advantage of having the data spread across a number of individual separate disks which arises from the redundant manner in which the data can be stored in the disk array. The remarkable benefit of this feature is that if any single disk in the array fails, the unit continues to function without loss of data. This is possible because redundancy information is stored separate from the data. The redundancy information, as will be explained, can be a copy of the data or other information that can be used to reconstruct any data that was stored on a failed disk.

Secondly, performance increases for specific applications are possible as the effective seek time for finding records on a given disk can potentially be reduced by allowing multiple simultaneous accesses of different blocks on different disks. Alternatively, with a different architecture, the rate at which data is transferred to and from the disk array can be increased significantly over that of a single disk utilizing parallel reads and writes of the data spread across the disks in the array. This function is referred to as “striping the data”.

Finally, disk array subsystem maintenance is typically simplified because it is possible to replace (“hot swap”) individual disks and other components while the system continues to function. You no longer have to bring down the system to replace a disk.

Disk Arrays



Why Disk Arrays?

- **High availability through data mirroring or data parity protection.**
- **Better I/O performance through implementation of RAID technology at the hardware level.**
- **Convenience – automatic disk recovery and data reconstruction when mirroring or data parity protection is used.**

RAID Technologies

RAID is an acronym for **R**edundant **A**rray of **I**ndependent **D**isks. The term was coined in 1988 in a paper describing array configuration and application by researchers and authors Patterson, Gibson and Katz of the University of California at Berkeley. The word redundant implies that data, functions and/or components have been duplicated in the array's architecture. Duplication of data, functions, and hardware ensures that even in the event of a failed drive or other components, data is not lost and is continuously available.

The industry currently has agreed upon six RAID configuration levels and designated them as RAID 0 through RAID 5. The physical configuration is dictated to some extent by the choice of RAID level; however, RAID conventions specify more precisely how data is stored on disk.

RAID 0	Data striping
RAID 1	Disk mirroring
RAID 2	Parallel array, hamming code
RAID 3	Parallel array with parity
RAID 4	Data parity protection, dedicated parity drive
RAID 5	Data parity protection, interleaved parity

With Teradata, the RAID 1 is most commonly used. RAID 5 (data parity protection) is also available with some arrays.

There are other RAID technologies that are defined by specific vendors or are accepted in the data processing industry. For example, RAID 10 or RAID 1+0 (or RAID 0+1) is considered to be "striped mirroring". RAID level classifications do not imply superiority of one mode over another. Each mode has its rightful application. In fact, these modes of operation can be combined within a single system configuration, within product limitations, to obtain maximum flexibility and performance.

The advantages of RAID 1 (compared to RAID 5) include:

Superior Performance

- Mirroring provides the best read and write throughput.
- Maximizes the performance capabilities of controllers and disk drives.
- Best performance when a drive has failed.
- Less reconstruction impact when a drive has failed.

Superior Availability

- Less susceptible to a double disk failure in a RAID drive group.
- Faster reconstruction of a failed drive - shorter vulnerability period during reconstruction.

Superior Price/Performance - the performance advantage of RAID 1 outweighs the additional cost for typical Teradata warehouses.

RAID Technologies

RAID – Redundant Array of Independent Disks

RAID technology provides data protection at the disk drive level. With RAID 1 and RAID 5 technologies, access to the data is continuous even if a disk drive fails.

RAID technologies available with Teradata:

RAID 1	Disk mirroring, used with NetApp (LSI Logic) and EMC^2 Disk Arrays.
RAID 5	Data parity protection, interleaved parity, RAID 5 provides more capacity, but less performance than RAID 1.

For Teradata:

RAID 1	Most useful with typical Teradata data warehouses (e.g., Active Data Warehouses). Most frequently used RAID technology.
RAID 5	Most useful when creating archival data warehouses that require less expensive storage and where performance is not as important. Not frequently used with Teradata systems (not covered in this class).

RAID 1 – Mirroring

RAID 1 is data mirroring protection. The **RAID 1** technology requires each primary data disk to have a companion disk or mirror. The contents of the primary disk and the mirror disk are identical.

When data is written on the primary disk, a write also occurs on the mirror disk. The mirroring process is invisible to the user. For this reason, RAID 1 is also called transparent mirroring.

With RAID solutions, mirroring is managed by the controller, which provides a higher level of performance. Performance is improved because data can be read from either the primary (data) drive or the mirror. The controller decides which read/write assembly (drive actuator) is closest to the requested data.

If the primary data disk fails, the mirror disk can be accessed without data loss. There is a minor performance penalty if a drive fails because the array controller can read from either drive if both drives are available. If either disk fails, the disk array controller can copy the data from the remaining drive to a replacement drive while normal operations continue.

RAID 10 – Striped Mirroring

When user data is to be written to the array, the controller instructs the array to write a block of data to one drive pair to the defined stripe depth. Subsequent data blocks are written concurrently to contiguous sectors in the next drive pair to the defined stripe depth. In this manner, data are striped across the array of drives, utilizing multiple drives and actuators.

With LSI Logic arrays, striped mirroring is automatic when you create a drive group (with RAID 1 technology) that has multiple mirrored pairs of disks.

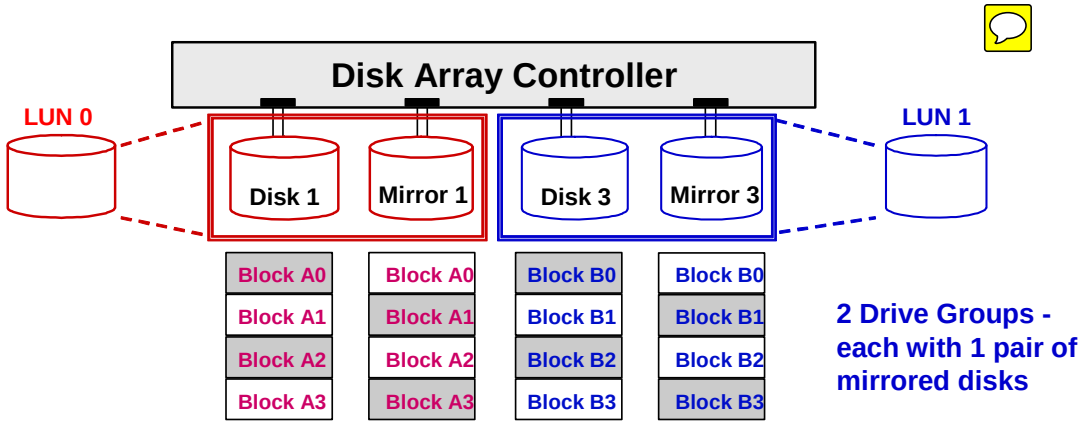
If an application (e.g., Teradata Database) uniformly distributes data, striped mirroring (RAID 10 or 1+0) and mirroring (RAID 1) will have similar performance.

If an application (database) partitions data, striped mirroring (RAID 10) can lead to performance gains over mirroring (RAID 1) because array controllers equally spread I/O's between channels in the array.

Striped Mirroring is NOT necessary with Teradata.

RAID 1 – Mirroring

- 2 Drive Groups each with 1 mirrored pair of disks
- Operating system sees 2 logical disks (LUNs) or volumes
- If LUN 0 has more activity, more disk I/Os occur on the first two drives in the array.



Notes:

- If the physical drives are 600 GB each, then each LUN or volume is effectively 600 GB.
- If both logical units (or volumes) are assigned to an AMP, then the AMP will have approximately 1.2* TB assigned to it.

* Actual MaxPerm space will be a little less.

RAID 1 Summary

RAID 1 characteristics include:

- Data is fully replicated
- Easy to understand technology
- Follows a traditional approach
- Transparent to the operating system
- Redundant drive is affected only by write operations

RAID 1 advantages include:

- High I/O rate (small logical block size)
- Maximum data availability
- Minor performance penalty with single drive failure
- No performance penalty in write intensive environments

RAID 1 disadvantage is:

- Only 50% of total disk space is available for user data. Therefore, RAID 1 has 50% overhead in disk space usage.

Summary

- RAID 1 provides high data availability and performance, but storage costs are high.
- Striped mirroring is not necessary with Teradata.

RAID 1 for Teradata - most useful with typical Teradata data warehouses (e.g., Active Data Warehouses).

RAID 5 for Teradata - most useful when creating archival data warehouses that require less expensive storage and where performance is not as important.

RAID 1 Summary

Characteristics

- data is fully replicated
- striped mirroring is possible with multiple pairs of disks in a drive group
- transparent to operating system

Advantages (compared to RAID 5)

- Provides maximum data availability
- Mirroring provides the best read and write throughput
- Maximizes the performance capabilities of controllers and disk drives
- Minimal performance issues when a drive has failed
- Less reconstruction impact when a drive has failed

Disadvantage

- 50% of disk space is used for mirrored data

Summary

- RAID 1 provides best data availability and performance, but storage costs are higher.
- ***Striped Mirroring is NOT necessary with Teradata.***

Cliques

A **clique** is a set of Teradata nodes that share a common set of disk arrays. In the event of node failure, all vprocs can migrate to available nodes in the clique. All nodes in the clique must have access to the same disk arrays.

The illustration on the facing page shows a three-node clique. In this example, each AMP has 24 AMP vprocs.

In the event of node failing, the remaining nodes will attempt to absorb all vprocs from the failed node.

Large Cliques

A **large clique** is usually a set of 8 Teradata nodes that share a common set of disk arrays via a set of Fibre Channel switches. In the event of a node failure, AMP vprocs can migrate to the other available nodes in the clique. In this case, work is distributed among 7 nodes and the performance degradation is approximately 14%.

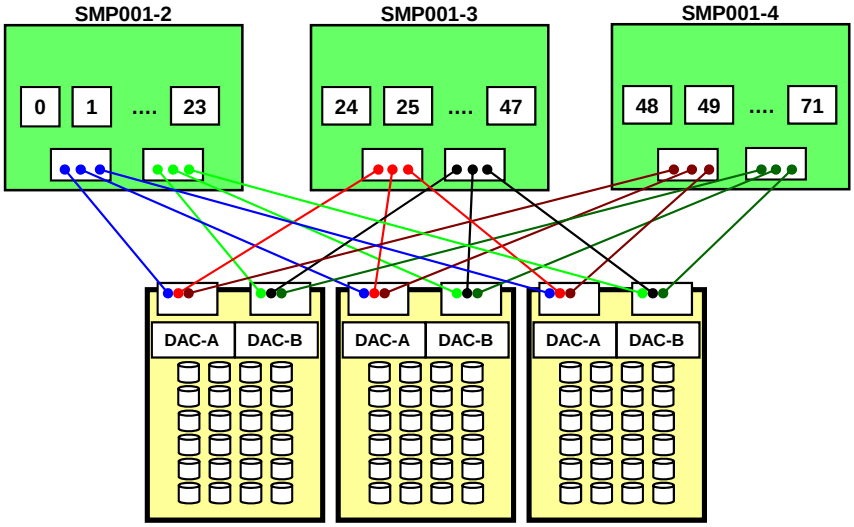
After the failed node is recovered/repaired and rebooted, a second restart of Teradata is needed to reuse the node that had failed. The restart will redistribute the AMPs to the recovered node.

Acronyms:

DAC – Disk Array Controller

Cliques

Clique – a set of SMPs that share a common set of disk arrays.



Example of a 2650 clique (3 nodes, no HSN) – 24 AMPs/node.

Teradata Vproc Migration

If a TPA node (running Teradata) fails, Teradata restarts and the AMP vprocs that were executing on the failed node are started on other nodes within the clique.

PE vprocs that are assigned to channel connections do not migrate to another node. PE vprocs that are assigned to gateway connections may or may not (depending on configuration) migrate to another node within the clique.

If a node fails, the vprocs from the failed node are distributed between the remaining nodes in the clique. The vconfig.out file determines the node on which vprocs will start if all of the nodes in the clique are available.

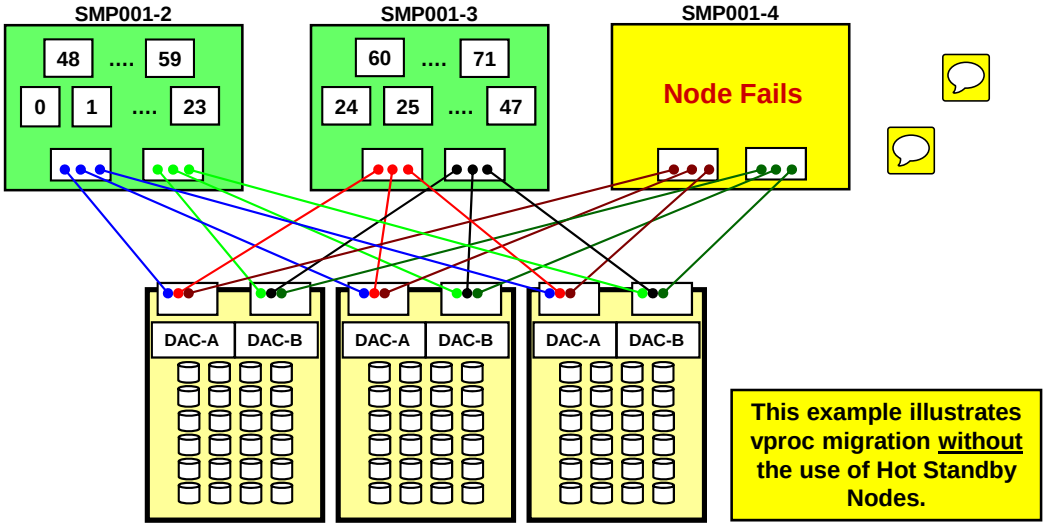
The following is from a "Get Config" command following the failure of SMP001-4.

DBS LOGICAL CONFIGURATION

Vproc Number	Rel. Vproc#	Node ID	Movable	Crash Count	Vproc State	Config Status	Config Type	Cluster/ Host No.	RcvJrnl/ Host Type
0*	1	1-02	Yes	0	ONLINE	Online	AMP	0	On
1	2	1-02	Yes	0	ONLINE	Online	AMP	1	On
2	3	1-02	Yes	0	ONLINE	Online	AMP	2	On
3	4	1-02	Yes	0	ONLINE	Online	AMP	3	On
:	:	:	:	:	:	:	:	:	:
22	23	1-02	Yes	0	ONLINE	Online	AMP	22	On
23	24	1-02	Yes	0	ONLINE	Online	AMP	23	On
24	1	1-03	Yes	0	ONLINE	Online	AMP	24	On
25	2	1-03	Yes	0	ONLINE	Online	AMP	25	On
26	3	1-03	Yes	0	ONLINE	Online	AMP	26	On
:	:	:	:	:	:	:	:	:	:
46	23	1-03	Yes	0	ONLINE	Online	AMP	46	On
47	24	1-03	Yes	0	ONLINE	Online	AMP	47	On
48	25	1-02	Yes	0	ONLINE	Online	AMP	48	On
49	26	1-02	Yes	0	ONLINE	Online	AMP	49	On
50	27	1-02	Yes	0	ONLINE	Online	AMP	50	On
:	:	:	:	:	:	:	:	:	:
59	36	1-02	Yes	0	ONLINE	Online	AMP	59	On
60	25	1-03	Yes	0	ONLINE	Online	AMP	60	On
61	26	1-03	Yes	0	ONLINE	Online	AMP	61	On
62	27	1-03	Yes	0	ONLINE	Online	AMP	62	On
:	:	:	:	:	:	:	:	:	:
71	36	1-03	Yes	0	ONLINE	Online	AMP	71	On

Teradata Vproc Migration

Clique – a set of SMPs that share a common set of disk arrays.



After vproc migration, the two remaining nodes each have 36 AMPs.

- **After failed node is repaired, a second restart is needed for failed node to rejoin the configuration.**

Hot Standby Nodes (HSN)

A **Hot Standby Node (HSN)** is a node that is part of a clique and the hot standby node is not configured (initially) to execute any Teradata vprocs. If a node in the clique fails, the AMPs from the failed node move to the hot standby node. The performance degradation is 0%.

When the failed node is recovered/repaired and restarted, it becomes the new hot standby node. A second restart of Teradata is not needed.

Characteristics of a hot standby node are:

- A node that is a member of a clique.
- Does not normally participate in the trusted parallel application (TPA).
- Can be brought into the TPA to compensate for the loss of a node in the clique.

Hot Standby Nodes are positioned as a performance continuity feature.

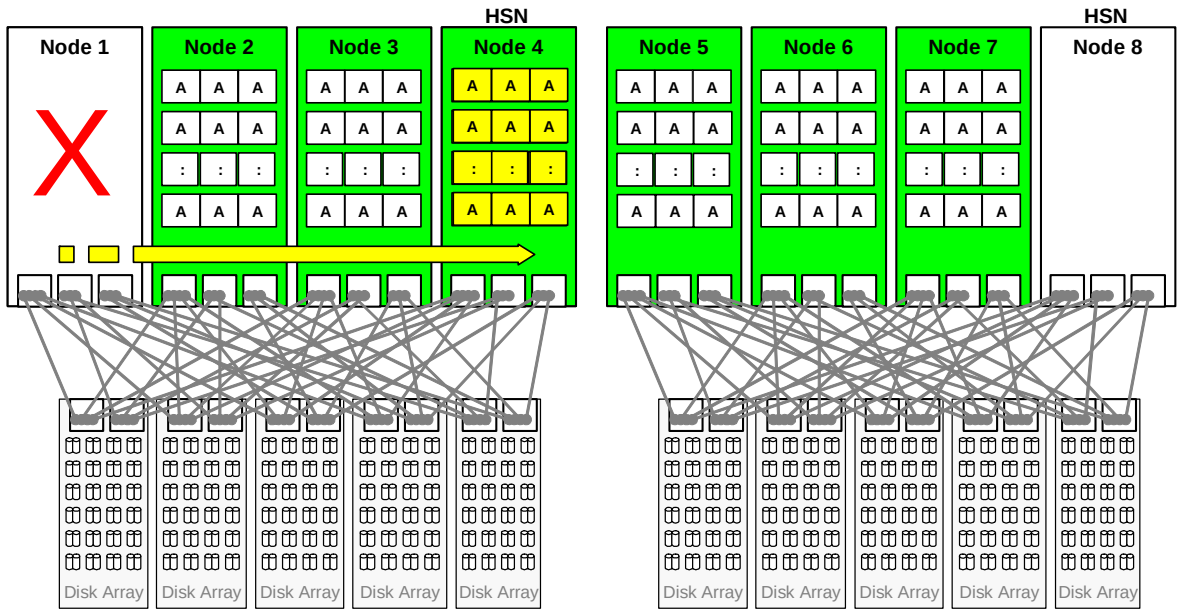
Large Cliques

A **large clique** can also utilize a Hot Standby Node (Node).

For example, an 8-node large clique with a Hot Standby Node would consist of 7 nodes running Teradata and 1 Hot Standby Node. The performance degradation would be 0% for an all-AMP operation when a node fails in the clique. This configuration is often referred to as a 7+1 configuration.

Large Clique configurations have not been supported since the introduction of the 5500.

Hot Standby Nodes (HSN)



1. Performance Degradation is 0% as AMPs are moved to the Hot Standby Node.
2. When Node 1 is recovered, it becomes the new Hot Standy Node.

This example illustrates vproc migration using a Hot Standby Node.

Performance Degradation with Node Failure

The facing page displays 2 examples of the performance degradation with all-AMP operations that occur when a node fails. Note: WL - Workload

The top example illustrates two 3-node cliques and the performance degradation of 50% for an all-AMP operation when a node fails in one of the cliques.

From a simple perspective, if you have 3 nodes in a clique and you lose a node, you would logically think 33% performance degradation. In reality, the performance cost or degradation is 50%. Assume 3 nodes, 72 AMPs, and you execute an all-AMPs query. This query uses 240 CPU seconds per node to complete the query. The 3 nodes use a total of 720 CPU seconds to do the work. Another way to look at is that each AMP needs 10 CPU seconds or 72 AMPs x 10 CPU seconds equals 720 CPU seconds of work to be done.

A node fails and now there are 2 nodes to complete the query. There are still 72 AMPs and the query still needs 720 CPUs seconds to complete the query, but now there are only 2 nodes. Each node will need about 360 CPUs seconds to complete the query. Each node has about 50% more work to do. This is why it is considered a 50% performance cost.

Another way of looking at a query is from the response time back to the user. From a user perspective, let's assume that response time back to the user with all 4 nodes normally active is 1 minute (60 seconds) of wall clock time. The wall clock response time with only 2 active nodes is 90 seconds. (Since there are fewer nodes, the query is going to take longer to complete.) From the user perspective, the response time is 50% longer (30/60).

It is true that if you take 67% of 90, you will get 60 and you may think that the degradation is 33%. However, 90 seconds is not the normal response time. This normal response time is 60 seconds and the exception is 90 seconds, therefore the performance is worse by 50%. The percentage is calculated from the "normal".

The bottom example illustrates two 3-node cliques, each with a hot standby node (2 TPA nodes) and the performance degradation of 0% for an all-AMP operation when a node fails in the clique. This configuration is often referred to as a 2+1 configuration.

Restarts

In the first (top) example, after the failed node is recovered/repaired and rebooted, a second restart of Teradata is needed to reuse the node that had failed. The restart will redistribute the AMPs to the recovered node.

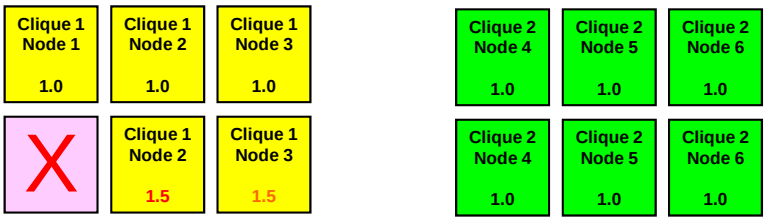
With a hot standby node, when the failed node is recovered/repaired and restarted, it becomes the new hot standby node within the clique. A second restart of Teradata is not needed.

Performance Degradation with Node Failure

2 Cliques without HSN nodes (3 nodes) – performance degradation of 50% with node failure.

Workload = 6.0
Clique WL = 3.0
Node WL = 1.00

Workload = 6.0
Clique WL = 3.0
Node WL = 1.5



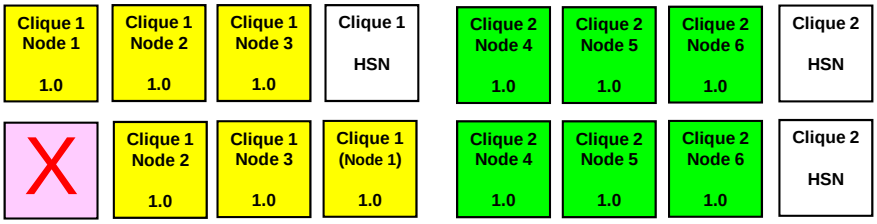
When a node fails, Teradata restarts.

After the node is repaired, a second restart of Teradata is required to allow the node to rejoin the configuration.

2 Cliques each with a HSN (3+1 nodes) – performance degradation of 0% with node failure.

Workload = 6.0
Clique WL = 3.0
Node WL = 1.00

Workload = 6.0
Clique WL = 3.0
Node WL = 1.0



When a node fails, Teradata restarts.

After the node is repaired, it becomes the new Hot Standby Node. A second restart of Teradata is not required.

Fallback

Fallback protects your data by storing a second copy of each row of a table on an alternative "fallback AMP". If an AMP fails, the system accesses the fallback rows to meet requests. Fallback provides AMP fault tolerance at the table level. With Fallback tables, if one AMP fails, all of the table data is still available. Users may continue to use Fallback tables without any loss of available data.

When a table is created, or any time after its creation, the user may specify whether or not the system should keep a fallback copy. If Fallback is specified, it is automatic and transparent to the user.

Fallback guarantees that the two copies of a row will always be on different AMPs. Therefore, if either AMP fails, the alternate row copy is still available on the other AMP.

Certainly there is a benefit to protecting your data. However, there are costs associated with that benefit. They are: twice the disk space for storage and twice the I/O for Inserts, Updates, and Deletes. (However, the Fallback option does not require any extra I/O for SELECT operations and the fallback I/O will be performed in parallel with the primary I/O.)

The benefits of Fallback include protecting your data from hardware (disk) failure, protecting your data from software (node) failure, automatic recovery and minimum recovery time after repairs or fixes are complete.

A hardware (disk) or software (vproc) failure causes an AMP to be taken off-line until the problem is corrected.

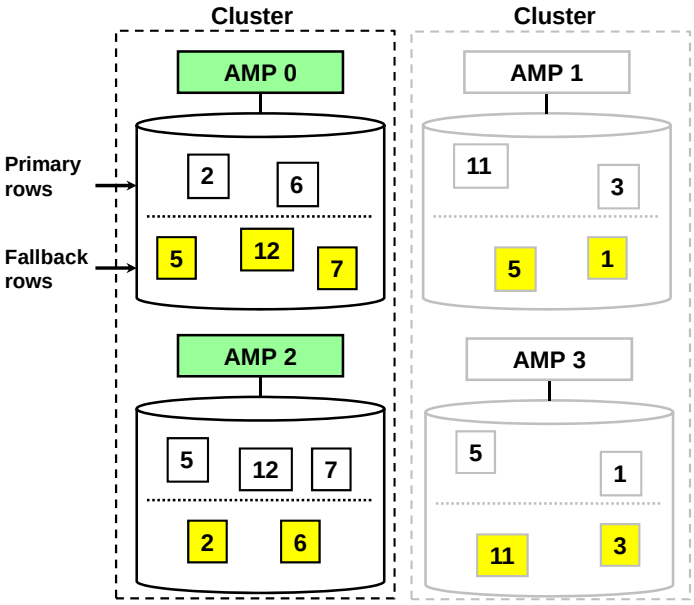
During this period, Fallback tables are fully available to users.

When the AMP is brought back on-line, the associated Vdisk is refreshed to reflect any changes during the off-line period.

Fallback

A Fallback table is fully available in the event of an unavailable AMP.

A Fallback row is a copy of a "Primary row" which is stored on a different AMP.



Benefits of Fallback

- Permits access to table data during AMP off-line period.
- Adds a level of data protection beyond disk array RAID.
- Automatic restore of data changed during AMP off-line.
- Critical for high availability applications.

Cost of Fallback

- Twice the disk space for table storage.
- Twice the I/O for Inserts, Updates and Deletes.

Loss of any two AMPs in a cluster causes RDBMS to halt!

Fallback Clusters

A **cluster** is a group of AMPs that act as a single fallback unit. Clustering has no effect on the distribution of the Primary rows of a table. The Fallback row copy however, will always go to a different AMP in the same cluster.

The cluster size is set when Teradata is configured and the only choice for new systems is 2-AMP clusters. Years ago, AMP clusters ranged from 2 to 16 AMPs per cluster and were commonly set as groups of 4 AMPs. Starting with 5450 systems, all clusters are defined as 2 AMP clusters.

Should an AMP fail, the primary and fallback row copies stored on that AMP cannot be accessed. However, their alternate copies are available through the other AMPs in the same cluster.

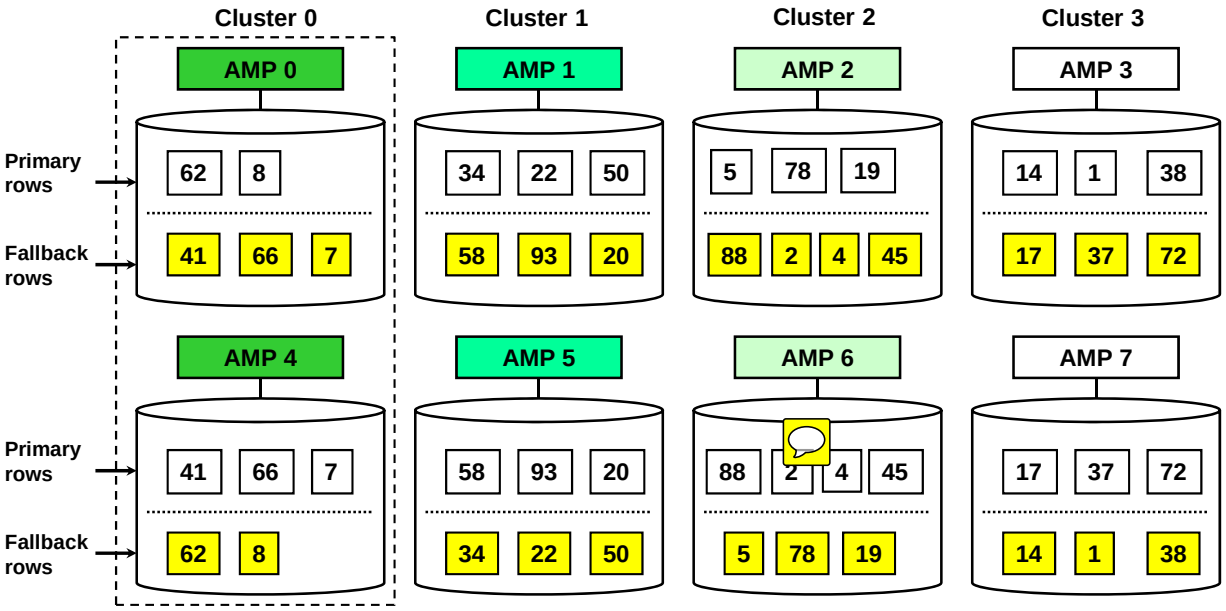
The loss of an AMP in a cluster has no effect upon other clusters. It is possible to lose one AMP in each cluster and still have full access to all Fallback-protected table data. If both AMPs fail in a cluster, then Teradata halts.

While an AMP is down, the remaining AMPs in the cluster must do their own work plus the work of the down AMP.

A small cluster size (e.g., 2 AMP cluster) reduces the chances of have 2 down AMPs in a single cluster which would cause a non-operational configuration. With today's new systems, a typical cluster size of 2 AMPs provides the best option to maximize availability.

Fallback Clusters

- A Fallback cluster is a defined set of 2 AMPs across which fallback is implemented.
- Loss of one AMP in the cluster permits continued table access.
- Loss of two AMPs in the cluster causes the RDBMS to halt.



Fallback and RAID Protection

RAID 1 mirroring and RAID 5 data parity protection provide protection in the event of a disk drive failure.

Fallback provides another level of data protection beyond disk mirroring or data parity protection.

Examples of other failures that Fallback provides protection against include:

- Multiple drive failures in the same drive group
- An array is not available (e.g., both disk array controllers fail in the disk array)
- An AMP is not available (e.g., a software problem)

Fallback and RAID Protection

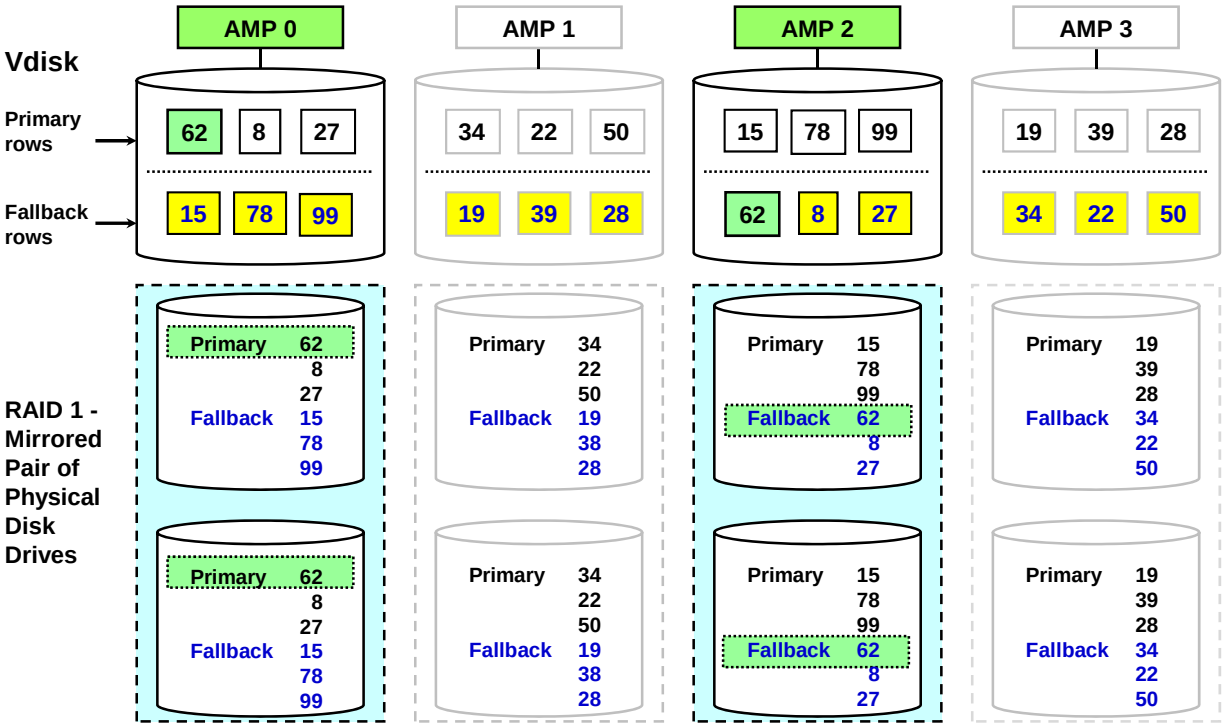
- RAID 1 Mirroring or RAID 5 Data Parity Protection provides protection in the event of disk drive failure.
  - Provides protection at a hardware level
  - Teradata is unaware of the RAID technology used
- Fallback provides an additional level of data protection and provides access to data when an AMP is not available (not online).
- Additional types of failures that Fallback protects against include:
  - Multiple drives fail in the same drive group,
  - Disk array is not available
    - Both disk array controllers fail in a disk array
    - Two of the three power supplies fail in a disk array
  - AMP is not available (e.g., software or data error)
- The combination of RAID 1 and Fallback provides the highest level of availability.

Fallback and RAID 1 Example

The next set of pages contains an example of how Fallback and RAID 1 Mirroring work together.

Fallback and RAID 1 Example

This example assumes that RAID 1 Mirroring is used and the table is fallback protected.



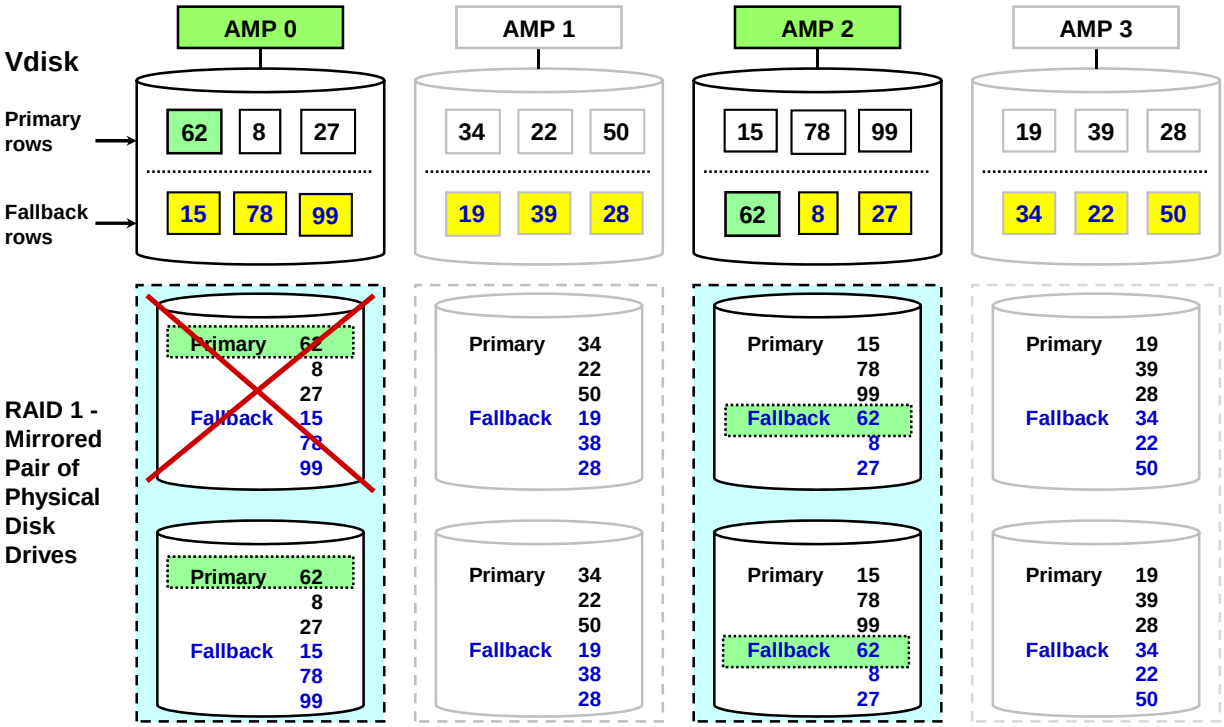
Fallback and RAID 1 Example (cont.)

The example of how Fallback and RAID 1 Mirroring work together is continued.

In this example, one disk drive has failed in the first drive group. Is Fallback needed? No. As a matter of fact, Teradata doesn't even realize that the drive has failed. The disk array continues to provide access to the data directly from the second disk drive in the drive group. The disk array controller will send a "fault" or error message to the AWS.

Fallback and RAID 1 Example (cont.)

Assume one disk drive fails. Is Fallback needed in this example?



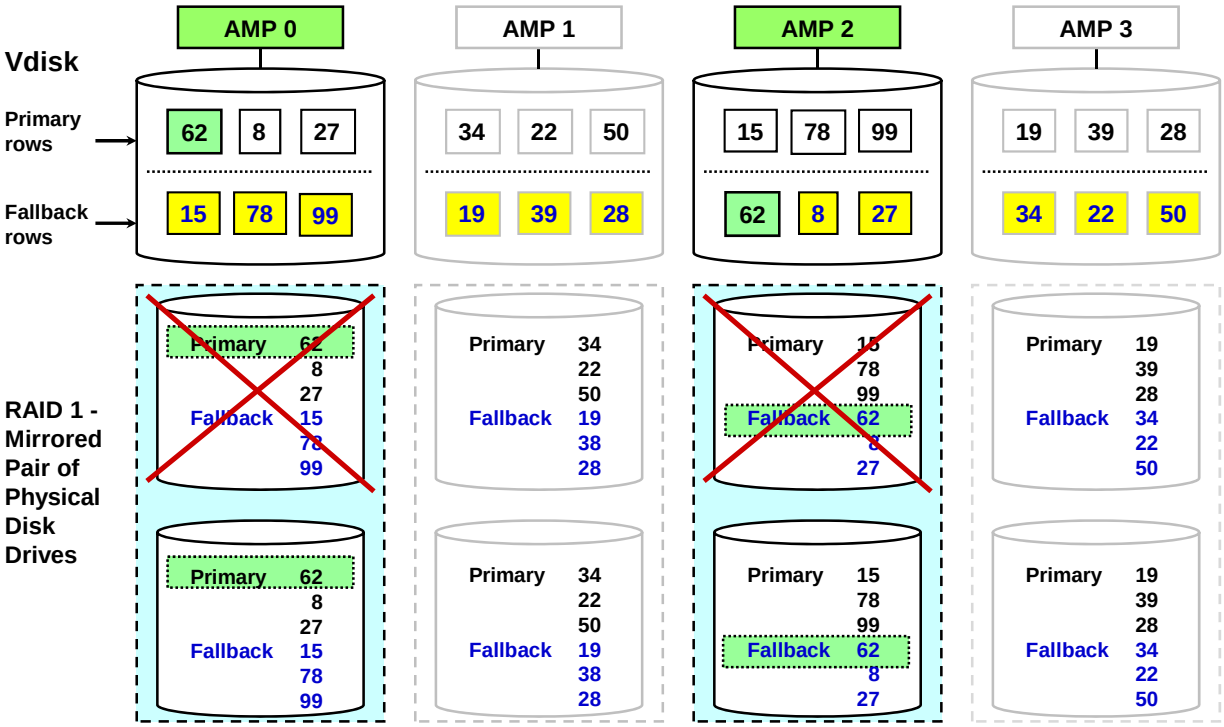
Fallback and RAID 1 Example (cont.)

The example of how Fallback and RAID 1 Mirroring work together is continued.

In this example, assume two disk drives have failed – one in the first drive group and one in the third drive group. Is Fallback needed? No. Like before, Teradata doesn't even realize that the drives have failed. The disk array continues to provide access to the data directly from the second disk drive each of the drive groups. The disk array controller will send "fault" or error messages to the AWS.

Fallback and RAID 1 Example (cont.)

Assume two disk drives have failed. Is Fallback needed in this example?



Fallback and RAID 1 Example (cont.)

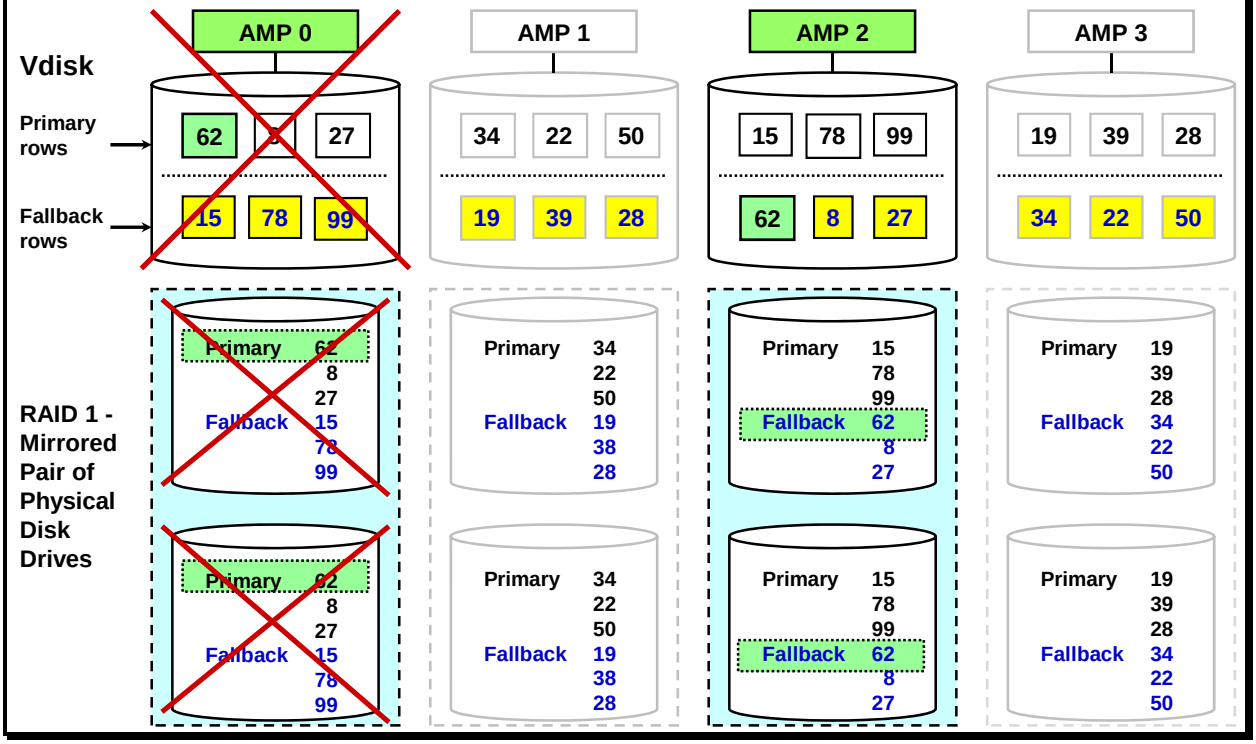
The example of how Fallback and RAID 1 Mirroring work together is continued.

In this example, assume two disk drives have failed – both failed drives are in the first drive group. Is Fallback needed? Yes, if you need to access the data in this table. When multiple disk drives fail in a drive group, the data (Vdisk) is not available and the AMP goes into a FATAL state. At this point, Teradata does realize that an AMP is not available and Teradata restarts. The disk array controller will send "fault" or error messages to the AWS.

The AWS will also get "fault" messages indicating that Teradata has restarted.

Fallback and RAID 1 Example (cont.)

Assume two disk drives have failed in the same drive group. Is Fallback needed?



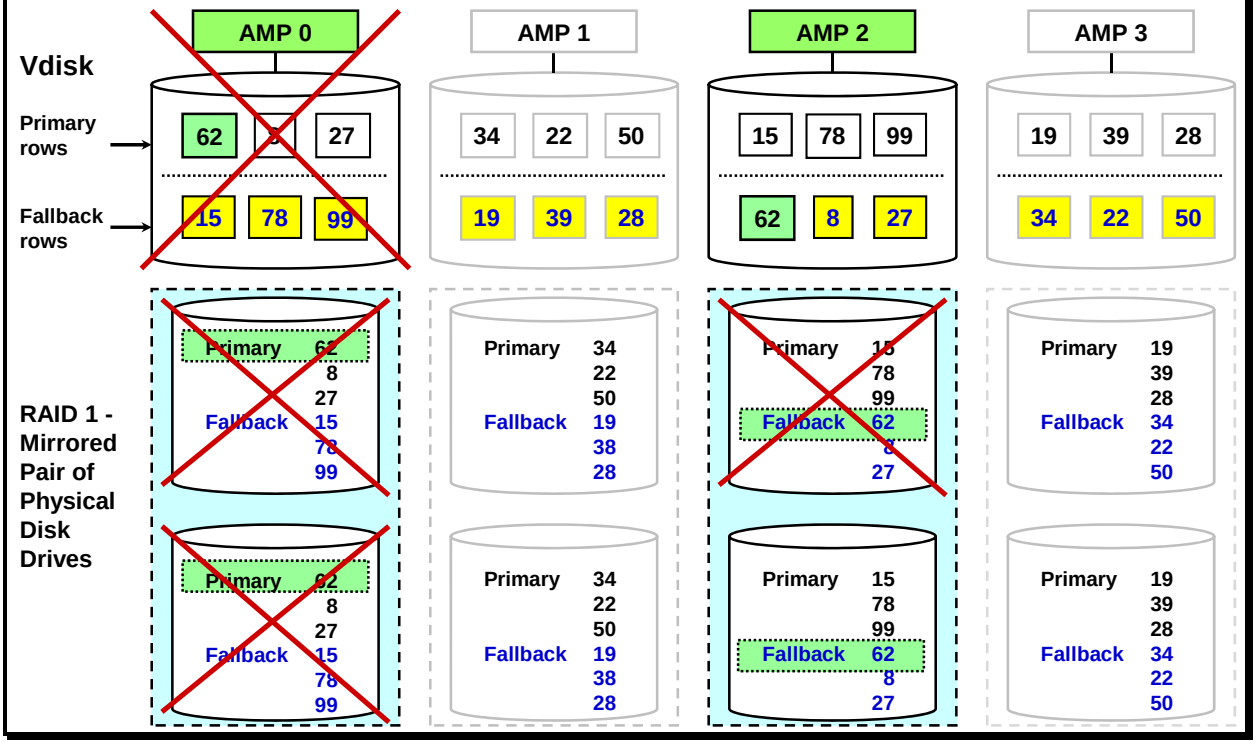
Fallback and RAID 1 Example (cont.)

The example of how Fallback and RAID 1 Mirroring work together is continued.

In this example, assume three disk drives have failed – two failed drives are in the first drive group and one failed drive is in the third drive group. Is Fallback needed? Yes, if you need to access the data in this table. When multiple disk drives fail in a drive group, the data (Vdisk) is not available and the AMP goes into a FATAL state. However, the third AMP is still operational and online.

Fallback and RAID 1 Example (cont.)

Assume three disk drive failures. Is Fallback needed? Is the data still available?



Fallback vs. non-Fallback Tables Summary

Fallback tables have a major advantage in terms of availability and recoverability. They can withstand an AMP failure in each cluster and maintain full data availability. A second AMP failure in any cluster results in a system halt. A manual restart of the system is required in this circumstance.

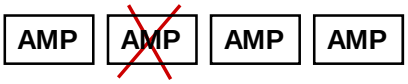
Non-Fallback tables are affected by the loss of any one AMP. The table continues to be accessible, but only for those AMPs that are still on-line. A one-AMP Primary Index access is possible, but a full table scan is not.

Fallback tables are easily recovered after a failure due to the availability of Fallback rows. **Non-Fallback tables** may only be restored from external medium in the event of a disaster.

Fallback vs. non-Fallback Tables Summary

FALLBACK TABLES

One AMP Down



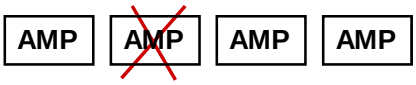
- Data fully available

Two or more AMPs Down

- If different cluster, data fully available
- If same cluster, Teradata halts

Non-FALLBACK TABLES

One AMP Down



- Data partially available; queries that avoid down AMP succeed.

Two or more AMPs Down

- If different cluster, data partially available; queries that avoid down AMP succeed.
- If same cluster, Teradata halts

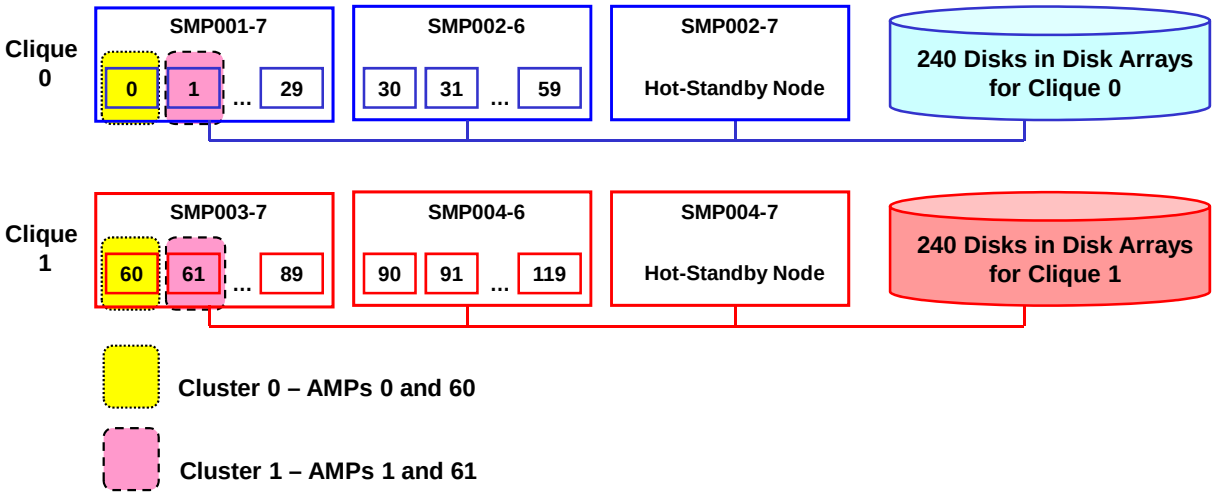
Clusters and Cliques

As you know, a **cluster** is a group of AMPs that act as a single fallback unit. A **clique** is a set of Teradata nodes that share a common set of disk arrays. Clusters provide data access protection in the event of an AMP failure (usually because of a Vdisk failure). Cliques provide protection from SMP node failures.

The best availability for Teradata is to spread clusters across different cliques. The "Default Cluster" function of the CONFIG utility does this automatically.

The example on the facing page illustrates a 4+2 node system. Each clique consists of 3 nodes (2 TPA plus one Hot Standby Node – HSN) connected to a set of disk arrays with 240 disks. This example assumes each node is configured with 30 AMPs.

Clusters and Cliques



To provide the highest availability, the goal is to interleave clusters across cliques and cabinets.

Locks

Locking prevents multiple users who are trying to change the same data at the same time from violating the data's integrity. This concurrency control is implemented by **locking** the desired data. Locks are automatically acquired during the processing of a request and released at the termination of the request. In addition, users can specify locks.

There are four types of locks: Exclusive, Write, Read, and Access.

Exclusive locks are only applied to databases or tables, never to rows. They are the most restrictive type of lock; all other users are locked out. Exclusive locks are used rarely, most often when structural changes are being made to the database.

Write locks enable users to modify data while locking out all other users except readers not concerned about data consistency (Access lock readers). Until a Write lock is released, no new read or write locks are allowed.

Read locks are used to ensure consistency during read operations. Several users may hold concurrent read locks on the same data, during which no modification of the data is permitted.

Access locks can be specified by users who are not concerned about data consistency. The use of an access lock allows for reading data while modifications are in process. Access locks are designed for decision support on large tables that are updated only by small single-row changes. Access locks are sometimes called “stale read” locks, i.e. you may get ‘stale data’ that hasn’t been updated.

Three levels of database locking are provided:

- **Database** - locks all objects in the database
- **Table** - locks all rows in the table or view
- **Row Hash** - locks all rows with the same row hash

The type and level of locks are automatically chosen based on the type of SQL command issued. The user has, in some cases, the ability to upgrade or downgrade the lock.

For example, if an SQL UPDATE command is executed without a WHERE clause, a WRITE lock is placed on the table. If an SQL UPDATE command is executed **with** a WHERE clause that specifies a Primary Index value, then a row hash lock is used.

Locks

There are four types of locks:

Exclusive	– prevents any other type of concurrent access
Write	– prevents other reads, writes, exclusives
Read	– prevents writes and exclusives
Access	– prevents exclusive only

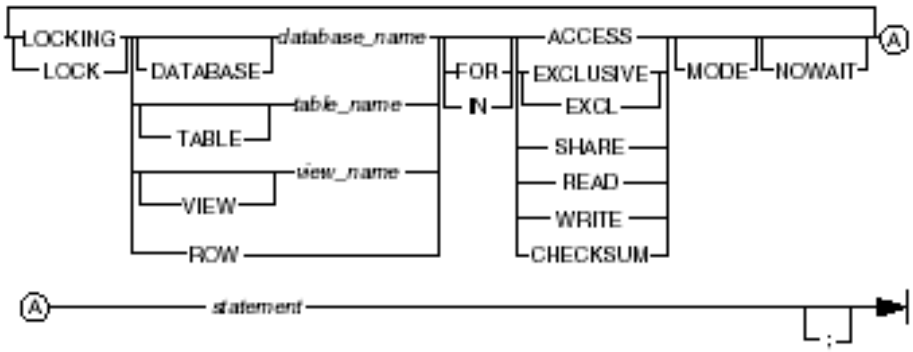
Locks may be applied at three levels:

Database	– applies to all tables/views in the database
Table/View	– applies to all rows in the table/views
Row Hash	– applies to all rows with same row hash

Lock types are automatically applied based on the SQL command:

SELECT	– applies a Read lock
UPDATE	– applies a Write lock
CREATE TABLE	– applies an Exclusive lock

Locking Modifier



This option precedes an SQL statement and locks a database, table, view, or row hash. The locking modifier overrides the default usage lock that Teradata places on a database, table, view, or row hash in response to a request.

Note: The DROP TABLE access right is required on the table in order to upgrade a READ or WRITE LOCK to an EXCLUSIVE LOCK.

ACCESS

Access locks have many advantages. This allows quick access to data, even if other requests are updating the data. They also have minimal effect on locking out others – when you use an access lock; virtually all requests are compatible with your lock except exclusive locks

NOWAIT

If a resource is locked and an application does not want to wait for that lock to be released, the Locking Modifier NOWAIT option can be used. The NOWAIT option indicates that if the lock cannot be obtained, then the statement will be aborted.

This option is used in situations where it is not desirable to have a statement wait for resources, possibly also tying up resources in the process of waiting.

Example:

LOCKING TABLE tablename FOR WRITE NOWAIT UPDATE ….. ;

***** Failure 7423 Object already locked and NOWAIT.**
Transaction Aborted. Statement# 1, Info =0

The user is informed with a 7423 error status code that indicates the lock could not be placed due to an existing, conflicting lock.

Locking Modifier

The locking modifier overrides the default usage lock that Teradata places on a database, table, view, or row hash in response to a request.

Certain locks can be upgraded or downgraded:

LOCKING ROW FOR ACCESS **SELECT * FROM Table_A;**

An "Access Lock" allows the user to access (read) an object that has a READ or WRITE lock associated with it.

In this example, even though an access row lock was requested, a table level access lock will be issued because the SELECT causes a full table scan.

Note: A "Locking Row" request must be followed by a SELECT.

LOCKING TABLE Table_B FOR EXCLUSIVE **UPDATE Table_B SET A = 2011;**

This request asks for an exclusive lock, effectively upgrading the lock.

LOCKING TABLE Table_C FOR WRITE NOWAIT **UPDATE Table_C SET A = 2012;**

The NOWAIT option is used if you do not want your transaction to wait in a queue.

NOWAIT effectively says to abort the the transaction if the locking manager cannot immediately place the necessary lock. Error code 7423 is returned if the lock request is not granted.

Rules of Locking

As the facing page illustrates, a new lock request must wait (queue) behind other incompatible locks that are either in queue or in effect. The new Read lock must wait until the write lock ahead of it is released before it goes into effect.

In the second example, the second Read lock request may occupy the same position in the queue as the Read lock that was already there. When the current Write lock is released, both requests may be given access concurrently. This only happens when locks are compatible.

When an SQL statement provides row hash information, a row hash lock will be used. If multiple row hashes within the table are affected, a table lock is used.

Rules of Locking

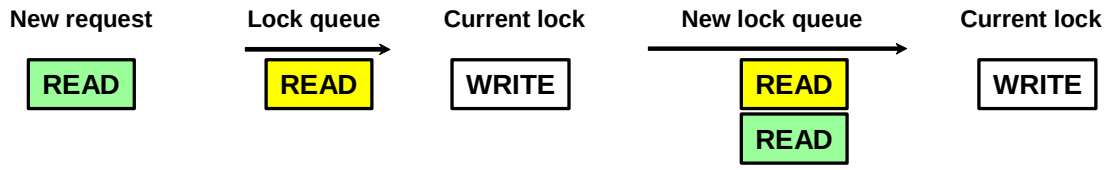
Rule

Lock requests are queued behind all outstanding incompatible lock requests for the same object.

LOCK REQUEST	LOCK LEVEL HELD				
	NONE	ACCESS	READ	WRITE	EXCLUSIVE
ACCESS	Granted	Granted	Granted	Granted	*Queued*
READ	Granted	Granted	Granted	*Queued*	*Queued*
WRITE	Granted	Granted	*Queued*	*Queued*	*Queued*
EXCLUSIVE	Granted	*Queued*	*Queued*	*Queued*	*Queued*

Example 1 – New READ lock request goes to the end of queue.

Example 2 – New READ lock request shares slot in the queue.



Access Locks

Access locks have many advantages. They allow quick access to data, even if other requests are updating the data. They also have minimal effect on locking out others - when you use an access lock; virtually all requests are compatible with yours.

When doing large aggregations of numbers, it may be inconsequential if certain rows are being updated during the summation, particularly if one is only looking for approximate totals. Access locks are ideal for this situation.

Looking at Example 3, what happens to the Write lock request when the Read lock goes away? Looking at the chart, it will be “Granted” since Write and Access are considered compatible.

Another example not shown on the facing page:

> Assume user1 is in ANSI mode and has updated a row, but hasn't entered COMMIT yet. The actual row in the table is updated on disk; the before-image is located in the TJ of the WAL log in case user1 decides to ROLLBACK.
>
> If user2 accesses this row with an access lock, the updated row on disk is returned - even though it is locked and not committed yet. Assume user1 issues a ROLLBACK, then the before-image in the TJ is used to rollback the row on disk. If user2 selects the row a second time, user2 will get the row (original) that is now on disk.

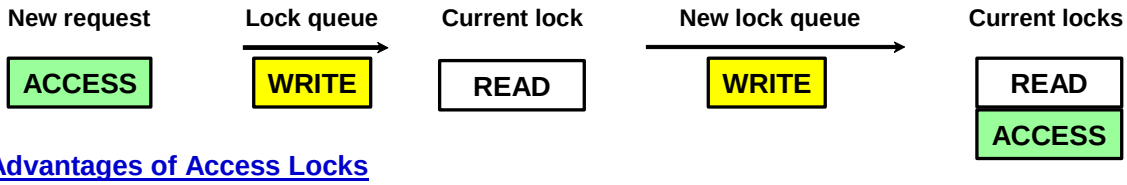
Access Locks

Rule

Lock requests are queued behind all outstanding incompatible lock requests for the same object.

LOCK REQUEST	LOCK LEVEL HELD				
	NONE	ACCESS	READ	WRITE	EXCLUSIVE
ACCESS	Granted	Granted	Granted	Granted	*Queued*
READ	Granted	Granted	Granted	*Queued*	*Queued*
WRITE	Granted	Granted	*Queued*	*Queued*	*Queued*
EXCLUSIVE	Granted	*Queued*	*Queued*	*Queued*	*Queued*

Example 3 – New ACCESS lock request granted immediately.



Advantages of Access Locks

- Permit quicker access to table in multi-user environment.
- Have minimal 'blocking' effect on other queries.
- Very useful for aggregating large numbers of rows.

Disadvantages of Access Locks

- May produce erroneous results if during table maintenance.

Transient Journal

The **Transient Journal** exists to permit the successful rollback of a failed transaction. Transactions are not committed to the database until an End Transaction request has been received by the AMPs, either implicitly or explicitly. Until that time, there is always the possibility that the transaction may fail in which case the participating table(s) must be restored to their pre-transaction state.

The **Transient Journal** maintains a copy of all before images of all rows affected by the transaction. If the event of transaction failure, the before images are reapplied to the affected tables, the images are deleted from the journal and a rollback operation is completed. In the event of transaction success, at the point of transaction commit, the before images for the transaction are discarded from the journal.

In Summary, if a Transaction fails (for whatever reason), the before images in the transient journal are used to return the data (in the tables involved in the transaction) to its original state.

Transient Journal

Transient Journal – provides transaction integrity

- A journal of transaction "before images" (UNDO rows) maintained within WAL.
- Provides for automatic rollback in the event of TXN failure.
- Is automatic and transparent.
- "Before images" are reapplied to table if a transaction fails.
- "Before images" are discarded upon transaction completion.

Successful TXN

BEGIN TRANSACTION	
UPDATE Row A	– Before image Row A recorded *(Add $100 to checking)*
UPDATE Row B	– Before image Row B recorded *(Subtract $100 from savings)*
END TRANSACTION	– Discard before images

Failed TXN

BEGIN TRANSACTION	
UPDATE Row A	– Before image Row A recorded
UPDATE Row B	– Before image Row B recorded
(Failure occurs)	
(Rollback occurs)	– Reapply before images
(Terminate TXN)	– Discard before images

Recovery Journal for Down AMPs

After the loss of any AMP, a **Down-AMP Recovery Journal** is started automatically. Its purpose is to log any changes to rows which reside on the down AMP. Any inserts, updates, or deletes affecting rows on the down AMP, are applied to the Fallback copy within the cluster. The AMP that holds the Fallback copy logs the Row ID in its Recovery Journal.

This process continues until such time as the down AMP is brought back on-line. As part of restart activity, the Recovery Journal is read and changed rows are applied to the recovered AMP. When the journal has been exhausted, it is discarded and those tables that are fallback-protected are fully recovered.

Recovery Journal for Down AMPs

Recovery Journal is:

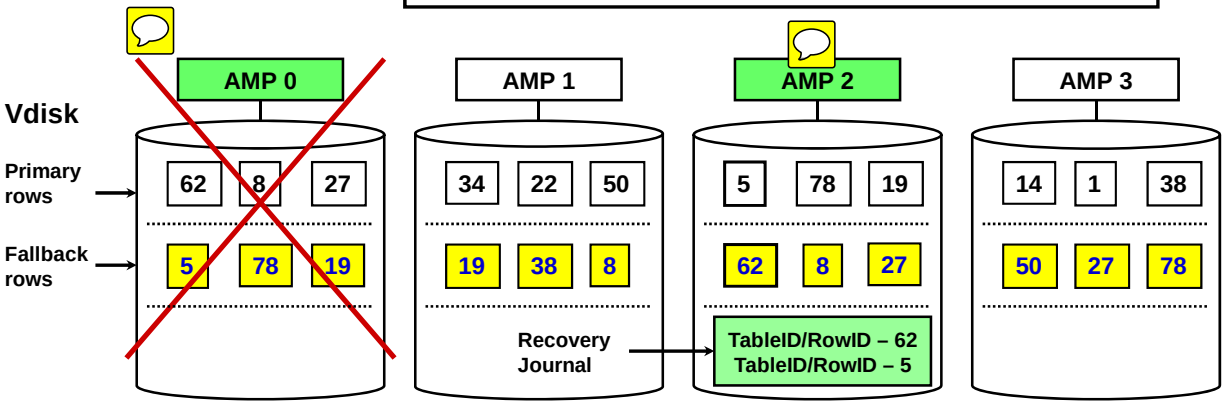
- Automatically activated when an AMP is taken off-line.
- Maintained by the other AMPs in a cluster.
- Totally transparent to users of the system.

While AMP is off-line

- Journal is active.
- Table updates continue as normal.
- Journal logs Row IDs of changed rows for down-AMP.

When AMP is back on-line

- Restores rows on recovered AMP to current status.
- Journal discarded when recovery complete.



Permanent Journal

The purpose of the **Permanent Journal** is to provide selective or full database recovery to a specified point in time. It permits recovery from unexpected hardware or software disasters. The Permanent Journal also has the effect of reducing the need for full table backups which can be costly both in time and resources.

The Permanent Journal is an optional journal and its features must be customized to the specific needs of the installation. The journal may capture before images (for rollback), after images (for rollforward), or both. Additionally, the user must specify if single images (default) or dual images (for fault-tolerance) are to be captured.

A Permanent Journal may be shared by multiple tables or multiple databases.
The journal captures images concurrently with standard table maintenance and query activity. The cost in additional required disk space may be calculated in advance to ensure adequate disk reserve.

The journal is periodically dumped to external media, thus reducing the need for full table backups – in effect, only the changes are backed up.

Permanent Journal

The Permanent Journal is an optional, user-specified, system-maintained journal which is used for recovery of a database to a specified point in time.

The Permanent Journal:

- **Is used for recovery from unexpected hardware or software disasters.**
- **May be specified for ...**
  - **One or more tables**
  - **One or more databases**
- **Permits capture of Before Images for database rollback.**
- **Permits capture of After Images for database rollforward.**
- **Permits archiving change images during table maintenance.**
- **Reduces need for full table backups.**
- **Provides a means of recovering NO FALLBACK tables.**
- **Requires additional disk space for change images.**
- **Requires user intervention for archive and recovery activity.**

Archiving and Recovering Data

The purpose of the **ARC utility** is to allow for the archiving and restoring of database objects which may have been damaged or lost. There are several scenarios where restoring objects from external media may be necessary.

- Restoring of non-Fallback tables after a disk failure.
- Restoring of tables which have been corrupted by batch processes which may have left the data in an 'uncertain' state.
- Restoring of tables, views or macros which have been accidentally dropped by the user.
- Miscellaneous user errors resulting in damaged or lost database objects.

Teradata's Backup and Recovery (BAR) architecture provides solutions from Teradata Partners. Two examples are:

- NetVault – from BakBone software
- NetBackup – from Symantec (Veritas NetBackup by Symantec)

The **ASF2 utility** is an older utility that provides an X Windows based front-end for creation and execution of ARC command scripts. It is designed to run on UNIX MP-RAS.

Archiving and Recovering Data

ARC

- The Archive/Restore utility (arcmain)
- Runs on IBM, UNIX MP-RAS, Windows 2003, and Linux systems
- Archives and restores data from/to Teradata Database
- Restores or copies data from archive media
- Permits data recovery to a specified checkpoint using Permanent Journals

Backup and Recovery (BAR)

- Example of BAR choices from different Teradata Partners
  - NetBackup - Veritas NetBackup by Symantec
  - Tivoli Storage Manager – utilizes TARA
- Provides Windows front end for ARC
- Easy creation of scripts for archive/recovery
- Provides job scheduling and tape management functions
- BAR was previously referred to as Open Teradata Backup (OTB)

Module 8: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 8: Review Questions

Match the item to a lettered description.

_____ 1.	Database locks	a. Provides for TXN rollback in case of failure
_____ 2.	Table locks	b. Teradata Backup and Recovery applications
_____ 3.	Row Hash locks	c. Protects all rows of a table
_____ 4.	FALLBACK	d. Logs changed rows for down AMP
_____ 5.	Cluster	e. Provides for recovery to a point in time
_____ 6.	Recovery journal	f. Applies to all tables and views within
_____ 7.	Transient journal	g. Multi-platform archive utility
_____ 8.	ARC	h. Lowest level of protection granularity
_____ 9.	NetBackup/Tivoli	i. Protects tables from AMP failure
_____ 10.	Permanent journal	j. Protects database from a physical drive failure
_____ 11.	Disk Array	k. Group of AMPs used by Fallback

Notes

Module 9

Introduction to MPP Systems

After completing this module, you will be able to:

- **Specify a major difference between a 6650 and a 6690 system.**
- **Specify a major difference between a 2650 and a 2690 system.**
- **Define the purpose of the major subsystems that are part of an MPP system.**
- **Specify the names of the Teradata (TPA) nodes in a 6690 cabinet.**

Notes

Table of Contents

Teradata Systems 9-4
SMP Architecture 9-6
Hyper-Threading and Multi-Core CPUs 9-8
Comparing Performance of Servers 9-10
Cabinet or Rack Pictures 9-12
Teradata 6650 Systems 9-14
Teradata 6650 Cabinets 9-16
Adding SSD to a 6650 (Future) 9-18
Teradata 6650 Configuration Examples 9-20
Teradata 6690 Systems 9-22
Teradata 6690 Cabinets 9-24
Teradata Extended Nodes 9-26
Making Sense of the Different Platforms 9-28
Linux Coexistence Combinations 9-30
Teradata Appliance Introduction 9-32
Teradata 2650/2690 Appliances 9-34
Teradata 2650/2690 Cabinets 9-36
Appliance Configuration Examples 9-38
What is the BYNET™? 9-40
BYNET 32 Switches 9-42
BYNET 64 Switches 9-44
BYNET Expansion Switches 9-46
BYNET Expansion to 1024 Nodes 9-46
Server Management with SWS 9-48
Node Naming Conventions 9-50
Summary 9-52
Module 9: Review Questions 9-54

Teradata Systems

As the competitive needs of businesses change, the system architecture changes over time. To be best-in-class, an information processing system in today's environment will typically have the following characteristics.

- Utilization of multiple processors in multiple nodes to achieve acceptable performance. Easily scalable in both processing power and data storage capacity with adherence to all industry-standard interfaces.

- Be capable of handling a very large databases, rapidly process complex queries, maintain data security, and be accessible to the total enterprise. Support on-line transaction processing as well as decision support applications.

- In today's global and highly competitive markets, computing systems (especially enterprise servers) need to be available to the world 24 hours a day.

TPerf (**T**raditional **Perf**ormance) is a power metric that has been used in a rigorous and consistent manner for each generation of the Teradata platform since the model 5100. It is a metric for how fast a node can process data. TPerf is maximized when there is a balance between CPU and I/O bandwidth. When used to compare different Teradata configurations, the TPerf metric is similar to other throughput metrics, such as rows/second or transactions/second that a node processes where actual data volumes in terms of bytes are not reflected in the metric. Data capacity is not a function of a general or design center TPerf used by sales and engineering to compare Teradata systems, that is, this metric assumes there is a constant database volume in place when comparing one system to another.

TPerf is a power metric that measures the throughput performance of the TPerf workload. It is not a response time metric for specific queries and operations. Response time depends on a number of factors in the Teradata architecture in addition to the ones that TPerf gauges, i.e., CPU power and I/O performance. Other factors influencing response time include, but are not limited to:

1. Parallelism provided by the number of AMPs
2. Concurrency (competition among queries)
3. Workload mix
4. Workload management

TPerf is analogous to the pulling Power of a train locomotive. The "Load" is the work the Node operates on. The data space is analogous to the freight cars in a train. You would need twice as big a locomotive to pull twice as many cars. You would need a

To have the same performance with twice as much data and load on a system, you would need a system with a TPerf that is twice (2x) as large.

Teradata Systems

Examples of systems used with the Teradata Database include:

Teradata Systems

5400/5450	– up to 10 nodes/cabinet
5500/555x/56xx	– up to 9 nodes/cabinet
6650/6680/6690	– up to 4 nodes/cabinet with associated storage
15xx/16xx/25xx/26xx	– various Appliance systems

The basic building block is the SMP (Symmetric Multi-Processing) node.

The power of these nodes will be measured by TPerf – Traditional Performance.

- **The Teradata metric for total power of a node or system.**
- **Determined by measuring system elements and calculating the performance with a representative set of workloads.**

Key differences:

- **Speed and capacity of SMP nodes and systems**
- **Cabinet architecture**
- **BYNET interface cards, switches and speeds** ***BYNET V4 – up to 4096 nodes**

SMP Architecture

The SMP or "processing node" is the basic building block for Teradata systems. The processing node contains the primary processor logic (CPUs), memory, and I/O functionality.

Teradata is supported on non-Teradata SMP servers with 4 or fewer physical CPU sockets. A Teradata license can only be purchased for an SMP server with up to 4 physical CPUs. The server might have 4 hyper-threading CPUs which look like 8 logical CPUs to the operating system. The server may have two quad-core CPUs which appears to the operating system as 8 CPUs.

Basic definitions of the CPUs used with Teradata servers:

- Hyper-Threading CPUs – one physical CPU (chip) socket, but with 2 control (context) areas – makes 1 CPU look like 2 logical CPUs.
- Dual-core CPUs – one physical CPU (chip) socket, but with two control (context) areas and 2 execution cores – makes 1 CPU look like 2 physical CPUs.
- Quad-core CPUs – one physical CPU (chip) socket, but with four control (context) areas and 4 execution cores – makes 1 CPU look like 4 physical CPUs.
- Quad-core CPUs with Hyper-Threading – one physical CPU (chip) socket, but with 8 control (context) areas and 4 execution cores – makes 1 CPU look like 4 physical CPUs or 9 logical CPUs.
- Six-core CPUs with Hyper-Threading – one physical CPU (chip) socket, but with 12 control (context) areas and 6 execution cores – makes 1 CPU look like 6 physical CPUs or 12 logical CPUs.

5400/5450 nodes have 2 physical chips using Hyper-Threading, effectively 4 logical CPUs.

5500H nodes have 2 dual-core chips, effectively 4 CPUs.

5555C nodes have 1 quad-core chips, effectively 4 CPUs

5550H and 5555H nodes have 2 quad-core chips, effectively 8 CPUs.

5600H nodes have 2 quad-core chips using hyper-threading, effectively 16 CPUs per node.

2650, 2690, 5650H, 6650H, 6680, and 6690 nodes have 2 six-core chips using hyper-threading, effectively 24 CPUs per node.

SMP Architecture

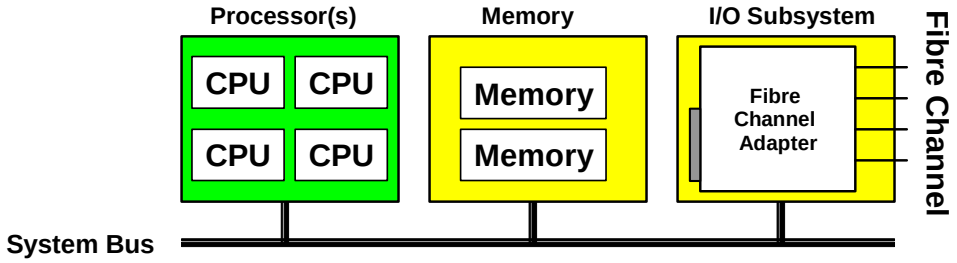
SMP (Symmetrical Multi-Processing) Node – basic building block of MPP systems.

- **Hyper-Threading CPUs – one CPU socket (chip) with 1 execution core and 2 control (context) areas – makes 1 CPU chip look like 2 logical CPUs.**
- **Dual-core CPUs – one CPU socket with 2 execution cores – makes 1 chip look like 2 physical CPUs.**
- **Quad-core CPUs – one CPU socket with 4 execution cores – makes 1 chip look like 4 physical CPUs.**
- **Quad-core CPUs with Hyper-Threading – one chip socket with 4 execution cores each with 2 control areas – makes 1 CPU chip socket look like 8 logical CPUs**
- **Six-core CPUs with Hyper-Threading – one chip socket with 6 execution cores each with 2 control areas – makes 1 CPU chip socket look like 12 logical CPUs**

Other names include node, compute node, processing node, 24-way node, etc.

Key hardware components of a node include:

- **CPUs and cache memory**
- **Memory**
- **System Bus**
- **I/O Subsystem**



Hyper-Threading and Multi-Core CPUs

The facing page illustrates the concept of Hyper-Threading and Multi-Core CPUs.

With Hyper-Threading, 2 physical CPUs appear to the Operating System as 4 logical or virtual CPUs. With Dual-Core, 2 physical CPUs appear to the Operating System as 4 physical CPUs. The SMP's BIOS automatically tells the Operating System that there are 4 CPUs. The Operating System will schedule work as though there are actually 4 CPUs in either case.

The reason for a performance gain with Hyper-Threading is as follows. When one of the logical processors (control unit) is setting up its data and instruction registers from cache or memory, the execution unit can be executing instructions from the other logical processor. In this way, the execution unit doesn't have to wait for one of the control units to set up its data and instruction registers – it is effectively kept busy a larger percentage of the time.

Some of the benefits of Hyper-Threading include:

- No software changes required
- Symmetric
- Improved CPU Efficiency

The reason for a performance gain with Dual-Core CPUs is that there are two control areas and two execution units. One CPU socket is really two physical CPUs. Quad-Core CPUs provide even more processing power with one CPU socket providing four physical CPUs.

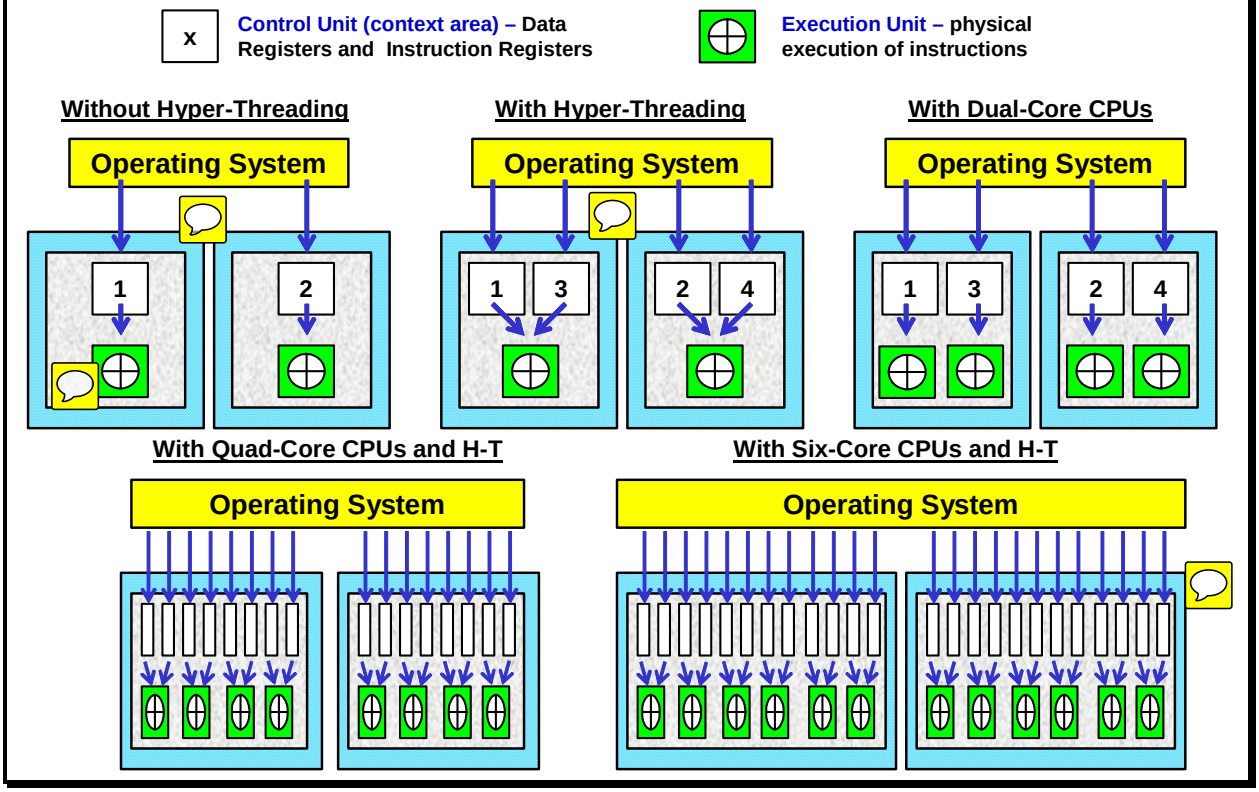
With Quad-Core, 2 physical CPUs appear to the Operating System as 8 physical CPUs. The SMP's BIOS effectively tells the Operating System that there are 8 CPUs.

With Quad-Core and Hyper-Threading, 2 physical CPUs appear to the Operating System as 16 CPUs. The SMP's BIOS effectively tells the Operating System that there are 16 CPUs.

Notes:

- The Operating System schedules work across logical or physical CPUs.
- The Windows Task Manager or UNIX "pinfo" command actually identifies the CPUs (e.g., 8 with quad-core) for which work can be scheduled.

Hyper-Threading and Multi-Core CPUs



Comparing Performance of Servers

TPerf is a metric for total Power of a Node or system

- TPerf = Traditional Performance
- Analogous to the pulling Power of a train locomotive.

The "Load" is the work the Node operates on. The data space is analogous to the freight cars in a train. You would need twice as big a locomotive to pull twice as many cars.

To have the same performance with twice as much data and load on a system, you would need a system with a TPerf that is twice (2x) as large.

Acronym: H-T is Hyper-Threading

Teradata's Design Center establishes typical system configurations for different Teradata system models. For example, one design center configuration for a 6650 system is cliques of 3+1 nodes, 42 AMPs per node, and two 600 GB mirrored disks for each node. The design center power rating is called TPerf-dc.

The process for deriving design center TPerf for a Teradata platform consists of five steps:

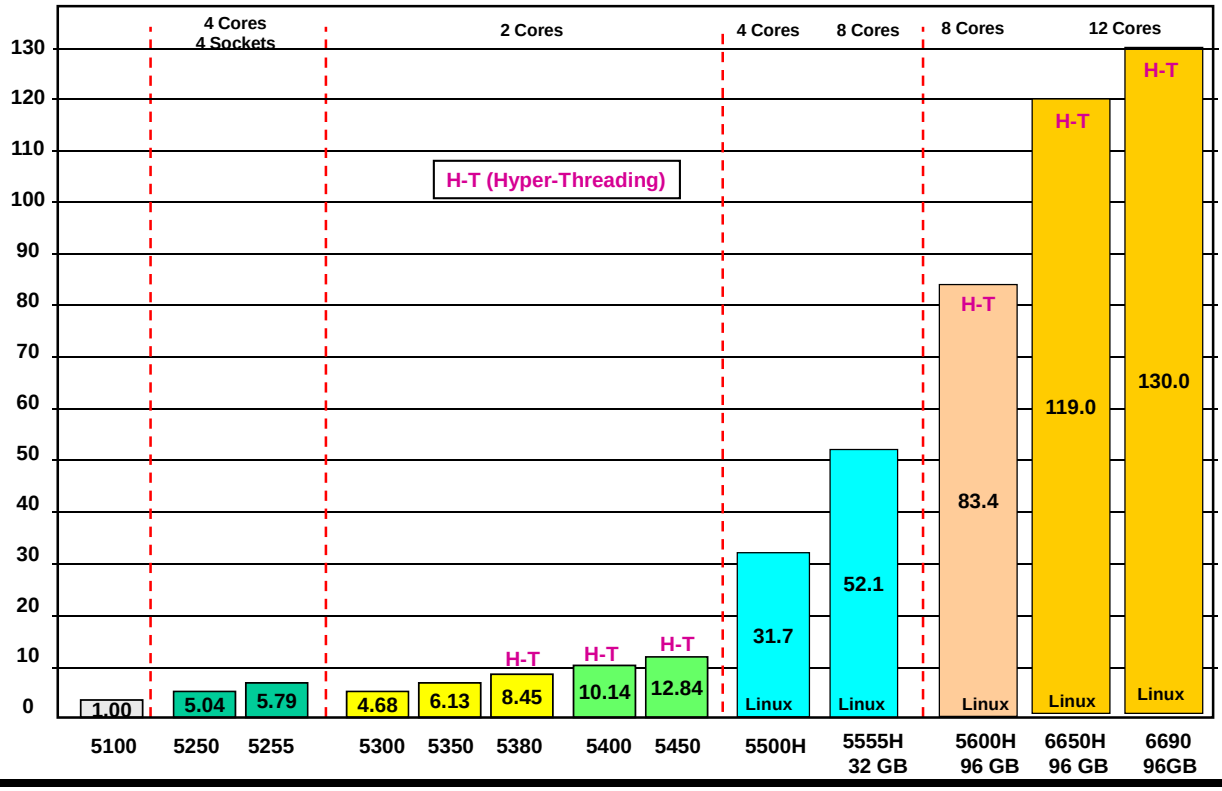
1) A diverse se of performance tests is executed on the platform design center configuration for a Teradata platform model.
2) The CPU and IO resource usage and throughput are measured.
3) An analytical model is used to calculate the CPU and IO resource usage of a weighted blend of workloads.
4) The blended workload is compared against the resource capabilities provided by the design center platform configuration.
5) The TPerf metric is then calculated.

This design center TPerf (TPerf-dc) represents system throughput potential, in other words, how much work could be done in a given amount of time given a high concurrency workload for that design center hardware configuration. Any system with the same configuration and the same workload mix used in the model will deliver overall performance that matches the level indicated by the TPerf-dc metric.

TPerf-dc usually does not describe the throughput potential for deployed configurations of Teradata systems. The reality is that business demands require a wide variety of Teradata system configurations to meet specific performance and pricing needs and no customer workload is the same as that for the TPerf-dc model.

TPerf-dc plays only a small part in any attempt to estimate response time expectations for the design center configuration and TPerf workload – all the other factors listed above must be considered.

Comparing Performance of Servers

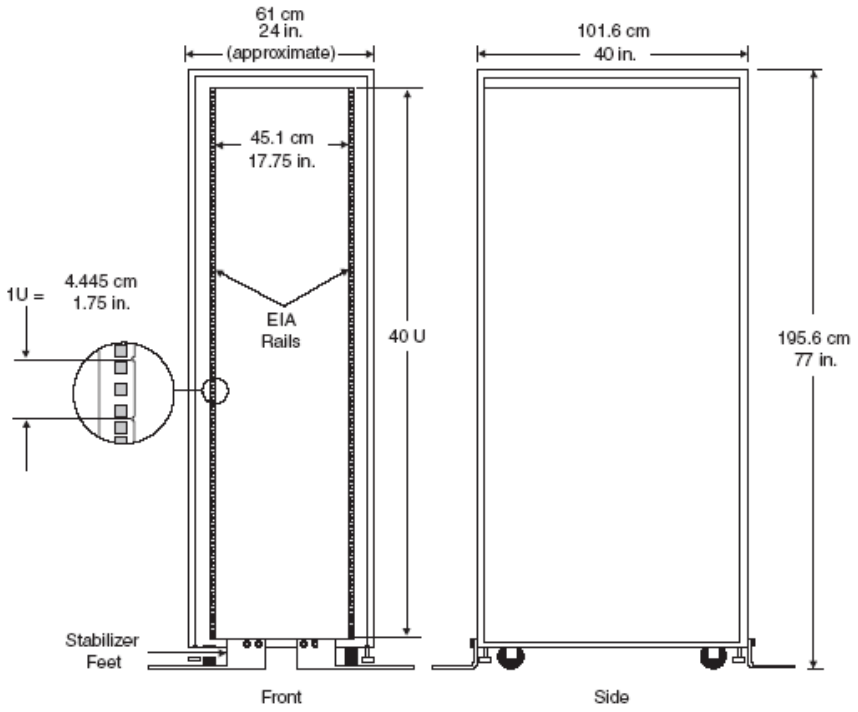


Cabinet or Rack Pictures

The Rack Cabinet is an industry standard, 40U rack frame used to house Teradata processing nodes and/or disk arrays.

Measurements

- The "U" in the 40U rack term represents a unit of vertical measurement for placement of chassis in the rack. [1U = 4.445 cm (1.75 in.)] This diagram illustrates the depth of older cabinet which was 40".



Teradata systems use an industry standard rack mount architecture and individual chassis that conform to industry standards. Each chassis occupies a specific number of U spaces in the rack. Examples of types of chassis that can be placed in a rack or cabinet include.

- Processing Node (54xx , 55xx, 56xx, and 66xx nodes – 2U
- BYNET Switch (BYA32S) – 1U
- Server Management Chassis (CMIC) – 1U

The 55xx and 66xx systems use a rack that is 44" deep (4" deeper than previous rack).

Older systems (e.g., 5650) used a separate Teradata SWS (Service Workstation) for operational maintenance of the system. The last SWS was a Dell PowerEdge T710 Server and was available as deskside or rack mount server.

Newer systems (e.g., 6690) utilize a VMS (Virtualized Management Server) which consolidates CMIC, SWS, and Teradata Viewpoint functions into a single chassis.

Cabinet or Rack Pictures

Node Chassis

Processor /Storage Cabinet

Notes:

- Cabinet Size = 24" W X 77"H X 44" D without doors and side panels
- Improved cable management
  - Larger exit hole in base
  - Supports inter-rack cabling

Teradata 6650 Systems

The Teradata Active Enterprise Data Warehouse 6650 platform is scalable from one to 4,096 Teradata nodes, and can handle more than 15 petabytes of data to support the complex workloads in an active warehouse environment.

The 6650 processing nodes are the newest release of Teradata Servers which supports the Teradata Warehouse solution. These nodes are similar to the 5650 processing nodes, utilizing the Intel Westmere™ six-core CPUs with hyper-threading enabled.

The Teradata Active Enterprise Data Warehouse platform is made up of a combination of cabinet types, depending on the system configuration:

- Processing/storage cabinet
- BYNET cabinet
- Teradata Managed Server (TMS) cabinet

The 6650 provides high availability via the following features:

- Hot standby nodes (HSN): One node in a clique can be configured as a hot standby node. Eliminates the degradation of database performance in the event of a node failure in the clique. Tasks assigned to the failed node are completely redirected to the hot standby node.

- Hot spare disks: One or more disks per array can be configured as hot spare disks. In the event of a disk failure on a RAID mirrored pair, the contents of the failed disk are copied into a hot spare disk from the mirrored surviving disk to repair the RAID pair. When the failed drive is replaced, a copy back operation occurs to restore data to the replaced drive.

- Fallback: Data protection can be provided at the table level by automatically storing a copy of each permanent data row of a table on a different or “fallback” AMP. If an AMP fails, the Teradata Database can access the fallback copy and continue operation.

The design center recommendations has a different number of AMPs and associated storage per AMP varies depending on the configuration.

- 1+1 clique – 48 AMPs/node; 192 disks per node
- 2+1 clique – 30 AMPs/node; 120 disks per node
- 3+1 clique – 42 AMP/node; 84 disks per node

Teradata 6650 Systems

Features of the 6650 system include:

- **The Teradata 6650 platform is the first release of unified Node/Storage within a single cabinet in the Active Enterprise Data Warehouse space.**
- **The 6650 is designed to reduce floor space utilization.**
  - **The UPS/batteries are not used with the 6650**
  - **In the event of site wide power loss, data integrity is provided by WAL.**
- **The 6650 utilizes up to two Intel® 2.93 GHz six-core CPUs**
  - **Two models – 6650C and 6650H**
    - **6650C nodes utilize 1 socket with one six-core CPU and 48 GB of memory**
    - **6650H nodes utilize 2 sockets with two six-core CPUs and 96 GB of memory**
    - **6650C can be used to co-exist with previous generations and 6650H will co-exist with future Active EDW Platform mixed storage offerings**
- **The 6650 can be configured in 1+1, 2+1, and 3+1 cliques.**
  - **A 6650 clique consists of either one or two processing/storage cabinets. Each cabinet contains processing nodes and a disk array.**
- **The 6650 can be upgraded to use SSD drives.**
  - **6650 is an *SSD Ready* platform and prepares for the introduction of Solid State Drives (SSD) in the Active EDW space.**

Teradata 6650 Cabinets

The facing page illustrates various 6650 cabinet configurations.

The 66xx and later systems utilize an industry standard rack mount cabinet which provide for excellent air flow and cooling. Similar to previous rack-based systems, this rack contains individual subsystem chassis that are housed in standard rack frames. Subsystems are self-contained, and their configurations — either internal or within a system — are redundant. The design ensures overall system reliability, enhances its serviceability, and enables time and cost efficient upgrades.

The key chassis in the rack/cabinet is the node chassis. The SMP node chassis is 2U in height.

A Hot Standby Node is required with 6650 systems.

- For 6650 systems, a clique has a maximum of three TPA nodes with one HSN node.

Cabinet Build Conventions

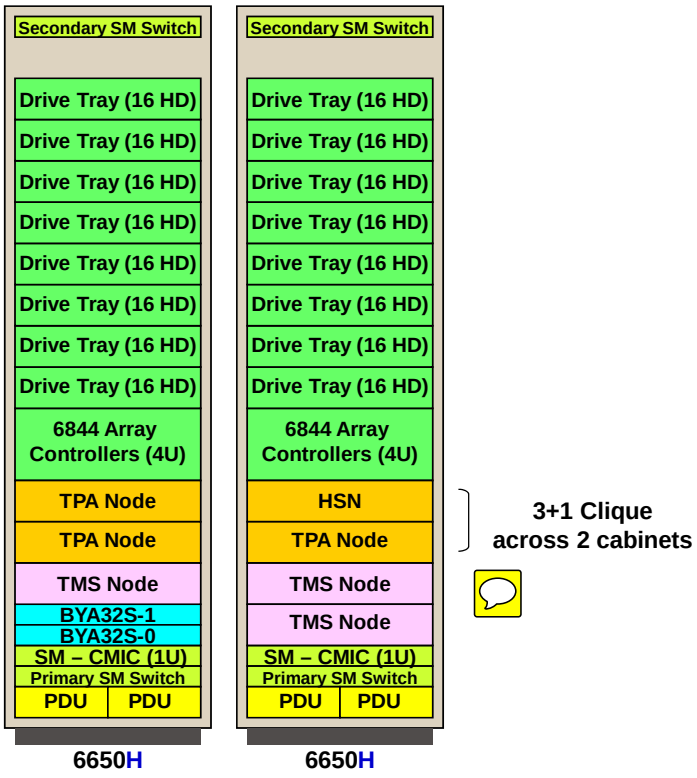
The placement of the hardware components in a cabinet follows these general cabinet build conventions:

- A 6650 clique consists of either one or two processing/storage cabinets. Each cabinet contains processing nodes and a disk array. The following clique configurations are available:
  - A two-cabinet 3+1 clique. The first cabinet contains two processing nodes and one disk array. The second cabinet contains one processing node, one hot standby node, and one disk array.
  - A two-cabinet 2+1 clique. The first cabinet contains one processing node and one disk array. The second cabinet contains one hot standby node, one processing node, and one disk array.
  - A two-cabinet 1+1 clique. The first cabinet contains one processing node and one disk array. The second cabinet contains one hot standby node and one disk array.
  - A one-cabinet 1+1 clique. The cabinet contains one processing node, one hot standby node, and one disk array.
- There is 1 CMIC in first cabinet of each two-cabinet clique. If a system only has one clique, then there is a CMIC in the second cabinet.

Teradata 6650 Cabinets

6650 Characteristics

- Integrated Cabinet with nodes and arrays in same cabinet.
- NetApp array with 2 controllers and 8 drive trays.
  - 300, 450, or 600 GB drives
- With 2+1 clique, each AMP is typically assigned to 4 disks (2 mirrored pairs).
  - Usually 30 AMPs/node
- With 3+1 clique, each AMP is typically assigned to 2 disks (1 mirrored pair).
  - Usually 42 AMPs/node
- No UPSs in cabinet.



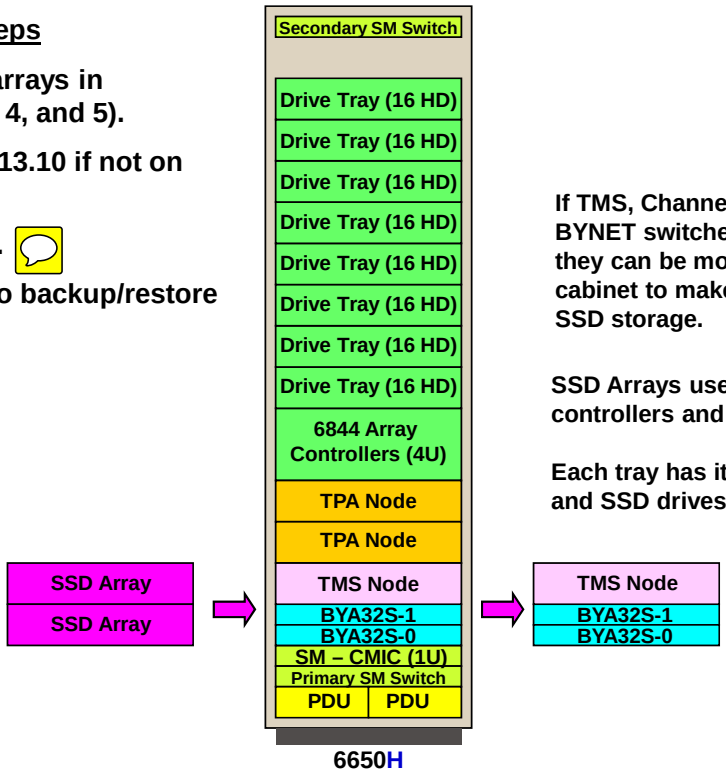
Adding SSD to a 6650 (Future)

The facing page illustrates a future option to add Solid State Disks (SSD) to a 6650 cabinet.

Adding SSD to a 6650 (Future)

SSD Upgrade Steps

- Place SSD arrays in positions 3, 4, and 5).
- Upgrade to 13.10 if not on 13.10.
- Enable TVS.
- Reconfig (no backup/restore required).



If TMS, Channel Servers and/or BYNET switches are installed, they can be moved to a another cabinet to make room for the SSD storage.

SSD Arrays use SAS based controllers and 400 GB SSD.

Each tray has its own controllers and SSD drives.

Teradata 6650 Configuration Examples

The facing page includes two examples of 6650 cliques. Typically, a 6650 node in a 3+1 clique will be configured with 42 AMPs, 2 disks per AMP, and 96 GB of memory.

Current configurations of the 6650 include:

6650H – Design Center

		Configuration			Allows upgrade to		
Clique	Drive Options	HDDs per Node	HDDs per Clique	CPU COD Available	SSD per Node	Disks per AMP	AMPs per Node
3+1 H	300GB	84	252	Yes	28	2	42
	450GB	84	252	Yes	28	2	42
	600GB	84	252	Yes	28	2	42
2+1 H	300GB	120	240	Yes	28	2	30
	450GB	120	240	Yes	28	2	30
	600GB	120	240	Yes	28	2	30
1+1 H	300GB	192	192	Yes	28	2	48
	450GB	192	192	Yes	28	2	48
	600GB	192	192	Yes	28	2	48

These configurations provide an effective future SSD upgrade path while maintaining optimum AMPs per node for a 6650H.

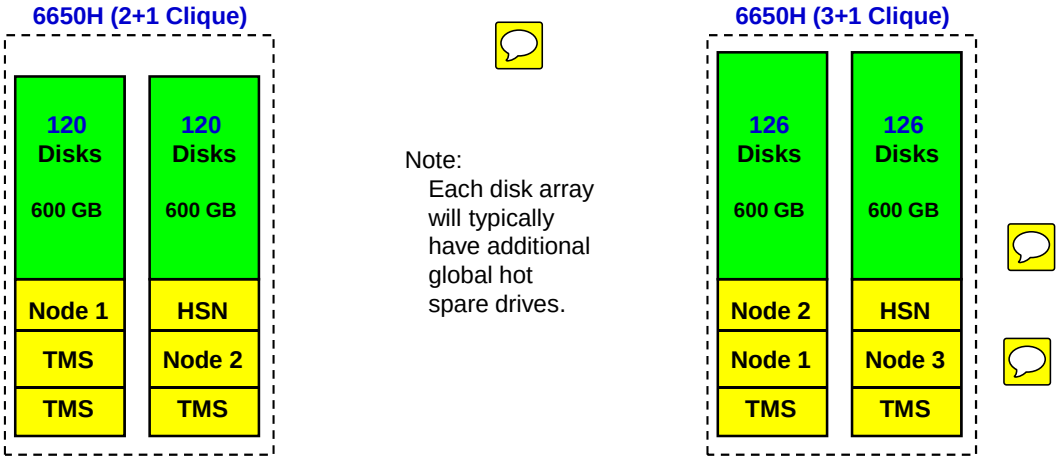
6650C – Design Center

		Configuration			Allows upgrade to		
Clique	Drive Options	HDDs per Node	HDDs per Clique	CPU COD Available	SSD per Node	Disks per AMP	AMPs per Node
3+1 C	300GB	42	126	Yes	14	2	21
	450GB	42	126	Yes	14	2	21
	600GB	42	126	Yes	14	2	21
2+1 C	300GB	60	120	Yes	14	2	15
	450GB	60	120	Yes	14	2	15
	600GB	60	120	Yes	14	2	15
1+1 C	300GB	96	96	Yes	14	2	24
	450GB	96	96	Yes	14	2	24
	600GB	96	96	Yes	14	2	24

These configurations provide an effective future SSD upgrade path while maintaining optimum AMPs per node for a 6650C.

For both 6650H and 6650C, if CPU Only Capacity on Demand is active, it should be removed to take full advantage of the increased I/O now available. Following the Optimum Performance Configurations will allow the customer to avoid a data reload and maintain their systems AMPs per node ratio thereby reducing the impact of an upgrade.

Teradata 6650 Configuration Examples



6650H (2+1 nodes/clique)

30 AMPs / Node
60 AMPs / Clique

120 Disks per Node
240 Disks per Clique

Each Vdisk – 4 Disks (RAID 1)
Each Vdisk – 1.08 TB*

Clique – 60 AMPs x 1.08 TB = 65 TB*

* Actual MaxPerm space is app. 90%.

6650H (3+1 nodes/clique)

42 AMPs / Node
126 AMPs / Clique

84 Disks per Node
252 Disks per Clique

Each Vdisk – 2 Disks (RAID 1)
Each Vdisk – 540 GB*

Clique – 126 AMPs x 540 GB = 68 TB*

Teradata 6690 Systems

The Teradata 6690 platforms utilize Solid State Drives (SSD) and Hard Disk Drives (HDD) within a single cabinet in the Active Enterprise Data Warehouse space.

- Requires Teradata Virtual Storage (TVS).
- SSD and HDD Storage is maintained within the same drive tray.

The Teradata Active Enterprise Data Warehouse platform is made up of a combination of cabinet types, depending on the system configuration:

- Processing/storage cabinet
- BYNET cabinet
- Teradata Managed Server (TMS) cabinet

Note: A Service Workstation (SWS) is installed in one TMS cabinet. A system may have additional TMS cabinets.

6690 nodes are based on the 6650 processing nodes. Characteristics include:

- Up to two Intel Westmere six-core CPU's
  - 12 MB L2 cache with Hyper-threading
  - Small performance increase over 5650; 6680H (126 TPerf)

- 450 GB OS drives support 96GB memory

- 300 GB dump drive for restart performance

Teradata 6690 Systems

Features of the 6690 system include:

- **The Teradata 6690 platforms utilize Solid State Drives (SSD) and Hard Disk Drives (HDD) within a single cabinet in the Active Enterprise Data Warehouse space.**
  - **Requires Teradata Virtual Storage (TVS).**
  - **SSD and HDD Storage is maintained within the same drive tray.**
- **The 6690 is designed to reduce floor space utilization (similar to 6650).**
  - **The UPS/batteries are not used with the 6690 cabinet.**
  - **Data integrity in event of site wide power loss is provided by WAL.**
- **A 6690 nodes uses the Intel six-core Westmere CPUs with hyper-threading enabled. The 6690 has a faster CPU (2.93 GHz. versus 3.06 GHz) than the previous 6680 node.**
  - **These systems can be configured in 1+1 or 2+1cliques.**
  - **A 6690 clique is contained within 1 processing/storage cabinet.**
- **No co-existence with Active Warehouse 5xxx and not planned for with 6650 systems.**
  - **The 6690 is ideal for new customers and/or floor sweeps.**
  - **The 6690 will co-exist with future Active EDW Platform mixed storage offerings.**

Teradata 6690 Cabinets

Each Teradata 6690 cabinet can be configured in a 1+1 or 2+1 clique configuration.

- A processing/storage cabinet contains one clique.
- A cabinet with a 2+1 clique contains two processing nodes, one hot standby node, and four disk arrays.
- A cabinet with a 1+1 clique contains one processing node, one hot standby node, and four disk arrays.

Virtualized Management Server (VMS)

The VMS is available with the 2690 Appliance and the 6690 Enterprise Warehouse Server.

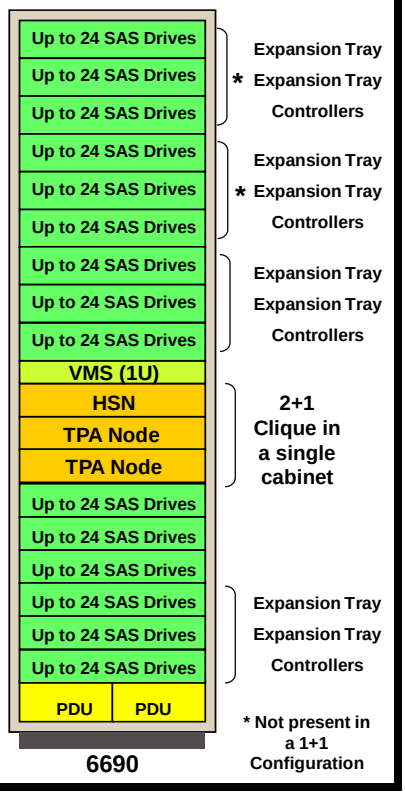
Characteristics of the VMS include:

- 1U Server that VIRTUALIZES system and cabinet management software onto a single server
- **Teradata System VMS** – provides complete system management functionality
  - Cabinet Management Interface Controller (CMIC)
  - Service Workstation (SWS)
  - Teradata Viewpoint (single system only)
  - **Automatically installed on base/first cabinet**
- The VMS allows full rack solutions without an additional cabinet for traditional Viewpoint and SWS
- Eliminates need for expansion racks reducing customers' floor space and energy costs
- **For multi-system monitoring and management traditional Teradata Viewpoint is required.**

Teradata 6690 Cabinets

6690 Characteristics

- Integrated Cabinet with nodes and SSD and HDD arrays in same cabinet.
- Each NetApp drive tray can hold up to 24 SSD and/or HDD drives.
  - SSD drives are 400 GB.
  - HDD drives (10K RPM) are 600 GB.
  - Possible maximum of 360 disks in the cabinet.
- One NetApp tray has 2 controllers and supports 2 additional expansion trays.
- 6690 feature – Virtualized Management Server (VMS)
  - Consolidated CMIC, SWS, Teradata Viewpoint
- No UPSs in cabinet.
- There is no room for BYNET switches in this cabinet. Therefore, BYNET switches are located in a separate cabinet.



Teradata Extended Nodes

Additional specialized nodes are available to Teradata 55xx, 56xx, and 66xx systems. The various type and possible uses are listed on the facing page.

General Notes:

- All TPA nodes (Teradata Nodes running the Teradata Database) must execute the same Operating System.
- Non-TPA Nodes and/or Managed Servers, can execute the same or a different Operating System; this is the "mixed OS support".
- A Non-TPA Node is a Teradata Server (Node) that is BYNET connected, but does not run the Teradata Database. A Non-TPA Node can communicate to the Teradata Database through TCP/IP emulation across the BYNET.
- A Managed Server is a Teradata Server (Node) that resides in the Teradata System Cabinet (rack mounted) and is connected through a dedicated Ethernet network to the Teradata Database Instance.
- The purpose of both Non-TPA Nodes and Managed Server Nodes is flexibility. These nodes can be used similar to external application servers for BAR, ETL/ELT, BI, etc. Some of the advantages of Non-TPA or Managed Server nodes include a single point of management/maintenance, "pre-built" dedicated network to Teradata Database, and they can often be installed into existing Cabinets, minimizing additional footprint in the data center.

Teradata Extended Nodes

Examples of extended node types:

- **Hot Standby Nodes (HSN)**
  - BYNET connected
  - spare node that is part of a clique and is used in the event of a node failure.
  - Located in same cabinet as other nodes; managed by SWS
- **Channel Server (used as interface between Teradata and mainframe (e.g., IBM)**
  - BYNET connected
  - Maximum of 3 ESCON and/or FICON adapters – allows host channel connections
  - Node with 1 Quad-core CPU and 24 GB of memory – improves Teradata performance by offloading the channel workload
  - Located in same cabinet as other nodes; managed by SWS
- **Teradata Managed Server (TMS) Nodes** 
  - Not BYNET connected
  - Dell server integrated in processor cabinet for use with Teradata applications
    - Can be utilized as a Viewpoint, SAS, BAR, Ethernet, TMSM, Data Mover, etc. node
  - Located in same cabinet as other nodes; managed by SWS
- **Non-TPA Nodes** 
  - BYNET connected
  - Can be used to execute application software (e.g., ETL)
  - Located in same cabinet as other nodes; managed by SWS

Making Sense of the Different Platforms

The facing page attempts to provide some perspective of the different platforms.

The 4400, 4800, 4850, 5200, and 5250 nodes are based on the Intel Eclipse chassis and Aspen baseboard technology. These nodes are often referred to as Eclipse nodes.

The 4455, 4851, 4855, 5251, and 5255 nodes are based on the Intel Koa baseboard technology. These nodes may be referred to as Koa nodes.

The 4470, 4900 and 5300 nodes are based on the INTEL Dodson baseboard technology and may be referred to as Dodson nodes.

The 4475, 4950 and 5350 nodes are based on the INTEL Hodges baseboard technology and may be referred to as Hodges nodes.

The 4480, 4980, and 5380 nodes are based on the INTEL Harlingen baseboard technology and may be referred to as Harlingen nodes.

The 5400 and 5450 nodes are based on the INTEL Jarrell baseboard technology and may be referred to as Jarrell nodes.

The 155x, 25xx, and 55xx nodes are based on the INTEL Alcolu baseboard technology and may be referred to as Alcolu nodes.

The following dates indicate when these systems were generally available to customers (GCA – General Customer Availability).

– 5100M	January, 1996 (not described in this course)
– 4700/5150	January, 1998 (not described in this course)
– 4800/5200	April, 1999
– 4850/5250	June, 2000
– 4851/4855/5251/5255	July, 2001
– 4900/5300	March, 2002
– 4950/5350	December, 2002
– 4980/5380	August, 2003
– 5400E/5400H	March, 2005
– 5450E/5450H	April, 2006
– 5500E/5500C/5500H	March, 2007
– 2500/5550H	January, 2008
– 2550/2555/5555C/H	October, 2008 (2550) and March, 2009 (2555/5555)
– 1550	December, 2008
– 1600/2580/5600C/H	March, 2010
– 5650C/H	July, 2010
– 6650C/H and 6680	April, 2011
– 2690	October, 2011
– 6690	February, 2012

Making Sense of the Different Platforms

	Model	CPU	BYNET
2003 2004	5350/5380 (2 – 512 nodes)	Intel Xeon 2.8/3.06 GHz	BYNET V2.1
2005 2006	5400/5450H (1–1024 nodes)	Intel Xeon 3.6/3.8 GHz	BYNET V3.0
2007	5500H (1–1024 nodes)	Intel Dual-core Xeon CPUs 2.66 GHz	BYNET V3.1
2008 2009	5550/5555H (1–1024 nodes)	Two Intel Quad-core Xeon CPUs 2.33 GHz	BYNET V3.1/V3.2
2010	5600/5650H (1–4096 nodes)	Two Intel quad or six-core CPUs 2.66/2.93 GHz	BYNET V4.0
2011	6650H/6680/6690 (1–4096 nodes)	Two Intel six-core CPUs 2.93/3.06 GHz	BYNET V4.0

Linux Coexistence Combinations

The facing page illustrates possible Linux coexistence combinations.

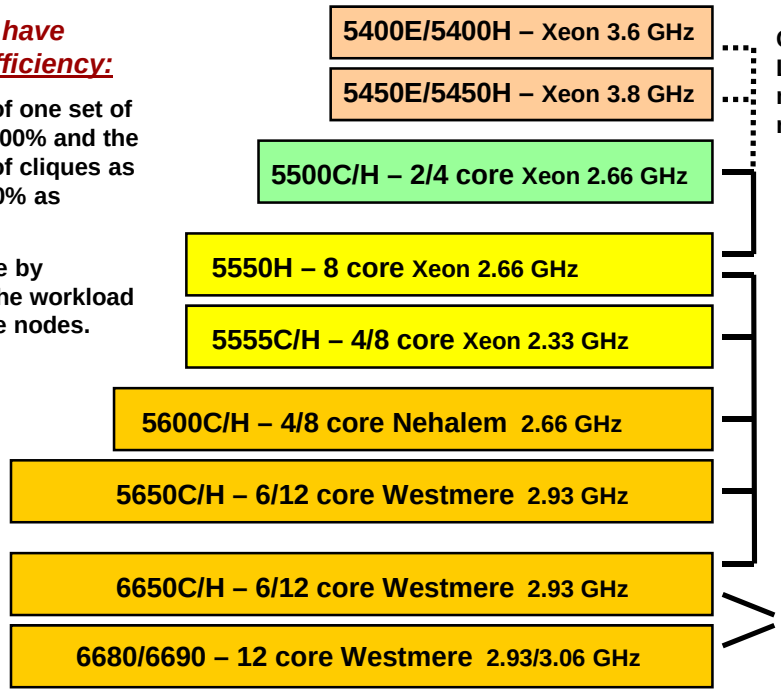
Linux Coexistence Combinations

Coexistence systems contain a mixture of node and storage generations that operate as a single MPP system running the same software.

Goal is to have Parallel Efficiency:

Utilization of one set of cliques at 100% and the other sets of cliques as close to 100% as possible.

This is done by balancing the workload between the nodes.



Conversion to 64-bit Linux is required if the nodes are not already running 64-bit Linux.

May need to leverage larger Linux memory.

66xx systems can coexist with future systems.

Teradata Appliance Introduction

A Teradata appliance is a Teradata server which is optimized specifically for high DSS performance. The first Teradata appliance was the 2500 introduced in 1Q2008. Characteristics of the Teradata appliances include:

- Delivered Ready to Run
  - Integrated system fully staged and tested
  - Includes a robust set of tools and utilities
- Rapid Time to Value
  - System live within hours
- Competitive Price Point
  - Capacity on Demand available if needed
- Easy Data and Application Migration to a Teradata EDW/ADW

What is an Appliance?

An appliance is an instrument or device designed for a particular use. The typical characteristics of an appliance are:

- **Combination of hardware and software designed for a specific function** – for example, the 25xx hardware/software is optimized for fast table scans & "Deep Dive" Analytics.
- **Fixed/limited function** – designed specifically for Decision Support workloads, the hardware is not configured or optimized for ADW.
- **Fixed capacity/configuration** - have a fixed configuration and limited upgrade paths.
- **Ease of installation** – fully staged and the integrated design greatly reduces the number of cabinet interconnect cables.
- **Simple to operate** – appliances are Teradata system! They have all the Server Management and capabilities used in the MPP systems.

Teradata Load 'N Go Services make is easy to quickly implement a new system

- Load data from operational systems
- Five easy steps completed in about one month
  - Step 1 - Build the base database structure
  - Step 2 - Easy Set-Up Options
  - Step 3 - Build and test the load scripts using the TPT Wizard
  - Step 4 - Conduct the initial load
  - Step 5 - Document and turn load/reload process over to customer
- No transformations or consolidation into an enterprise data model
- Users have access to data quickly
- Enabling new business insights

The firmware in the disk array controllers for 25xx systems has been specifically optimized for scan-based workloads. The disk array controller pre-fetches entire cylinder to cache when a cylinder index is accessed by Teradata.

Introduction to Teradata Appliances

- **What is an Appliance?**
  - An appliance is an device designed for a specific function.
  - Fixed/limited function and fixed capacity/configuration.
  - Easy to install and simple to operate.
- **Data Warehouse Appliance**
  - Teradata nodes and storage is integrated into a single cabinet.
  - Delivered ready to run with rapid time to value.
  - System live within hours, fully staged and tested.
- **Powerful**
  - Purpose-built for high analytical performance.
  - Optimized for fast file scans and heavy “deep dive” analytics.
- **Cost-Effective**
  - Competitive price point.
  - Easy data and application migration to a Teradata Enterprise Data Warehouse.
- **Ideal for Entry Level Data Warehouses, Analytical Sand Boxes, and Test and Development Systems.**

Teradata 2500

Teradata 2650/2690 Appliances

Teradata 2650 Appliance

The Data Warehouse Appliance 2650 can have up to 9 nodes in a cabinet. The nodes utilize the Intel Westmere six-core CPU with hyper-threading and 96 GB of memory per node. The Data Warehouse Appliance 2650 comes standard with the BYNET over Ethernet switch. For scalability requirements beyond 275TB you can configure BYNET V4, but special approval is required.

Teradata 2690 Appliance

The Data Warehouse Appliance 2690 can have up to 8 nodes in a cabinet. The nodes utilize the Intel Westmere six-core CPU (3.06 GHz) with hyper-threading and 96 GB of memory per node. Cliques consist of 2 nodes and no HSN. The Data Warehouse Appliance 2690 comes standard with the BYNET over Ethernet switch.

Teradata 2650/2690 Appliances

Teradata appliances utilize a fully integrated cabinet design with nodes and disk arrays in the same cabinet. Two examples of appliances are:

Teradata 2650 Systems

- Nodes use 2 Intel Six-core Westmere CPUs at 2.93 GHz; 96 GB of memory per node
- 24 AMPs per node
  - 24 SAS 300 or 600 GB drives, or 12 SAS 2 TB drives per node
- A 2650 cabinet can house up to 9 nodes.
  - Cliques are in 3 node configurations (no HSN); Cabinets can have 1, 2, or 3 cliques.

Teradata 2690 Systems

- Nodes use 2 Intel Six-core Westmere CPUs at 3.06 GHz; 96 GB of memory per node
- Each node has 2 hardware compression boards
- 24 AMPs per node
  - 24 SAS 300, 600, or 900 GB drives per node (2.5" drives @ 10K RPM)
- A 2690 cabinet can house up to 8 nodes.
  - Cliques are in 2 node configurations (not HSN); a cabinet can have between 1 and 4 cliques.
- Utilizes VMS (Virtualized Management Server)
  - Consolidated CMIC, SWS, Teradata Viewpoint

Teradata 2650/2690 Cabinets

With the 2650, you can have 3 cabinet type configurations: a 1/3 cabinet, 2/3 cabinet and full cabinet. With a full cabinet you have 9 nodes. The disk drives that are supported are 300GB or 600GB drives or 108 2 TB 3.5" drives. To also help improve loading you can configure the system with 10GB Ethernet copper or fiber ports. Cliques are configured with 3 nodes and no HSN.

A 1/3 cabinet is designed for lower CPU/Node density per cabinet. A 2/3 cabinet is designed for medium CPU/Node density per cabinet. It is a good solution for mid-size capacity options and provides flexible solutions for adding an integrated SWS or TMS. A fully populated 2650 cabinet is designed for high CPU/Node density per cabinet. It is a good solution for a high capacity system driving high CPU utilization.

With the 2690, a cabinet can be configured with up to 4 cliques in 2 node clique configurations (no HSN). A full cabinet will have 8 nodes. The disk drives that are supported are 300GB, 600GB, or 900GB drives.

One important new feature of the 2690 is hardware compression.

- With automatic block level compression, customers can get as much as 3x the customer data space, so the amount of available storage has tripled.
- System level scan rate (what's also known as effective scan rate), has increased 3x as well because with compression because 3x more data is scanned.
- Also, hot cache memory, which is where frequently used results are stored until not needed, has tripled as well because the data/results being stored are compressed.

With compression, the system can be pushed higher because compression CPU work has been moved out of the nodes, and that CPU is available for Teradata work.

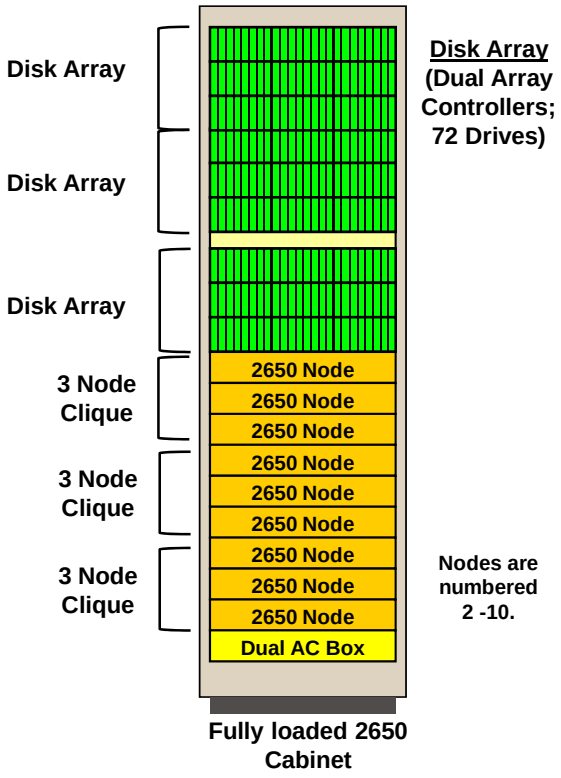
The Teradata Virtualized Management Server is a standard feature on the Data Warehouse Appliance 2690. This 1U managed server rack mounts in the appliance node cabinet and essentially consolidates all Teradata management functionality into one server. The VMS contains the following functionality:

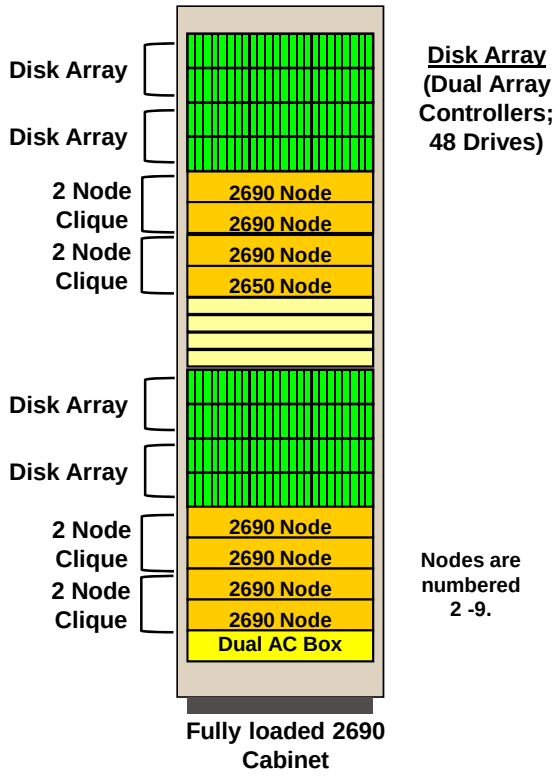
- Teradata Viewpoint, single system: Teradata Viewpoint is the robust web portal that manages workloads, queries, and systems.
- SWS: The Teradata SWS is the software that monitors the physical cabinet. This includes the nodes, disks, and connectivity.
- CMIC: The CMIC monitors all the disk controllers and cabling

The VMS is a key reason why full racks can be shipped without having to have a separate expansion cabinet for this functionality. Some considerations include:

- Traditional Viewpoint is still available, but it is priced and licensed differently. Please see the Teradata Viewpoint OCI for more information. Also note that VMS Viewpoint can only monitor one system, not multiple
- If more than one node cabinet is required, the expansion cabinet will also have a VMS but will only contain the CMIC software as the others aren't needed.

Teradata 2650/2690 Cabinets





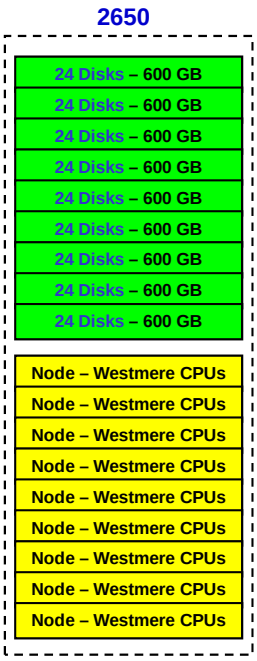
Appliance Configuration Examples

The examples on the facing page show a typical AMP and Disk configurations for 2650 and 2690 systems.

Notes:

- 2650 systems utilize SAS disks (Serial Attached SCSI) – 300 GB and 600 GB disk drives
- 2650 systems can utilize 2 TB SATA disks (Serial Advanced Technology Attachment)
- 2690 systems can utilize 300, 600, or 900 GB SAS disk drives.

Appliance Configuration Examples



(Up to 9 Nodes in a cabinet)

2650 Disk Options

- 300 or 600 GB SAS Disks
  2.5" – 216 in cabinet
- 2 TB Disks
  3.5" – 108 in cabinet

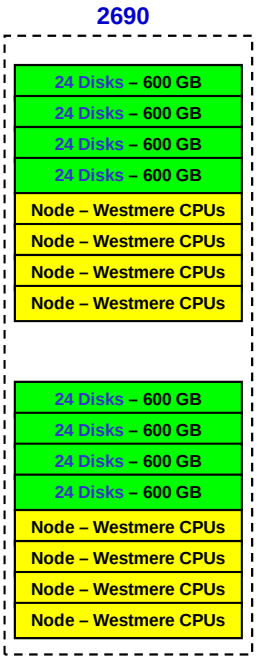
2650 Clique

- 3 Node Cliques share 3 drive trays
- 96 GB Memory / Node
- 24 AMPs / Node
- 72 AMPs /Clique

- 24 Disks / Node (RAID 1)
- 72 Disks / Clique

2650 Cabinet with 9 nodes

- 216 AMPs
- 864 GB memory in cabinet



(Up to 8 Nodes in a cabinet)

2690 Disk Options

- 300, 600, or 900 GB SAS Disks
  2.5" – 192 in cabinet

2690 Clique

- 2 Node Cliques share 2 drive trays
- 96 GB Memory / Node
- 24 AMPs / Node
- 48 AMPs /Clique

- Includes hardware compression.

- 24 Disks / Node (RAID 1)
- 48 Disks / Clique

2690 Cabinet with 8 nodes

- 192 AMPs
- 768 GB memory in cabinet

What is the BYNET™?

The BYNET (BanYan Network) provides high performance networking capabilities for MPP systems. The BYNET is a dual-redundant, bi-directional, multi-staged network based on a Banyan network topology. The BYNET enables multiple processing nodes (SMP nodes) to communicate in a high speed, loosely-coupled fashion.

BYNET communication occurs in a point-to-point, multi-cast, or broadcast fashion. A connection request contains an address or routing tag for the intended receiving node or group of nodes. Once the connection is made, a circuit is established for the duration of the connection. The BYNET works much like a telephone network where many callers can establish connections, including conference calls.

The BYNET interconnect provides a peak bandwidth of x Megabytes (MB) per second for *each* node per direction connected to a network.

- V1 – 10 MB
- V2 – 60 MB
- V3 – 93.75 MB
- V4 – 240 MB

For example, a BYNET v4 network provides 240 MB x 2 (bi-directional) x 2 (BYNETs) = 960 MB/sec per node. A 10-node 5600 system with a dual BYNET network has the potential raw capability of 9600 MB (or 9.6 GB) per second total bandwidth for point–to–point connection. However, the total available broadcast bandwidth is 960 MB per second for a dual network system of any size.

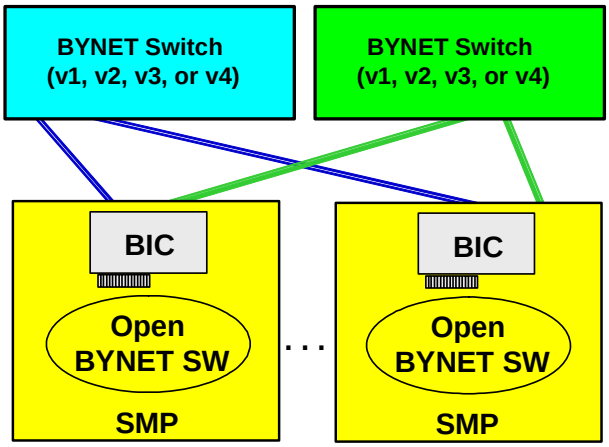
Other features of the BYNET network include:

- Guaranteed delivery - a message from a node is guaranteed to be delivered without error to the receiving node(s); multiple levels of error checking and acknowledgment are used to ensure this.
- Fault tolerant - multiple connection paths are available in each network; dual network feature provides an active backup network should one network be lost.
- Flexible network usage - nodes communicate in point-to-point or broadcast fashion.
- Self-configuring - the BYNET automatically determines network topology at start-up; enables ease of installation.
- Self-diagnosis and automatic fault recovery - automatically detects and reports errors; reconfigures routing of connections to avoid inoperable processing nodes.
- Load balancing - traffic is automatically and dynamically distributed throughout the networks.

What is the BYNET?

What is the BYNET (BanYan NETwork)?

- High speed interconnect (network) for processing nodes in MPP systems. The BYNET is a dual redundant network.
- BYNET works much like a telephone network where many callers (nodes) can establish connections, including conference calls.
- BYNET Version 3 Switches – 375 MB/Sec per node
- BYNET Version 4 Switches – 960 MB/Sec per node



BYNET Switch Examples

- BYNET 4 switch (v2.1 – 240 MB/sec)
- BYNET 32 switch (BYA32S)
  - Can execute at v3 or v4 speed
- BYNET 64 switch (v3.0 – 12U switches)
- BYNET 64 switch (v4.0 – 5U switches)

BIC (BYNET Interface Card) Examples
(these can run at v3 or v4 speeds)

- BIC2SX – used with 54xx nodes
- BIC2SE – used with 5500 nodes and later

BYNET 32 Switches

The facing page contains of an example of a BYNET 32 switches. Examples of other BYNET switches are listed below. This is not an inclusive list.

BYNET 4 Switch Version 2 (BYA4G) – a PCI card designed to interconnect up to 4 SMPs. This switch is a BYNET v2 switch (60 MB/sec.) designed for 485x systems. The BYA4G is a PCI card that is placed into a PCI slot of an SMP.

BYNET 4 Version 2.1 Switch (BYA4M) – PCI card designed to interconnect up to 4 SMPs. This switch is a BYNET v2.1 switch (60 MB/sec.) designed for 4900 systems. The BYA4M is a PCI card that is placed into a PCI slot of an SMP.

BYNET 4 Switch Version 2.1 (BYA4MS) – PCI card designed to interconnect up to 4 SMPs. This BYNET V2.1 switch (60 MB/sec.) was designed for 4980 systems. The BYA4MS has a shorter form factor – S is for shorter.

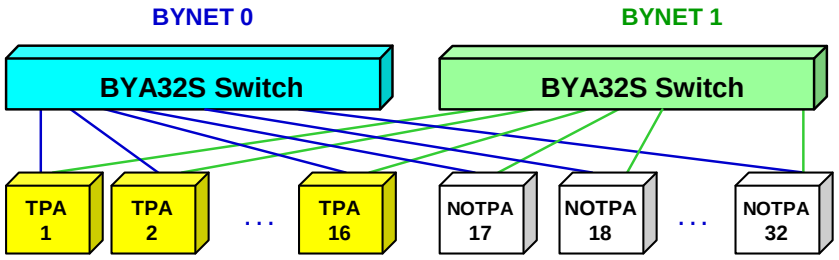
BYNET 32 Switch (BYA32S) – this switch can run at v3 or v4 speeds depending on the system and type of BICs. Up to 16 TPA nodes and 16 NOTPA nodes can be connected to this switch. This 1U chassis switch resides in a Base or System Cabinet.

- Includes an Ethernet Interface for BYNET status & error reporting and chassis management.

Note on BYNET cables:

- There is a physical difference between BYNET v2 and BYNET v3/v4 cables. The BYNET v3/v4 cables have a “Quick Disconnect” connector whereas the BYNET v2 cables have a “Micro D” connector with 2 screws. The number of wires inside the cables is the same.

BYNET 32 Switches



BYNET 32 switch (BYA32S) is a 1U chassis used in an processor rack.

- **This 32-port switch can execute at v3 or v4 speeds.**
- **Up to 16 TPA nodes can be connected.**
- **An additional 16 HSN, Channel, or non-TPA nodes can be connected.**

BYNET 64 Switches

For configurations greater that 16 TPA/HSN nodes, BYNET 64 switches must be used.

BYNET 64 Node Switch Version 2 (BYA64GX chassis) – this switch is actually composed of 8 BYA8X switch boards in the BYA64GX chassis. Each BYA8X switch board allows up to 8 SMPs to interconnect (i.e., 8 switches x 8 SMPs each = 64 SMPs). The BYA64GX is actually a backpanel that allows the 8 BYA8X switch boards to interconnect.

This 12U chassis resides in either the BYNET V2 64 Node Switch cabinet or the BYNET V2 64/512 Node Expansion Cabinet.

Note: BYA8X switch board (in BYA64GX chassis): This is Stage A base switch board. Each board supports 8 links to nodes. The BYA64GX chassis can contain a maximum of 8 BYA8X switches, allowing for 64 links to nodes. In systems greater than 64 nodes, the BYA8X switch boards also connect the BYA64GX chassis to BYB64G chassis through X-port connectors, one on each BYA8X board.

BYNET Switch Cabinets

Even though the BYNET switch cabinets are different for BYNET v2, v3, and v4. However, the basic purpose is the same - the purpose is to house BYNET 64 switches.

The **BYNET 64 Node Switch Cabinet** (shown on facing page) can be used for configurations from 2 through 64 nodes and must be used for configurations greater than 16 nodes. All nodes in the configuration are interconnected from the BYNET (V2 or V3) node interface to the BYNET (V2 or V3) 64 Node Switch chassis (BYA64GX). Two BYNET (V2 or V3) 64 Node Switch Cabinets are required for the base dual redundant BYNET V2 networks.

The **BYNET 512 Node Expansion Cabinet Version 2 (or 3)** (not shown) is for used for configurations that begin with 64 nodes or less and has expanded beyond 64 node maximum configuration supported by the BYNET BYA64GX chassis (in the BYNET 64 Node Switch Cabinet). Above 64 nodes, the BYNET BYB64G chassis (effectively a 512 node switch chassis) is used to interconnect multiple BYNET 64 node switch chassis. The simple configuration rules are:

- Each group of 2 to 64 nodes requires two BYNET V2 64 node switch chassis; a minimum of two is required for dual redundancy.

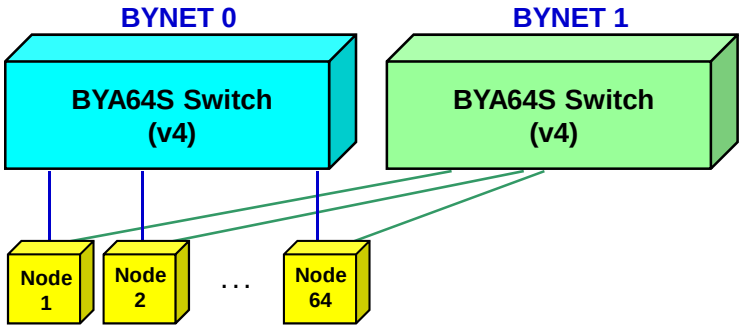
- For configurations with greater than 64 nodes, each BYNET V2 64 node switch chassis must have a complimentary BYNET V2 512 node switch chassis.

BYNET 64 Switches

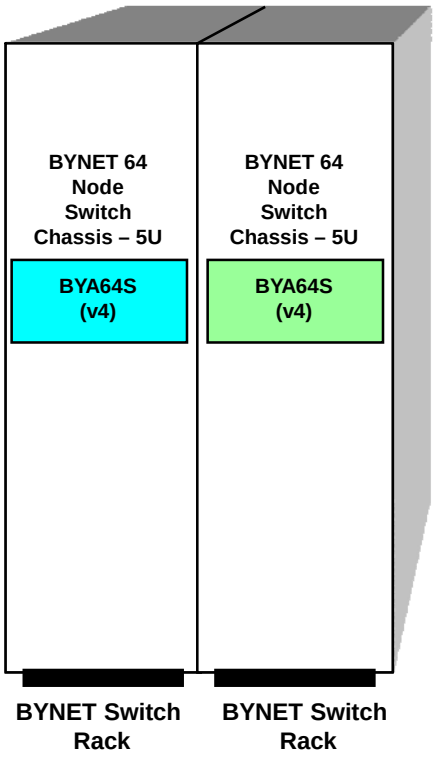
A BYNET 64 Switch is a separate chassis located inside a BYNET rack or cabinet.

- **BYNET v3 64 Switches (BYA64GX) – 12U in height**
  - **375 MB/sec per node for both BYNET channels**
- **BYNET v4 64 Switches (BYA64S) – 5U in height**
  - **960 MB/sec per node for both BYNET channels**

Two BYNET switch racks are needed to house these two BYNET 64 switches.



Nodes connect to BYA switches.



BYNET Expansion Switches

With BYNET v3, the BYA64GX and BYC64G switches are physically identical. What makes them different is the firmware that is loaded onto the BYNET switch and how they are cabled together.

- The base chassis is the same as the BYNET version 2 base chassis. This includes including sheet metal and backpanel, power supplies, power control board, and fans.

- The v3 BYA8QX switch is new within the BYA64GX and BYC64G switches.

BYNET V3 64-node Switch Chassis

The BYNET V3 64-node Switch Chassis are used in 5400H systems with greater than 16 nodes. Each switch chassis resides in its own cabinet or co-reside with a BYNET V3 1024-node Expansion Switch Chassis. Each BYNET V3 64-node Switch Chassis provides the BYNET switching for its own BYNET V3 fabric. Therefore, for redundancy; two 64-node Switch Chassis are needed. In systems with greater than 64 nodes, two BYNET 64-node switches are needed for every 64 nodes.

BYNET V3 1024-node Expansion Switch Chassis

The BYNET V3 1024-node Expansion Switch Chassis (marketing name) is used in 5400H systems with greater than 64 nodes. The 1024-node switch resides in its own cabinet or co-resides with a BYNET 64-node switch.

The total number of 1024-node switch chassis needed in a system is a power of 2 based on the number of nodes.

- For systems with 65 - 128 nodes, two 1024-node switches are needed per BYNET fabric (total of 4).

- For systems with 129 – 256 nodes, four 1024-node switches are needed per BYNET fabric (total of 8).

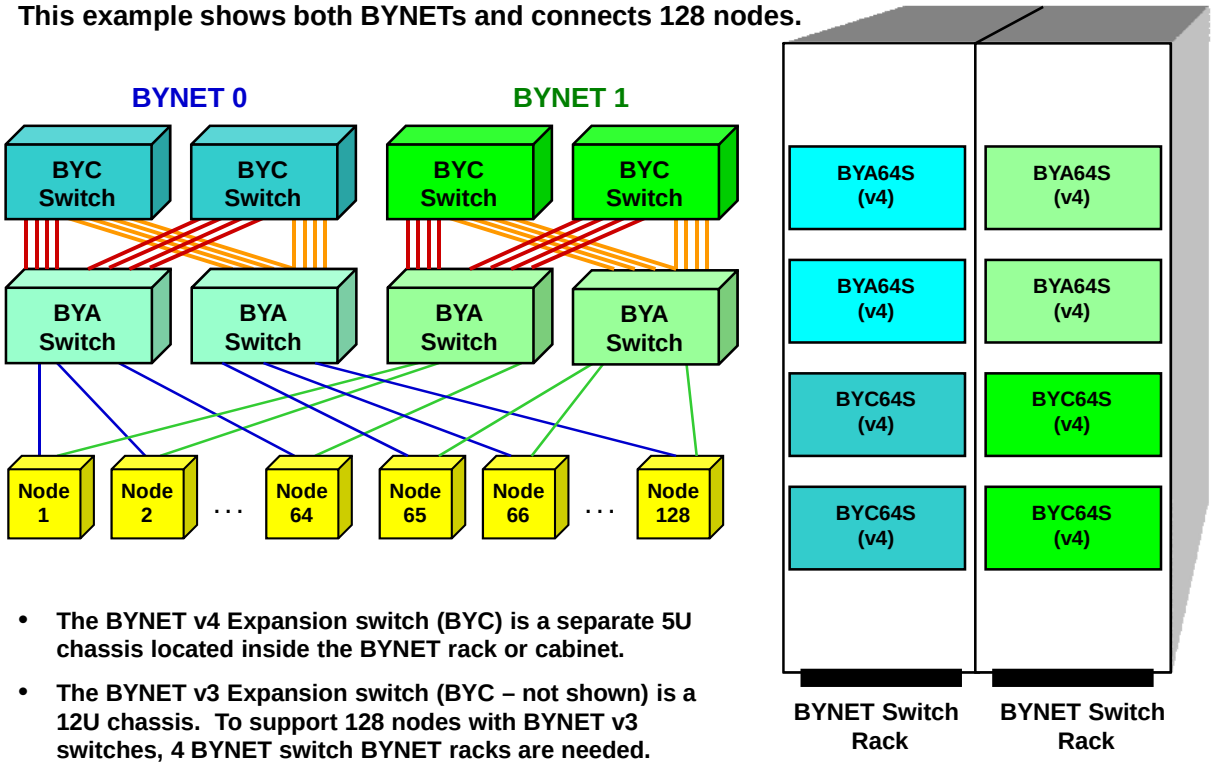
- For systems with 257 – 512 nodes eight 1024-node switches are needed per BYNET fabric (total of 16).

BYNET Expansion to 1024 Nodes

BYNET v3/v4 support configurations up to 1024 nodes. For BYNET v3, in order to interconnect more than 512 nodes additional BYOX and BYCLK hardware is needed.

BYNET Expansion Switches

This example shows both BYNETs and connects 128 nodes.



- The BYNET v4 Expansion switch (BYC) is a separate 5U chassis located inside the BYNET rack or cabinet.
- The BYNET v3 Expansion switch (BYC – not shown) is a 12U chassis. To support 128 nodes with BYNET v3 switches, 4 BYNET switch BYNET racks are needed.

Server Management with SWS

The SWS (Service Workstation) provides a single operational view for Teradata MPP Systems and the environment to configure, monitor, and manage the system. The SWS effectively is the central console for MPP systems.

The SWS is one part of the Server Management subsystem that provides monitoring and management capabilities of MPP systems. Prior to the SWS, other server management environments were:

1st Generation Server Management (3600) – Server Management (SM) processing, storage and display occurred on AWS.

2nd Generation Server Management (5100, 48xx/52xx, 49xxx/53xx) – most SM processing occurs on CMICs and Management Boards. The AWS still provides all the storage and display.

3rd Generation Server Management (54xx systems and beyond) – most SM processing occurs on CMICs and Management Boards. The SWS still provides all the storage and display. The Server Management subsystem uses industry standard parts, **a Server Management Node** and **Ethernet switches** to implement an Ethernet based Server Management solution. This new Server Management is referred to a Third Generation Server Management (SM3G).

One of the reasons for the new Server Management subsystem is to better adhere to industry standards. Ethernet-based management is now the industry standard for chassis vendors.

Virtualized Management Server (VMS)

The Teradata Virtualized Management Server is a standard feature on the Data Warehouse Appliance 2690 and the 6690. This 1U managed server rack mounts in the appliance node cabinet and essentially consolidates all Teradata management functionality into one server. The VMS contains the following functionality:

- Teradata Viewpoint, single system: Teradata Viewpoint is the robust web portal that manages workloads, queries, and systems.
- SWS: The Teradata SWS is the software that monitors the physical cabinet. This includes the nodes, disks, and connectivity.
- CMIC: The CMIC monitors all the disk controllers and cabling

The VMS is a key reason why full racks can be shipped without having to have a separate expansion cabinet for this functionality. Some considerations include:

- Traditional Viewpoint is still available, but it is priced and licensed differently. Please see the Teradata Viewpoint OCI for more information. Also note that VMS Viewpoint can only monitor one system, not multiple
- If more than one node cabinet is required, the expansion cabinet will also have a VMS but will only contain the CMIC software as the others aren't needed.

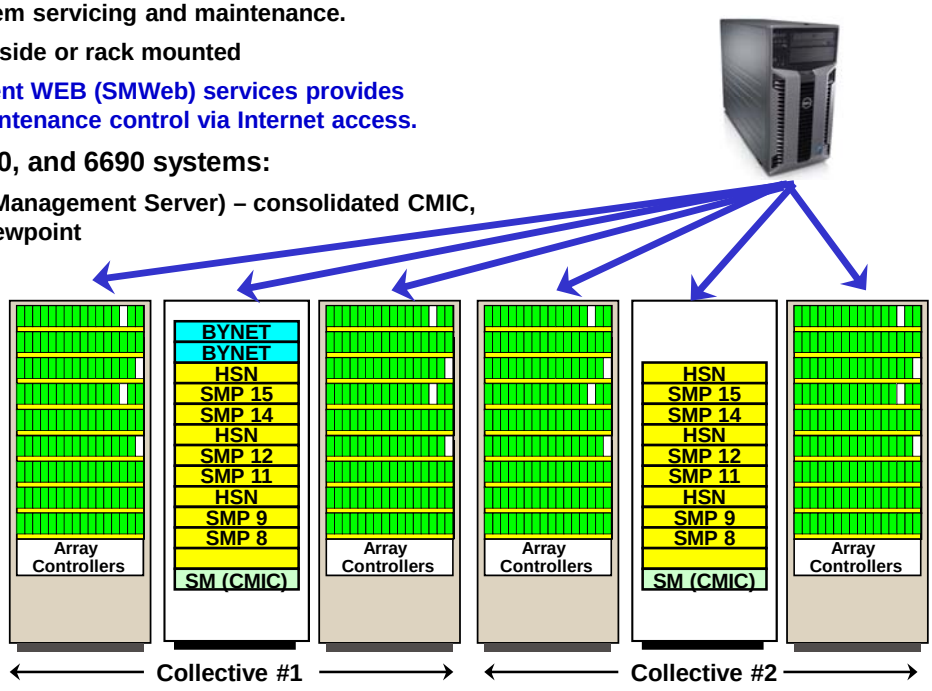
Server Management with SWS

For 1600, 56xx, and 66xx systems:

- The SWS (Service Workstation) is a Linux workstation that is dedicated to system servicing and maintenance.
  - May be deskside or rack mounted
- Server Management WEB (SMWeb) services provides operational & maintenance control via Internet access.

Option for 1650, 2690, and 6690 systems:

- VMS (Virtualized Management Server) – consolidated CMIC, SWS, Teradata Viewpoint



SMWeb services provide the ability to:

- connect to AWS type display
- connect to nodes
- power on/off/reset
- manage alerts
- obtains h/w or s/w status information

Node Naming Conventions

The examples on the facing page show AWS naming conventions for cabinets or racks. Each chassis consists of a number of internal components (processors, fans, power supplies, management boards, etc.). The chassis numbering for 52xx/53xx cabinets starts at 1 from the top of the cabinet to bottom of the cabinet. The chassis numbering for 54xx and 55xx cabinets starts at 1 from the bottom of the cabinet to the top of the cabinet.

54xx/55xx Chassis Numbering Conventions

A standard chassis numbering convention is used for the 54xxE, 54xxH/LC, 55xxC/H, Storage, and BYNET cabinets. The chassis numbers are not defined by hardware, but only by convention and numbering defined in the CMIC configuration file. Chassis numbers begin with one and go up to 22. Chassis numbers are assigned to the position; chassis numbers begin for each type of chassis as defined below and are not skipped if a chassis is not installed.

All Cabinets

In all cabinets, chassis **1** is the bottom UPS, the numbering continues upward until all UPS(s) are assigned. Up to 5 UPS(s) can exist in the cabinet.

Node Cabinets

Chassis 6 - CMIC in the Node cabinets

Chassis 7 through 16 – Nodes; the bottom node chassis starts **7** and continues up to **16**. The chassis number is assigned to the position, if no node is installed the chassis number is skipped. If only 8 TPA nodes in a rack, then nodes are numbered 9 to 16.

Chassis 17 and 18 – BYA32Gs

Chassis 19 through 22 – FC switches (if present)

Storage Cabinets

Chassis 4 – SM Chassis (CMIC) in a Storage cabinet (if present)

Chassis 5 and 6 – Disk Array Controller chassis; lower disk array is 5, the upper is 6. Disk Array names are DAMCxxx-y-z where xxx is collective number, y is cabinet number, and z is chassis number.

BYNET Cabinets

Chassis 4 and 5 – BYNET 64 switches (Chassis 4 - BYC64, Chassis 5 - BYA64)

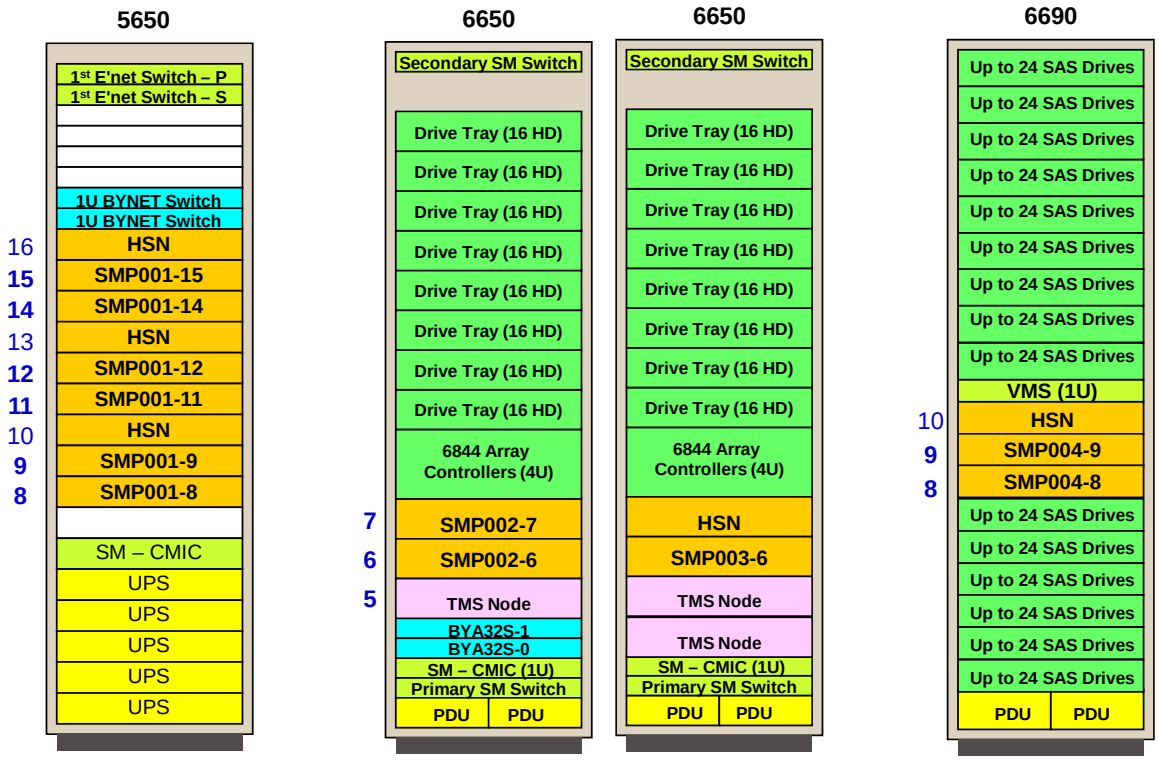
54xx/55xx Collective Conventions

A collective is made up of the node and disk array cabinets that are part of the same server management group (usually the same clique).

- Include the first BYNET Cabinet to the first Node Cabinet Collective
- Include the second BYNET Cabinet to the second Node Cabinet Collective
- Include the third BYNET Cabinet to the third Node Cabinet Collective, etc
- Remember, only one BYNET Cabinet may be configured in any 54xx Collective

The SM3G Collectives are defined in software using the **CMIC Configuration Utility**. The CMIC Configuration Records (**CMICConfig.xml**) contain configuration information for all the chassis in a CMIC's collective. All SM3G chassis must reside on the same Primary and Secondary management networks.

Node Naming Conventions



Summary

The facing page summarizes the key points and concepts discussed in this module.

Summary

	Data Mart Appliance	Extreme Data Appliance	Data Warehouse Appliance	Extreme Performance Appliance	Active Enterprise Data Warehouse
Purpose	Test/ Development or Smaller Data Marts	Analytics on Extreme Data Volumes from New Data Types	Data Warehouse or Departmental Data Marts	Extreme Performance for Operational Analytics	Enterprise Scale for both Strategic and Operational Intelligence EDW/ADW
Possible Uses	Departmental Analytics, Entry level EDW	Analytical Archive, Deep Dive Analytics	Strategic Intelligence, Decision Support, Fast Scan	Operational Intelligence, Lower Volume, High Performance	Active Workloads, Real Time Update, Tactical and Strategic response times

Module 9: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 9: Review Questions

1. What is a major difference between a 6650 system as compared to a 6690 system?

   _____

2. What is a major difference between a 2650 node and a 2690 node?

   _____

3. What does the acronym represent and briefly define the purpose of the following subsystems?

   BYNET  _____

   SWS  _____

4. Specify the names of the two TPA nodes in 6690 cabinet #2.

    _____  _____

Play the numbers games – match the number to a definition.

___	1. 3	a.	Typical # of AMPs per node in a 6650 3+1 clique
___	2. 8	b.	Maximum number of nodes that can be in a 2690 cabinet
___	3. 24	c.	Maximum number of drives in one NetApp 6844 disk array
___	4. 42	d.	Number of nodes in a 2650 clique
___	5. 128	e.	Large disk drive size (GB) for a 2690 disk array
___	6. 900	f.	Typical # of AMPs in a 2690 node

Notes

Module 10

How Teradata uses MPP Systems

After completing this module, you will be able to:

- **Identify items that are placed into FSG cache.**
- **Identify a purpose for the WAL Depot and the WAL Log.**
- **Describe the fundamental relationship between Linux, logical units, and disk array controllers.**
- **Describe the fundamental relationship between Vdisks, Pdisks, LUNs, and partitions.**

Notes

Table of Contents

Teradata and the Processing Node 10-4
FSG Cache 10-4
Memory and the Teradata Database 10-6
5555H Example 10-6
SMP Memory – Summary 10-8
Determining FSG Cache 10-8
O.S. Managed Memory and FSG Cache 10-10
WAL – Write Ahead Logic 10-12
WAL Concepts 10-14
Linux Vproc Number Assignment 10-16
Disk Arrays from a O.S. Perspective 10-18
Logical Units and Partitions 10-20
EMC^2 Notes 10-20
Teradata and Disk Arrays 10-22
Teradata 6650 (2+1) Logical View 10-24
Teradata 6650 (3+1) Logical View 10-26
Example of 1.2 TB Vdisk (pre-TVS) 10-28
Teradata File System Concepts 10-30
Teradata Vdisk Size Limits 10-30
Teradata 13.10 Large Cylinder Support 10-32
When to Use This Feature 10-32
Full Cylinder Read 10-34
Summary 10-36
Module 10: Review Questions 10-38

Teradata and the Processing Node

The example on the facing page illustrates a 5650H processing node running Linux and Teradata.

Memory will initially be allocated for the operating system and Teradata vprocs. PDE will calculate how much memory to allocate to itself for FSG (File Segment Cache) based on memory not being used by the operating system and the Teradata vprocs. PDE software will manage the FSG memory space.

Practical experience (for most environments) indicates that the operating system (e.g., Linux) may need more than this initial allocation during startup. For these reasons, PDE is not assigned all of the remaining memory for FSG cache, but a percentage (e.g., 90%) of the remaining memory.

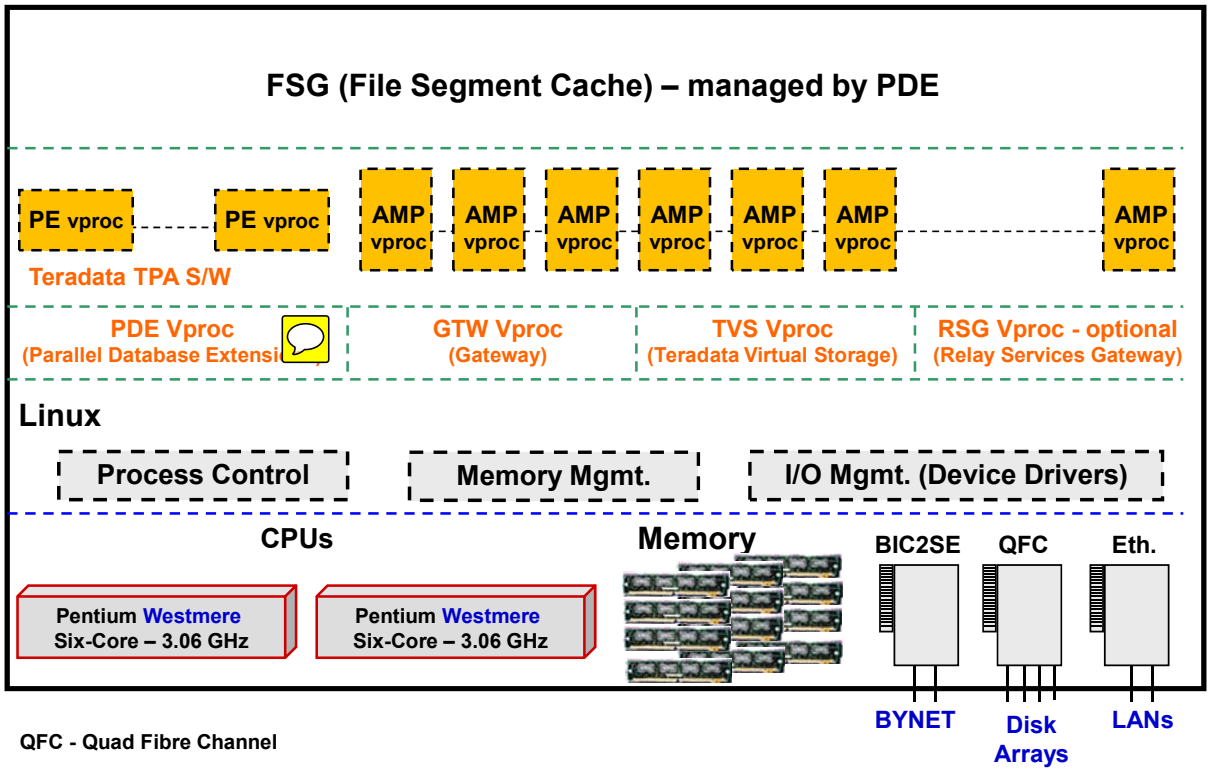
Also note that LAN and Channel adapters (PBSA) also require memory for network and channel activity. For example, each channel adapter uses memory buffers up to 500 MB in size. For 56xx systems, LAN and Channel Adapters not utilized within a TPA node. These are implemented in “Extended Node Types”.

FSG Cache

FSG Cache is primarily used by the AMPs to access memory resident database segments.

When the Teradata Database needs to read a database block, it checks FSG Cache first.

Teradata and the Processing Node



Memory and the Teradata Database

The example on the facing page assumes a 5650H node with 96 GB of memory executing the Teradata Database. This example assumes 42 AMPs, 2 PEs, PDE, GTW, RSG, and TVS vprocs for a total of 48 vprocs in this node. This means that memory will have to be allocated for the 48 vprocs.

The operating system, device drivers, and Teradata vprocs for a 6650H Linux node with 96 GB of memory will use approximately 18 GB of memory. PDE will use a FSG Cache Percent (CTL parameter) to calculate how much memory to allocate to itself for FSG (File Segment Cache) based on the available memory (96 GB – 18 GB).

Practical experience (for most environments) indicates that the operating system (e.g., Linux) may need more than this initial allocation during startup. Parsing Engines and AMPs will typically use more than their initial allocation of memory (80 MB). For example, redistribution buffers for an AMP may use an additional 130 MB of memory for a total of 210 MB of memory per AMP.

For these reasons, PDE is not assigned all of the remaining memory for FSG cache, but a percentage of the remaining memory. The default of 90% for FSG Cache Percent works for most 66xx systems. 90% of 78 GB (96-18) = 70.2 GB of FSG cache.

This can be verified by using the ctl utility hardware function, it can be determined that 42 AMPs have an average of 1.669 GB of memory. 42 x 1.669 = 70.1 GB of FSG cache.

5555H Example

Assume a 5555H node with 32 GB of memory executing the Teradata Database.

Assume that a typical 5555H node will have 25 AMPs, 2 PEs, PDE, GTW, RSG, and TVS vprocs for a total of 31 vprocs. This means that memory will have to be allocated for the 31 vprocs.

The operating system, device drivers, and Teradata vprocs for a 5555H Linux node with 32 GB of memory may use as much as 5.8 GB of memory. PDE will use a FSG Cache Percent (CTL parameter) to calculate how much memory to allocate to itself for FSG (File Segment Cache) based on the available memory (32 GB – 5.8 GB).

The 5.8 GB is based on the Design Center recommendation for a 5555H node with 32 GB of memory.

For these reasons, PDE is not assigned all of the remaining memory for FSG cache, but a percentage of the remaining memory. The default of 80% for FSG Cache Percent works for most 5555 systems.

Memory and the Teradata Database

Example of 6650 (Linux) node with 2 PEs and 42 AMPs and 96 GB of memory:

Memory

O.S., Device Drivers, and space for vprocs ≈ 18 GB

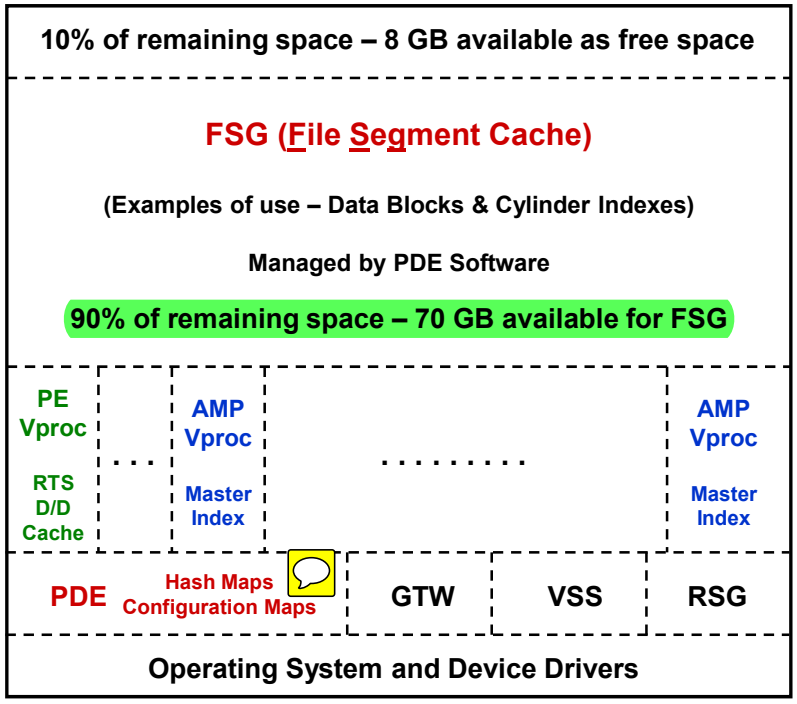
96 GB
– 18 GB
78 GB

FSG Cache 90%
FSG Cache ≈ 70 GB

Free Memory ≈ 8 GB

Examples of objects that are memory resident:

Hash Maps
Configuration Maps
Master Indexes
RTS – Request-to-Steps Cache
D/D – Data Dictionary Cache



Ex. 96 GB Memory

SMP Memory – Summary

Practical experience (for most environments) indicates that Linux and Teradata vprocs need more memory than initially allocated during normal processing periods. Prior to V2R5, it was recommended that at least 20 MB to 40MB of additional free memory be available for each AMP. With 32-bit systems, it is recommended that a node have at least 60 – 80 MB of free memory available for each AMP. With 64-bit systems, each AMP may use up to 210 MB of memory. This would be an additional 130 MB of memory per AMP.

This is accomplished by not giving 100% of the remaining memory to FSG. It is always recommended that the FSG Cache Percent be set to a value less than 100%. The default of 90% for FSG Cache Percent works well for most 56xx and 66xx configurations. 80% usually works well for 5555 configurations.

Determining FSG Cache

The "ctl" utility can be used to determine how much FSG cache memory is actually available to a node.

Using the "ctl" utility, the ***hardware*** command will report the amount of FSG cache for each AMP. The values below represent the average amount of FSG memory per AMP. Examples are shown below.

For a 5555H node with 25 AMPs, the report will indicate 838,016 KB/per AMP.

838,016 KB/AMP x 25 = 20,950,500 KB or approximately 21 GB of FSG cache.

For a 2555H node with 36 AMPs, the report will indicate 582,016 KB/per AMP.

582,016 KB/AMP x 36 = 20,952,576 KB or approximately 21 GB of FSG cache.

For a 5600H node with 40 AMPs, the report will indicate 1,753,856 KB/per AMP.

1,753,856 KB/AMP x 40 = 70,154,240 KB or approximately 70 GB of FSG cache.

For a 5650H node with 47 AMPs, the report will indicate 1,472,000 KB/per AMP.

1,472,000 KB/AMP x 47 = 69,184,000 KB or approximately 69.2 GB of FSG cache.

For a 6650H node with 42 AMPs, the report will indicate 1,669,120 KB/per AMP.

1,669,120,000 KB/AMP x 42 = 70,103,040 KB or approximately 70.1 GB of FSG cache.

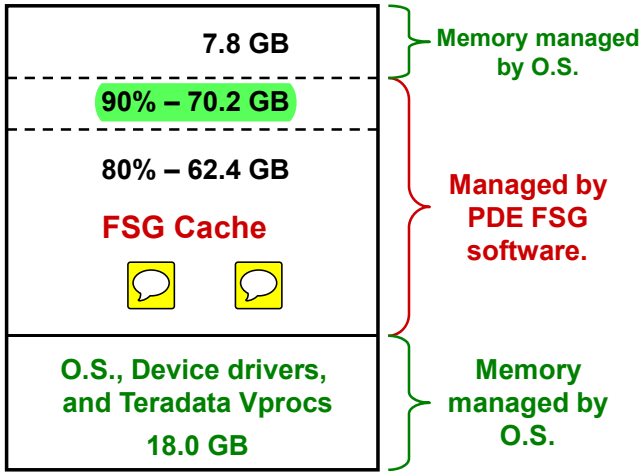
SMP Memory – Summary

Based on the configuration and FSG Cache Percent value, PDE will determine the amount of memory to allocate for FSG cache.

However, vprocs (especially AMPs) will use more than their initial memory allocations during normal processing (e.g., redistribution buffers, aggregations buffers, hash join buffers, etc.).

Some basic guidelines for AMPs are:

64-bit systems – assume 210 MB per AMP



Ex. 96 GB Memory

FSG – pool of memory managed by PDE and each AMP uses what it needs.

ctl Parameter – FSG Cache Percent – for 66xx, the design center recommendation is 90% and this works for most configurations.

O.S. Managed Memory and FSG Cache

The facing page lists examples of how Operating System managed memory (free memory) and FSG cache is used.

Memory managed and used by the operating system and the vprocs is sometimes called "free memory". The main code (on a TPA node) that uses free memory is the operating system and Teradata vprocs

A brief description of Teradata Vprocs:

- AMP Access module processors perform database functions, such as executing database queries. Each AMP owns a portion of the overall database storage.
- GTW Gateway vprocs provide a socket interface to Teradata Database on Windows and Linux systems. On MP-RAS systems, the same functionality is provided by gateway software running directly on the system nodes within the PDE vproc.
- Node (or Base) PDE vproc - the node vproc handles PDE and operating system functions not directly related to AMP and PE work. Node vprocs cannot be externally manipulated, and do not appear in the output of the Vproc Manager utility.
- PE Parsing engines perform session control, query parsing, security validation, query optimization, and query dispatch.
- RSG Relay Services Gateway provides a socket interface for the replication agent, and for relaying dictionary changes to the Teradata Meta Data Services (MDS) utility.
- TVS Manages Teradata Database storage. AMPs acquire their portions of database storage through the TVS (previous releases named this VSS) vproc.

When Teradata needs to read a database block, it checks FSG Cache first.

Examples of how FSG Cache is used

- Permanent data blocks
- Cylinder Indexes
- Spool data blocks
- Transient Journals
- Permanent Journals
- Synchronized scan (sync scan) data blocks

O.S. Managed Memory and FSG Cache

Memory managed by the O.S. is referred to as "free memory".

- **Teradata Vprocs**
  - **AMP** – includes AMP worker tasks
  - **PE** – Session control, Parser, Optimizer, Dispatcher
  - **PDE (Parallel Database Extensions)** – messaging, FSG space management, etc.
  - **GTW (Gateway)** – Logon Security, Session Context, Connection to Client
  - **RSG (Relay Services Gateway)** – Optional; Replication Gateway, MDS auto-update
  - **TVS (Teradata Virtual Storage)** – manages Teradata Virtual Storage
- **Administrative and/or user programs such as:**
  - **kernel resources and administrative program text and data**
  - **message buffers (ex., TCP/IP)**

Memory managed by PDE is called FSG cache. FSG cache is primarily used by the AMPs to access memory resident database segments.

- **When Teradata needs to read a database block, it checks FSG Cache first.**
  - **Permanent data blocks**
  - **Cylinder Indexes**
  - **Spool data blocks**
  - **Journal blocks; Transient Journal and/or Permanent Journals**
  - **Synchronized scan (sync scan) data blocks**

WAL – Write Ahead Logic

WAL (Write Ahead Logic) is a recoverability/reliability feature that can possibly provide performance improvements in the area of database writes. In general, I/O increases with WAL and, therefore, it may reduce throughput for I/O bound workloads. However, the overall performance is expected to be better with WAL since the benefit of CPU improvement outweighs the I/O cost. There is some additional CPU cost for maintaining the WAL log so WAL may reduce throughput for CPU-bound workloads, but is minimal.

Simple example: Assume Teradata Mode, an implicit transaction, and you are doing an UPDATE of a single row in a block that has 300 rows in it.

1. Data block is read into FSG Cache.
2. UNDO row is written to WAL Log (effectively a before-image or TJ type row)
3. The data block in memory is changed and is marked as changed (not immediately written to disk - called deferred write).
4. REDO row is written to the WAL Log (effectively an after-image) - **writing a single REDO row is faster than writing a complete block**
5. The lock is released and the user gets a transaction completed message. Note the updated block is still in memory and hasn't been written to disk yet.

   Note: Other users might be doing updates on rows in the same block and there might be multiple updates to the same block in memory.

6. At some point (maybe a half-second second later), the block needs to be written to disk. This is a deferred write and is done in the background.

6A. If the updated block has not changed size, then it can be written back-in-place. Before physically writing the block back-in-place, the updated block is first written to the WAL depot. After the datablock is successfully written to the WAL Depot, it is then physically written back-in-place.

   Why is the block effectively written twice back to disk? A write operation can fail (called interrupted write) and this can corrupt a block on disk and potentially corrupt all 300 rows. This is a very rare occurrence, but can happen. The WAL Log only has 1 row (REDO row) of the row that has changed. Therefore, by writing the block first to the WAL Depot before writing back-in-place, Teradata ensures that a good copy of the entire datablock is written back-to-disk. The WAL Depot is ONLY used for blocks that haven't changed size - effectively write back-in-place operations. This is an extra internal I/O, but it provides data integrity and protection from interrupted write operations.

6B. If the block has changed size in memory (e.g., block expands to an additional sector), then the updated block is written to a new location on disk - it is not written to the WAL Depot. If there is an interrupted write, the original block has not been touched and the REDO rows along with the original data block can be used for recovery.

WAL can batch up modifications from multiple transactions and apply them with a single disk I/O, thereby saving I/O operations. WAL will help improve throughput for I/O-bound workloads. Obviously, Load utilities such as FastLoad and MultiLoad don't need to use WAL. Other functions such as FastPath operations use the WAL subsystem differently.

WAL – Write Ahead Logic

WAL – Write Ahead Logic

- Available with all Teradata systems – PDE (UNIX MP-RAS) and OpenPDE (Windows and Linux)
- Replaced Buddy Backup in PDE (UNIX MP-RAS) Teradata systems

WAL is a primarily an internal recoverability/reliability feature that also provides performance improvements in the area of database writes.

- All modifications are represented in a log and the log is forced to disk at key times.
- Data Blocks updated in Memory, but not written immediately to disk
- In place of the data block written to disk, the before image (UNDO row) and after image (REDO row) are written to a WAL buffer which is written to the WAL log on disk.
- WAL can batch up modifications from multiple transactions and apply them with a single disk I/O, thereby saving I/O operations. WAL will help improve throughput for I/O-bound workloads.
- Updated data blocks will be eventually aged out and written to disk.

Note: There are numerous DBS Control parameters to specify space allocations for WAL.

WAL Concepts

WAL has its own file system software and uses a fixed number of cylinders for the WAL Depot (varies by vdisk size and DBSControl parameters) and a dynamic number of cylinders for the WAL Log itself.

The WAL Depot consists of two types of slots:

- Large Depot slots
- Small Depot slots

The Large Depot slots are used by aging routines to write multiple blocks to the Depot area with a single I/O. The Small Depot slots are used when individual blocks that require Depot protection are written to the Depot area by foreground tasks.

The number of cylinders allocated to the Depot area is fixed at startup based on the settings of several internal DBS Control flags.

The number of Depot area cylinders allocated is per pdisk, so their total number depends on the number of Pdisks in your system. Sets of Pdisks belong to a subpool, and the system assigns individual AMPs to those subpools.

Because it does not assign Pdisks to AMPs, the system calculates the average number of Pdisks per AMP in the entire subpool from the vconfig GDO when it allocates Depot cylinders, rounding up the calculated value if necessary. The result is then multiplied by the specified values to obtain the total number of depot cylinders for each AMP. Using this method, each AMP is assigned the same number of Depot cylinders.

The concept is to disperse the Depot cylinders fairly evenly across the system. This prevents one pdisk from becoming overwhelmed by all the Depot writes for your system.

WAL (Write Ahead Logic) is a transaction logging scheme maintained by the File System in which a write cache for disk writes of permanent data is maintained using log records instead of writing the actual data blocks at the time a transaction is processed. Multiple log records representing transaction updates can then be batched together and written to disk with a single I/O thus achieving a large savings in I/O operations and enhancing system performance as a result.

The amount of space used for the WAL Log is dynamic. WAL contains before-images (TJ) and after-images (Redo) for transactions. For example, the number of TJ images is very dependent on the type of transaction. Updating every row in a large table places a lot of TJ images into WAL.

Note: Prior to the V2R6.2 release, Teradata systems running under UNIX MP-RAS systems utilized a facility referred to as “buddy backup”.

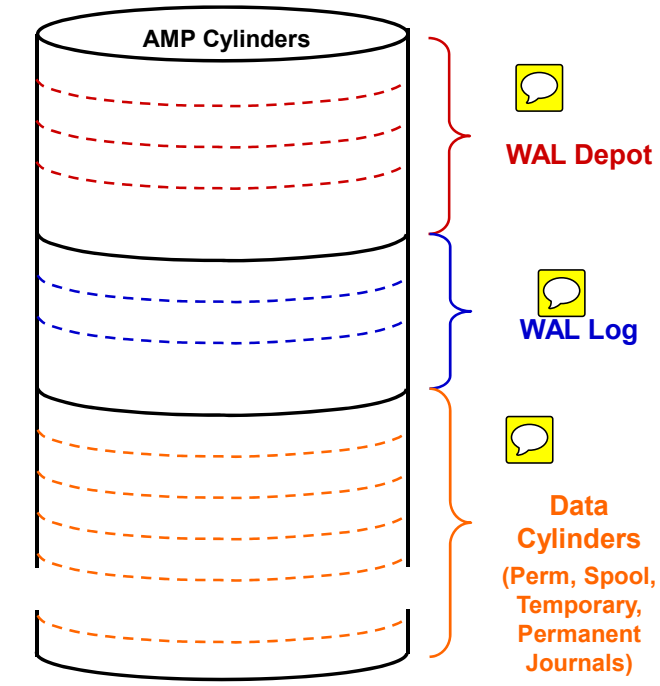
WAL Concepts

WAL Depot

- Fixed number of cylinders allocated to each AMP.
- Used for Write-in-Place operations.
- Teradata first writes data to WAL Depot and if successful, then writes to disk.
- WAL Logic will attempt to group a number of blocks to write to WAL Depot.

WAL Log

- Dynamic number of cylinders used by each AMP.
- Used for new block allocations on disk.
- Contains before-images (UNDO) and after-images (REDO) for transactions – used with both Write-in-Place or new block allocations.
- Updated data blocks will be eventually aged out and written to disk.



Allocation of cylinders is not contiguous.

Linux Vproc Number Assignment

The facing page describes how Vprocs are assigned numbers.

With OpenPDE systems, gateway software is implemented in as separate vproc (named GTW) from the PDE vproc. With MP-RAS systems (PDE), gateway software is incorporated into the PDE vproc.

Within a multi-node single clique system, it is possible for one of the nodes to have a second TVS vproc. This may seem like an anomaly, but this is normal.

For example, assume a 3+1 single clique system:

- In order for fallback to be most effective, a single clique is divided into two subpools of storage and AMPs which reference that storage. Fallback is then setup as a cluster size of two between the two subpools of AMPs. An Allocator (part of TVS vproc) only deals with a single sub-pool of storage. Since in this case we are dividing up two subpools into three nodes, one of the nodes has about half of its storage in one subpool and half of its storage in the other subpool. Therefore, that node needs to have two Allocator vprocs, one for each sub-pool of storage. Any system with more than one clique has only one sub-pool per clique and this anomaly goes away.

- A single node system (which is of course a single clique) also has two sub-pools for the same reason.

With Teradata 13.10 (and previous releases), vproc number ranges are:

AMPs – 0, 1, 2, …
PEs – 16383, 16382, 16381, …
GTW – 8192, 8193, 8194, …
VSS – 10238, 10237, 10236, …
PDE – 16384, 16385, 16386, …
RSG – 9215, 9216, 9217, … (Optional)

When a system is configured with PUT, the installer is presented with an option to choose large vproc numbers if configuring a system with more than 8,192 AMPs. Therefore, starting with Teradata 14.0, optional vproc number ranges are:

AMPs – 0, 1, 2, …
PEs – 30719, 30718, 30717, …
GTW – 22528, 22529, 22530, …
TVS – 28671, 28670, 28669, …
PDE – 30720, 30721, 30722, …
RSG – 26623, 26622, 26621, … (Optional)

Linux Vproc Number Assignment

Each Teradata Vproc is assigned a unique Vproc number in the system. For example:

Typical Vproc assignments:

AMP Vproc #s (start at 0 and increment by 1)

- First AMP 0
- Second AMP 1
- Third AMP 2

PE Vproc #s (start at 16383 and decrement by 1)

- First PE 16383
- Second PE 16382
- Third PE 16381

Optional Vproc assignments starting with Teradata 14.0:

AMP Vproc #s (start at 0 and increment by 1)

- First AMP 0
- Second AMP 1
- Third AMP 2

PE Vproc #s (start at 30719 and decrement by 1)

- First PE 30719
- Second PE 30718
- Third PE 30717

Appear in DD/D and utilities such as Teradata Administrator, Viewpoint etc.

Disk Arrays from a O.S. Perspective

The Operating System is used to read and/or write data to/from an individual disk. Disk arrays trick the operating system into thinking it is writing to a single disk. A disk array LUN looks to the operating system like a single disk. When the operating system gets ready to do a read or a write, the disk array controller steps in and says, “I’ll handle that for you”. The operating system says, “I am writing to a single disk and its address is c10t0d0s1”.

The operating system does not directly read or write to a disk in a disk array environment. The operating system communicates with the disk array controller. The operating system actually reads or writes the data from a logical unit (often referred to as a LUN or a Volume). A logical unit (LUN) or Volume is a logical disk and not a physical disk.

The operating system does not know (or care) if a LUN or Volume is RAID 0, RAID 1, or RAID 5. The operating system does not know if the drive group is one disk, two disk, or four disks. The operating system does not know if the data is spread across one disk or four disks. The operating system simply sees the logical unit as a single disk.

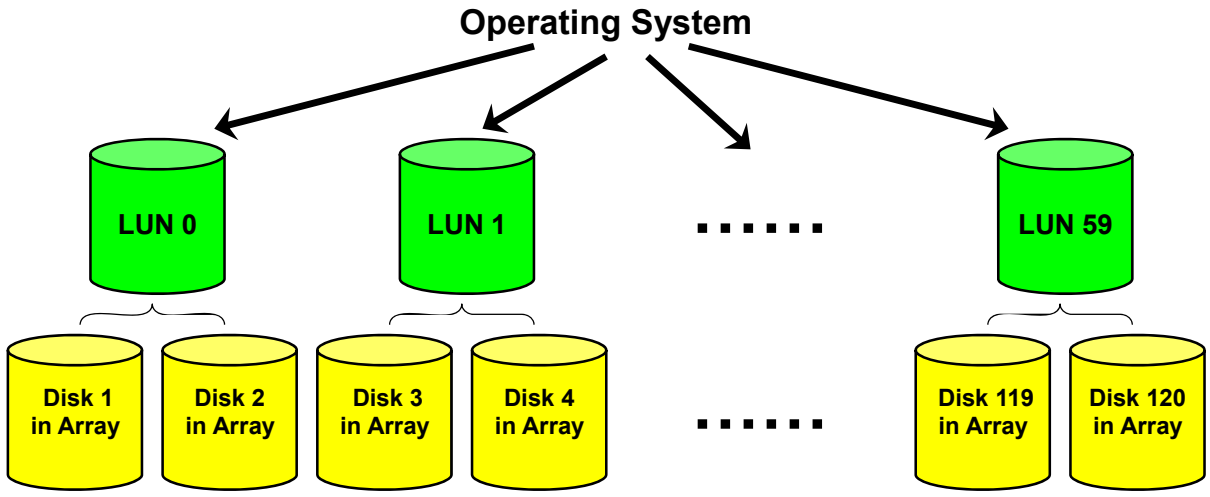
The standard operating system utilities that are used to manage, configure, and utilize a physical disk are also used to manage, configure, and utilize a logical disk or LUN. With the Teradata Database, the **PUT** utility is used to configure the disk array.

The array controller performs the actual input/output operations to its disks. The array controller is responsible for handling the different RAID technologies.

Disk Arrays from an O.S. Perspective

A logical unit (LUN) or Volume is a single disk to the operating system.

- The operating system does not know or care about the specific RAID technology being used for a LUN or Volume.
- The operating system uses LUNs to communicate with disk array controllers.
- It is possible to divide a LUN into one or more partitions (or slices for MP-RAS).



The operating system (e.g., Linux) thinks it is reading and writing to 60 logical disks.

Logical Units and Partitions

A logical unit (just like a physical disk) can be divided into multiple partitions (or slices with MP-RAS). A partition is a portion of a logical unit. A partition is typically used in one of two ways.

- Used to hold the Linux file system on SMP node internal disks.
- Provides a raw data storage area (raw disk partition) that is used by Teradata.

EMC² Notes

EMC² DMX disk arrays are configured with 4-way Hyper (disk slice) Meta volumes which are seen as LUNs at the host or SMP level.

Each drive is divided into 4 equal size pieces (effectively slices within the array). 4 slices (across 4 disks) make a LUN that is presented to the operating system.

Meta volumes are used to reduce the number of LUNs and minimize registry entries in a Windows system.

Acronym: FS – File System

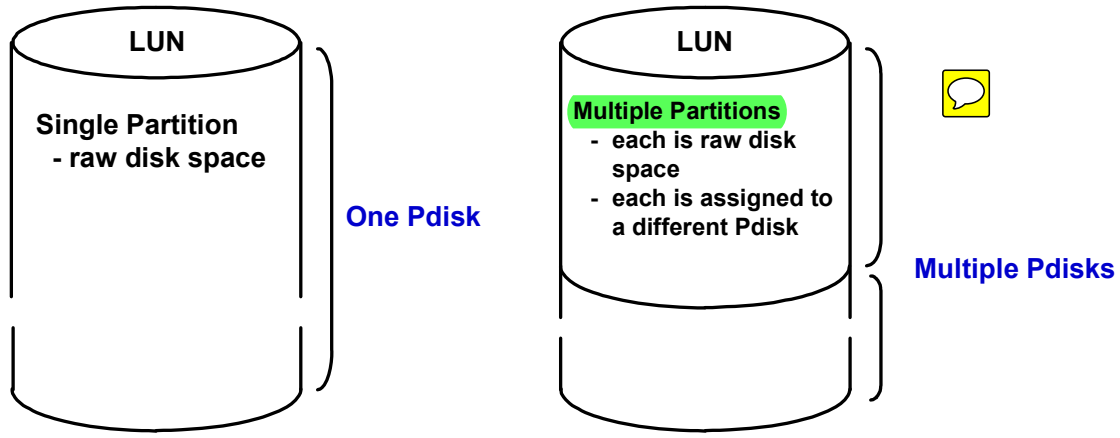
Logical Units and Partitions

With Linux, a logical unit (LUN) or Volume can be divided into one or more partitions.

- With MP-RAS systems, the portions of a LUN are referred to as slices.

How are partitions typically used by Teradata?

- Provides raw data storage area (raw disk partition) for Teradata.
- A Pdisk (Teradata) is a name that is assigned to a partition (slice) within a LUN.



Teradata and Disk Arrays

The Teradata Database has long been recognized as one of the most effective database platforms for the storage and management of very large relational databases.

The Teradata Database implementation executes as an application under the operating system. The two key pieces of software that make up the Teradata Database are the PE software and the AMP software.

Users access the Teradata Database by issuing SQL commands - usually from channel-attached hosts or LAN attached workstations. The user request is handled by *Channel Driver* or *Gateway* software and is passed to a Parsing Engine (PE) which processes the SQL request. PE software manages the user session, interprets (parses) the SQL request, creates an execution plan, and dispatches the steps of that plan to the AMP(s).

AMPs provide access to user data stored within tables that are physically stored on disk arrays.

Each AMP is associated with a Vdisk. Each AMP sees its Vdisk as a single disk. Teradata Database (AMP software) organizes its data on its disk space (Vdisk) using a Teradata Database "File System" structure. A "master index" is used to locate "cylinder indexes" which are used to locate data blocks that contain data rows.

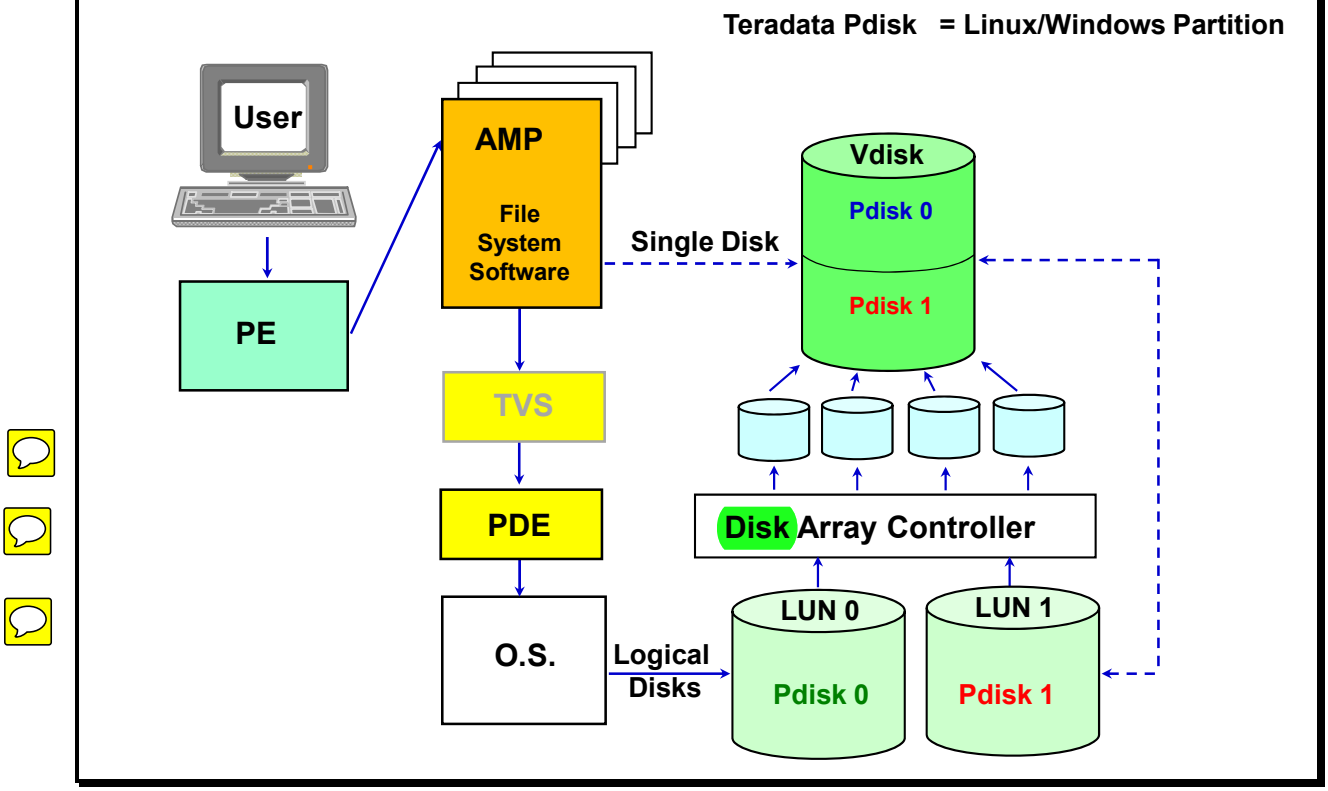
A Vdisk is actually composed of multiple slices (also called Pdisks - Physical disk) that are part of a LUN (Logical Unit) in a disk array. The operating system (e.g., Linux) and the array controllers work at the LUN level.

A logical unit (just like a physical disk) can be divided into multiple slices. A slice is a portion of a logical unit.

An AMP is assigned to a Vdisk. A Vdisk is composed of one or more Pdisks. In Linux, a Pdisk is assigned to a partition within a LUN.

The **PUT** utility is used to define a Teradata Database configuration.

Teradata and Disk Arrays



Teradata 6650 (2+1) Logical View

The facing page illustrates a logical view of the Teradata Database using a 6650 3+1 clique.

The design center configuration for a 6650H (Linux) 2+1 clique is as follows:

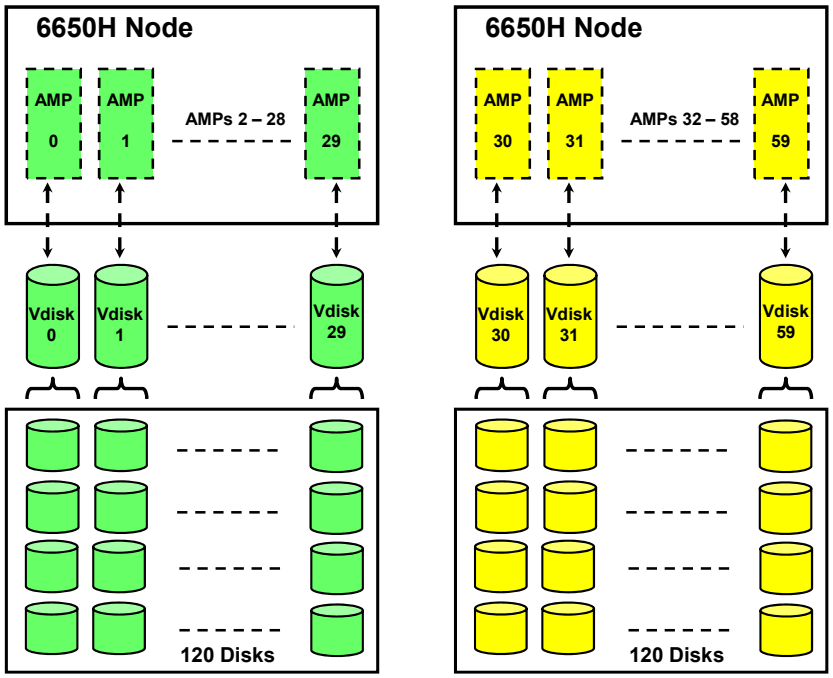
- 4 Drives per AMP
- 30 AMPs per Node

Each virtual AMP is assigned to a virtual disk (Vdisk). AMP 0 is assigned to Vdisk 0 which consists of 2 mirrored pairs of disks.

Each AMP has a Vdisk with 592,020 cylinders.

Note: The actual MaxPerm space that is available to an AMP is slightly less than the physical disk space because of file system overhead. Approximately 90 - 91% of the physical disk space is actually available as MaxPerm space.

Teradata 6650 (2+1) Logical View



Two Disk Arrays with 240 Disks – Logical View
Typical configuration is to assign each AMP with <u>two</u> mirrored pairs of disks.

Teradata 6650 (3+1) Logical View

The facing page illustrates a logical view of the Teradata Database using a 6650 3+1 clique.

The design center configuration for a 6650H (Linux) 3+1 clique is as follows:

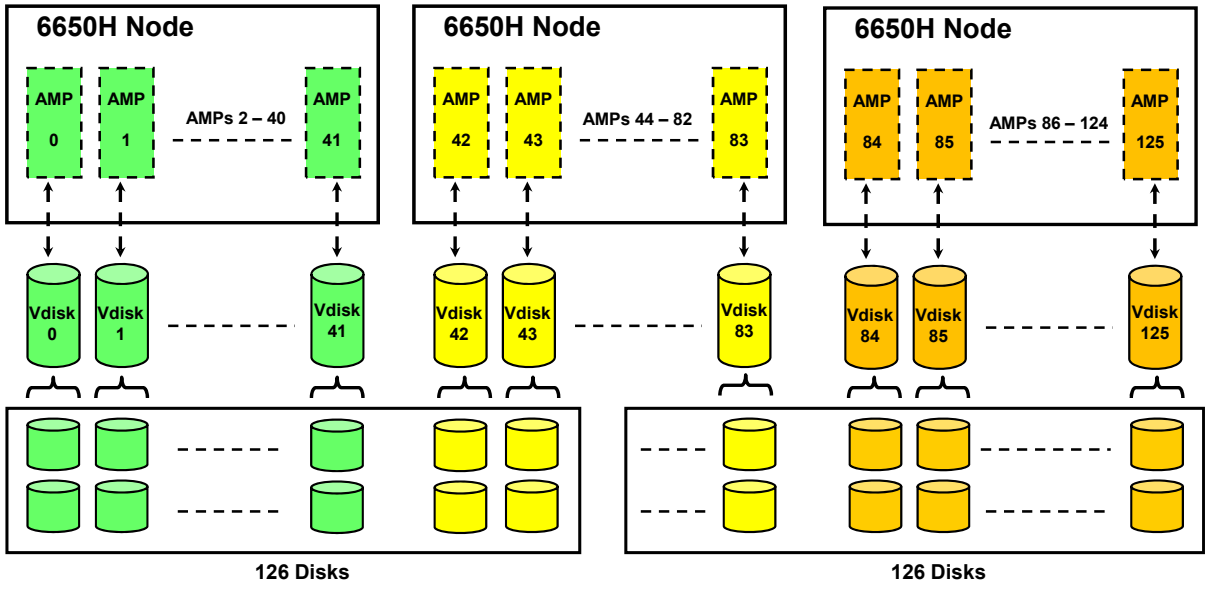
- 2 Drives per AMP
- 42 AMPs per Node

Each virtual AMP is assigned to a virtual disk (Vdisk). AMP 0 is assigned to Vdisk 0 which consists of 1 mirrored pair of disks.

Each AMP has a Vdisk with 295,922 cylinders.

Note: The actual MaxPerm space that is available to an AMP is slightly less than the physical disk space because of file system overhead. Approximately 90 - 91% of the physical disk space is actually available as MaxPerm space.

Teradata 6650 (3+1) Logical View



Two Disk Arrays with 252 Disks – Logical View
Typical configuration is to assign each AMP with one mirrored pair of disks.

Example of 1.2 TB Vdisk (pre-TVS)

A Vdisk effectively represents a set of disks in a disk array. In this example, a Vdisk represents a rank of 4 disks in a disk array that is configured to use RAID 1 technology.

If the disk array has 600 GB disks and RAID 1 protection is used, then one rank of disks (4 disks) has 1.2 TB of available disk space.

4 disks x 600 GB x .50 (parity is 50%) = 1.2 TB*

If the Vdisk is configured (assigned) with four 600 GB disks (RAID 1), then the associated AMP has 1.2 TB of perm disk space available to it.

The facing page contains a typical example of a 1.2 TB Vdisk. It would contain 592,021 cylinders; each cylinder is 3872 sectors in size. A cylinder is approximately 1.9 MB in size (3872 x 512 bytes).

With 600 GB disk drives, 592,021 cylinders are numbered from 0 to 592,020. Cylinder 0 contains control information used by the AMP and does not contain user data.

If 73 GB disk drives are used, the AMP's Vdisk will be as follows:

Total number of cylinders – 71,853
First Pdisk – 35,924 cylinders (numbered 0 through 35,923)
Second Pdisk – 35,929 cylinders (numbered 35,924 through 71,852)

If 146 GB drives are used, then the Vdisk will be as following:

Total number of cylinders – 144,482
First Pdisk – 72,237 cylinders (numbered 0 through 72,236)
Second Pdisk – 72,245 cylinders (numbered 72,237 through 144,481)

If 300 GB drives are used, then the Vdisk will be as following:

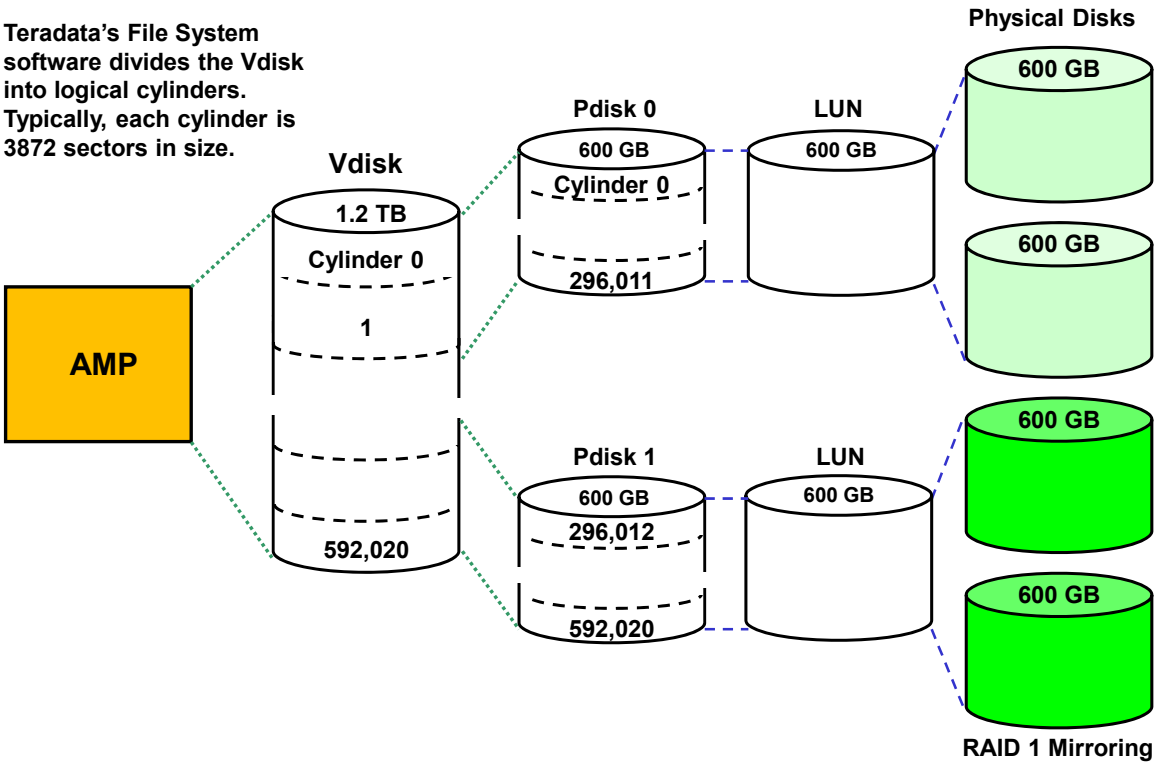
Total number of cylinders – 290,072
First Pdisk – 145,037 cylinders (numbered 0 through 145,036)
Second Pdisk – 145,035 cylinders (numbered 145,037 through 290,071)

The configuration of LUNs/partitions and the assignment Pdisks/Vdisks to AMPs is done through the **PUT** utility.

As mentioned previously, the actual space that is available to an AMP is slightly less that the numbers used above because of file system overhead. The actual MaxPerm space is approximately 90-91% of the physical disk space. In the example on the facing page, each AMP will have approximately 1080 GB of MaxPerm space, not 1200GB.

Example of 1.2 TB Vdisk (pre-TVS)

Teradata's File System software divides the Vdisk into logical cylinders. Typically, each cylinder is 3872 sectors in size.



Teradata File System Concepts

Each AMP has its own disk space managed by the Teradata Database file system software. The file system software groups physical blocks into logical cylinders.

One AMP can address/manage up to 700,000 cylinders. Each cylinder has a cylinder index (CI). Although an AMP can address this large number of cylinders, typically an AMP will only be responsible for a much smaller number of cylinders. For example, an AMP that manages 292 GB of disk space will have 144,482 cylinders.

When an AMP is initialized (booted), it reads the Cylinder Indexes and creates an in-memory Master Index to the Cylinder Indexes.

Notes:

- Teradata Database V2R5 to 13.0 – each Cylinder Index is 12 KB in size. The cylinder size is still 3872 sectors.

- Teradata uses both of the cylinder indexes as alternating cylinder indexes for write (INSERT, UPDATE, and DELETE) operations for all of the supported operating systems.

Teradata Vdisk Size Limits

For Teradata releases up to Teradata 13.0, the maximum amount of space that one AMP can access is based on the following calculation:

700,000 logical cylinders x 3872 sectors/cylinder x 512 bytes/sector

This equals 1,387,724,800,000 bytes or approximately 1.26 TB where a TB is 1024^4.

Teradata File System Concepts

The cylinder size is 3872 sectors.

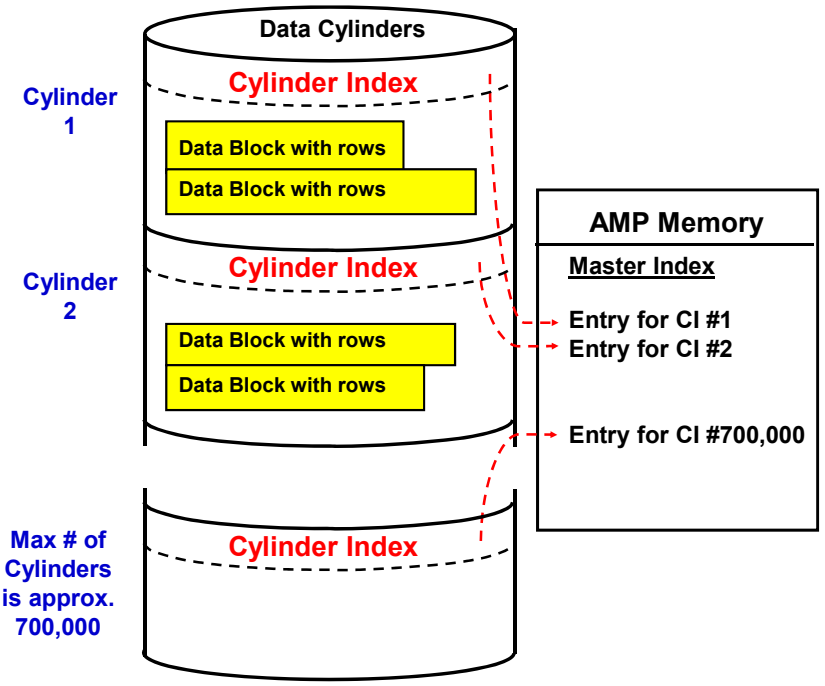
For a 1.2 TB Vdisk, there are 592,021 cylinders.

Note: However, the amount of actual MaxPerm space is approximately 90% of the actual disk space because of overhead (cylinder indexes, etc.).

MaxPerm per AMP
1.2 TB x .90 ≈ 1.08 TB

Note:
The maximum disk space an AMP can address is:

700,000 cylinders
x 3872 sectors/cylinder
x 512 bytes/sector
= 1.26 Terabytes



Size of Cylinder Index space: 24K

Teradata 13.10 Large Cylinder Support

Prior to Teradata 13.10, the maximum space available to an AMP is approximately 1.2 TB. This feature increases the maximum space available to an AMP to approximately 7 TB. Benefits of this feature are listed on the facing page.

Only systems that are newly initialized (sysinit) with Teradata 13.10 will have large cylinders enabled. Existing systems that are upgraded to 13.10 will have to be initialized (sysinit) in order to utilize large cylinders.

A cylinder contains Perm, Spool, Temporary, Permanent Journal, or WAL data, but NOT a combination. For an existing system, large cylinders results in fewer cylinders that are available for different types of data. Fewer cylinders can result in low cylinder conditions occurring more quickly and possibly more Mini-CylPacks.

If the larger cylinder size is used on an existing system where each AMP has much less space than 1.2 TB, then the number of available cylinders will be much less. For example:

- Assume a 5650 system with disk arrays populated with 600 GB drives. An AMP will typically have 4 drives assigned to it (2 sets of mirrored disks). Therefore, the AMP will have approximately 1200 GB of available space. This space is divided into approximately 592,000 cylinders.
  - Note: The actual MAXPERM space available to an AMP in this example is approximately 1080 GB (90% of 1200 GB).
- If this system is configured with large cylinders, then the system will only have approximately 99,000 cylinders. Large cylinders consume more physical space, resulting in fewer overall cylinders.

When to Use This Feature

A customer should consider enabling Large Cylinders if:

- Initial system will be sized above the current 1.2 TB per AMP limit
- It is possible that future expansion would cause the per AMP limit of 1.2 TB to be exceeded.
- If the customer anticipates the need to utilize larger row sizes (e.g., 1 MB rows) in a future release.

A customer should NOT enable Large Cylinders if:

- AMPs on the system are sized considerably less than 1.2 TB with no plans to expand beyond that limit. Large cylinders consume more physical space, resulting in fewer overall cylinders.

Teradata Large Cylinder Support

Starting with Teradata 13.10, this feature increases the maximum space available to an AMP to approximately 7.2 TB.

Benefits of this feature are:

- To utilize larger disk drives, AMPs must be to address more storage space.
  - Example, 2650 systems utilizing 2 TB disk drives
- Customers that require a large capacity of storage space have the option of increasing their storage per AMP, rather than increasing the number of AMPs.
- The maximum row size will most likely increase in future releases. Larger cylinders are more space efficient for storing large rows.
  - The maximum row size (~64 KB) is unchanged in 14.0

If large cylinders are enabled for a Teradata 13.10 or 14.0 system, then the maximum space that an AMP can access is 6 times greater or approximately 7.2 TB.

Max # of cylinders	x	#sectors in cylinder	x	sector size	
~700,000	x	23,232	x	512 bytes	= 7.2 TB

- Each cylinder index has increased to 64 KB to accommodate more blocks in a large cylinder.

Only newly initialized (sysinit) systems can have large cylinders enabled.

- Existing systems upgraded to 13.10 have to be initialized (sysinit) in order to utilize large cylinders.

Full Cylinder Read

Full Cylinder Read allows retrieval operations to run more efficiently by reading a list of cylinder-resident data blocks with a single I/O operation. This reduces I/O overhead from once per data block to once per cylinder.

A data block is a disk-resident structure that contains one or more rows from the same table and is the smallest I/O unit for the Teradata Database file system. Data blocks are stored in physical disk sectors or segments, which are grouped in cylinders.

Full Cylinder Read improves the performance of systems with both fine-grained operations and decision support workload. It eliminates the tradeoffs for short queries and concurrent updates versus strategic queries.

Performance may benefit from Full Cylinder Read during operations such as:

- Full-table scan operations under conditions such as:
- Large select
- Merge insert/select and Merge delete
- Aggregation: Sum, Average, Minimum/Maximum, Count
- Join operations that involve many data blocks, such as merge joins, product joins, and inner/outer joins

Starting with Teradata 13.10, this feature no longer needs to be tuned using a Cylinder Slots/AMP setting. This allows for more extensive use of cylinder read without the need to reserve portions of the FSG cache for Cylinder Read when Cylinder Read is not being used.

Prior to Teradata 13.10, it was necessary to specify the number of cylinder slots per AMP that would be available. The default number of cylinder slots per AMP is:

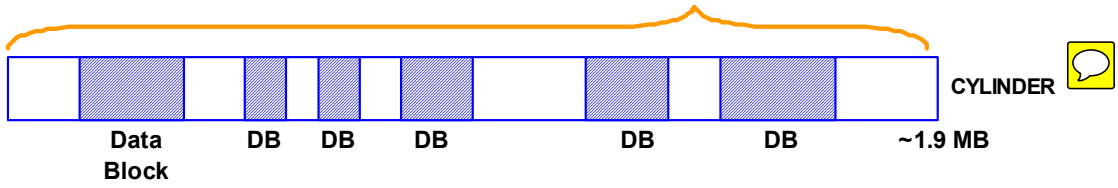
- 6 on 32-bit systems with model numbers lower than 5380.
- 6 on 32-bit coexistence systems with “older nodes.” An “older node” is a node for a system with a model number lower than 5380.
- 8 on the 32-bit systems with model numbers at 5380 or higher.
- 8 on 64-bit systems.

Teradata Database Customer Support sets the CR flag to ON and uses the ctl (control) utility to modify the number of cylslots.

Memory allocated to cylinder slots can only be used for cylinder reads. The benefit of cylinder reads is likely to outweigh the reduction in generic FSG cache.

Full Cylinder Read

The Full Cylinder Read feature allows data to be retrieved with a single cylinder (large) read, rather than individual reads of blocks.



Enables efficient use of disk & CPU performance resources for the following table scan operations under specific conditions. Examples include:

- large selects and aggregates: sum, avg, min, max, count
- joins: merge joins, product joins, inner/outer joins
- merge delete merge insert/select into empty or populated tables
- full table update/deletes

With Teradata 13.10, it is no longer necessary to specify a number of cylinder slots to make available for this feature.

- This 13.10 enhancement allows for more extensive use of cylinder reads without the need to reserve portions of the FSG cache for the Cylinder Read feature.
- Prior to 13.10, the number of cylinder slots was set using the ctl utility. The default was 8 for 64-bit operating systems.

Summary

The facing page summarizes the key points and concepts discussed in this module.

Summary

- Memory managed and used by the operating system and the vprocs is sometimes called “free memory”.
- PDE software manages FSG Cache.
  - FSG Cache is primarily used by the AMPs to access memory resident database segments.
- The operating system and Teradata does not know or care about the RAID technology being used.
- A LUN or Volume looks like a single disk to the operating system.
  - With Linux or Windows, a LUN or Volume is considered a partition and the raw partition is assigned to a Teradata Pdisk.

Module 10: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 10: Review Questions

1. Which two are placed into FSG cache?
   a. Hash maps
   b. Master Index
   c. Cylinder Indexes
   d. Permanent data blocks

2. What is the WAL Depot used for?
   a. UNDO Rows
   b. New data blocks
   c. Master Index updates
   d. Write-in-place data blocks

3. Which two are placed into the WAL Log?
   a. REDO Rows
   b. UNDO Rows
   c. New data blocks
   d. Master Index updates
   e. Write-in-place data blocks

4. Describe the fundamental relationship between Linux, logical units, and disk array controllers.

5. Describe the fundamental relationship between AMPs, Vdisks, Pdisks, Partitions, and LUNs.

Notes

Module 11

Teradata Virtual Storage

After completing this module, you will be able to:

- **List two benefits of Teradata Virtual Storage.**
- **List the two operational modes of TVS.**
- **Identify the difference between temperature and performance.**
- **Identify typical data that is identified as hot data.**

Notes

Table of Contents

Teradata Virtual Storage 11-4
Teradata Virtual Storage Concepts 11-6
  Allocation Map and Statistics Overhead 11-6
  TVAM 11-6
Teradata Virtual Storage Terminology 11-8
  Teradata Virtual Storage Components 11-8
TVS Operational Modes 11-10
Expanding Data Storage Concepts 11-12
Multi-Temperature Concepts 11-14
Storage Performance vs. Data Temperature 11-16
Teradata with Hybrid Storage 11-18
What Goes Where? 11-20
Multi-Temperature Data Example 11-22
Teradata 6690 Cabinets 11-24
  Virtualized Management Server (VMS) 11-24
HHD to SSD Drive Configurations 11-26
Summary 11-28
Module 11: Review Questions 11-30

Teradata Virtual Storage

Teradata Virtual Storage (TVS) is designed to allow the Teradata Database to make use of new storage technologies. It will allow you to store data that is accessed more frequently on faster devices and data that is accessed less frequently on slower devices. It will also allow Teradata to make use of solid state drives (SSD), for example, whenever the technology is available at a competitive price. Solid state refers to the use of semiconductor devices.

Teradata Virtual Storage is responsible for:

- pooling clique storage and allocating cylinders from the storage pool to individual AMPs
- tracking where data is stored on the physical media
- maintaining statistics on the frequency of data access and on the performance of physical storage media

These capabilities allow Teradata Virtual Storage to provide the following benefits:

- Storage optimization, data migration, and data evacuation

  Teradata Virtual Storage maintains statistics on frequency of data access ("data temperature") and on the performance ("grade") of physical media. This allows the Teradata Virtual Storage product to intelligently place more frequently accessed data on faster physical storage. As data access patterns change, Teradata Virtual Storage can move ("migrate") storage cylinders to faster or slower physical media within each clique. This can improve system performance over time.

  Teradata Virtual Storage can migrate data away from a physical storage device in order to prepare for removal or replacement of the device. This process is called "evacuation.". Complete data evacuation requires a system restart, but Teradata Virtual Storage supports a "soft evacuation" feature that allows much of the data to be moved while the system remains online. This can minimize system down time when evacuations are necessary.

- Lower Barriers to System Growth

  Device management features of Teradata Virtual Storage provide the ability to pool storage within each clique. Each storage device (pdisk) can be shared, if necessary, by all AMPs in the clique. If the number of storage devices is not a multiple of the number of AMPs in the clique, the extra storage will be shared. Consequently, storage can be added to the system in smaller increments, as needs and opportunities arise.

Teradata Virtual Storage

What is Teradata Virtual Storage (TVS)?

- **TVS (Teradata 13.0) is a change to the way in which Teradata accesses storage.**
- **Purpose is to manage a Multi-Temperature Warehouse.**
- **Pools all of the cylinders within a clique's disk space and allocates cylinders from this storage pool to individual AMPs.**

Advantages include:

- **Simplifies adding storage to existing cliques.**
  - Improved control over storage growth. You can add storage to the clique-storage-pool versus to every AMP.
  - Allows sharing of storage devices among AMPs.
- **Enables mixing drive sizes / speeds / technologies**
  - Enables the "mixing" of storage devices (e.g., spinning disks, Solid-State Disks – SSD).
- **Enables non-intrusive migration of data.**
  - The most frequently accessed data (hot data cylinders) can migrate to the high performing cylinders and infrequently accessed data (cold data cylinders) can migrate to the lower performing cylinders.

Teradata Virtual Storage Concepts

The facing page illustrates the conceptual differences with and without Teradata Virtual Storage

One of benefits of Teradata Virtual Storage is the ease of adding storage to an existing system.

Before Teradata Virtual Storage,

- Existing systems have integral number of drives / AMP
- Today adding storage requires an additional drive per AMP – means 50% or 100% increase in capacity

With Teradata Virtual Storage, you can add any number of drives.

- Added drives are shared by all AMPs
- These new disks may have different capacities and / or performance than those disks which already reside in the system.

Cylinders IDs (with TVS) are unique in the system and are 8 bytes in length as compared to 4 bytes in length before TVS (Teradata 12.0 and before).

Allocation Map and Statistics Overhead

The file system requires space on each pdisk for its allocation map and statistics areas. The number of cylinders required depends on the pdisk size as specified in the vconfig GDO.

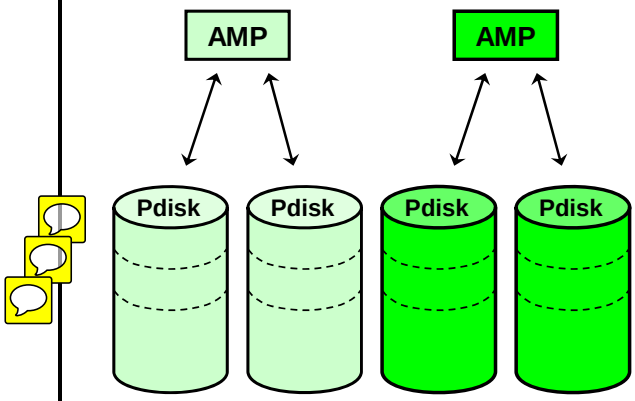
TVAM

TVAM is a support utility to control and monitor Teradata Virtual Storage. TVAM …

- Includes “-optimize” command to cause forced migration
- Includes “evacuate” and “un-join” command to enable removing a drive

Teradata Virtual Storage Concepts

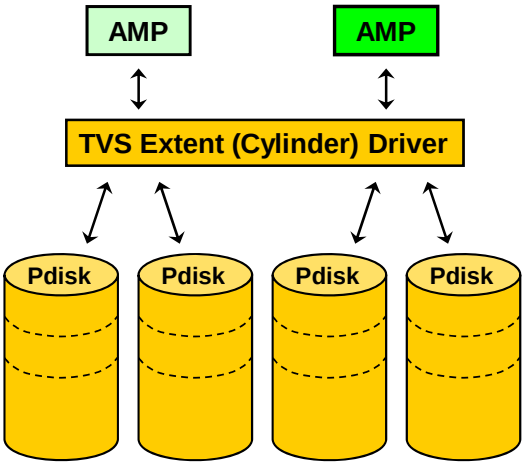
Pre-TVS – AMPs own storage



Cylinders were addressed by drive # and cylinder #.

TVS owns storage

AMPs don't know physical location of a cylinder and it can change.



All of the cylinders in clique are effectively in a pool that is managed by that TVS vproc.

Cylinders are assigned a unique cylinder id (virtual id) across all of the pdisks.

Teradata Virtual Storage Terminology

The facing page lists and defines some of the key terms used with Teradata Virtual Storage (TVS).

A subpool is a set of pdisks. There is typically one subpool per clique. Single clique systems have 2 subpools, so we can spread the AMP clusters across the subpools to achieve fallback. It is very important to understand that TVS is configured on a clique by clique basis. For a multi-clique system each clique typically has one subpool. This is where we configure the AMP clusters across the cliques. No two AMPs in the same cluster should be configured in the same clique.

TVS will take all the cylinders it finds in the clique (actually the subpool), and will divide that by the number of AMPs in the clique. This is the maximum that each AMP can allocate and is communicated back to the AMP so that it can size its master index. Each AMP can allocate cylinders as it needs cylinders up to that maximum. If some AMPs allocate more or less than other AMPs at any given time, it does not cause a problem because the space is not over-subscribed and no AMP can allocate more than its maximum.

Teradata Virtual Storage Components

1. The DBS to File System interface is unchanged.

2. The file system calls SAM (Storage Allocation Manager) at startup to obtain the list of allocated extents which can be used to rebuild or verify the MI (Master Index) and WMI (WAL Master Index). SAM also reports the maximum size of the vdisk for this AMP.

3. The file system makes calls on the SAM library interface in order to allocate and free the extents (cylinders) of storage. SAM provides an opaque handle for each allocated extent virtualizing the storage from the file system's point of view.

4. SAM sends messages to this AMP's Allocator to allocate/free the extents requested. Usually this will be on the same node as the AMP, but may require a node hop if the AMP and Allocator (part of VSS Vproc) have migrated due to a node failure.

5. The Allocator keeps the Extent Driver apprised of all virtual to physical extent mappings. Sometimes this requires a message to the node agent on another node in case of vproc migration. The Allocator keeps a copy of its translation maps on physical storage in case of node crash. It performs these I/Os directly.

6. The file system uses extent handles when communicating FSGids to FSG.

7. FSG passes the extent handle as the disk address for disk I/O to the Extent Driver.

8. The Extent Driver translates the handle to a physical extent address before passing the request to the disk driver.

Teradata Virtual Storage Terminology

TVS Software

- Consists of TVS (previously named VSS) vproc which includes **Allocator** and **Migrator** code
- Includes Extent Driver (in kernel) which interfaces between PDE and operating system

Cylinder (or Extent)

- Allocation unit of disk space (currently 3872 sectors) from TVS to an AMP

Pdisk

- Partition/slice of a physical disk; associated with a TVS vproc

Subpool

- Group of disks (effectively a set of pdisks) assigned to a TVS vproc
- Fallback clustering is across subpools; a single AMP is associated with a specific subpool
- 1 subpool/clique except for single-clique systems which have 2 subpools

Storage Performance

- TVS profiles the performance of all the disk drives (e.g., spinning disks versus SSD)
- With spinning disks, outer zones have a faster transfer rate than inner zones on a disk.

Temperature

- Frequency of access of a cylinder (Hot – Cold). TVS gathers metrics on data access frequency.

Migration

- Movement of cylinders between disks or between locations within a disk.

TVS Operational Modes

Teradata Virtual Storage operates in one of two modes:

- Teradata Traditional
  - Mimics prior Teradata releases
- Intelligent Placement (a.k.a., 1D)
  - Data temperature based placement

Teradata Traditional (TT) mode Characteristics:

When using configurations modeled with the standard interface, the TVS software is used in Teradata Traditional (TT) Mode. TT mode is available for all operating systems.

In TT mode, TVS software uses similar placement algorithms as pre-TVS Teradata software. There is no migration of hot data to fast locations in TT mode

Use Teradata Traditional mode when

- No mixing of array models in a clique ***AND***
- No mixing of disk sizes in an array ***AND***
- All Pdisks are the same size (homogeneous storage) ***AND***
- Performance capability of all Pdisks is equal (within 5%) ***AND***
- Migration is not desired ***AND***
- The number of Pdisks is an integer multiple of the AMPs in the clique. This is not a strict requirement. In this case, any fractional Pdisks will go unused in TT mode.

Intelligent Placement (1D – 1 Dimensional) mode Characteristics

Intelligent Placement is only available for Linux 64-bit operating systems.

This mode is used when any of the following are true:

- Mixing of array models in a clique
- Mixing of disk sizes in an array
- Pdisks in a clique are different sizes

When TVS software is used in Intelligent Placement (1D) Mode:

- TVS software uses advanced data placement algorithms
- Migration of hot data to fast locations and cold data to slower locations can be enabled in 1D mode.
- Use Intelligent Placement (1D) when
  - Pdisks are multiple sizes ***OR***
  - Performance capability of any Pdisks is different (> 5%) ***OR***
  - The number of Pdisks of each size in the clique is not a multiple of the number of amps in the clique ***OR***
  - Migration is desired

TVS Operational Modes

Teradata Virtual Storage (Storage Allocation) operates in one of two modes:

- **Teradata Traditional – works like prior Teradata releases**
- **Intelligent Placement (a.k.a., 1D) – data temperature based placement**

	Operational Mode	
	Teradata Traditional	Intelligent Placement
Operating System Support	All	Linux
Mixed Disk and/or Mixed Array	No	Yes
Small Growth Increments	No	Yes
Data Migration	No	Yes
Disk Evacuation	No	Yes

Note:
Evacuation is used to migrate all allocated extents (cylinders) from the selected storage to different devices. This may be done when a disk goes bad or if a disk is to be removed from the storage pool.

Expanding Data Storage Concepts

When adding **non-shared storage** to a clique on a system with Teradata Virtual Storage Services (TVS), the number of devices (Pdisks) added should be a multiple of the number of AMPs in the clique. The allocation method can be either 1D migration or Teradata Traditional (TT).

When adding **shared storage** to a clique on a system with Teradata Virtual Storage Services (TVS), the storage added will be shared among all AMPs. The allocation method is 1D migration only.

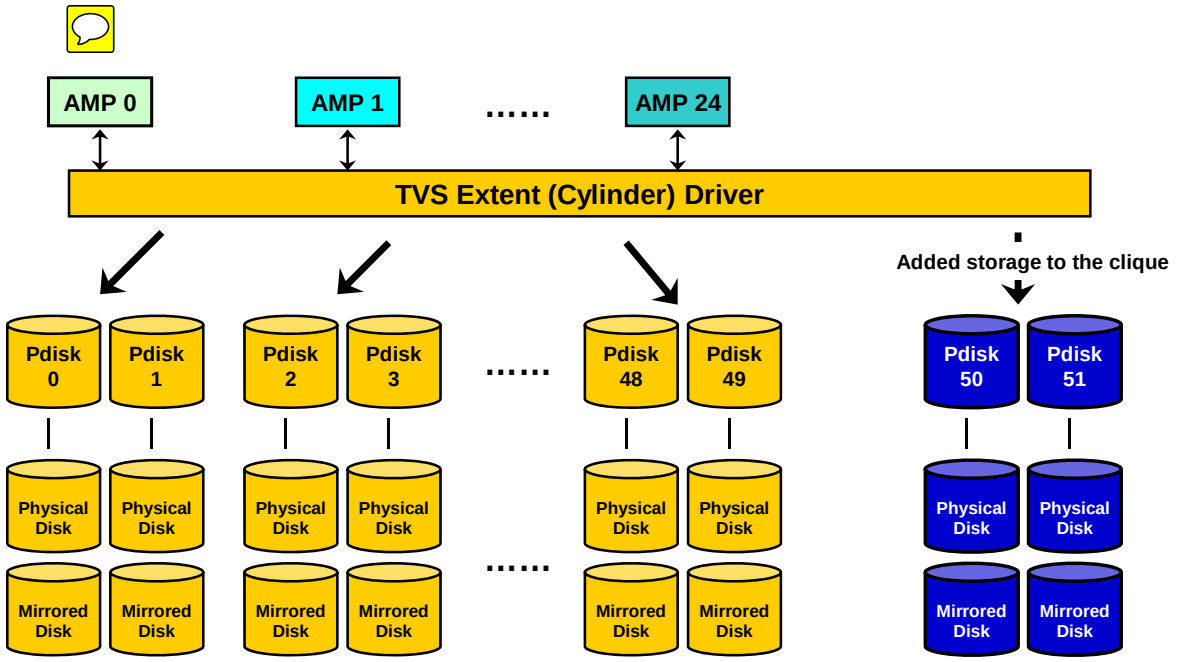
In addition to utilizing the existing storage capacity more efficiently with temperature based placement, TVS simplifies the ability to alter the storage capacity of a Teradata system. As previously mentioned, database generations prior to Teradata Database 13.0 typically allocated the entire capacity of each drive in the system to a single AMP. That design parameter, coupled with the need for each AMP to have identical amounts of available storage meant that system-wide storage capacity could only be increased by adding enough new disks (of the same size) to provide each AMP in the system with an entire new drive. Thus, a system with 100 AMPs would typically have a minimum increment of growth of 100 drives (actually 200 drives using RAID-1) which have identical performance and capacities to the existing drives.

With TVS, storage capacity can be added in a much more flexible manner. Instead of requiring each drive to be dedicated to a single AMP, TVS can subdivide a device into equivalent groups of cylinders which can then be allocated to each AMP, allowing even single drive pairs to be equally shared by all of the AMPs in a clique. This “fine grained virtualization” enables the growth of storage capacity using devices with differing capacities and speeds. For multi-clique or co-existence systems, new storage capacity would still have to be added to each clique in an amount that is proportional to the number of AMPs in each clique.

Expanding Data Storage Concepts

Storage can be added to a clique and is shared between all the AMPs within the clique.

Expanded storage within a clique is treated as "cold storage".



Multi-Temperature Concepts

The facing page identifies two key areas associated with Multi-Temperature environments.

With today's disk technology, the user experiences faster data transfer with data that is located on the outer zones of a disk drive. This is because there is more data accessed per disk revolution. Data located on the inner zones experience slower data transfer because more disk revolutions are needed to access the same amount of data.

Teradata Virtual Storage can track data temperature over time and can move data to appropriate region.

A Multi-Temperature Warehouse has the ability to prioritize the use of system resources based on business rules while maximizing utilization of storage with ever increasing capacity

Teradata Virtual Storage enhances performance with multi-temperature data placement.

Multi-Temperature Concepts

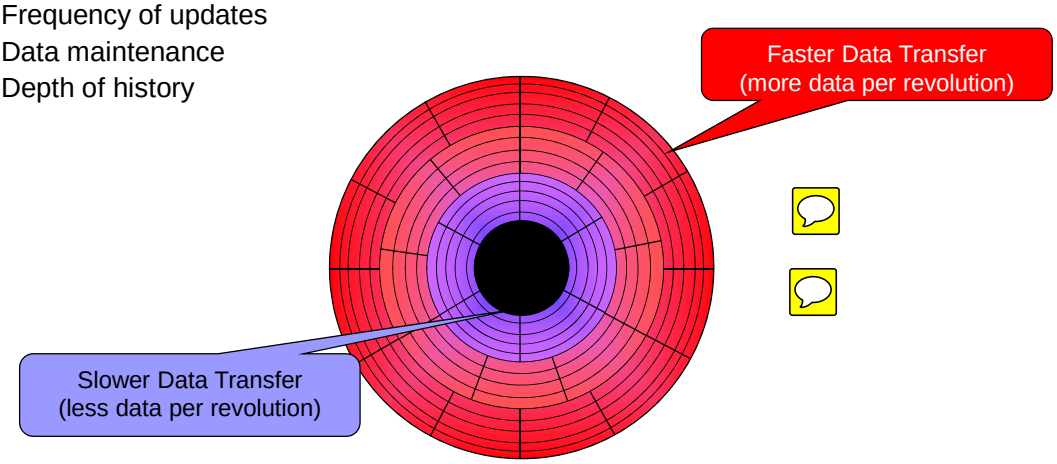
Two related concepts for Multi-Temperature Data:

✓ **Performance of Storage**

- Differences between different drives on different controllers (spinning disk vs. SSD)
- Differences between different areas within a drive
  - Outer zones on a disk have fastest transfer rates.

✓ **Data Access Pattern (Frequency) or Temperature is determined by:**

- Frequency of access
- Frequency of updates
- Data maintenance
- Depth of history



Storage Performance vs. Data Temperature

For the purposes of describing the performance characteristics of storage devices, we'll use terms like "fast", "medium" and "slow" to describe the relative response times (grade) of the physical cylinders that comprise each device. The important thing to keep in mind is that temperatures (hot, warm, and cold) refer to data access and grade (fast, medium, slow) refer to the speed of physical devices.

PUT executes the TVS Profiler on one node per clique. This is done during initial install while the system is essentially idle. The TVS Profiler measures and records the cylinder response times of one disk of each size (i.e., 146 GB, 300 GB, 450 GB, 600 GB, etc.) and type (i.e., Cheetah-5). These metrics are then used for all disks of the same size and type in the clique throughout the life of the system. This can also be done using the TVAM utility.

Data temperature values are maintained by the TVS Allocator vproc. They are viewable at an extent level via the new TVAM (Teradata Virtual Administration Menu) command and at a system, AMP, and table level via the Ferret utility SHOW command.

The data temperatures are calculated as a function of both frequency of access and "recency" of access and are measured relative to the temperatures of the rest of the user data stored in the warehouse.

This concept of "recency" is important because it allows the data temperatures to cool off as time passes so that even if a table or partition has a lot of historical access, the temperature of that data appears lower than data that has the same amount of access during a more recent time period. This trait of data becoming cooler over time is commonly referred to as data aging. But just because data is older doesn't imply that it will only continue to get cooler. In fact, cooler data can become warm/hot again as access increases. For example, sales data from this quarter may remain hot until several months in the future as current quarter/previous quarter comparisons are run in different areas of the business. After 6 months, that sales data may cool off until nearly a year later when it becomes increasingly queried (i.e. becomes hotter) by comparison against the new current quarter's sales data.

Teradata Virtual Storage enhances performance with multi-temperature data placement.

Storage Performance vs. Data Temperature

Storage Performance relative response times – (e.g., fast, medium, slow).

- Profiles the performance of all the disk drives (e.g., SSD versus spinning disks)
- Identifies performance zones (usually 10) on each spinning disk drive

Data Access Frequency – referred to as "Data Temperature" (e.g., hot, warm, cold).

- TVS records information about data access (called Profiling and Metric Collection)
  - How long it takes to access data (I/O response times)
  - How often data is accessed (effectively the data temperature)

TVS places data for optimal access based upon storage performance, type of data (WAL, Depot, Spool, etc.) and the results of metric collection.

- Initial Data Placement
- Migration of data based upon data temperature

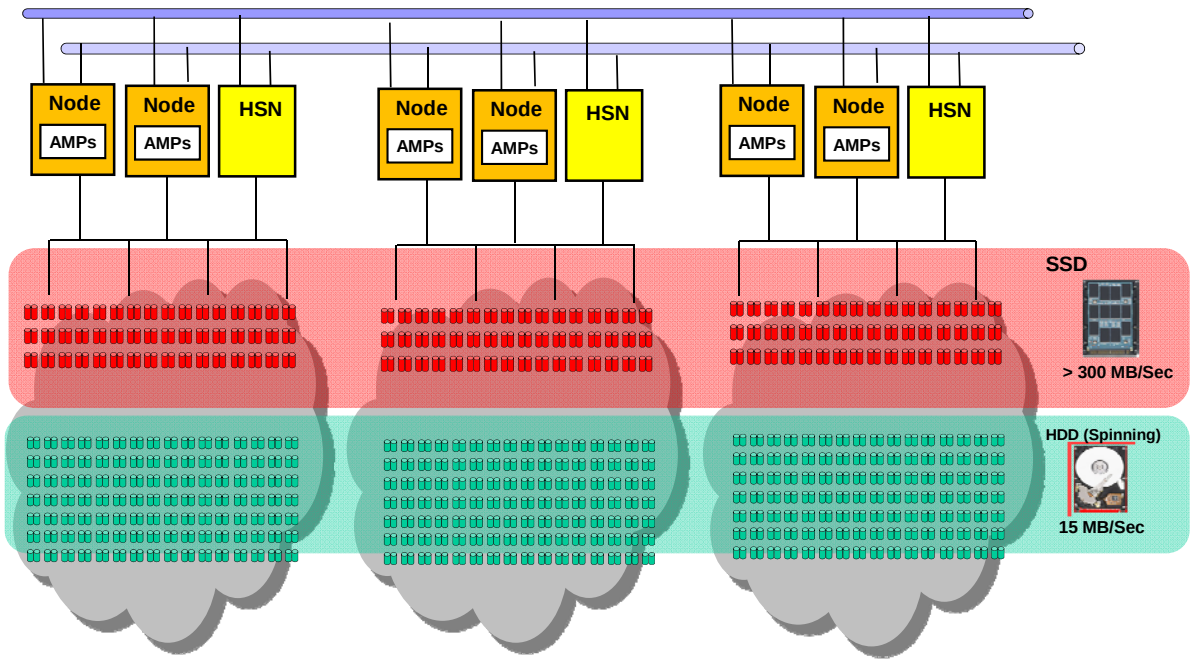
Three types of Data Migration:

- Background Process During Queries – moves 10% of data in about a one week
- Optimize Storage Command (Database Off-Hours) - moves 10% of data in about eight hours
  - Ignores other work – just runs “flat out”
- Anticipatory Migration to Make Room in Fast Reserve, Fast or Warm Storage for Hotter Data (when needed)

Teradata with Hybrid Storage

The facing page illustrates an example of a Teradata system with both HDD and SDD drives.

Teradata with Hybrid Storage



What Goes Where?

Virtualization is the Key to Managing Multi-Temperature Data.

- TVS “knows” the performance characteristics of the different storage devices
- TVS virtualizes location information for the database and stores the translation information in its meta-data
- TVS collects usage information and adds it to the meta-data
- TVS uses these metrics to assign a temperature to each cylinder (extent) of data
  - Each time cylinder is accessed it heats up
  - Over time all cylinders cool off
- TVS migrates data from one performance zone to another as appropriate to match temperature to storage performance

Initial Data Placement

- Based on several factors and controls
- File System indicates to TVS expected temperature of the data, and its use (permanent tables, spool, WAL, other temp tables)
- TVS allocates from appropriate performance device
  - SSDs for hot data
  - Inside cylinders of HDD for cold data
  - All the rest of HDD for all else (warm)

Initial Data Temperature for Permanent Data

- All new cylinders are assigned an initial temperature
- Defaults for each type are specified in DBSControl
- When loading into empty tables, if don’t want the default then temperature can be specified by the user via Querybanding the Session doing the loading
- When adding data to existing tables, new data assumes the temperature of the data already in the table at the location it is inserted.
  - Possible to forcibly change temperature of existing table or part of table via Ferret – this is not a recommended management tool
  - Changing temperature does not move data, just make it subject to normal migration
  - Over time, temperature will return to ambient

What Goes Where?

Migration is done at the Cylinder level.

Depot, WAL, and Spool cylinders are allocated as HOT

- 20% of Fast storage (SSD) is reserved for this Depot, WAL, and Spool.
- This region is called the Fast Reserve.
  - This does not limit the total amount of WAL or Spool.
- When Fast Reserve is full, use Fast or even Medium for WAL and Spool allocations.
- These cylinders types are not subject to “cooling off”, their temperature is static.

Loading perm data into an empty table defaults to HOT

- This is assumed to be a short-lived staging table.
- If not, this default can be changed with DBSControl.
- Another option is to specify Initial Data Temperature with Query Band.
  - **SET QUERY_BAND = 'TVSTemperature=COLD;' UPDATE FOR SESSION;**

Note: The UPDATE option is used so that this query band statement will not completely replace, but rather supplement, any other query band name:value pairs already specified for the session.

Multi-Temperature Data Example

The facing page illustrates an example of using a Multi-Temperature Warehouse.

Example of Multi-Temperature with a PPI Table:

- If this is time based (e.g., DATE), then rows of the table are physically grouped by DATE and the groups ordered by DATE, even though hash ordered within the partition for each DATE value.
- Because the rows are logically grouped together, they reside in a set of cylinders
- Based on usage patterns, all the cylinders of a partition will have same temperature.
- As usage drops, partition cools off, eventually its cylinders get migrated out of FAST to MEDIUM, then eventually to SLOW storage.
- Newly loaded partition will assume temperature of previous latest (probably HOT).

While TVS can monitor data temperatures, it can't change or manipulate the temperature of your data because data temperatures are primarily dictated by the workloads that are run on the warehouse. That is, the more queries that are run against a particular table (or tables) the higher its temperature(s). The only way to change a table's temperature is to alter the number of queries that are run against it.

For technical accuracy, TVS temperature is measured at a cylinder level not a data level.

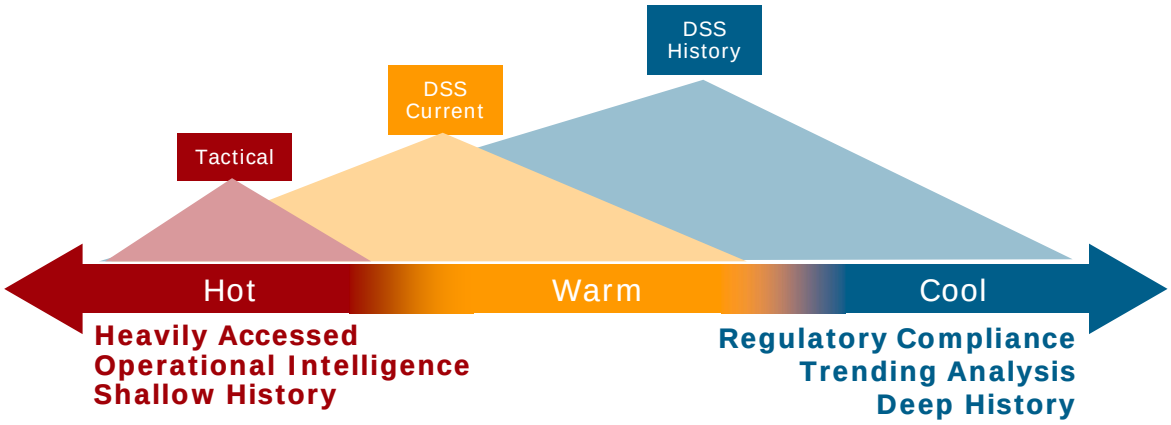
The facing page illustrates the result of data migration with Teradata Virtual Storage.

Teradata Virtual Storage enables you to more easily, more cost effectively, mix data with different levels of importance on the same system

Advantages of Teradata Virtual Storage:

- Allows incremental growth of storage
- Provides lower cost method for adding "Cold" data to Enterprise Warehouse w/o Performance Penalty to Bread and Butter Workload
- Enhances multi-generation co-existence

Multi-Temperature Data Example



Hybrid Storage Data Usage Model

The closer this model fits your data, the more useful the Hybrid system will be.

Teradata 6690 Cabinets

Each Teradata 6690 cabinets can be configured in a 1+1 or 2+1 clique configuration.

- A processing/storage cabinet contains one clique.
- A cabinet with a 2+1 clique contains two processing nodes, one hot standby node, and four disk arrays.
- A cabinet with a 1+1 clique contains one processing node, one hot standby node, and four disk arrays.

Virtualized Management Server (VMS)

The VMS is available with the 6690 Enterprise Warehouse Server.

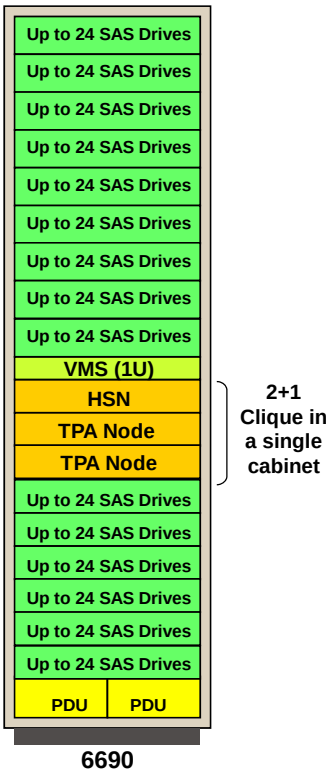
Characteristics of the VMS include:

- 1U Server that VIRTUALIZES system and cabinet management software onto a single server
- **Teradata System VMS** – provides complete system management functionality
  - Cabinet Management Interface Controller (CMIC)
  - Service Workstation (SWS)
  - Teradata Viewpoint (single system only)
  - **Automatically installed on base/first cabinet**
- The VMS allows full rack solutions without an additional cabinet for traditional Viewpoint and SWS
- Eliminates need for expansion racks reducing customers' floor space and energy costs
- **For multi-system monitoring and management traditional Teradata Viewpoint is required.**

Teradata 6690 Cabinets

6690 Characteristics

- Integrated Cabinet with nodes and SSD and HDD arrays in same cabinet.
- Each NetApp Drive Tray can hold up to 24 SSD and/or HDD drives.
  - SSD drives are 400 GB.
  - HDD drives (10K RPM) are 600 GB.
  - Possible maximum of 360 disks in the cabinet.
- The primary requirement for planning a 6690 system is the completion of the Data Temperature assessment.
- There is a range of configurations to meet the requirements of most customers' data temperature assessments.



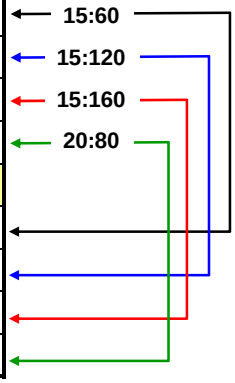
HHD to SSD Drive Configurations

The facing page lists possible hybrid system configurations.

HHD to SSD Drive Configurations

- **There are four preset HDD to SSD configurations (ratios of SSD:HDD per node) which vary slightly between 1+1 and 2+1 cliques.**

Solid State Devices (SSD)	Hard Disk Drives (HDD)
# of SSD per Clique/per Node	# of HDD per Clique/per Node
1+1 Configuration	
16*	60
16*	120
18*	160
20	80
2+1 Configuration	
30/15	120/60
30/15	240/120
30/15	320/160
40/20	160/80



- **PUT requires specific and even SSD numbers in a clique, thus the difference between a 1+1 and 2+1 disks per node (16 or 18 vs. 15). 18 includes GHS drives.**
- **6690 nodes are typically configured with 30 AMPs per node.**

Summary

The facing page summarizes the key points and concepts discussed in this module.

Summary

- TVS is a change to the way in which Teradata accesses storage.
- Advantages include:
  - Simplifies adding storage to existing cliques
  - Enables mixing drive sizes / speeds / technologies
  - Enables non-intrusive migration of data
- Purpose is to manage a Multi-Temperature Warehouse.
- Two related concepts for Multi-Temperature Data
  - Performance of Storage
  - Data Access Pattern – Frequency
- Pools all of the cylinders within a clique's disk space and allocates cylinders from this storage pool to individual AMPs.

Module 11: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 11: Review Questions

1. List two capabilities of using Teradata Virtual Storage.

2. List the two operational modes of Teradata Virtual Storage.

3. Which choice is associated with data temperature?
   a. Skewed data
   b. Frequency of access
   c. Solid State Disk Drives
   d. Inner tracks versus outer tracks on a spinning disk

4. Which data level is migrated from hot to cold storage?
   a. Row
   b. Block
   c. Cylinder
   d. Subtable

5. Which two types of data are typically considered to be HOT data?
   a. WAL
   b. DBC tables
   c. Spool data
   d. History data

Notes

Module 12

Physical Database Design Overview

After completing this module, you should be able to:

- **Understand the stages of database design.**
- **List and describe the input requirements for database design.**
- **List and describe the outputs and objectives for database design.**
- **Describe the differences between a Logical, Extended, and Physical Data Model.**

Notes

Table of Contents

The Stages of Database Development....12-4
Example of Data Model – ER Diagram....12-6
Customer Service Logical Model....12-8
Relational Terms Review....12-10
Domains....12-12
Attributes....12-14
Entities and Relationships....12-16
Decomposable Data....12-18
Normal Forms....12-20
Normalization....12-22
Normalization Example....12-24
Denormalizations....12-34
Derived Data....12-36
Pre-Joins....12-38
Exercise 1: Choose Indexes....12-40
Tables Index Selection....12-42
Database Design Components....12-44
Extended Logical Data Model....12-46
Physical Data Model....12-48
The Principles of Index Selection....12-50
Transactions and Parallel Processing....12-52
Module 12: Review Questions....12-54

The Stages of Database Development

Four core stages are identified as being relevant to any database design task. They are:

- **Requirement Analysis** involves eliciting the initial set of information and processing requirements from users.

- **Logical Modeling** determines the contents of a database independent of a particular physical implementation's exigencies.

  – **Conceptual Modeling** transforms the user requirements into a number of individual user views normally expressed as entity-relationship diagrams.

  – **View Integration** combines these individual user views into a single global schema expressed as key tables. The logical model is implemented by taking the conceptual model as input and transforming it into the data model supporting the target relational database management system (RDBMS). The result is the relational data model.

- **Activity Modeling** determines the volume, usage, frequency, and integrity analysis of a database. This process also consists of placing any constraints on domains and entities in addition to addressing any legal and ethical issues including referential integrity.

- **Physical Modeling** transforms the logical model into a definition of the physical model suitable for a specific software and hardware configuration. In relational terms, this is usually some schema expressed in a dialect of the data definition language of SQL.

Outputs from these stages are shown on the right of the facing page and are as follows:

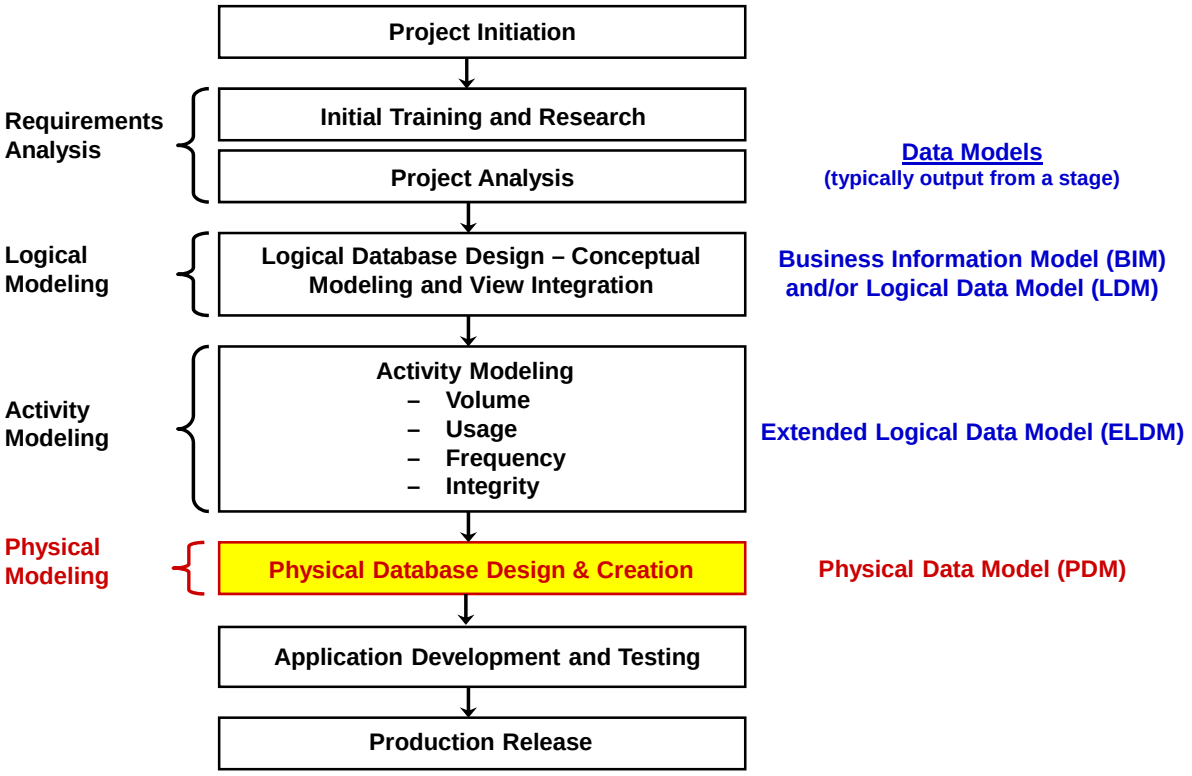
- Business Information Model (BIM)
  – shows major entities and their relationships
  – also referred to as "Business Model"
  – BIM acronym – also used for "Business Impact Model"

- Logical Data Model (LDM) - should be in Third Normal Form (3NF)
  – BIM plus all tables, minor entities, PK – FK relationships
  – constraints and attributes (columns)

- Extended Logical Data Model (ELDM)
  – LDM plus demographics and frequencies

- Physical Data Model (PDM)
  – ELDM plus index selections and any denormalizations

The Stages of Database Development

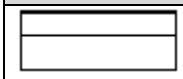
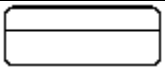
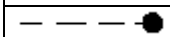
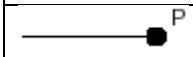
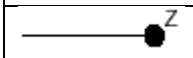
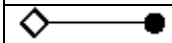
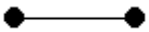
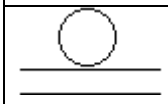
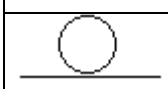


Example of Data Model – ER Diagram

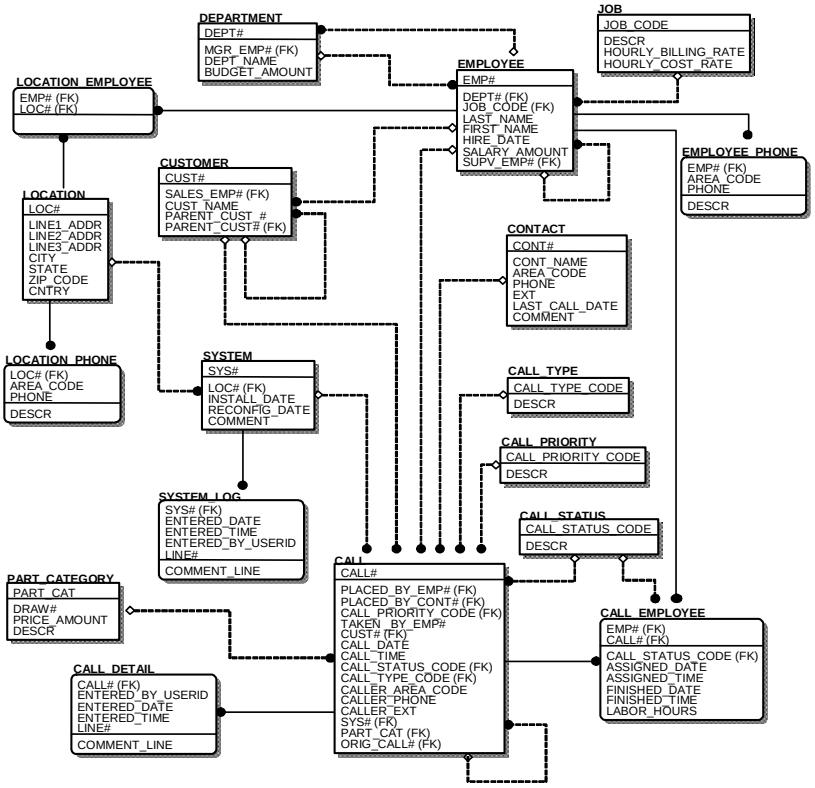
The Customer Service database is designed to handle information pertaining to phone calls by customers to Customer Service employees. The CALL table is the central table in this database. On the facing page is the **Entity-Relationship (E-R) diagram** of the Customer Service database. This type of model depicts entities and the relationships between them. The E-R diagram provides you with a high-level perspective.

ERD Convention Overview

The following conventions are generally used in ER diagramming. Symbols in this module are consistent with the ERwin data modeling tool's conventions.

Convention	Example
	Independent entity. An independent entity does not depend on another entity for its identification. It should have a single-column PK. PK attribute appears above the horizontal line.
	Dependent entity. A dependent entity depends on one or more other entities for its identification. It generally has multiple columns in its PK, one or more of which is also an FK. All PK attributes appear above the horizontal line.
(FK)	A Foreign Key. An attribute in the entity that is the PK in another, closely related entity. FK columns are shown above or below the horizontal dividing line in all entities, depending on the nature of the relationship. For 1:1 and 1:M relationships, their FKs are below the horizontal line. For M:M relationships the FKs participating in the PK are above the horizontal line.
	One-to-Zero, One, or Many occurrences (1:0-1-M). Solid lines indicate a relationship (join path) between two entities. The dot identifies the child end of a parent-child relationship between two entities.
	The dotted line indicates that the child does not depend on the parent for identification.
	One-to-At least One or More occurrences (1:1-M)
	One-to-Zero, or at most One occurrence (1:0-1)
	Zero or One-to-Zero, One, or Many occurrences (0-1:0-1-M). The diamond shape on the originating end indicates the relationship is optional. Physically, this means that a NULL value can exist for an occurrence of any row of the entity positioned at the terminating end (filled dot) of the relationship.
	Many-to-Many occurrences (M:M). A many-to-many relationship, also called a non-specific relationship, represents a situation where an instance in one entity relates to one or more instances in a second entity and an instance in the second entity also relates to one or more instances in the first entity.
	Indicates that each parent instance must participate in one and only one sub-type as shown in the LDM.
	Indicates that parent instances may or may not participate in one of the sub-types as shown in the LDM.

Example of Data Model – ER Diagram



Customer Service Logical Model

While the E-R diagram (previous page) was very helpful, it lacked the detail necessary for broad user acceptance. How many columns are in the CALL table? What is the Primary Key (PK) of the CALL table?

The logical model of the Customer Service database is depicted on the facing page. It shows many more table-level details than the E-R diagram does. You can see the individual column names for every table. In addition, there are codes to indicate PKs and Foreign Keys (FKs), as well as columns which are System Assigned (SA) or which allow No NULLS (NN) or No Duplicates (ND). Sample data values are also depicted.

This is the type of model that comes about as a result of Relational Data Modeling. This example most closely represents a “Logical Data Model” or LDM.

TERADATA
Raising Intelligence

Customer Service Logical Model
(ERA Methodology Diagram)

CALL

CALL#	TAKEN BY EMP#	CUST#	PLACED BY CONT#	PLACED BY EMP#	ORIG CALL#	CALL DATE	CALL TIME	CALL STATUS CODE	CALL TYPE CODE	CALL PRIORITY CODE	CALLER AREA CODE	CALLER PHONE	CALLER EXT	SYS#	PART CAT
PK,SA	FK,NN	FK	FK	FK	FK	NN	NN	FK,NN	FK,NN	FK,NN				FK	FK
1	1002	4		1004		030215	0905	1	H	4				1	5

CALL DETAIL

CALL#	ENTERED BY USERID	ENTERED DATE	ENTERED TIME	LINE#	COMMENT LINE
PK					
FK					
1	LJC	030215	1625	1	When the

CALL PRIORITY

CALL PRIORITY CODE	DESCR
PK	NN,ND
1	TOP

CALL STATUS

CALL STATUS CODE	DESCR
PK	NN,ND
1	OPEN

SYSTEM

SYS#	LOC#	INSTALL DATE	RECONFIG DATE	COMMENT
PK	FK,NN			
547	1	030212		

CALL EMPLOYEE

CALL#	EMP#	CALL STATUS CODE	ASSIGNED DATE	ASSIGNED TIME	FINISHED DATE	FINISHED TIME	LABOR HOURS
PK		FK,NN,NC	NN	NN			
FK	FK						
1	1004	1	891215	0905	891215	1625	8.5

EMPLOYEE PHONE

EMP#	AREA CODE	PHONE	DESCR
PK			
FK			
1001	415	1234567	

LOCATION PHONE

LOC#	AREA CODE	PHONE	DESCR
PK			NN
FK			
27	415	1234567	OFFICE

LOC EMP

LOC#	EMP#
PK	
FK	FK
1	1001

CONTACT

CONT#	CONT NAME	AREA CODE	PHONE	EXT	LAST CALL DATE	COMMENT
PK	NN	NN			NN	
8010	CSB	408	7654321		030321	

CUSTOMER

CUST#	CUST NAME	PARENT CUST#	SALES EMP#
PK	NN,ND	FK	FK,NN
4	TDAT	3	1023

CALL TYPE

CALL TYPE CODE	DESCR
PK	NN,ND
H	HDWR

PART CATEGORY

PART CAT	DRAW#	PRICE AMT	DESCR
PK	NN,ND	NN	
1	A7Z348	1.27	CLIP

DEPARTMENT

DEPT#	DEPT NAME	BUDGET AMOUNT	MGR EMP#
PK	NN,ND		FK,NN
403	EDUC	932000	1005

EMPLOYEE

EMP#	SUPV EMP#	DEPT#	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SALARY AMOUNT
PK,SA	FK	FK	FK	NN				
1001	1003	401	412101	NOMAR	JOE	890114	450824	50000.00

SYSTEM LOG

SYS#	ENTERED DATE	ENTERED TIME	ENTERED BY USERID	LINE#	COMMENT LINE
PK					
FK					
547	030212	1738	LJC	1	We added

JOB

JOB CODE	DESCR	HOURLY BILLING RATE	HOURLY COST RATE
PK	NN,ND		
412101	F.E.		

LOCATION

LOC#	CUST#	LINE1 ADDR	LINE2 ADDR	LINE3 ADDR	CITY	STATE	ZIP CODE	CNTRY
PK,SA	FK,NN	NN			NN	NN	NN	
1	4	100 N.			ATLANTA	GA	30096	USA

Relational Terms Review

Relational theory uses the terms Relations, Tuples, and Attributes. Most people are more comfortable with the terms Tables, Rows, and Columns.

Additional Relational terminology (such as Domains) will be discussed more completely on the following pages.

Acronyms:

PK – Primary Key
FK – Foreign Key
SA – System Assigned
UA – User Assigned
NN – No NULLS
ND – No Duplicates
NC – No Changes

It is very important that you start with a well-documented relational model in Third Normal Form. This model is used as the basis for an ELDM.

Knowledge of the hardware and software environment is crucial to doing physical database design for Teradata. The final PDM should be optimized for site-specific implementation. It is also crucial that you think in terms of the large volume of data that is usually stored in Teradata databases. When working with such large-scale databases, extraneous I/Os can have a great impact on performance.

By understanding how the Teradata Database System works, you can make constructive physical design decisions that have a positive impact on your system's performance.

Relational Terms Review

Operational File Systems	Relational Theory	Logical Models & RDBMS systems
File	Relation	Entity or Table
Record	Tuple	Row
Field	Attribute	Column

Table A two-dimensional representation of data composed of rows and columns.

Row One occurrence in a relational table – a record.

Column The smallest category of data in the model – a field or attribute.

Domain The definition of a pool of valid values from which column values are drawn.

EMPLOYEE

EMP#	LAST NAME	FIRST NAME	MI	NETWORK ID
PK, SA	NN	NN		FK, ND, NN
01029821	Smith	John	A	JS129101

Domains

The following statements are true for domains and their administration in relational database management systems:

- A domain defines the SET of all possible valid values, which may appear in all columns based within that domain.
- A domain value is a fundamental non-decomposable unit of data.
- A domain must have a domain name and a domain data type. Valid domain data types are:

INTEGER	Any integer value
DECIMAL	Whole and fractional values
CHARACTER	Alpha-numeric values
DATE	Valid Gregorian calendar dates
TIME	24 hour notation
BIT STRING	Digitized data (e.g. photos, x-rays)

Domain Values

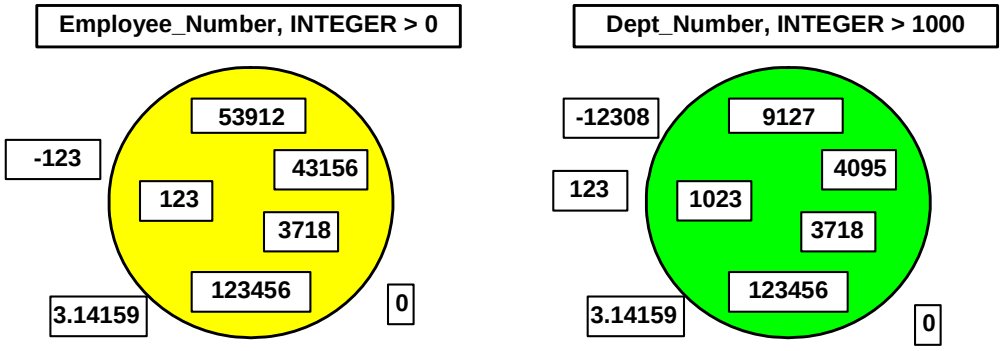
A domain defines the conceptual SET, or range, of all valid values that may appear in any column based upon that domain.

Sometimes domains are restricted to specific values. For example:

- Would you ever want negative employee numbers?
- Has there ever been, or will there ever be, an employee with the employee number of ZERO?

Domains

Domain – the definition of a pool of valid values from which column values are drawn.



Question: Does an Employee_Number of 3718 and a Dept_Number of 3718 represent the same value?

Attributes

Types of Attributes

The types of attributes include:

- Primary Key (PK): Uniquely identifies each row in a table
- Foreign Key (FK): Identifies the relationship between tables
- **Non-Key Attributes:** All other attributes that are not part of any key. They are descriptive only, and do not define uniqueness (PK) or relationship (FK).
- **Derived Attributes:** An attribute whose value can be calculated or otherwise derived from other existing attributes. Example: NetPay is derived by calculating GrossPay - TaxAmt.

Derived Attribute Issues

The attributes from which derived attributes are calculated are in the design, so carrying the derived attribute in addition creates redundant data. Derived attributes may be identified and defined in order to validate that the model can in fact deduce them, but they are not shown in the ER Diagram, because carrying redundant data goes against relational design theory and principles.

There are several good reasons to avoid carrying redundant data:

- The data must be maintained in two places, which involves extra work, time and expense.
- There is a risk (likelihood) of the copies getting out of sync with each other, causing data inconsistency.
- It takes more physical storage.

Attributes

Types of Attributes

- **Primary Key (PK):** Uniquely identifies each row in a table
- **Foreign Key (FK):** Identifies the relationship between tables
- **Non-Key Attributes:** All other attributes that are not part of any key. They are descriptive only, and do not define uniqueness (PK) or relationship (FK).
- **Derived Attributes:** An attribute whose value can be calculated or otherwise derived from other existing attributes.

  Example: Count of current employees in a department. A SUM of Employee table meets this requirement.

Derived Attribute Issues

- Carrying a derived attribute creates redundant data.
- Reasons to avoid carrying redundant data:
  - The data must be maintained in two places which possibly causes data inconsistency.
  - It takes more physical storage

Entities and Relationships

The entities and their relationships are shown in table form on the facing page. The naming convention used for the tables and columns makes it easy to find the PK of any FK.

Acronyms:

PK – Primary Key
FK – Foreign Key
SA – System Assigned
UA – User Assigned
NN – No NULLS
ND – No Duplicates
NC – No Changes

Relationship Descriptions

Many-to-many relationships are usually implemented by an associative table (e.g., Order_Part table).

Examples of relationships are shown below.

1:1 and 1:M Relationships

(PK)		(FK)
Country	Has	LOCATIONs
Customer	Has	LOCATIONs
Employee	Generates	ORDERs
	Generates	SHIPMENTs
	Receives	SHIPMENTs
Location	Generates	ORDERs
	Generates	SHIPMENTs
	Receives	SHIPMENTs
	Has Individual	PARTs
Order	Requisitions individual	PARTs

M : M Relationships

Order/Part Category Show kinds of PARTs on an ORDER before it is filled (Direct.)

Order/Shipment Shows which PARTs belong to which ORDERs and SHIPMENTs after the ORDER is filled (INDIRECT).

Entities and Relationships

There are three types of relationships: 1:1 1:M M:M

1:1 Relationships are rare.

Ex. One employee has only one Network ID and a Network ID is only assigned to one Employee.

EMPLOYEE

EMPLOYEE NUMBER	EMPLOYEE L_NAME	NETWORK ID
PK, SA		FK, ND, NN
30547	SMITH	BS100421
21289	NOMAR	JN450824

NETWORK_USERS

NETWORK ID	VIRTUAL FLAG	SecurID
PK, UA		ND
BS100421		231885
JN450824	Y	348145

1:M and M:M Relationships are common.

CUSTOMER

CUST ID	CUST NAME	CUST ADDRESS
PK, SA		
1001	MALONEY	100 Brown St.
1002	JONES	12 Main St.

Examples:

1:M – A Customer can place many orders.

M:M – An Order can have many parts on it. The same part can be on many Orders. An "associative" table is used to resolve M:M relationships.

ORDER

ORDER #	ORDER DATE	CUST ID
PK, SA		
		FK, NN
1	2005-12-24	1001
2	2006-01-23	1002
3	2006-02-07	1001

ORDER_ITEM

ORDER #	ITEM ID	ITEM QTY
PK		
FK	FK	
1	6001	3
1	6200	1
2	6001	5

ITEM

ITEM ID	ITEM DESC	RETAIL PRICE
PK		
6001	Paper	15.00
6200	Printer	300.00

Decomposable Data

Data may be either **decomposable** or **atomic**. Decomposable data can be broken down into finer, smaller units while atomic data is already at its finest level.

There is a Relational Rule that "Domains must not be decomposable." If you normalize your relational design and create your tables based on domains, you will have columns that do not contain decomposable data.

In practice, you may have columns that contain decomposable data. This will not cause excessive problems if those columns are not used for access. You should create a column for any individual character or number that is used for access.

A good example of decomposable data is a person's name:

- Name can be broken down into last name and first name.
- Last name and first name are good examples of atomic data since they really can't be broken down into meaningful finer units.

There are several benefits to designing your system in this manner. You should get increased performance because there will be fewer Full Table Scans due to partial value index searches. Also, if the columns are NUSI's, you will increase the chance of using NUSI Bit Mapping. Finally, you will simplify the coding of your SQL queries.

Remember that storage and display are separate issues.

Decomposable Data

RELATIONAL RULE: Domains must not be decomposable.

- Atomic level data should be defined.
- Continue to normalize through the lifetime of the system.
- Columns with multiple domains should be decomposed to the finest level of ANY access.
- Create a column for an individual character or number if it is used for access.
- Storage and display are separate issues.

The GOAL:

- Eliminate FTS (Full Table Scans) on partial value index searches.
- Simplify SQL coding.

Normal Forms

Normalization is a set of rules and a methodology for making sure that the attributes in a design are carried in the correct entity to map accurately to reality, eliminate data redundancy and minimize update anomalies.

Stated simply: ***One Fact, One Place!***

- 1NF, 2NF and 3NF are progressively more refined and apply to non-key attributes regarding their dependency on PK attributes.

- 4NF and 5NF apply to dependencies between or among PK attributes.

For most models, normalizing to 3NF meets the business requirements.

Normalization provides a rigorous, relational theory based way to identify and eliminate most data problems:

- Provides precise identification of unique data values

- Creates data structures which have no anomalies for access and maintenance functions

Later in the module, we will discuss the impact of denormalizing a model and the effect it may have (good or bad) on performance.

By implementing a model that is in Third Normal Form (3NF), you might gain the following Teradata advantages.

- Usually more tables – therefore, more primary index choices

  – Possibly fewer full table scans
  – More Data control

- Fewer Columns per Row – usually smaller rows

  – Better user isolation from the data
  – Better application separation from the data
  – Better blocking
  – Less transient and permanent journaling space

These advantages will be discussed in Physical Design and Implementation portion of this course.

Normal Forms

Once you've identified the attributes, the question is which ones belong in which entities?

- **A non-key attribute should be placed in *only one* entity.**
- **This process of placing attributes in the correct entities is called *normalization*.**

First Normal Form (1NF)

- **Attributes must not repeat within a table. No repeating groups.**

Second Normal Form (2NF)

- **An attribute must relate to the entire Primary Key, not just a portion.**
- **Tables with a single column Primary Key (entities) are always in Second Normal form.**

Third Normal Form (3NF)

- **Attributes must relate to the Primary Key and not to each other.**
- **Cover up the PK and the remaining attributes must not describe each other.**

Normalization

The facing page illustrates violations of First, Second and Third Normal Form.

First Normal Form (1NF)

The rule for 1NF is that attributes must not repeat within a table. 1NF also requires that each row has a unique identifier (PK). In the violation example, there are six columns representing sales amount.

Second Normal Form (2NF)

The rule for 2NF is that attributes must describe the entire Primary Key, not just a portion. In the violation example, the ORDER DATE column describes only the ORDER portion of the Primary Key.

Third Normal Form (3NF)

The rule for 3NF is that attributes must describe only the Primary Key and not each other. In the violation example, the JOB DESCRIPTION column describes only the JOB CODE column and not the EMPLOYEE NUMBER (Primary Key) column.

Fourth (4NF) and Fifth (5NF) Normal Forms

4NF and 5NF are covered here only for your information. The vast majority of models never apply these levels. Essentially these Normal Forms are designed to impose the same level of consistency within a PK composed of more than two columns as the first 3NFs impose on attributes outside the PK.

Entities with more than two columns in the PK often contain no non-key attributes. If non-key attributes do exist, 4NF and 5NF violations are unlikely because bringing the model into 3NF compliance precludes them.

Usually 4NF and 5NF violations occur when the definition of the information to be represented is ambiguous (e.g. the user has either not really understood what they are asking for, or they have failed to state it clearly enough for the designer to understand it). 4NF and 5NF really represent two flip sides of the same issue: The PK must contain the minimum number of attributes that accurately describe all of the business rules.

Formal Definitions:

4NF: The entity's PK represents a single multi-valued fact that requires all PK attributes be present for proper representation. Attributes of a multi-valued dependency are functionally dependent on each other.

5NF: The entity represents, in its key, a single multi-valued fact and has no unresolved symmetric constraints. A 4NF entity is also in 5NF if no symmetric constraints exist.

Normalization

Normalization is a technique for placing non-key attributes in tables in order to:

- Minimize redundancy
- Provide optimum flexibility
- Eliminate update anomalies

First Normal Form (1NF)

attributes must not repeat within a table.

SALES HISTORY

EMP NUMBER	FIGURES FOR LAST SIX MONTHS					
	SALES	SALES	SALES	SALES	SALES	SALES
PK, SA						
2518	32389	21405	18200	27590	29785	35710

Second Normal Form (2NF)

attributes must describe the entire Primary Key, not just a portion.

ORDER PART

ORDER NUMBER	PART NUMBER	ORDER DATE	QUANTITY
PK			
FK	FK		
100	1234	2005-02-15	200
100	2537	2005-02-15	100

Third Normal Form (3NF)

attributes must describe only the Primary Key and not each other.

EMPLOYEE

EMPLOYEE NUMBER	EMPLOYEE NAME	JOB CODE	JOB DESCRIPTION
PK, SA		FK	
30547	SMITH	9038	INSTRUCTOR
21289	NOMAR	9038	INSTRUCTOR

Normalization Example

The facing page contains an illustration of a simple order form that a customer may use.

It is possible to simply convert this data file into a relational table, but it would not be in Third Normal Form.

Dr. Codd Mnemonic

Every non-key attribute in an entity must depend on:

The KEY	- 1st Normal Form (1NF)
The WHOLE key	- 2nd Normal Form (2NF)
And NOTHING BUT the Key	- 3rd Normal Form (3NF)

-- E.F. Codd

Normalization Example

One of the order forms a customer uses is shown below.

Order # _____ Order Date _____

Customer ID _____
Customer Name _____
Customer Address _____
Customer City _____ State _____ Zip _____

Item ID	Item Description	Item Price	Item Quantity	Item(s) Total Price
_____	_____	_____	_____	_____
_____	_____	_____	_____	_____
_____	_____	_____	_____	_____
_____	_____	_____	_____	_____
			Order Total	_____

A listing of the fields is:

Order #
Order Date
Customer ID
Customer Name
Customer Address
Customer City
State
Zip

Repeats
- **Item ID**
- **Item Description**
- **Item Price**
- **Item Quantity**
- **Item(s) Total Price**

Order Total

Normalization Example (cont.)

The tables on the facing page represent the normalization to 1NF for the previous order form example.

Recall that the rule for 1NF is that attributes must not repeat within a table.

Negative effects of violating 1NF include:

- Places artificial limits on the number of repeating items (attributes)
- Sorting on the attribute becomes very difficult
- Searching for a particular value of the attribute is more complex

Normalization Example (cont.)

A modeler chooses to remove the repeating groups and creates two tables as shown below.

Order Table	Order-Item Table
Order # ←	Order #
Order Date	Item ID
Customer ID	Item Description
Customer Name	Item Price
Customer Address	Item Quantity
Customer City	Item(s) Total Price
State	
Zip	
Order Total	

This places the data in first normal form.

Normalization Example (cont.)

The tables on the facing page represent the normalization to 2NF for the previous order form example.

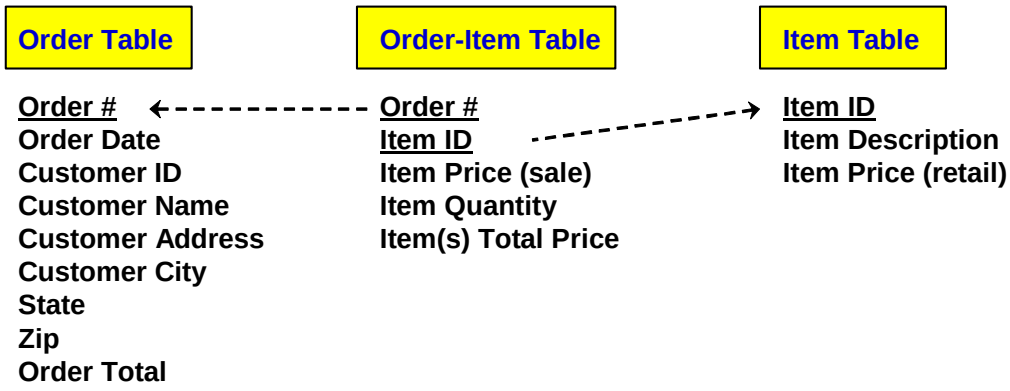
Recall that the rule for 2NF is that attributes must describe the entire Primary Key, not just a portion.

Negative effects of violating 2NF include:

- More disk space may be used
- Redundancy is introduced
- Updating is more difficult
- Can also comprise the integrity of the data model

Normalization Example (cont.)

A modeler checks that attributes describe the entire Primary Key.



This places the data in second normal form.

As an option, the item price may be kept at the Order-Item level in the event a discount or different price is given for the order. The Item table may identify the retail price.

The Order Total and Item(s) Total Price are derived data and may or may not be included.

Normalization Example (cont.)

The tables on the facing page represent the normalization to 3NF for the previous order form example.

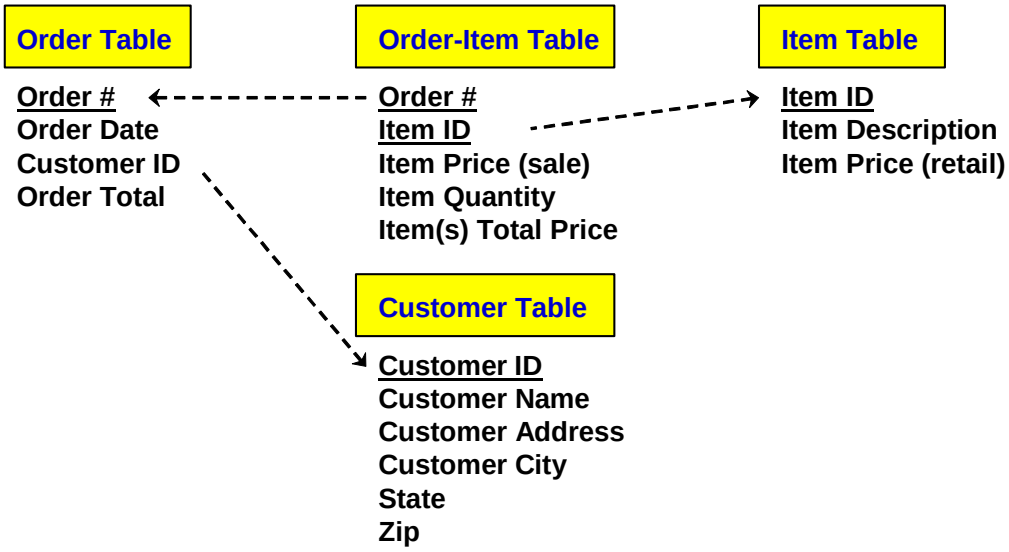
Recall that the rule for 3NF is that attributes must describe only the Primary Key and not each other.

Negative effects of violating 3NF include:

- More disk space may be used
- Redundancy is introduced
- Updating is more costly

Normalization Example (cont.)

A modeler checks that attributes only describe the Primary Key.



These tables are now in third normal form. If the item sale price is always the same as the retail price, then the item price only needs to be kept in the item table.

The Order Total and Item(s) Total Price are derived data and may or may not be included.

Normalization Example (cont.)

The facing page completes this example and illustrates the tables in a logical format showing PK-FK relationships.

Normalization Example (cont.)

The tables are shown below in 3NF with PK-FK relationships.

ORDER

ORDER #	ORDER DATE	CUSTOMER ID
PK, SA		
		FK
1	2005-02-27	1001
2	2005-04-24	1002

CUSTOMER

CUST ID	CUST NAME	CUST ADDRESS	CUST CITY	CUST STATE	CUST ZIP
PK, SA					
1001	MALONEY	100 Brown St.	Dayton	OH	45479
1002	JONES	12 Main St.	San Diego	CA	92127

ORDER_ITEM

ORDER #	ITEM ID	SALE PRICE	ITEM QUANTITY
PK			
FK	FK		
1	5001	15.00	2
1	5002	300.00	1
2	5001	15.00	1

ITEM

ITEM ID	ITEM DESCRIPTION	RETAIL PRICE
PK		
5001	PS20 Electric Pencil Sharpener	15.00
5002	MFC140 Multi-Function Printer	300.00

Note that Items Total Price & Order_Total are not shown in this model.

How are Items Total Price & Order_Total handled?

Denormalizations

This course recommends that the corporate database tables that represent the company's business be maintained in Third Normal Form (3NF). Due to the large volume of data normally stored in a Teradata system, denormalization may be necessary to improve performance. If you do denormalize, make sure that you are aware of all the trade-offs involved.

It is also recommended that, whenever possible, you keep the normalized tables from the Logical Model as an authoritative source and add additional denormalized tables to the database. This module will cover the various types of denormalizations that you may choose to use. They are:

- Derived Data
- Repeating Groups
- Pre-Joins
- Summary and/or Temporary Tables
- Partitioning (Horizontal or Vertical)

Complete the Logical Model before choosing to use these denormalizations.

There are a few costs in normalizing your data. Typically, the advantages of having a data model in 3NF **outweigh** the costs of normalizing your data.

Costs of normalizing to 1NF include:

- you use more disk space
- you have to do more joins

Costs of normalizing to 2NF when already in 1NF include:

- you have to do more joins

Costs of normalizing to 3NF when already in 2NF include:

- you have to do more joins

A customer may choose to implement a semantic layer between the data tables and the end users. The simplest definition of a semantic layer is as the view layer that uses business terminology and does presentation.

The semantic layer can also be viewed as a logical construct to support a presentation layer which may interface directly with some end-user access methodology. The "semantic layer" may change column names, derive new column values, perform aggregation, or whatever else the presentation layer needed to support the users.

Denormalizations

Denormalize only when all of the trade-offs of the following are known. Examples of denormalizations are:

- **Derived data**
- **Pre-Joins**
- **Repeating groups**
- **Partitioning (Horizontal or Vertical)**
- **Summary and/or Temporary tables**

Make these choices AFTER completing the Logical Model.

- **Keep the Logical Model pure.**
- **Keep the documentation of the physical model up-to-date.**

Denormalization may increase or decrease system costs.

- **It may be positive for some applications and negative for others.**
- **It generally makes new applications harder to implement.**
- **Any denormalization automatically reduces data flexibility.**
- **It introduces the potential for data problems (anomalies).**
- **It usually increases programming cost and complexity.**

Note: Only a few denormalization examples are included in this module. Other techniques will be discussed throughout the course.

Derived Data

Attributes whose values can be determined or calculated from other data are known as Derived Data. Derived Data can be either integral or stand-alone, examples of which are shown on the facing page. You should notice that integral Derived Data requires no additional I/O and no denormalization. Stand-alone Derived Data, on the other hand, requires additional I/O and may require denormalization.

Creating temporary tables to hold Derived Data is a good strategy when the Derived Data will be used frequently and is stable.

Handling Derived Data

Storing Derived Data is a normalization violation that breaks the rule against redundant data. Whenever you have stand-alone Derived Data, you must decide whether to calculate it or store it. This decision should be based on the following demographics:

- number of tables and rows involved
- access frequency
- column data value volatility and column data value change schedule

All above demographics are determined through Activity Modeling – also referred to as Application and Transaction Modeling. The following table gives you guidelines on what approach to take depending on the value of the demographics.

Guidelines apply when you have a large number of tables and rows. In cases where you have a small number of tables and rows, calculate the Derived Data on demand.

Access Frequency	**Change Rating**	**Update Frequency**	**Recommended Approach**
High	High	Dynamic	Denormalize the model or use Temporary Table
High	High	Scheduled	Use Temporary Table or produce batch report
High	Low	Dynamic	Use Temporary Table
High	Low	Scheduled	Use Temporary Table or produce batch report
Low	?	?	Calculate on demand

Note that, in general, using summary/temporary tables is preferable to denormalization.

The example on the facing page shows an example of using a derived data column (Employee Count) to identify the number of employees in a department.

This count can be determined by doing a count of employees from the Employee table.

Derived Data

Derived data is an attribute whose value can be determined or calculated from other data. Storing a derived item is a denormalization (redundant data).

Normalized

DEPARTMENT

DEPT NUM	DEPT NAME
PK, SA	NN, ND
UPI	
1001	ENGINEERING
1002	EDUCATION

EMPLOYEE

EMPLOYEE NUMBER	EMPLOYEE NAME	DEPT NUM
PK, SA	NN	FK
UPI		
22416	JONES	1002
30547	SMITH	1001
82455	NOMAR	1002
17435	NECHES	1001
23451	MILLER	1002

Carrying the count of the number of employees in a department is a normal forms violation. The number of employees can be determined from the Employee table.

Denormalized

DEPARTMENT

DEPT NUM	DEPT NAME	EMPLOYEE COUNT
PK, SA	NN, ND	Derived Data
UPI		
1001	ENGINEERING	2
1002	EDUCATION	3

EMPLOYEE

EMPLOYEE NUMBER	EMPLOYEE NAME	DEPT NUM
PK, SA	NN	FK
UPI		
22416	JONES	1002
30547	SMITH	1001
82455	NOMAR	1002
17435	NECHES	1001
23451	MILLER	1002

Pre-Joins

Pre-Joins can be created in order to eliminate Joins to small, static tables (Minor Entities). The example on the facing page shows a Pre-Join table that contains columns from both the JOB and EMPLOYEE tables above it.

Although this is a violation of Third Normal Form, there are several reasons that you may want to use it:

- It is a good performance technique for the Teradata DBS especially when there are known queries.
- It is a good way to handle situations where you have tables with fewer rows than AMPs.
- You still have your original Minor Entity to maintain data consistency and avoid anomalies.

Costs of pre-joins include:

- Additional space is required
- More maintenance and I/Os are required.

Pre-Joins

To eliminate joins to a small table (possibly static), consider including their attribute(s) in the parent table.

NORMALIZED

JOB

JOB CODE	JOB DESCRIPTION
PK, SA	NN, ND
UPI	
1015	PROGRAMMER
1023	ANALYST

EMPLOYEE

EMPLOYEE NUMBER	EMPLOYEE NAME	JOB CODE
PK, SA		FK
UPI		
22416	JONES	1023
30547	SMITH	1015

DENORMALIZED

EMPLOYEE

EMPLOYEE NUMBER	EMPLOYEE NAME	JOB CODE	JOB DESCRIPTION
PK, SA			
UPI			
22416	JONES	1023	ANALYST
30547	SMITH	1015	PROGRAMMER

Reasons you may want Pre-Joins:

- Performance technique when there are known queries.
- Option to handle situations where you have tables with fewer rows than AMPs.

A Join Index (Teradata feature covered later) provides a way of creating a "pre-join table". As the base tables are updated, the Join Index is updated automatically.

Exercise 1: Choose Indexes

At right is the EMPLOYEE table from the CUSTOMER_SERVICE database. The legend below explains the abbreviations you see below the column names. The following pages contain fifteen more PTS tables.

Choose the best indexes for these tables. Remember, you must choose exactly one Primary Index per table, but you may choose up to 32 Secondary Indexes.

Primary Keys do not have to be declared. Any Primary Key which is declared must have all columns of the PK defined as NOT NULL, and will be implemented by Teradata as a Unique index (UPI or USI).

– REMEMBER –

The Primary Key is the logical reference for the Logical Data Model. The Primary Index is the physical access mechanism for the Physical Data Model. They may be but will not always be the same.

Exercise 1: Choose Indexes

The next page contains a portion of the logical model of the PTS database.

Indicate the candidate index choices for all of the tables. An example is shown below.

The Teradata database supports four index types:

UPI (Unique Primary Index)
NUPI (Non-Unique Primary Index)
USI (Unique Secondary Index)
NUSI (Non-Unique Secondary Index)

EMPLOYEE

50,000 Rows	EMP#	SUPV EMP#	DEPT#	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SAL AMT
PK/FK	PK,SA	FK	FK	FK	NN	NN	NN	NN	NN
PI/SI	UPI	NUSI	NUSI	NUSI	NUSI				

LEGEND

PK = Primary Key (implies NC, ND, NN)
FK = Foreign Key
NC = No Change
SA = System Assigned Value
ND = No Duplicates
UA = User Assigned Value
NN = No Nulls

Tables Index Selection

On the facing page, you will find some of the tables in the PTS database.

Choose the best indexes for these tables. Remember that you must choose exactly one Primary Index per table, but you may choose up to 32 Secondary Indexes.

Tables Index Selection

LOCATION

	LOC#	CUST#	LINE1 ADDR	LINE2 ADDR	LINE3 ADDR	CITY	STATE	ZIP	CNTRY
PK/FK	PK,SA	FK,NN	NN			NN			
PI/SI									

ORDER

	ORD#	CUST#	LOC#	ORD DATE	CLOSE DATE	UPD DATE	UPD TIME	UPD USER	
PK/FK	PK,SA	FK,NN	FK,NN	NN		SA,NN	SA,NN	FK,NN	
PI/SI									

PART

	PART#	PART CAT	SER#	LOC#	SYS#	SHIP#	ORD#	STAT	UPD DATE
PK/FK	PK,SA	FK,NN	FK,NN	FK,NN		FK,NN	FK,NN	SA,NN	
PI/SI									

Database Design Components

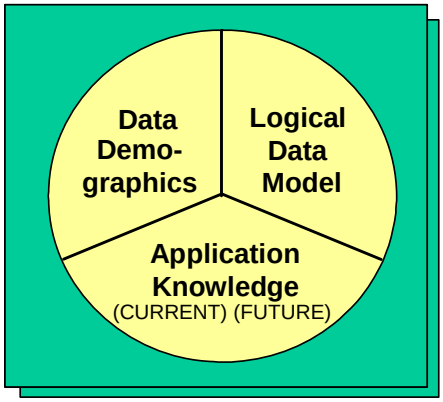
Each System Development Phase adds to the design. As we mentioned earlier, they are:

- Logical Data Modeling
- Extended Data Modeling (also known as Application and Transaction Modeling; we will call it Activity Modeling).
- Physical Data Modeling

First and foremost, make sure the system is designed as a function of business usage and not the reverse.

Let usage drive design.

Database Design Components



- A good logical model reduces application workload.
- Thorough application knowledge produces dependable demographics.
- Proper demographics are needed to make sound index choices.
- Though you don't know users' access patterns, you will need that information in the future. For example, management may want to know why there are two copies of data.
- For DSS, OLAP, and Data Warehouse systems, aim for even distribution and let Teradata parallel architecture handle the changing access needs of the users.

Extended Logical Data Model

At right is the **Extended Logical Data Model (ELDM)**, which includes data demographic information pertaining to data distribution, sizing and access.

Information provided by the ELDM results from user input about transactions and transaction rates.

The Delete Rules and Constraint Numbers (from a user-generated list) are provided as an aid to application programmers, but have no effect on physical modeling.

The meaning and importance of the other ELDM data to physical database design will be covered in coming modules of this course.

Extended Logical Data Model

EXTENDED LOGICAL DATA MODEL

- **It provides demographics of data distribution, sizing and access.**
- **It maps applications and transactions to the related tables, columns and row sets.**
- **It is the main information source for creating the physical data model**

TABLE NAME: Employee

DESCRIPTION: Someone who works for our company and on payroll.

ROW COUNT: 50,000 **TABLE TYPE:** Entity

EMPLOYEE

	EMPLOYEE NUMBER	SUPERVISOR EMPLOYEE NUMBER	DEPARTMENT NUMBER	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SALARY AMOUNT	
PK/FK	PK, SA	FK	FK	FK	NN					
DEL RULES		N	N	P						
CONSTR#	101	101								
VALUE ACC FREQ	10K	0	8K	1K	200	0	0	0	0	
JOIN ACC FREQ	17K	50	12K	6K	0	0	0	0	0	
JOIN ACC ROWS	136K	10K	96K	50K	0	0	0	0	0	
DISTINCT VALUES	50K	7K	2K	3K	40K	NA	NA	NA	NA	
MAXIMUM ROWS/VAL	1	30	40	4K	2K	NA	NA	NA	NA	
MAX ROWS NULL	0	1	18	40	0	NA	NA	NA	NA	
TYPICAL ROWS/VAL	1	7	23	15	1	NA	NA	NA	NA	
CHANGE RATING	0	3	2	4	1	NA	NA	NA	NA	
SAMPLE DATA	8326	647	2431	18	OZ	WIZ				

Physical Data Model

The model at right is the **Physical Data Model (PDM)**, which contains the same information as the ELDM except that index selections and other physical design choices such as data protection mechanisms (e.g., Fallback) have been added.

A complete PDM will define all tables, indexes and views to be implemented. Due to physical design considerations, the PDM may differ from the logical model. In general, the more the PDM differs from the logical model, the less flexible it is and the more programming it requires.

Physical Data Model

PHYSICAL DATA MODEL

- **A collection of DBMS constructs that define the tables, indexes and views to be implemented.**
- **The main tables represent the entities of the business function.**
- **It may differ from the logical model due to implementation issues.**
- **The more it differs, the less flexible it is and the more programming it requires.**

TABLE NAME: Employee

DESCRIPTION: Someone who works for our company and on payroll.

FALLBACK: YES **PERM JRNL:** NO **IMPLEMENTATION:** 3NF

ROW COUNT: 50,000 **TABLE TYPE:** Entity

EMPLOYEE

	EMPLOYEE NUMBER	SUPERVISOR EMPLOYEE NUMBER	DEPARTMENT NUMBER	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SALARY AMOUNT	
PK/FK	PK, SA	FK	FK	FK	NN					
DEL RULES		N	N	P						
CONSTR#	101	101								
VALUE ACC FREQ	10K	0	8K	1K	200	0	0	0	0	
JOIN ACC FREQ	17K	50	12K	6K	0	0	0	0	0	
JOIN ACC ROWS	136K	10K	96K	50K	0	0	0	0	0	
DISTINCT VALUES	50K	7K	2K	3K	40K	NA	NA	NA	NA	
MAXIMUM ROWS/VAL	1	30	40	4K	2K	NA	NA	NA	NA	
MAX ROWS NULL	0	1	18	40	0	NA	NA	NA	NA	
TYPICAL ROWS/VAL	1	7	23	15	1	NA	NA	NA	NA	
CHANGE RATING	0	3	2	4	1	NA	NA	NA	NA	
PI/SI	**UPI**		**NUSI**	**NUSI**						
SAMPLE DATA	8326	647	2431	18	OZ	WIZ				

The Principles of Index Selection

The right-hand page illustrates the many factors that impact Index selection. As you can see, they represent all three of the Database Design Components (Logical Data Model, Data Demographics and Application Knowledge).

Index selection can be summarized as follows:

- Start with a well-documented 3NF logical model.
- Develop demographics to create the ELDM.
- Make index selections based upon these demographics.

The Principles of Index Selection

There are many factors which guide the designer in choosing indexes:

- The way the system uses the index.
- The space the index requires.
- The table type.
- The number of rows in the table.
- The type of data protection.
- The column(s) most frequently used to access rows in the table.
- The number of distinct column values.
- The maximum rows per value.
- Whether the rows are accessed by values or through a Join.
- The primary use of the table data (Decision support, Ad Hoc, Batch Reporting, Batch Maintenance, OLTP).
- The number of INSERTS and when they occur.
- The number of DELETEs and when they occur.
- The number of UPDATEs and when they occur.
- The way transactions are written.
- The way the transactions are parceled.
- The level and type of locking a transaction requires.
- How long a transaction hold locks.
- How normalized the data model is.

Through lecture and exercises, this course points out the importance and use of all these factors.

Transactions and Parallel Processing

One additional goal of this course is to point out what causes all-AMP operations. In some cases, they are accidental and can be changed into one-or two-AMP operations.

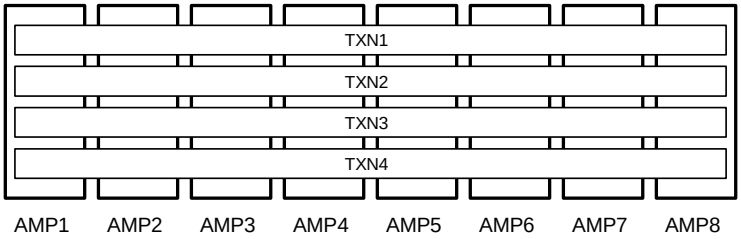
To have the maximum number of transactions that need only one-or two-AMPs, you require a good logical model (Third Normal Form), a good physical model (what you will learn about in this course), and good SQL coding (we will provide some examples).

Transactions and Parallel Processing

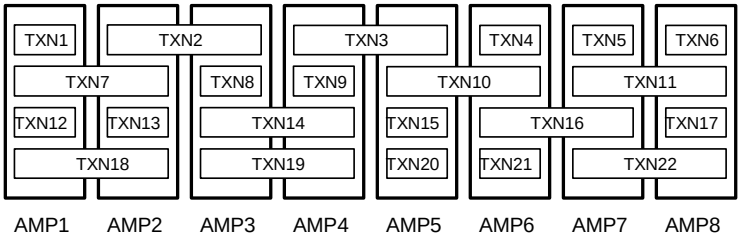
Teradata does all-AMP processing very efficiently.

However, one-AMP and two-AMP processing is even more efficient. It allows the existing configuration to support a greater workload.

Ideal for Decision Support (DSS), Ad Hoc, Batch Processing, and some Batch Maintenance operations.



Best for OLTP, tactical transactions, and preferred for many Batch Maintenance operations. Created by a good Logical Model AND a good Physical Model AND good SQL coding.



This course points out the methods of maximizing the use of one-AMP and two-AMP transactions and when all-AMP operations are needed.

Module 12: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 12: Review Questions

1. Which three are benefits to creating a data model in 3NF? ____ ____ ____

   a. Minimize redundancy
   b. To reduce update anomalies
   c. To improve distribution of data
   d. To improve flexibility of access
   e. To reduce number of I/Os to access data

2. Which data model would include the definition of a partitioned primary index? ____

   a. Logical data model
   b. Physical data model
   c. Business information model
   d. Extended logical data model

3. Which two factors should be considered when deciding to denormalize a table? ____ ____

   a. Volatility
   b. Performance
   c. Distribution of data
   d. Connectivity of users

4. Which is a benefit of implementing data types at the domain level? ____

   a. Reduce storage space
   b. Avoid data conversion
   c. Provides consistent display of data
   d. Reduce need for secondary indexes

Notes

Module 13

TERADATA
Raising Intelligence

Data Distribution and Hashing

After completing this module, you will be able to:

- **Describe the data distribution form and method.**
- **Describe Hashing.**
- **Describe Primary Index hash mapping.**
- **Describe the reconfiguration process.**
- **Describe a Block Layout.**
- **Describe File System Read Access.**

Notes

Table of Contents

Data Distribution 13-4
Hashing 13-6
  Enhanced Hashing Algorithm Starting with Teradata 13.10 13-6
Hash Related Expressions 13-8
Hashing – Numeric Data Types 13-10
Multi-Column Hashing 13-12
  Multi-Column Hashing (cont.) 13-14
Additional Hash Examples 13-16
Using Hash Functions to View Distribution 13-18
  Identifying the Hash Buckets 13-18
  Identifying the Primary AMPs 13-18
Primary Index Hash Mapping 13-20
Hash Maps 13-22
Primary Hash Map 13-24
Hash Maps for Different Systems 13-26
Fallback Hash Map 13-28
Reconfiguration 13-30
Row Retrieval via PI Value – Overview 13-32
Names and Object IDs 13-34
Table ID 13-36
    Spool File Table IDs 13-36
Row ID 13-38
AMP File System – Locating a Row via PI 13-40
Teradata File System Overview 13-42
Master Index Format 13-44
Cylinder Index Format 13-46
Data Block Layout 13-48
Example of Locating a Row – Master Index 13-50
Example of Locating a Row – Cylinder Index 13-52
Example of Locating a Row – Data Block 13-54
Accessing the Row within the Data Block 13-56
AMP Read I/O Summary 13-58
Module 13: Review Questions 13-60

Data Distribution

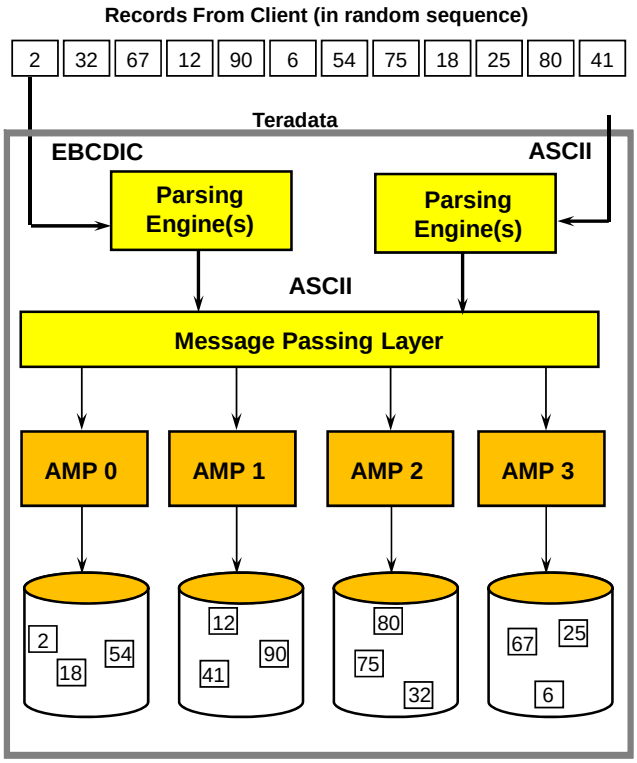
Parsing Engines (PE) are assigned either to channel connections (e.g., IBM Mainframe) or to LAN connections. Data is always stored by the AMPs in 8-bit ASCII. If the input is in EBCDIC, the PE converts it to ASCII before any hashing and distribution takes place.

A USER may have a COLLATION = EBCDIC, ASCII, MULTINATIONAL, or HOST. If the HOST is an EBCDIC host or COLLATION = EBCDIC, then the AMPs convert from ASCII to EBCDIC before doing any comparisons or sorts. MULTINATIONAL allows sites to create their own collation file. Otherwise, all comparisons and sorts use the ASCII collating sequence.

Teradata has no concept of pre-allocated table space. The rows of all hashed tables are distributed randomly across all AMPs and then randomly within the space available on the selected AMP.

Data Distribution

Data distribution is dependent on the hash value of the primary index.



Hashing

Hashing is the mechanism by which Teradata utilizes the Primary Index to distribute rows of data. The Hashing Algorithm acts like a mathematical "blender". It takes up to 64 columns of mixed data as input and generates a single 32-bit binary value called a **Row Hash**.

- The Row Hash is the logical storage locator of the row. A part of this value is used in determining the AMP to which the row is distributed.

- Teradata uses the Row Hash value for distribution, placement and retrieval of rows.

The Hashing Algorithm is random but consistent. Although consecutive PI values do not normally produce consecutive hash values, identical **Primary Index (PI)** values always generate the same Row Hash (assuming that the data types hash identically). Rows with the same Row Hash are always distributed to the same AMP.

Different PI values rarely produce the same Row Hash. When this does occur, they are known as **Hash Synonyms** or **Hash Collisions**.

Note: Upper and lower case values hash to the same hash value. For example, 'Jones' and 'JONES' generate the same hash value.

Enhanced Hashing Algorithm Starting with Teradata 13.10

This enhancement is targeted to reduce the number of hash collisions for character data stored as either Latin or Unicode, notably strings that contain primarily numeric data. Reduction in hash collisions reduces access time per AMP and produces a more balanced row distribution which in-turn improves parallelism. Reduced access time and increased parallelism translate directly to better performance.

This capability is only available starting in TD 13.10. This feature is available to new systems and requires a System Initialization (sysinit) for existing systems. It is anticipated that typically this activity would be performed during technology refresh opportunities.

Hashing

- The Hashing Algorithm creates a fixed length value from any length input string.
- Input to the algorithm is the Primary Index (PI) value of a row.
- The output from the algorithm is the Row Hash.
  - A 32-bit binary value.
  - Used to identify the AMP of the row and the logical storage location of the row in the AMP.
  - Table ID + Row Hash is used to locate the Cylinder and Data Block.
- Row Hash uniqueness depends directly on PI uniqueness.
  - Good data distribution depends directly on Row Hash uniqueness.
- The algorithm produces random, but consistent, Row Hashes.
  - The same PI value and data type combination always hash identically.
  - Rows with the same Row Hash will always go to the same AMP.
- Teradata has a new "Enhanced Hashing Algorithm" starting with Teradata 13.10 new systems and fresh installs (sysinit).
  - Solves the problem of too many hash synonyms when character columns contain numeric data.
  - Problem most commonly occurs with long strings of numeric data in CHAR or VARCHAR columns as either Latin or Unicode.

Hash Related Expressions

The Teradata Database includes extensions to Teradata SQL, known as **hash functions**, which allow the user to extract statistical properties from the current index, evaluate those properties for other columns to determine their suitability as a future primary index, or more effectively design the primary index of rows. These statistics also help minimize hash synonyms and enhance data distribution uniformity. Hash functions are valid within a Teradata SQL statement where other functions (like SUBSTRING or INDEX) can occur.

HASHROW — this function returns the row hash value of a given sequence of expressions in BYTE (4) data type. For example, the following statement returns the average number of rows per row hash where C1 and C2 constitute an index (or potential index) of table TabX

```
SELECT COUNT(*) (FLOAT) / COUNT (DISTINCT(HASHROW (C1,C2))
    FROM TabX;
```

HASHBUCKET — this function returns the bucket number that corresponds to a *hashrow*. The bucket number is an integer type. The following example returns the number of rows in each hash bucket where C1 and C2 are an index (or potential index) of table TabX:

```
SELECT HASHBUCKET (HASHROW(C1,C2)), COUNT(*)
    FROM TabX
    GROUP BY 1 ORDER BY 1;
```

Query results can be treated as a histogram of table distribution among the hash buckets.

HASHAMP and HASHBACKAMP — this function returns the identification number of the primary or fallback AMP corresponding to a *hashbucket*. With Teradata V2R6.2 (and before), HASHAMP accepts only integer values between 0 and 65,535 as its argument. In this example, HASHAMP is used to determine the number of primary rows on each AMP where C1 and C2 are to be the primary index of table TabX:

```
SELECT HASHAMP (HASHBUCKET (HASHROW (C1, C2))), COUNT(*)
    FROM TabX
    GROUP BY 1 ORDER BY 1;
```

Query results can be treated as a histogram of the table distribution among the AMPs.

Further information on these functions and their uses can be found in the *Teradata RDBMS SQL Reference*.

Note the examples on the facing page. This example was captured on a 26 AMP system using a hash map with 1,048,576 entries.

The row hash of the literal 'Teradata' is the same with 16-bit or 20-bit hash bucket numbers. However, the target AMP numbers are different for a system with 65,536 hash buckets as compared to the same system with 1,048,576 hash buckets.

Hash Related Expressions

- **The SQL hash functions are:**

  HASHROW (column(s)) HASHBUCKET (hashrow)
  HASHAMP (hashbucket) HASHBAKAMP (hashbucket)

- **Example 1:**

```
SELECT   HASHROW ('Teradata')                                    AS "Hash Value"
        ,HASHBUCKET (HASHROW ('Teradata'))                       AS "Bucket Num"
        ,HASHAMP (HASHBUCKET (HASHROW ('Teradata')))             AS "AMP Num"
        ,HASHBAKAMP (HASHBUCKET (HASHROW ('Teradata')))          AS "AMP Fallback Num" ;
```

Hash Value	Bucket Num	AMP Num	AMP Fallback Num
F5C4BC93	1006667	12	25

AMP Numbers based on 26-AMP system with 1,048,576 hash buckets.

- **Example 2:**

```
SELECT   HASHROW ('Teradata')     AS "Hash Value 1"
        ,HASHROW ('Teradata  ')   AS "Hash Value 2"
        ,HASHROW ('  Teradata')   AS "Hash Value 3" ;
```

Hash Value 1	Hash Value 2	Hash Value 3
F5C4BC93	F5C4BC93	01989D47

Note: Literals are converted to Unicode and then hashed.

Hashing – Numeric Data Types

The hashing algorithm will hash the same numeric value in different data types to the same value.

A DATE data type and an INTEGER data type hash to the same value. An example follows:

```
CREATE TABLE tableE
      (c1_int          INTEGER
      ,c2_date         DATE)
UNIQUE PRIMARY INDEX (c1_int);

INSERT INTO tableE (1010601, 1010601);
INSERT INTO tableE (NULL, NULL);

SELECT c1_int, HASHROW (c1_int), HASHROW (c2_date) from tableE;
```

c1_int	HASHROW (c1_int)	HASHROW (c2_date)
1010601	1213C458	1213C458
?	00000000	00000000

A second example follows:

```
CREATE TABLE tableF
      (c1_int          INTEGER
      ,c2_int          INTEGER
      ,c3_char         CHAR(4)
      ,c4_char         CHAR(4))
UNIQUE PRIMARY INDEX (c1_int, c2_int);

INSERT INTO tableF (0, NULL,'0', NULL);

SELECT HASHROW (c1_int) AS "Hash c1"
           ,HASHROW (c2_int) AS "Hash c2"
           ,HASHROW (c3_char) AS "Hash c3"
           ,HASHROW (c4_char) AS "Hash c4"
FROM    tableF;
```

Hash c1	Hash c2	Hash c3	Hash c4
00000000	00000000	2BB7F6D9	00000000

Note: The BTEQ commands .SET SIDETITLES and .SET FOLDLINE were used to display the output on the bottom of the facing page.

Hashing – Numeric Data Types

- **The Hashing Algorithm hashes the following numeric data types to the same hash value:**
  - **BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL(x,0), DATE**

Example:

```
CREATE TABLE tableA
      (c1_bint       BYTEINT
      ,c2_sint       SMALLINT
      ,c3_int        INTEGER
      ,c4_bigint     BIGINT
      ,c5_dec        DECIMAL(8,0)
      ,c6_dec2       DECIMAL(8,2)
      ,c7_float      FLOAT
      ,c8_char       CHAR(10))
   UNIQUE PRIMARY INDEX
     (c1_bint, c2_sint);

INSERT INTO tableA (5, 5, 5, 5, 5, 5, 5, '5');
```

```
SELECT   HASHROW (c1_bint)     AS "Hash Byteint"
        ,HASHROW (c2_sint)     AS "Hash Smallint"
        ,HASHROW (c3_int)      AS "Hash Integer"
        ,HASHROW (c4_bigint)   AS "Hash BigInt"
        ,HASHROW (c5_dec)      AS "Hash Dec80"
        ,HASHROW (c6_dec2)     AS "Hash Dec82"
        ,HASHROW (c7_float)    AS "Hash Float"
        ,HASHROW (c8_char)     AS "Hash Char"
FROM    tableA;
```

Output from SELECT

Hash Byteint	609D1715
Hash Smallint	609D1715
Hash Integer	609D1715
Hash BigInt	609D1715
Hash Dec80	609D1715
Hash Dec82	BD810459
Hash Float	E40FE360
Hash Char	551DCFDC

Multi-Column Hashing

The hashing algorithm uses multiplication and addition as commutative operators for handling a multi-column index.

If the data types hash the same, a multi-column index will hash the same for the same values in different columns. Note the example on the facing page.

Note: The result would be the same if 3.0 and 5.0 were used as decimal values instead of 3 and 5.

```
INSERT INTO tableB (5, 3.0);
INSERT INTO tableB (3, 5.0);

SELECT c1_int AS c1
       ,c2_dec AS c2
       ,HASHROW (c1_int) AS "Hash c1"
       ,HASHROW (c2_dec) AS "Hash c2"
       ,HASHROW (c1_int, c2_dec) as "Hash c1c2"
FROM   tableB;
```

c1	c2	Hash c1	Hash c2	Hash c1c2
5	3	609D1715	6D27DAA6	6C964A82
3	5	6D27DAA6	609D1715	6C964A82

Multi-Column Hashing

- The Hashing Algorithm uses multiplication and addition to create the hash value for a multi-column index.
- Assume PI = (A, B)

  [Hash(A) * Hash(B)] + [Hash(A) + Hash(B)] = [Hash(B) * Hash(A)] + [Hash(B) + Hash(A)]

- Example: A PI of (3, 5) will hash the same as a PI of (5, 3) if both c1 & c2 are equivalent data types.

```
CREATE TABLE tableB
      (c1_int      INTEGER
      ,c2_dec      DECIMAL(8,0))
UNIQUE PRIMARY INDEX (c1_int, c2_dec);

INSERT INTO tableB (5, 3);
INSERT INTO tableB (3, 5);
```

These two rows will hash the same and will produce a hash synonym.

```
SELECT   c1_int AS c1
        ,c2_dec AS c2
        ,HASHROW (c1_int) AS "Hash c1"
        ,HASHROW (c2_dec) AS "Hash c2"
        ,HASHROW (c1_int, c2_dec) as "Hash c1c2"
FROM     tableB;
```

*** *Query completed. 2 rows found. 5 columns returned.*

c1	c2	Hash c1	Hash c2	Hash c1c2
5	3	609D1715	6D27DAA6	6C964A82
3	5	6D27DAA6	609D1715	6C964A82

Multi-Column Hashing (cont.)

As mentioned before, the hashing algorithm uses multiplication and addition as commutative operators for handling a multi-column index.

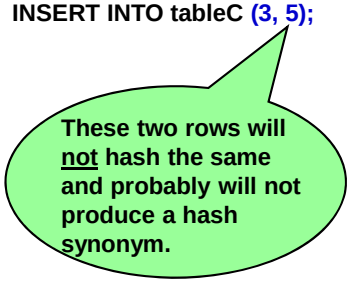
If the data types hash differently, then a multi-column index will hash differently for the same values in different columns. Note the example on the facing page.

Multi-Column Hashing (cont.)

- **A PI of (3, 5) will hash differently than a PI of (5, 3) if column1 and column2 are data types that do not hash the same.**
- **Example:**

```
CREATE TABLE tableC
      (c1_int      INTEGER
      ,c2_dec      DECIMAL(8,2))
UNIQUE PRIMARY INDEX (c1_int, c2_dec);

INSERT INTO tableC (5, 3);
INSERT INTO tableC (3, 5);
```




```
SELECT   c1_int AS c1
        ,c2_dec AS c2
        ,HASHROW (c1_int) AS "Hash c1"
        ,HASHROW (c2_dec) AS "Hash c2"
        ,HASHROW (c1_int, c2_dec) as "Hash c1c2"
FROM     tableC;
```

*** *Query completed. 2 rows found. 5 columns returned.*

c1	c2	Hash c1	Hash c2	Hash c1c2
5	3.00	609D1715	A4E56902	0E452DAE
3	5.00	6D27DAA6	BD810459	336B8C96

Additional Hash Examples

A numeric value of 0 hashes the same as a NULL. A character data type with a value of all spaces also hashes the same as a NULL. However, a character value of '0' hashes to a value different than the hash of a NULL.

Upper and lower case characters hash the same.

The following example shows that different numeric types with a value of 0 all hash to the same hash value.

```
CREATE TABLE tableA
          (c1_bint          BYTEINT,
           c2_sint          SMALLINT,
           c3_int           INTEGER,
           c4_dec           DECIMAL(8,0),
           c5_dec2          DECIMAL(8,2),
           c6_float         FLOAT,
           c7_char          CHAR(10))
UNIQUE PRIMARY INDEX (c1_bint, c2_sint);

.SET FOLDLINE
.SET SIDETITLES

INSERT INTO tableA (0,0,0,0,0,0,'0');

SELECT
      HASHROW (c1_bint)        AS "Hash Byteint"
     ,HASHROW (c2_sint)        AS "Hash Smallint"
     ,HASHROW (c3_int)         AS "Hash Integer"
     ,HASHROW (c4_dec)         AS "Hash Dec0"
     ,HASHROW (c5_dec2)        AS "Hash Dec2"
     ,HASHROW (c6_float)       AS "Hash Float"
     ,HASHROW (c7_char)        AS "Hash Char"
FROM    tableA;

     Hash Byteint        00000000
     Hash Smallint       00000000
     Hash Integer        00000000
     Hash Dec0           00000000
     Hash Dec2           00000000
     Hash Float          00000000
     Hash Char           2BB7F6D9
```

Note: An INTEGER value of 500 and a DECIMAL (8, 2) value of 5.00 will both have the same hash value.

Additional Hash Examples

- A NULL value for numeric data types is treated as 0.
- Upper and lower case characters hash the same.

Example:

```
CREATE TABLE tableD
     (c1_int              INTEGER
     ,c2_int              INTEGER
     ,c3_char             CHAR(4)
     ,c4_char             CHAR(4))
UNIQUE PRIMARY INDEX (c1_int, c2_int);

INSERT INTO tableD ( 0, NULL, 'EDUC', 'Educ' );

SELECT   HASHROW (c1_int)   AS "Hash c1"
        ,HASHROW (c2_int)   AS "Hash c2"
        ,HASHROW (c3_char) AS "Hash c3"
        ,HASHROW (c4_char) AS "Hash c4"
FROM     tableD;
```

Result:

Hash c1	Hash c2	Hash c3	Hash c4
00000000	00000000	6ED679D5	6ED679D5
Hash of 0	Hash of NULL	Hash of 'EDUC'	Hash of 'Educ'

Using Hash Functions to View Distribution

The Hash Functions can be used to view the distribution of rows for a chosen Primary Index.

Notes:

- HashRow – returns the row hash value for a given value(s)
- HashBucket – the grouping for a specific hash value
- HashAMP – the AMP that is associated with the hash bucket
- HashBakAMP – the fallback AMP that is associated with the hash bucket

Identifying the Hash Buckets

If you suspect data skewing due to hash synonyms or NUPI duplicates, you can use the HashBucket function to identify the number of rows in each hash bucket. The HashBucket function requires the HashRow of the columns that make up the Primary Index or the columns being considered for a Primary Index.

Identifying the Primary AMPs

The HASHAMP function can be used to determine data skewing and which AMP(s) have the most rows.

The Customer table on the facing page consists of 7017 rows.

Using Hash Functions to View Distribution

Hash Functions can be used to calculate the impact of NUPI duplicates and synonyms for a PI.

```
SELECT    HASHROW (Last_Name, First_Name)
          AS "Hash Value"
          ,COUNT(*)
FROM      customer
GROUP BY  1
ORDER BY  2 DESC;
```

Hash Value	Count(*)
2D7975A8	12
14840BD7	7
(Output cut due to length)	
E7A4D910	1
AAD4DC80	1

The largest number of NUPI duplicates or synonyms is 12.

```
SELECT    HASHAMP (HASHBUCKET
          (HASHROW (Last_Name, First_Name)))
          AS "AMP #"
          ,COUNT(*)
FROM      customer
GROUP BY  1
ORDER BY  2 DESC;
```

AMP #	Count(*)
7	929
6	916
4	899
5	891
2	864
3	864
1	833
0	821

AMP #7 has the largest number of rows.

Primary Index Hash Mapping

The diagram on the facing page gives you an overview of **Primary Index Hash Mapping**, the process by which all data is distributed in the Teradata DBS.

The Primary Index value is fed into the Hashing Algorithm, which produces the Row Hash. The row goes onto the Message Passing Layer. The **Hash Maps** in combination with the Row Hash determines which AMP gets the row. The Hash Maps are part of the Message Passing Layer interface.

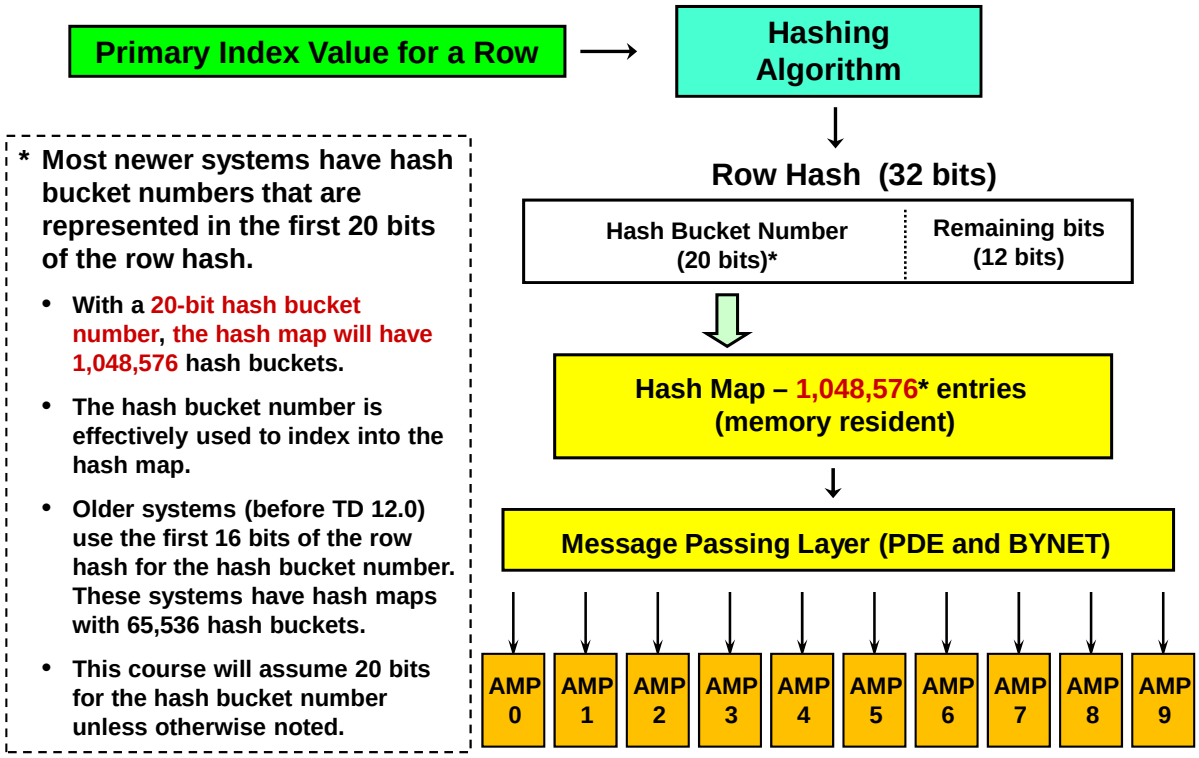
Starting with Teradata Database 12.0, Teradata supports either 65,536 or 1,048,576 hash buckets for a system. The larger number of buckets primarily benefits systems with thousands of AMPs, but there is no disadvantage to using the larger number of buckets on smaller systems.

The hash map is an array indexed by hash bucket number. Each entry of the array contains the number of the AMP that processes the rows in the corresponding hash bucket.

The RowHash is a 32-bit result obtained by applying the hash function to the primary index of the row. On systems with:

- 65,536 hash buckets, the system uses 16 bits of the 32-bit RowHash to index into the hash map.
- 1,048,576 hash buckets, the system uses 20 bits of the 32-bit RowHash as the index.

Primary Index Hash Mapping



* Most newer systems have hash bucket numbers that are represented in the first 20 bits of the row hash.

- With a 20-bit hash bucket number, the hash map will have 1,048,576 hash buckets.
- The hash bucket number is effectively used to index into the hash map.
- Older systems (before TD 12.0) use the first 16 bits of the row hash for the hash bucket number. These systems have hash maps with 65,536 hash buckets.
- This course will assume 20 bits for the hash bucket number unless otherwise noted.

Hash Maps

As you have seen, Hash Maps are the mechanisms that determine which AMP gets a row. They are duplicated on every TPA node in the system. There are 4 Hash Maps:

- Current Configuration Primary (designates where rows are stored)
- Current Configuration Fallback (designates where copies of rows are stored)
- Reconfiguration Primary (designates where rows move during a system reconfiguration)
- Reconfiguration Fallback (designates where copies of rows move during a reconfiguration)

Hash Maps are also used whenever there is a PI or USI operation.

Hash maps are arrays of Hash Map entries. There are 65,536 or 1,048,576 Hash Map entries. Each of these entries points to a single AMP in the system. The Row Hash generated by the Hashing Algorithm contains information that designates a particular entry on a particular Hash Map. This entry tells the system which AMP should be interrupted.

- Teradata Version 1 used a Hash Map with only 3643 hash buckets.
- Teradata Version 2 (prior to Teradata 12.0) used hash maps with 65,536 hash buckets. Starting with Teradata 12.0, the number of hash buckets in a hash map can be either 65,536 or 1,048,576. One of the important impacts of this change was that this increase provides for a more even distribution of data with large numbers of AMPs.

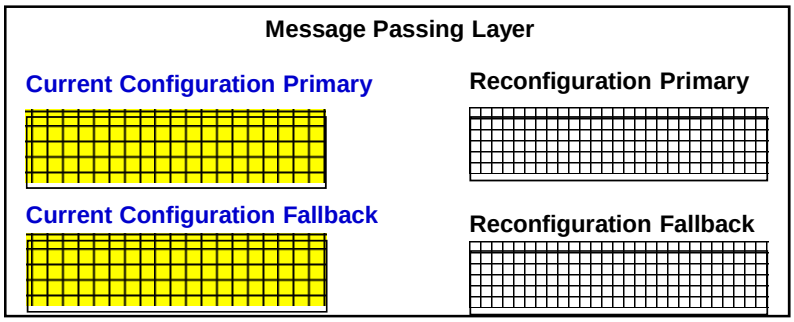
For systems upgraded to Teradata Database 12.0, the default number of hash buckets remains unchanged at 65,536 buckets. For new systems or following a sysinit, the default is 1,048,676 buckets.

Note: The Hash Maps are stored in GDO (Globally Distributed Object) files on each SMP and are loaded into the PDE memory space when PDE software is started – usually as part of the UNIX MP-RAS, Windows 2003, or Linux startup process.

Hash Maps

Hash Maps are the mechanism for determining which AMP gets a row.

- There are four (4) Hash Maps on every TPA node.
- By default, the two Current Hash Maps are loaded into PDE memory space of each TPA node when PDE software boots.



Hash Maps have either 65,536 or 1,048,576 entries. Each entry is 2 bytes in size.

- Starting with Teradata 12.0, new systems (or for systems that have a sysinit), the default number of hash buckets is 1,048,576.
- The increased number of hash buckets provides for a more even distribution of data with large numbers of AMPs.
- For systems upgraded to Teradata Database 12.0, the default number of hash buckets remains unchanged at 65,536 buckets.

Primary Hash Map

The diagram on the facing page is a graphical representation of a **Primary Hash Map**. (It serves to illustrate the concept; they really don't look like this.) The Hash Map utilized by the system is the Current Configuration Primary Hash Map. The Fallback Hash Map IS NOT an exact copy of the Primary Hash Map. The Primary Hash Map identifies which AMP the first (Primary) copy of a row belongs to. The Fallback Hash Map is only used for Fallback protected tables and identifies a different AMP in the same "cluster" for the second (Fallback) row copy.

Note: On most systems (i.e., systems since the 5450), clusters typically consist of 2 AMPs.

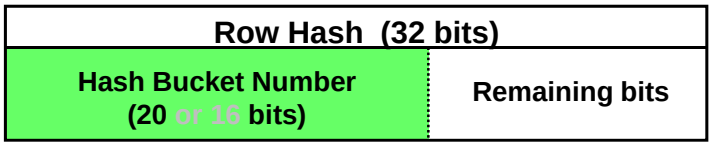
That portion of the Row Hash that points to a particular Hash Map entry is called the Hash Bucket Number (HBN). The hash bucket number is the first 16 or 20 bits of the Row Hash depending on the size of the hash maps. The hash bucket number points to a single entry in a Hash Map. As the diagram shows, the system looks at the particular Hash Map entry specified by the hash bucket number to determine which AMP the row belongs to.

The Message Passing Layer (or Communications Layer) uses only the hash bucket number portion of the Row Hash to determine which AMP gets the row when inserting a new row into a table. The AMP uses the entire 32 bit Row Hash to determine logical disk storage location of the row.

Teradata builds Hash Maps in a consistent fashion. The Primary Hash Map of systems with the same number of AMP vprocs is identical assuming the same number of buckets in the hash map (65,536 or 1,048,576 hash buckets). Fallback Hash Maps may differ due to clustering differences at each site.

The hash bucket number (prior to Teradata 12.0) was commonly referred to as the **Destination Selection Word (DSW).**

Primary Hash Map



PRIMARY HASH MAP – 14 AMP System

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	13	12	13	12	13	11	12	10	13	10	11	12	11	12	13	09
0001	13	07	08	10	08	08	11	11	09	09	10	12	09	09	10	13
0002	10	10	13	05	11	11	12	12	11	11	06	12	13	04	12	12
0003	07	06	13	03	06	08	13	02	13	13	01	00	07	08	05	07
0004	04	04	05	07	09	06	09	07	03	02	03	08	01	00	02	06
0005	01	00	05	04	08	10	10	05	08	08	06	09	07	06	05	11

Note:
This partial hash map (1,048,576 buckets) is associated with a 14 AMP System.

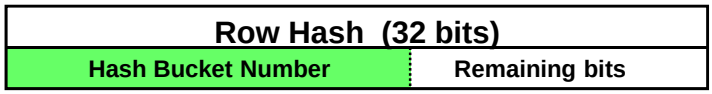
- Assume the Hash Bucket Number is the first 20 bits of the Row Hash.
- The Hash Bucket Number points to one entry within the map.
- The referenced Hash Map entry identifies the AMP for the row hash.

Hash Maps for Different Systems

The diagrams on the facing page show a graphical representation of a Primary Hash Map for an 8 AMP system and a Primary Hash Map for a 16 AMP system. These examples assume hash maps with 1,048,576 entries.

A data value which hashes to “00023 1AB” will be directed to different AMPs on different systems. For example, this hash value will be associated with AMP 5 on an 8 AMP system and AMP 14 on a 16 AMP system.

Hash Maps for Different Systems



PRIMARY HASH MAP – 8 AMP System

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	07	06	07	06	07	04	05	06	05	05	06	06	07	07	03	04
0001	07	07	02	04	01	00	05	04	03	02	03	05	01	00	02	06
0002	01	00	05	05	03	02	04	03	01	00	06	02	04	04	01	00
0003	07	06	03	03	06	06	02	02	01	00	01	00	07	07	05	07
0004	04	04	05	07	05	06	07	07	03	02	03	04	01	00	02	06
0005	01	00	05	04	03	02	06	05	01	00	06	05	07	06	05	07

PRIMARY HASH MAP – 16 AMP System

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	15	14	15	15	13	14	12	14	13	15	15	12	11	12	13	14
0001	13	14	14	10	15	08	11	11	15	09	10	12	09	09	10	13
0002	10	10	13	14	11	11	12	12	11	11	14	12	13	14	12	12
0003	15	15	13	14	06	08	13	14	13	13	14	14	07	08	15	07
0004	15	04	05	07	09	06	09	07	15	15	03	08	15	15	02	06
0005	01	00	05	04	08	10	10	05	08	08	06	09	07	06	05	11

Assume row hash of
00023 1AB

8 AMP system – AMP 05
16 AMP system – AMP 14

Portions of actual hash maps with 1,048,576 hash buckets.

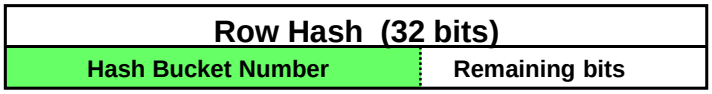
Fallback Hash Map

The diagram on the facing page is a graphical representation of a Primary Hash Map and a **Fallback Hash Map**.

The Fallback Hash Map is only used for Fallback protected tables and identifies a different AMP in the same “cluster” for the second (Fallback) row copy.

Note: These are the actual partial primary and fallback hash maps for a 14 AMP system with 1,048,576 hash buckets.

Fallback Hash Map



PRIMARY HASH MAP – 14 AMP System

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	13	12	13	12	13	11	12	10	13	10	11	12	11	12	13	09
0001	13	07	08	10	08	08	11	11	09	09	10	12	09	09	10	13
0002	10	10	13	05	11	11	12	12	11	11	06	12	13	04	12	12
0003	07	06	13	03	06	08	13	02	13	13	01	00	07	08	05	07
0004	04	04	05	07	09	06	09	07	03	02	03	08	01	00	02	06
0005	01	00	05	04	08	10	10	05	08	08	06	09	07	06	05	11

Assume row hash of
00023 1AB

Primary AMP – 05
Fallback AMP – 12

FALLBACK HASH MAP – 14 AMP System

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	06	05	06	05	06	04	05	03	06	03	04	05	04	05	06	02
0001	06	00	01	03	01	01	04	04	02	02	03	05	02	02	03	06
0002	03	03	06	12	04	04	05	05	04	04	13	05	06	11	05	05
0003	00	13	06	10	13	01	06	09	06	06	08	07	00	01	12	00
0004	11	11	12	00	02	13	02	00	10	09	10	01	08	07	09	13
0005	08	07	12	11	01	03	03	12	01	01	13	02	00	13	12	04

Notes:
14 AMP System with 2 AMP clusters; hash maps with 1,048,576 buckets.

Reconfiguration

Reconfiguration (Reconfig) is the process for changing the number of AMPs in a system and is controlled by the Reconfiguration Hash Maps. The system constructs Reconfiguration Hash Maps by reassigning Hash Map Entries to reflect a new configuration of AMPs. This is done in a way that minimizes the number of rows (and Hash Map Entries) reassigned to a new AMP. After rows are moved, the Reconfiguration Primary Hash Map becomes the Current Configuration Primary Hash Map, and the Reconfiguration Fallback Hash Map becomes the Current Fallback Hash Map.

The diagram on the right illustrates a 200 AMP to 300 AMP Reconfig for a system. The 1,048,576 Hash Map entries are distributed evenly across the 200 AMPs in the initial configuration (top illustration), with approximately 5243 entries referencing each AMP. Thus, there are 5243 Hash Map Entries pointing to AMP 1.

In a 300 AMP system, each AMP will have approximately 3496 referencing the AMP. It is necessary to change 1748 (5243 - 3496) of those and divide them between the new AMPs (AMP 200 through 299). The system does the same thing for the Hash Map Entries that currently point to the other AMPs. This constitutes the Reconfiguration Primary Hash Map. A similar process is done for the Reconfiguration Fallback Hash Map.

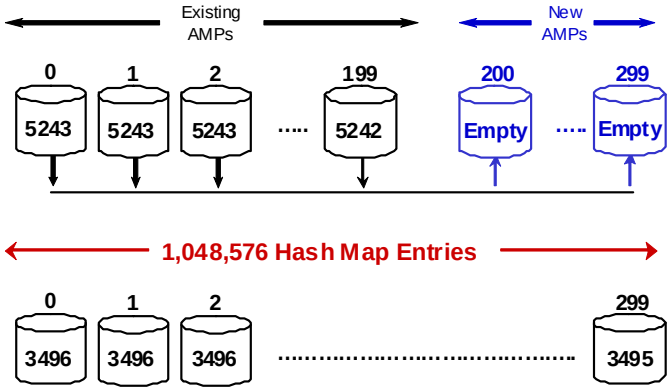
Once the new Hash Maps are ready, the system looks at every row on each AMP and checks to see if the Hash Bucket Number points to one of the Hash Map Entries which was changed. If so, then the row is moved to its new destination AMP.

The formula used to determine the percentage of rows migrating to new AMPs during a Reconfig is shown at the bottom of the right-hand page. Divide the Number of New AMPs by the Sum of the Old and New AMPs (the number of AMPs after the Reconfig). For example, the above 200 to 300 AMP Reconfig causes 33.3% of the rows to migrate.

Reconfiguration

If a 12.0 system (with 1,048,576 Hash buckets) has 200 AMPs, then each of the 200 AMPs will have approx. 5243 entries in the hash map.

If upgrading to 300 AMPs, then each of the 300 AMPs will have a similar number of entries (approx. 3496) in the hash map.



- The system creates new Hash Maps to accommodate the new configuration.
- Old and new maps are compared – each AMP reads its rows, and moves only those that hash to a new AMP.

$$\text{Percentage of Rows Moved to new AMPs} = \frac{\text{Number of New AMPs}}{\text{SUM of Old + New AMPs}} = \frac{100}{300} = \frac{1}{3} = 33.3\%$$

- It is not necessary to offload and reload data due to a reconfiguration.
- If the hash map size is changed (65,536 to 1,048,576), more data will be moved as part of a reconfiguration.

Row Retrieval via PI Value – Overview

The facing page illustrates the step-by-step process involved in Primary Index retrieval. The SELECT statement (shown on facing page) retrieves the row or rows where the PI is equal to a particular column value (or column values in the case of a multi-column PI).

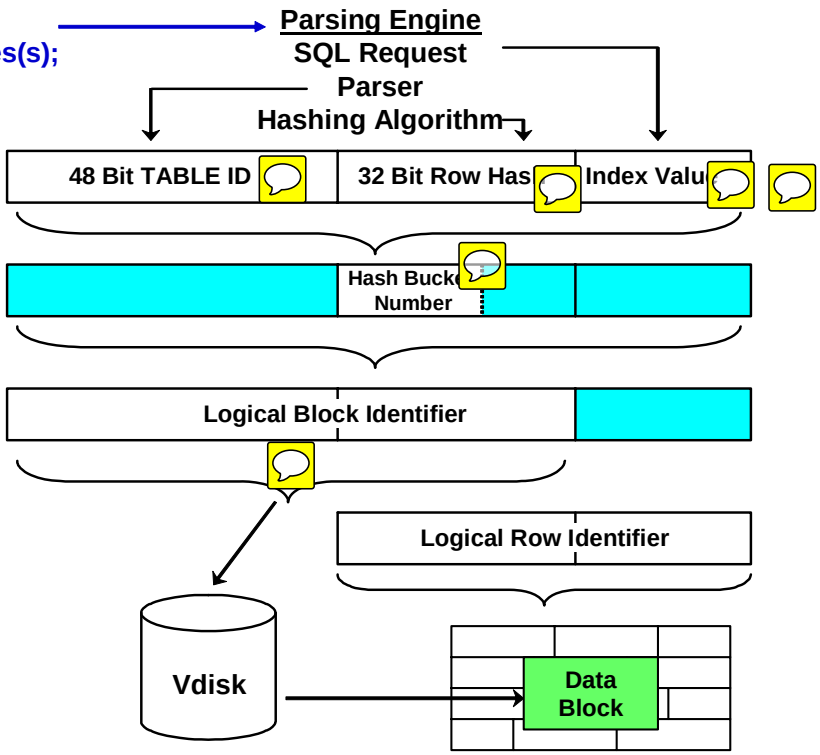
The PE parser always puts out a three-part message composed of the Table ID, Row Hash and Primary Index value. The 48 bit Table ID is looked up in the Data Dictionary, the 32 bit Row Hash value is generated by the Hashing Algorithm and the Primary Index value comes from the SQL request.

The Message Passing Layer (a.k.a., Communications Layer) Interface uses the Hash Bucket Number (first 16 or 20 bits of the Row Hash) to determine which AMP to interrupt and pass on the message.

The AMP uses the Table ID and Row Hash to identify and locate the proper data block, then uses the Row Hash and PI value to locate the specific row(s). The PI value is required to distinguish between Hash Synonyms.

Row Retrieval via PI Value – Overview

SELECT … FROM tablename
WHERE primaryindex = values(s);



With a PI row retrieval, only the AMP (whose number appears in the referenced Hash Map) is accessed by the system.

Names and Object IDs

DBC.Next is a Data Dictionary table that consists of a single row with 9 columns as shown below.

One of the counters is used to assign a globally unique numeric ID to every Database, User, Role, and Profile. A different counter is used to assign a globally unique numeric ID to every Table, View, Macro, Trigger, Stored Procedure, User-Defined Function, Join Index, and Hash Index.

DBC.Next always contains the next value to be assigned to any of these. Think of these columns as counters for ID values.

You may be interested in noting that DBC.Next only contains a single, short row but it requires a Table Header on every AMP, as does any table.

Columns and Indexes are also assigned numeric IDs, which are unique within their respective tables. However, column and index IDs are not assigned from DBC.Next.

DBC.Next columns	Values	Data Type
RowNum	1	CHAR(1)
DatabaseID	numeric	BYTE(4)
TableID	numeric	BYTE(4)
ProcsRowLock	numeric	BYTE(4)
EventNum	numeric	BYTE(4)
LogonSequenceNo	numeric	BYTE(4)
TempTableID	numeric	BYTE(4)
StatsQueryID	number	BYTE(4)
ReconfigID	number	INTEGER

Names and Object IDs

DBC.Next (1 row)

- Each Database/User/Profile/Role – is assigned a globally unique numeric ID.
- Each Table, View, Macro, Trigger, Stored Procedure, User-defined Function, Join Index, and Hash Index – is assigned a globally unique numeric ID.
- Each Column – is assigned a numeric ID unique within its Table ID.
- Each Index – is assigned a numeric ID unique within its Table ID.

- The DD keeps track of all SQL names and their numeric IDs.
- The PE's RESOLVER uses the DD to verify names and convert them to IDs.
- The AMPs use the numeric IDs supplied by the RESOLVER.

Table ID

The **Table ID** is the first part of the three-part message. It is a 48-bit number supplied by the parser. There are two major components of the Table ID:

- The first component of the Table ID is the **Unique Value**. Every table, view and macro is assigned a 32-bit Unique Value, which is assigned by the system table called DBC.Next. In addition to specifying a particular table, this value also indicates whether the table is a normal data table, Permanent Journal table or Spool file table.

- The second component of the Table ID is known as the **Subtable ID**. Teradata stores various types of rows of a table in separate blocks. For example, Table Header rows (described later) are stored in different blocks than primary data rows, which are stored in different blocks than Fallback data rows, and so on (more examples are shown on the facing page). Each separate set of blocks is known as a subtable. The Subtable ID is a 16-bit value that tells the file system which type of blocks to search for.

The facing page lists subtable IDs in decimal value for 2-AMP clusters. The SHOWBLOCKS utility will display the block allocations by subtable and uses decimal values to represent each subtable. If a Reference Index subtable was created, it would have subtable IDs of 1536 and 2560.

For convenience, Table ID examples throughout this course only refer to the Unique Value and omit the Subtable ID.

The Table ID, together with the Row ID, gives Teradata a way to uniquely identify every single row in the entire system.

Spool File Table IDs

Spool files are temporary work tables which are created and dropped as queries are executed. When a query is complete, all of the spool files that it used will be dropped automatically.

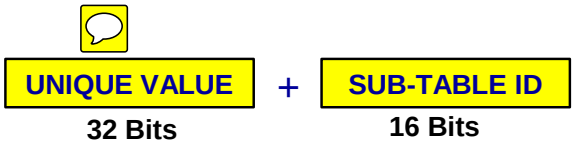
Like all tables, a spool file (essentially a temporary work table) requires a Table ID (or tableid). There is a range of tableids exclusively reserved for spool files (C000 0001 through FFFF FFFF) and the system cycles through them. Eventually, the system will cycle through all the tableids for spool files and reassign spool tableids starting at C000 0001.

Table ID

The Table ID is a Unique Value for Tables, Views, Macros, Triggers, Stored Procedures, Join Indexes, etc. that comes from DBC.Next dictionary table.

Unique Value also defines the type of table:

- **Normal data table**
- **Permanent journal**
- **Global Temporary**
- **Spool file**



Sub-table ID identifies the part of a table the system is looking at.

Sub-table type	Primary ID	Fallback ID	(shown in decimal format)
Table Header	0		
Data table	1024	2048	
1st Secondary index	1028	2052	
2nd Secondary index	1032	2056	
1st Reference index	1536	2560	
1st BLOB or CLOB	1792	2816	
2nd BLOB or CLOB	1794	2818	
Archive Online Subtable	18440	n/a	

Table ID plus Row ID makes every row in the system unique.

Examples shown in this manual use the Unique Value to represent the entire Table ID.

Row ID

The Row Hash is not sufficient to identify a specific row in a table. Since it is based on a Primary Index value, multiple rows can have the same Row Hash. This is due either to Hash Synonyms or NUPI Duplicates.

The **Row ID** makes every row within a table uniquely identifiable. For a non-partitioned table, the Row ID consists of the Row Hash plus a **Uniqueness Value**. The Uniqueness Value is a 32-bit numeric value, designed to identify specific rows within a single Row Hash value. When there are multiple rows with the same Row Hash within a table, the first row is assigned a Uniqueness Value of 1. Additional rows with the same Row Hash are assigned ascending Uniqueness Values.

For Primary Index retrievals, only the Row Hash and Primary Index values are needed to find the qualifying row(s). The Uniqueness Value is needed for Secondary Index support. Since a Row ID is a unique identifier of a row within a table, Teradata uses Row IDs as Secondary Index pointers.

Although Row IDs do identify every row in a table uniquely, they do not guarantee that the data itself is unique. In order to avoid the problem of duplicate rows (permitted in Multiset tables), the complete set of data values for a row (in a Set table) must also be unique.

Summary

- For a non-partitioned table (NPPI), the Row ID consists of the Row Hash + Uniqueness Value for a total of 8 bytes in length.

- For a partitioned table (PPI), the Row ID actually consists of the Partition Number + Row Hash + Uniqueness Value for a total of 10 or 16 bytes in length.

Row ID

On INSERT, Teradata stores both the data values and the Row ID.

ROW ID = ROW HASH and UNIQUENESS VALUE

Row Hash

- Row Hash is based on Primary Index value.
- Multiple rows in a table could have the same Row Hash.
- NUPI duplicates and hash synonyms have the same Row Hash.

Uniqueness Value

- The AMP creates a numeric 32-bit Uniqueness Value.
- The first row for a Row Hash has a Uniqueness Value of 1.
- Additional rows have ascending Uniqueness Values.
- Row IDs determine sort sequence within a Data Block.
- Row IDs support Secondary Index performance.
- The Row ID makes every row within a table uniquely identifiable.

Duplicate Rows

- Row ID uniqueness does not imply data uniqueness.

Note: The Row ID for a non-partitioned table is effectively 8 bytes long.

AMP File System – Locating a Row via PI

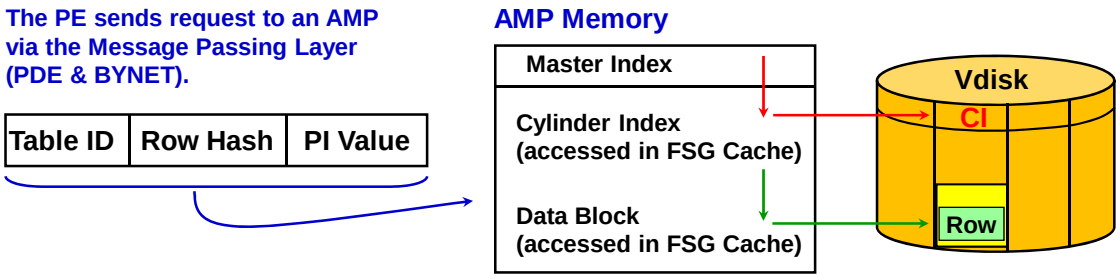
The steps on the right-hand page outline the process that Teradata uses to locate a row. We know that rows are distributed according to their Row Hash. More specifically, the Hash Bucket Number points to a single entry in a Hash Map which designates a particular AMP.

- Once the correct AMP has been found, the Master Index for that AMP is used to identify which Cylinder Index should be referenced.
- The Cylinder Index then identifies the correct Data Block.
- A search of the Data Block locates the row or rows specified by the original three-part message.
- The system performs either linear or indexed searches.

The diagram at the bottom of the facing page illustrates these steps in a graphical fashion.

AMP File System – Locating a Row via PI

- The AMP accesses its Master Index (always memory-resident).
  - An entry in the Master Index identifies a Cylinder # and the AMP accesses the Cylinder Index (frequently memory-resident).
- An entry in the Cylinder Index identifies the Data Block.
  - The Data Block is the physical I/O unit and may or may not be memory resident.
  - A search of the Data Block locates the row(s).



Teradata File System Overview

The Teradata File System software has these characteristics:

- part of AMP address space
- unaware of other AMP or File System instances
- AMP Interface to disk services
- uses PDE FSG services

The Master Index contains an entry (CID) for each allocated cylinder. (CID – Cylinder Index Descriptor)

On the facing page, SRD–A represents an SRD (Subtable Reference Descriptor) for table A. DBD–A1 and DBD–A2 represent data blocks for table A. (DBD – Data Block Descriptor)

On the facing page, SRD–B represents an SRD for table B. DBD–B1, etc. represent data blocks for table B.

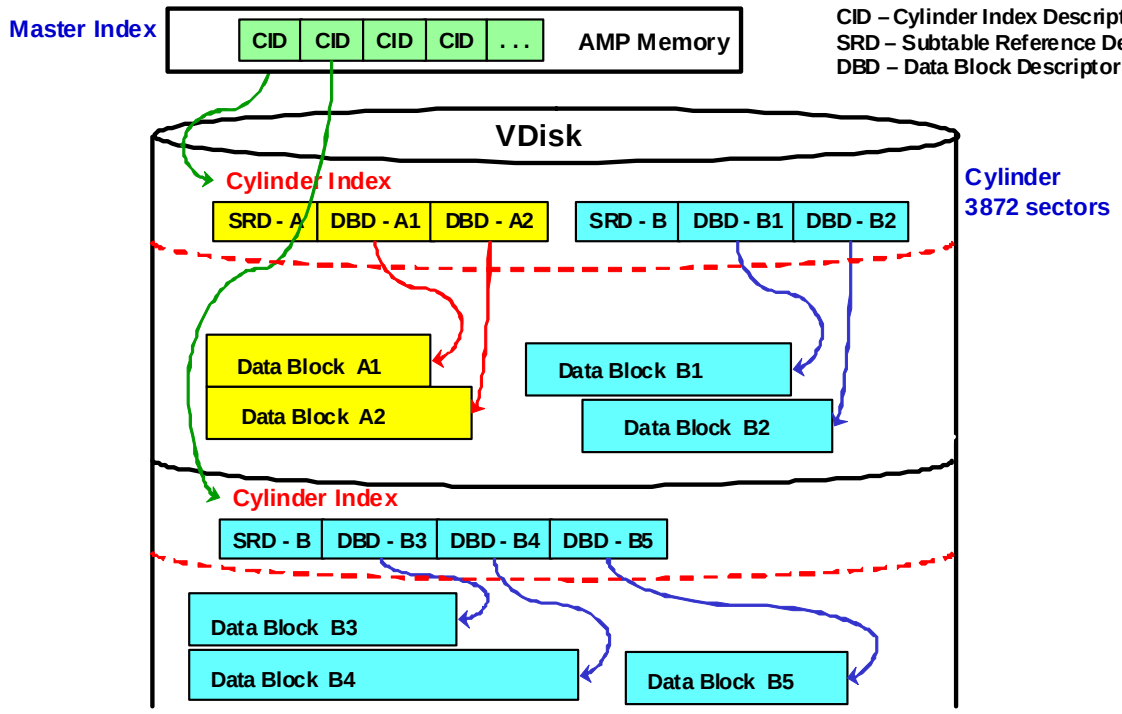
There are actually two cylinder indexes allocated for each cylinder. Each cylinder index is 12 KB in size. Therefore, there is 24 KB (48 sectors) allocated for cylinder indexes at the beginning of each cylinder.

Prior to Teradata 13.10 and Large Cylinder Support, cylinders are 3872 sectors.

Miscellaneous notes:

- Master index entries are 72 bytes long.
- A cylinder index is 12 KB in size for 2 MB cylinders and are 64 KB in size for 12 MB cylinders
- Data rows for PPI tables require an additional 2 bytes to identify the partition number and the spare byte is set to x'80' to identify the row as a PPI row. Secondary index subtable rows also have the Part # + Row Hash + Uniqueness ID) to identify data rows.

Teradata File System Overview



Master Index Format

The first cylinder in each Vdisk contains a number of control structures used by the AMP's File System software. Segment 0 (512 bytes) contains the Vdisk status and a number of structure pointers for the AMP. Following Segment 0 is the FIB (File System Information Block). The FIB contains global file system information – a key component is a status array that shows the status of cylinders (used, free, bad, etc.), and the sorted list of CIDs that are the descriptors for the cylinders currently in use. The FIB effectively contains the list of free or available cylinders. Unlike the Master Index (MI), the FIB is written to disk when cylinders are allocated, and it is read from disk when Teradata boots or when the MI needs to be rebuilt in memory. If necessary, software will allocate additional cylinders for these structures.

The Master Index is a memory resident structure that contains an entry for every allocated data cylinder on that AMP. Entries in the Master Index are sorted by the lowest Table ID and Row ID that can be found on the associated cylinder. The Master Index is used to identify which cylinder a specific row can be found in.

The key elements of the Master Index are:

- Master Index Header - 32 bytes (not shown)
- Cylinder Index Descriptors (CID) – one per allocated cylinder – 72 bytes in length
- Cylinder Index Descriptor Reference Array (not shown) – set of 4 byte pointers to the CIDs; these entries are sorted in descending order.

Note: This array is similar to the row reference array at the end of a data block.

Cylinders that contain no data are not listed in the Master Index. They appear in the Free Cylinder List (which is part of the FIB – File System Information Block) for the associated Vdisk. Entries in the Free Cylinder List are sorted by Cylinder Number.

Each Master Index entry (or CID) contains the following data:

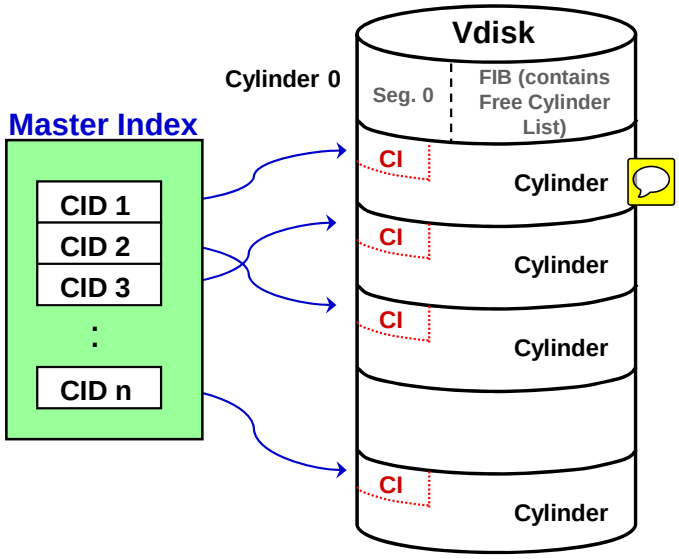
- Lowest Table ID in the cylinder
- Lowest Part # / Row ID value in the cylinder (associated with the lowest Table ID)
- Highest Table ID in the cylinder
- Highest Part # / Row hash (not Row ID) value in the cylinder (associated with the highest Table ID)
- Drive (Pdisk) and Cylinder Number
- Free sectors
- Flags

The maximum size of the Master Index is based on number of cylinders available to the AMP.

Master Index Format

Characteristics

- Memory resident structure specific to each AMP.
- Contains Cylinder Index Descriptors (CID) – one for each allocated Cylinder (72 bytes long).
- Each CID identifies the lowest Table ID / Part# / Row ID and the highest Table ID / Part# / Row Hash for a cylinder.
- Range of Table ID / Part# / Row IDs does not overlap with any other cylinder.
- Sorted list of CIDs.



Notes:

- The Master index and Cylinder Index entries include the partition #'s to support partition elimination for Partitioned Primary Index (PPI) tables.
- For non-partitioned tables, the partition number is 0 and the Master and Cylinder Index entries (for NPPI tables) will use 0 as the partition number in the entry.

Cylinder Index Format

Each cylinder has its own **Cylinder Index (CI)**. The Cylinder Index contains a list of the data blocks and free sectors that reside on the cylinder. The Cylinder Index is accessed to determine which data block a row resides in.

The key elements of the Cylinder Index include:

- Cylinder Index Header (not shown)
- Subtable Reference Descriptors (SRD) contain
  - Table ID
  - Range of DBDs (1st and count)
- Data Block Descriptors (DBD)
  - First Part # / Row ID
  - Last Part # / Row Hash
  - Sector number and size
  - Flags
  - Row count
- Free Sector Entries (FSE) – identifies free sectors in the cylinder. There is one FSE (for each free sector range in the cylinder. The set of FSEs effectively make up the "Free Block List" or also known as the "Free Sector List".
- Subtable Reference Descriptor Array (not shown) – set of 2 byte pointers to the SRDs; these entries are sorted in descending order. Note: This array is similar to the row reference array at the end of a data block.
- Data Block Descriptor Array (not shown) – set of 2 byte pointers to the DBDs; these entries are sorted in descending order. Note: This array is similar to the row reference array at the end of a data block.

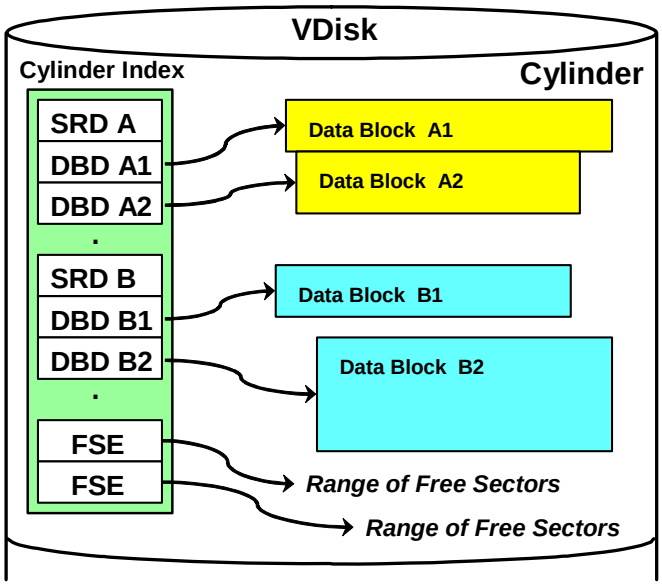
There are two cylinder indexes allocated for each cylinder. Each cylinder index is 12 KB in size. Therefore, there is 24 KB (48 sectors) allocated for cylinder indexes at the beginning of each cylinder.

The facing page illustrates a logical view of SRDs and DBDs and does not represent the actual physical implementation. For example, the SRD and DBD reference arrays are not shown.

Cylinder Index Format

Characteristics

- Located at beginning of each Cylinder.
- There is one SRD (Subtable Reference Descriptor) for each subtable that has data blocks on the cylinder.
- Each SRD references a set of DBD(s). A DBD is a Data Block Descriptor.
- One DBD per data block - identifies location and lowest Part# / Row ID and the highest Part # / Row Hash within a block.
- FSE - Free Segment (or Sector) Entry identifies free sectors.
- Note: Each Cylinder actually has two 12K Cylinder Indexes and the File System software alternates between them.



Data Block Layout

A Block is the physical I/O unit for Teradata. It contains one or more data rows, all of which belong to the same table. They must fit entirely within the block.

The maximum block size is 255 sectors or 127.5 KB.

A Data Block consists of three major sections:

- The Data Block Header (DB Header)
- The Row Heap
- The Row Reference Array

Rows cannot be split between blocks. Each row in a DB is referenced by a separate index to the row known as the Row Reference Array. The **Row Reference Array** is placed at the end of the data block just before the Block Trailer.

With tables that are not partitioned (Non-Partitioned Primary Index – NPPI), each row has at least 14 bytes of overhead in addition to the data values stored in that row. With tables that are partitioned (PPI), each row has at least 16 bytes of overhead in addition to the data values stored in that row. The partition number uses the additional two bytes.

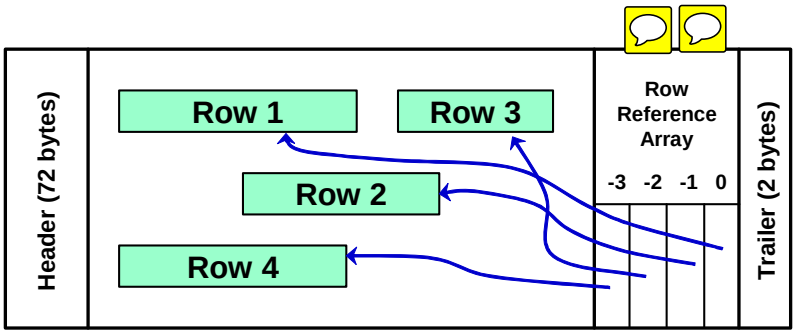
There are also 2 bytes of space used in the Row Reference Array for a 2-byte Reference Array Pointer. This 2-byte pointer identifies the offset of where the row starts within the block. If a row is an odd number of bytes in length, the Row Length specifies its precise length, but the system allocates whole words within the block for the row. Rows will start on an even address boundary.

- Teradata truly supports variable length rows.

- The max amount of user data that you can define in a table row is 64,243 bytes because there is a minimum of 12 bytes of overhead within the row. This gives a total of 64,255 bytes for the data row plus an additional 2 bytes for the row offset within the row reference array.

Data Block Layout

- A data block contains rows with same subtable ID.
  - Contains rows within range of Row IDs of associated DBD entry and the range of Row IDs does not overlap with any other data block.
  - Logically sorted set of rows.
- The maximum block size is 255 sectors (127.5 KB).
  - Blocks can vary in size from 1 sector to 255 sectors.
- A maximum row size is 64,255 bytes.



Example of Locating a Row – Master Index

In the example on the facing page, you can see how Teradata would use the Master Index to locate the data requested by a SELECT statement. The three-part message is Table ID=100, Row Hash=1000 and EMPNO=3755. After identifying the appropriate AMP, Teradata uses that AMP's Master Index to locate which cylinder contains this Table ID and Row Hash. By examining the Master Index, you can see that Cylinder Number 169 contains the appropriate row, if it exists in the system.

Teradata's File System software does a binary search of the CIDs based on Table ID / Part # / Row Hash or Table ID / Part # / Row ID to locate the cylinder number that has the row(s). The CI for that cylinder is accessed to locate the data block.

A user request for a row based on a Primary Index value will only have the Table ID / Part # / Row Hash.

A user request for a row based on a Secondary Index (SI) will have the Table ID / Row Hash for the SI value. The SI subtable row contains the Row ID(s) of the base table row(s). Teradata software uses the Table ID / Row ID(s) to locate the base table row(s). If a table is partitioned, the SI subtable row will have the Part # and the Row ID.

Free cylinders appear in the Free Cylinder List which is part of the FIB (File System Information Block) for the associated Vdisk.

Summary

- **There is only one entry for each cylinder on the AMP.**
- **Cylinders with data appear on the Master Index.**
- **Cylinders without data appear on the free Cylinder List (which is located within the FIB – File System Information Block).**
- **Each index entry identifies its cylinder's lowest Table ID / Partition # / Row ID.**
- **Index entries are sorted by Table ID, Partition #, and Lowest Row ID.**
- **Multiple tables may have rows on the same cylinder.**
- **A table may have rows on many cylinders on different Pdisks on an AMP.**
- **The Free Cylinder List is sorted by Cylinder Number.**

Example of Locating a Row – Master Index

Table ID	Part #	Row Hash	empno
100	0	1000	3755

```
SELECT  *
FROM    employee
WHERE   empno = 3755;
```

Master Index

Lowest			Highest			Pdisk and	Free Cylinder List	Free Cylinder List
Table ID	Part #	Row ID	Table ID	Part #	Row Hash	Cylinder Number	Pdisk 0	Pdisk 1
:	:	:	:	:	:	:	:	:
078	0	58234, 2	095	0	72194	204	124	761
098	0	00107, 1	100	0	00676	037	125	780
100	**0**	**00773, 3**	**100**	**0**	**01361**	**169**	168	895
100	0	01361, 2	100	0	02884	777	170	896
100	0	02937, 1	100	0	03602	802	183	914
100	0	03662, 1	100	0	03999	117	189	935
100	0	04123, 2	100	0	05888	888	201	941
100	0	05974, 1	100	0	07328	753	217	1012
100	0	07353, 1	120	0	00469	477	220	1234
123	1	00343, 2	123	2	01864	529	347	1375
123	2	06923, 1	123	3	00231	943	702	1520
:	:	:	:	:	:	:	:	:

To CYLINDER INDEX

Part # - Partition Number

What cylinder would have Table ID = 100, Row Hash = 00598?

Example of Locating a Row – Cylinder Index

Using the example on the facing page, the File System would determine that the data block it needs is the six-sector block beginning at sector 0789. The Table ID and Row Hash we are looking for (100 + 1000, n) falls between the lowest and highest entries of 100 + 00998, 1 and 100 + 01010.

The convention of 00998, 1 is as follows: 00998 is the Row Hash and 1 is the Uniqueness Value.

Teradata's File System software does a binary search of the SRDs based on Table ID and a binary search of the DBDs Partition #, Row Hash or Row ID to identify the data block(s) that has the row(s).

A user request for a row based on a Primary Index value will include the Table ID / Part # / Row Hash.

A user request for a row based on a Secondary Index (SI) will have the Table ID / Part # / Row Hash for the SI value. The SI subtable row contains the Row ID(s) of the base table row(s). Teradata software uses the Table ID / Part # / Row ID(s) to locate the base table row(s) for secondary index accesses. If a table is partitioned, the SI subtable row will have the Part # and the Row ID.

The example on the facing page illustrates a cylinder that only has one SRD. All of the data blocks in this cylinder are associated with the same subtable.

Summary

- **There is an entry (DBD) for each data block on this cylinder.**
- **These entries are sorted ascending on Table ID, Partition #, and Lowest Row ID.**
- **Only rows belonging to the same table and sub-table appear in a block.**
- **Blocks belonging to the same sub-table can vary in size.**
- **Blocks without data appear on the Free Sector List that is sorted ascending on sector number.**

Example of Locating a Row – Cylinder Index

Table ID	Part #	Row Hash	empno
100	0	1000	3755

```
SELECT  *
FROM    employee
WHERE   empno = 3755;
```

Cylinder Index - Cylinder #169

SRDs	Table ID	First DBD Offset	DBD Count	
SRD #1	100	FFFF	12	

DBDs	Part #	Lowest Row ID	Part #	Highest RowHash	Start Sector	Sector Count	Row Count
:	:	:	:	:	:	:	:
DBD #4	0	00867, 2	0	00902	1010	4	5
DBD #5	0	00938, 1	0	00996	0093	7	10
DBD #6	0	00998, 1	0	01010	0789	6	8
DBD #7	0	01010, 3	0	01177	0525	3	4
DBD #8	0	01185, 2	0	01258	0056	5	6
DBD #9	0	01290, 1	0	01333	1138	5	6
:	:	:	:	:	:	:	:

Free Block List
Free Sector Entries

Start Sector	Sector Count
:	:
0270	3
0301	5
0349	5
0470	4
0481	6
0550	5
:	:

This example assumes that only 1 table ID has rows on this cylinder and the table is not partitioned.

Part # - Partition Number

Example of Locating a Row – Data Block

A Block is the physical I/O unit for Teradata. It contains one or more data rows, all of which belong to the same table. They must fit entirely within the block.

The maximum block size is 255 sectors or 127.5 KB.

A Data Block consists of three major sections:

- The Data Block Header (DB Header)
- The Row Heap
- The Row Reference Array

Rows cannot be split between blocks. Each row in a DB is referenced by a separate "offset or pointer" to the row. These offsets are kept in the Row Reference Array. The **Row Reference Array** is placed near the end of the DB just before the Block Trailer.

The **DB Header** contains control information for both the Row Reference Array and the Row Heap. The DB Header is 72* bytes of information which contains the Table ID (6 bytes). It shows which table and subtable the rows in the block are from.

The **Row Heap** is where the rows reside in the DB. The rows may be in any physical order, are aligned on an even address boundary, and therefore have an even number of bytes allocated for them.

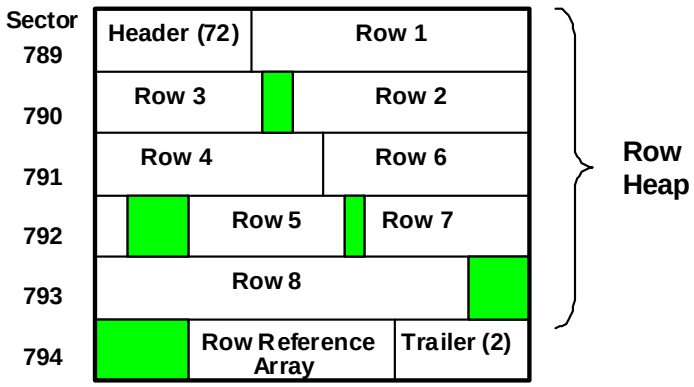
The Reference Array Pointers (2 bytes each), which point to the first byte of a row (Row Length), are maintained in reverse Row ID sequence. The Reference Array pointers are used to do both binary and sequential searches.

The Block Trailer (2 bytes) consists of a block version number which must match the block version number in the Data Block Header.

* Notes on amount of space used by DB Headers.

- If the DB is on a 32-bit system and has never been updated, then the DB Header is only 36 bytes long.
- If the DB is on a 64-bit system and has never been updated, then the DB Header is only 40 bytes long.
- If a data block is new or has been updated (either a 32-bit or 64-bit system), then the DB Header is 72 bytes long.
- The length of the block header for a compressed block is 128 bytes. Note that, in a compressed block, the header is not compressed and neither is the block trailer. Only the row data within the block is compressed. The extended block header has the normal block header at the start and then 56 extra bytes that contains information specific to the compressed block plus some extra filler bytes to allow for later additions without requiring data conversion.

Example of Locating a Row – Data Block



- A block is the physical I/O unit.
- The block header contains the Table ID (6 bytes).
- Only rows for the same table reside in the same data block.
  - Rows are not split across block boundaries.
- Blocks within a table vary in size. The system adjusts block sizes dynamically.
  - Blocks may be from 1 sector (512 bytes) to 255 sectors (127.5 KB).
- Data blocks are not chained together.
- Row Reference Array pointers are stored (sorted) in reverse sequence based on Row ID within the block.

Accessing the Row within the Data Block

Teradata's File System software does a binary search of the Row Reference Array to locate the rows that have a matching Row Hash. Since the Row Reference Array is sorted in reverse sequence based on Row ID, the system can do a binary or linear search.

The first row with a matching Row Hash has its Primary Index value compared with the Primary Index value in the request. The PI value must be checked to eliminate Hash Synonyms. The matching rows are then put into spool. If no matches are made, a message is returned that no rows are found.

In the case of a Unique Primary Index (UPI), the search ends with the first row found matching the criteria. The row is then returned.

In the case of a Non-Unique Primary Index (NUPI), the matching rows (same PI value and Row Hash) are put into spool. With a NUPI, the matching rows in spool are returned.

The example on the right-hand page illustrates how Teradata utilizes the Primary Index data value to eliminate synonyms. This is the conclusion of the example that we have been following throughout this module.

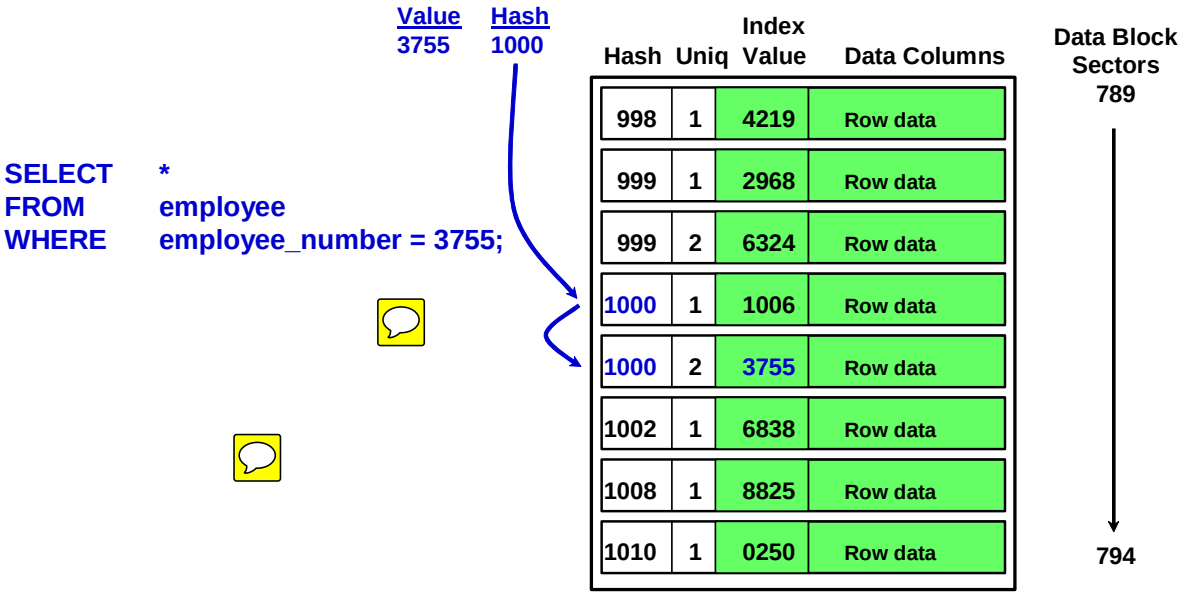
In earlier steps the Master Index was used to find that the desired row was in Cylinder 169. Then the Cylinder Index was used to find that the desired row was in the 6-sector block beginning in Sector Number 789. The diagram shows that block.

The objective is to find that row with **Row Hash=1000** and **Index Value=3755**. When the block is searched, the first row with Row Hash 1000 does not meet these criteria. Its Index Value is 1006, which means that it is a Hash Synonym. The system must continue its search to the next row, the only row that meets both criteria.

The diagram on the facing page shows the logical order of rows in the block with a binary search.

Accessing the Row within the Data Block

- Within the data block, the Row Reference Array is used to locate the first row with a matching Row Hash value within the block.
- The Primary Index data value is used as a row qualifier to eliminate synonyms.



AMP Read I/O Summary

You have seen that a Primary Index Read requires that the Master Index, Cylinder Index and Data Block all must be accessed. The number of I/Os involved in this process can vary.

The Master Index is always resident in memory. The Cylinder Index may or may not be resident in memory and the Data Block may or may not be resident in memory.

Factors that affect the number of I/Os involved include AMP memory, cache size and locality of reference. Often the Cylinder Index is memory resident so that a Unique Primary Index retrieval requires only a single I/O.

Note that no matter how many rows are in the table and no matter how many inserts are made, Primary Index access never gets any more complicated than Master Index to Cylinder Index to Data Block.

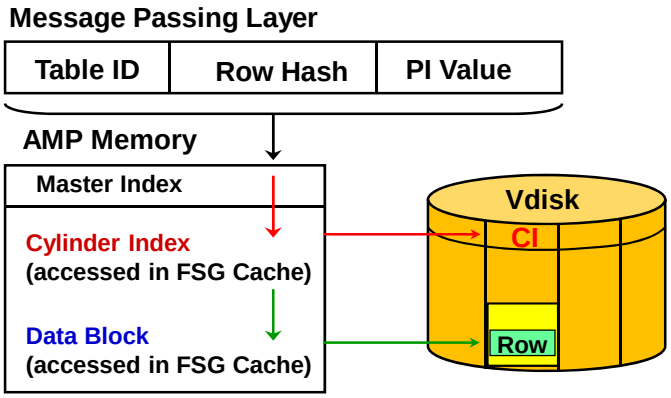
AMP Read I/O Summary

The Master Index is always memory resident.

The AMP reads the Cylinder Index if not memory resident.

The AMP reads the Data Block if not memory resident.

- **The amount of FSG cache size also has an impact if either of these steps require physical I/O.**
- **The data block may or may not be memory residence depending on recent accesses of this data block.**
- **The Cylinder Index is usually memory resident and a Unique Primary Index retrieval requires only one I/O.**



Module 13: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 13: Review Questions

1. The Row Hash for a PI value of 824 is the same for the data types of INTEGER and DECIMAL(18,0). True or False. ____
2. The first 16 or 20 bits of the Row Hash is referred to as the _____ _____ _____ .
3. The Hash Map consists of entries or buckets which identify an _____ number for the Row Hash.
4. The Current Configuration _____ Hash Map is used to locate the AMP to locate/store a row based on PI value.
5. The _____ utility is used to redistribute rows to a new system configuration with more AMPs.
6. When creating a new table, the Unique Value of a Table ID comes from the dictionary table named DBC._____ .
7. The Row ID consists of the _____ _____ and the _____ _____ .
8. The _____ _____ contains a Cylinder Index Descriptor (CID) for each allocated Cylinder.
9. The _____ _____ contains an entry for each data block in the cylinder.
10. The _____ _____ consists of a set of 2 byte pointers to the data rows in data block.
11. The maximum block size is approximately _____ and the maximum row size is approximately _____ .
12. The Primary Index data value is used as a row qualifier to eliminate hash _____ .

Notes

TERADATA
Raising Intelligence

File System Writes

After completing this module, you will be able to:

- **Describe File System Write Access.**
- **Describe what happens when Teradata inserts a new row into a table.**
- **Describe the impact of row inserts on block sizes.**
- **Describe how fragmentation affects performance.**

Notes

Table of Contents

AMP Write I/O 14-4
New Row INSERT – Part 1 14-6
New Row INSERT – Part 2 14-8
New Row INSERT – Part 2 (cont.) 14-10
New Row INSERT – Part 3 14-12
New Row INSERT – Part 4 14-14
  Alternate Cylinder Index 14-14
Blocking in Teradata 14-16
Block Size and Filling Cylinders 14-18
Variable Block Sizes 14-20
Block Splits (INSERT and UPDATE) 14-22
Space Fragmentation 14-24
Cylinder Full 14-26
Mini-Cylpack 14-28
Space Utilization 14-30
  Teradata 13.10 Auto Cylinder Pack Feature 14-30
Merge Datablocks (13.10 Feature) 14-32
  Merge Datablocks (Teradata 13.10) cont. 14-34
    How to use this Feature 14-34
File System Write Summary 14-36
Module 14: Review Questions 14-38
  Module 14: Review Questions (cont.) 14-40

AMP Write I/O

The facing page illustrates how Teradata performs write operations and it outlines steps required to perform an AMP Write operation.

WAL (Write Ahead Logging) is a recoverability/reliability feature that also provides performance improvements in the area of database writes. WAL is a Teradata V2R6.2 (and later) feature. WAL can batch up modifications from multiple transactions and apply them with a single disk I/O, thereby saving I/O operations. WAL will help improve throughput for I/O-bound workloads.

WAL is a log-based file system recovery scheme in which modifications to permanent data are written to a log file, the WAL log. The log file contains change records (Redo records) which represent the updates. At key moments, such as transaction commit, the WAL log is forced to disk. In the case of a reset or crash, Redo records can be used to transform the old copy of a permanent data block on disk into the version that existed at the time of the reset.

By maintaining the WAL log, the permanent data blocks that were modified no longer have to be written to disk as each block is modified. Only the Redo records in the WAL log must be written to disk. This allows a write cache of permanent data blocks to be maintained. WAL protects all permanent tables and all system tables but is not used to protect either the Transient Journal (TJ), since TJ records are stored in the WAL log, or any type of spool tables, including global temporary tables.

The WAL log is maintained as a separate logical file system from the normal table area. Whole cylinders are allocated to the WAL log, and it has its own index structure. The WAL log data is a sequence of WAL log records and includes the following:

- Redo records, used for updating disk blocks and insuring file system consistency during restarts.
- TJ records used for transaction rollback.

There is some additional CPU cost for maintaining the WAL log so WAL may reduce throughput for CPU-bound workloads. However, the overall performance is expected to be better with WAL since the benefit of I/O improvement outweighs the much smaller CPU cost.

If CHECKSUM = NONE and the New Block length = Old Block length, Teradata will attempt to update-in-place for any INSERT, DELETE, or UPDATE operations.

If the CHECKSUM feature is enabled for a table, any INSERT, UPDATE, or DELETE operation will cause a new data block to be allocated.

The FastLoad and MultiLoad utilities always allocate new data blocks for write operations. TPump follows the same rules as an SQL INSERT, UPDATE, or DELETE.

AMP Write I/O

For SQL writes, Teradata uses WAL logic to manage disk write operations.

- Read the Data Block if not in memory (Master Index > Cylinder Index > Data Block).
- Place appropriate entries (e.g., before-images) into the Transient Journal buffer (actually a WAL buffer) and write it to the WAL log on disk.
- Data blocks are updated in Memory, but not written immediately to disk.
- The after-image or changed image (REDO row) is written to a WAL buffer which is written to the WAL log on disk.
  - WAL can batch up modifications from multiple transactions and apply them with a single disk I/O, thereby saving I/O operations.
  - Updated data blocks in memory will be eventually aged out and written to disk.
- Make the changes to the Data Block in memory and determine the new block's length.
  - If the New Block has changed size, always allocate a new Data Block.
  - If the New Block length = Old Block length, Teradata will attempt to update-in-place for any INSERT, DELETE, or UPDATE operations.

These operations happen concurrently on the Fallback AMP.

New Row INSERT – Part 1

The facing page illustrates what happens when Teradata INSERTs a new row into a table. The three part message is Table ID = 100, Partition # = 0, Row Hash = 1123 and PI Value = 7923.

- The AMP uses its Master Index to locate the proper cylinder for the new row. As you can see, Cylinder #169 is where a row with Table ID = 100, Partition # = 0, and Row Hash = 1123 should be inserted.

- The next step is to access the Cylinder Index for Cylinder #169, as illustrated on the facing page.

Teradata's File System software does a binary search of the CIDs based on Table ID / Partition # / Row Hash to locate the cylinder number in which to insert the row. The CI for that cylinder is accessed to locate the data block.

Note: The Partition # (shown in the examples) does not exist in Teradata systems prior to V2R5.

New Row Insert – Part 1

INSERT INTO employee VALUES (7923, ...);

INSERT ROW	Table ID	Part #	Row Hash	data column values				
	100	0	1123	7923				

Master Index

Lowest			Highest			Pdisk and
Table ID	Part #	Row ID	Table ID	Part #	Row Hash	Cylinder Number
:	:	:	:	:	:	:
078	0	58234, 2	095	0	72194	204
098	0	00107, 1	100	0	00676	037
100	**0**	**00773, 3**	**100**	**0**	**01361**	**169**
100	0	01361, 2	100	0	02884	777
100	0	02937, 1	100	0	03602	802
100	0	03662, 1	100	0	03999	117
100	0	04123, 2	100	0	05888	888
100	0	05974, 1	100	0	07328	753
100	0	07353, 1	120	0	00469	477
123	1	00343, 2	123	2	01864	529
123	2	06923, 1	123	3	00231	943
:	:	:	:	:	:	:

Free Cylinder List Pdisk 0	Free Cylinder List Pdisk 1
:	:
124	761
125	780
168	895
170	896
183	914
189	935
201	941
217	1012
220	1234
347	1375
702	1520
:	:

To CYLINDER INDEX

Part # - Partition Number

New Row INSERT – Part 2

The example on the facing page is a continuation from the previous page. Teradata has determined that the new row must be INSERTed into Cylinder #169 in this example.

New Row Insert – Part 2

INSERT INTO employee VALUES (7923, ...);

INSERT ROW	Table ID	Part #	Row Hash	data column values				
	100	0	1123	7923				

Cylinder Index - Cylinder #169

SRDs	Table ID	First DBD Offset	DBD Count
SRD #1	100	FFFF	12

DBDs	Part #	Lowest Row ID	Part #	Highest RowHash	Start Sector	Sector Count	Row Count
:	:	:	:	:	:	:	:
DBD #4	0	00867, 2	0	00902	1010	4	5
DBD #5	0	00938, 1	0	00996	0093	7	10
DBD #6	0	00998, 1	0	01010	0789	6	8
DBD #7	0	01010, 3	0	01177	0525	3	4
DBD #8	0	01185, 2	0	01258	0056	5	6
DBD #9	0	01290, 1	0	01333	1138	5	6
:	:	:	:	:	:	:	:

Free Block List
Free Sector Entries

Start Sector	Sector Count
:	:
0270	3
0301	5
0349	5
0470	4
0481	6
0550	5
:	:

Read the block into memory (FSG cache).

To Data Block

New Row INSERT – Part 2 (cont.)

The example on the facing page is a continuation from the previous page. Teradata has determined that the new row hash value falls with the range of the data block that starts at sector 525 and is 3 sectors long.

If the block that has been read into memory (FSG Cache) has enough contiguous free bytes, then the row is inserted into this space within the block. The row reference array and the Cylinder Index are updated.

If the block that has been read into memory (FSG Cache) does not have enough contiguous free bytes, but it does have enough free bytes within the entire block, the software will defragment the block and insert the row. The row reference array and the Cylinder Index are updated.

Note: The block header contains a field that indicates the total number of free bytes within the block.

Also note that the Row Reference Array expands by 2 bytes to reflect the added row. If the block now has 5 rows, the Row Reference Array will increase from 8 bytes to 10 bytes in length.

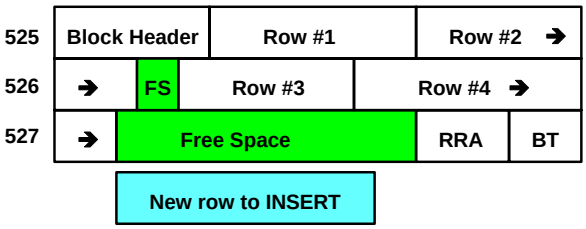
Acronyms:

FS – Free Space
RRA – Row Reference Array
BT – Block Trailer

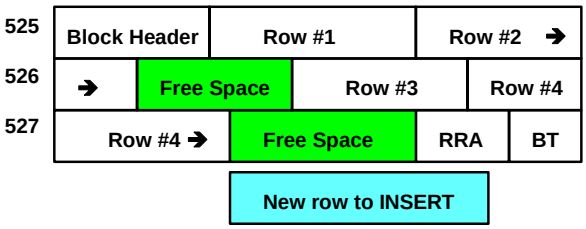
New Row Insert – Part 2 (cont.)

Read the block into memory (FSG cache).

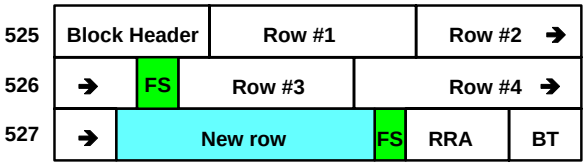
1. If the block has enough free <u>contiguous</u> bytes, then insert row into block and update CI.



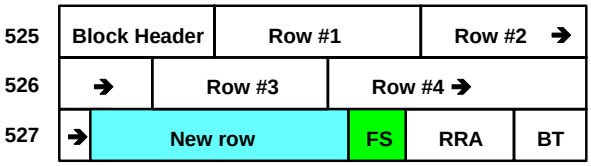
2. If the block has enough free bytes, then <u>defragment</u> the block and insert row into block and update CI.



Row is inserted into contiguous free space.



Block is defragmented and row is inserted into contiguous free space.



FS - Free Space; RRA - Row Reference Array; BT - Block Trailer

New Row INSERT – Part 3

The File System then accesses the 3-sector block which starts at sector 525 and makes it available in AMP memory.

The row is placed into the block, and the new block length is computed. In this example, inserting the row has caused the block to expand from 3 sectors to 4 sectors.

Note that the Row Reference Array expands by 2 bytes to reflect the added row. If the block now has 5 rows, the Row Reference Array will increase from 8 bytes to 10 bytes in length.

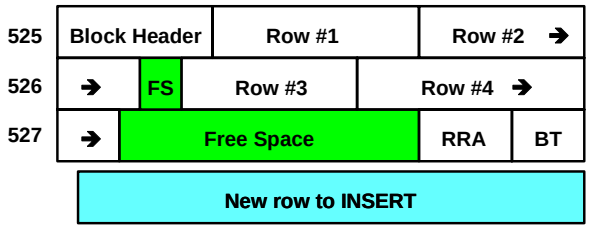
Acronyms:

- FS – Free Space
- RRA – Row Reference Array
- BT – Block Trailer

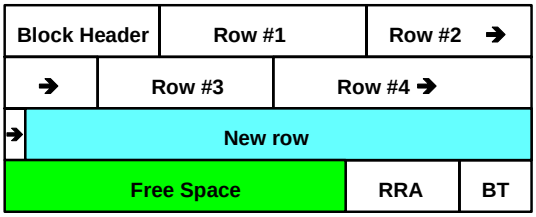
New Row Insert – Part 3

3. If the new row is larger than the total free space within the block, then the Insert expands the block by as many sectors as needed (in memory) to hold the row.

   In this example, the block is expanded by one sector in memory.



In memory, block is expanded.



4. The next step is to locate the first block on the Free Block List equal to, or greater than 4 sectors.

New Row INSERT – Part 4

The File System searches the Free Sector (or Block) List looking for the first Free Block whose size is equal to or greater than the new block's requirement. It does not have to be an exact match.

- Upon finding a 5-sector free block starting at sector 0301, the system allocates a new 4-sector block (sectors 301, 302, 303, 304) for the new data block, leaving a free block of one sector (305) remaining.
- The new data block is written to disk.
- The old, 3-sector data block is placed onto the Free Sector List (or Free Block List).
- The modified CI will be copied to the buddy node (FSG Cache) and the modified CI will be written back to disk (eventually).

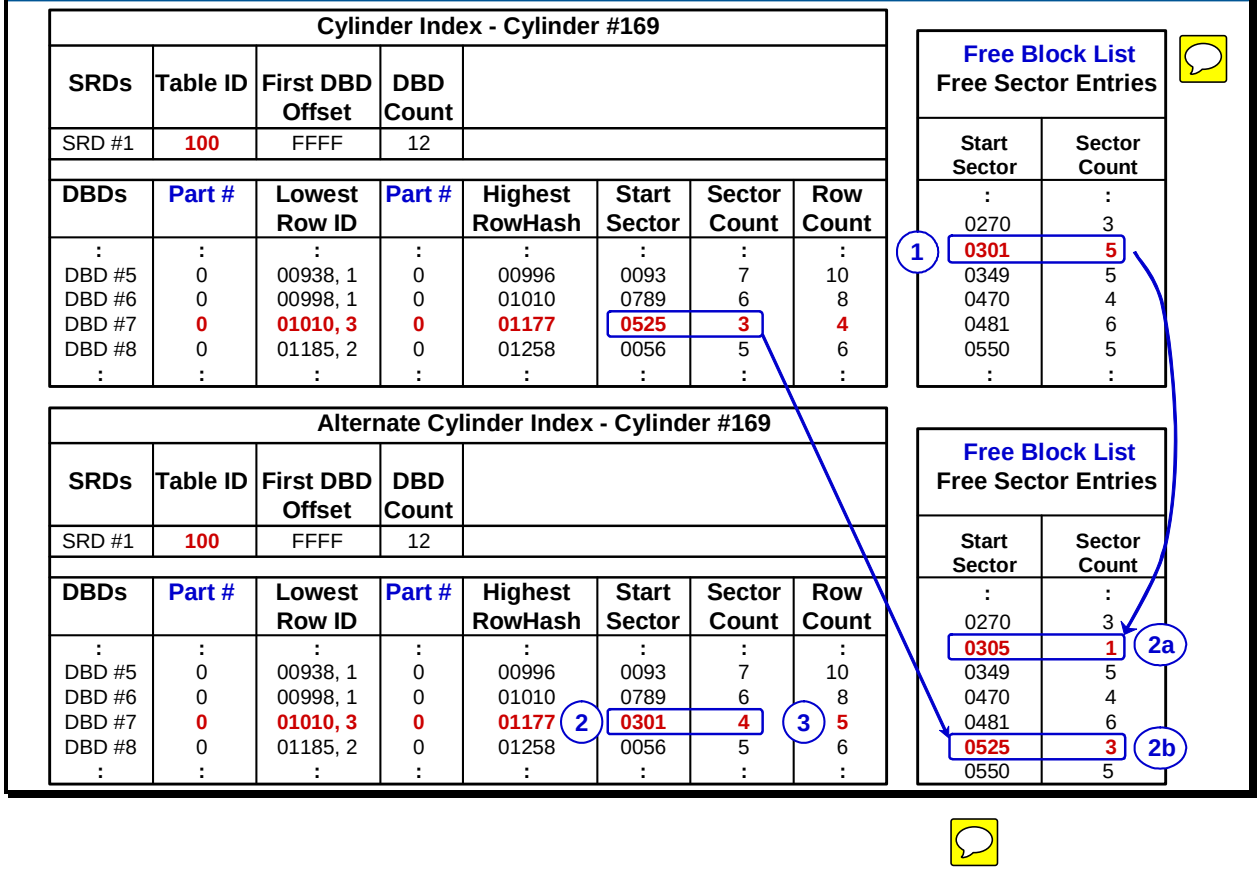
If a transaction failure occurs (or the transaction is aborted), the Transient Journal is used to undo the changes to both the data blocks and the Cylinder Indexes. Before images of data rows are written to the Transient Journal. Before images of Cylinder Indexes are not written to the Transient Journal because Teradata uses the Alternate Cylinder Index for the changes. If a transaction fails, the before image in the Transient Journal is used to return the data row(s) back to the state before the transaction.

Alternate Cylinder Index

Starting with V2R6.2 and with WAL, space for 2 Cylinder Indexes (2 x 12 KB = 24 KB) is allocated at the beginning of every cylinder. Characteristics include:

- Two Cylinder Indexes are used – Teradata alternates between the two Cylinder Indexes.
- Changes are written to an "Alternate Cylinder Index".
- When a CI is changed, it is not updated in place. This provides for better I/O integrity.

New Row INSERT – Part 4



Blocking in Teradata

Tables supporting Data Warehouse and Decision Support users generally have their block size set very large to accommodate more rows per block and reduce the number of block I/Os needed to do full table scans. Tables involved in online applications and heavy data maintenance generally have smaller block sizes.

Extremely large rows, called Oversized Rows, are very costly. Each Oversized row requires its own block and costs one I/O every time it is touched. Oversized rows are common in non-relational data models and appear in poor relational data models.

Blocking in Teradata

Definitions

Largest Data Block Size

- The largest multi-row data block allowed. Impacts when a block split occurs.
- Determined by:
  - Table level attribute DATABLOCKSIZE
  - System default - PermDBSize parameter (DBS Control) – 5555 default is 254 sectors (127 KB)

Large (or typical) Row

- The largest fixed length row that allows multiple rows/block.
- Defined as ((Largest Block - 74) / 2);
  - Block header is 72 bytes and trailer is 2 bytes.

Oversized Row

- A row that requires its own Data Block (one I/O per row):
- A fixed length row that is larger than Large Row.

Example:

- Assume DATABLOCKSIZE = 65,024 (127 sectors x 512 bytes)
  - Largest Block = 65,024 bytes
  - Large Row ≤ 32,475 bytes ((65,024 – 74) / 2)
  - Oversize row > 32,476 bytes

Block Size and Filling Cylinders

Teradata supports a maximum block size of 255 sectors. With newer, larger, and faster systems, it typically makes sense to use a large block size for transactions that do full table or partition scans. A large block may help to minimize the number of I/Os needed to access a large amount of data.

Therefore, it may seem that using the largest possible block size of 255 sectors would be a good choice. However, a maximum block size of 254 sectors is actually a better choice in most situations. Why?

With 254 sector blocks, a cylinder can hold 15 blocks.
With 255 sector blocks, a cylinder can only hold 14 blocks.

Why?

A cylinder consists of 3872 sectors and 48 sectors are used for the cylinder indexes.
The available space for user data blocks is 3872 – 48 = 3824 sectors.

3824 ÷ 254 = 15.055 or 15 blocks
3824 ÷ 255 = 14.996 or 14 blocks

15 x 254 = 3810 sectors of a cylinder are utilized or 99.6%
14 x 255 = 3570 sectors of a cylinder are utilized or only 93.4%

Assume an empty staging table and using FastLoad to load data into the table. With 255 sector blocks, the table will use 6% more cylinders to hold the data.

By using a system default (PermDBSize) or data block size (DATABLOCKSIZE) of 254 sectors will effectively utilize the space in cylinders more efficiently than 255 sector blocks.

The same is true if you are considering 127 or 128 sector blocks.

127 sector blocks – cylinder can hold 30 blocks – utilize 99.6% of cylinder
128 sector blocks – cylinder can hold 29 blocks – utilize 97.1% of cylinder

Therefore, 127 or 254 sector blocks are typically better choices. A greater percentage of cylinder space can be utilized with these choices.

Block Size & Filling Cylinders

What is the difference between choosing maximum data block size of 254 or 255 sectors?

- With 254 sector blocks, a cylinder can hold 15 blocks.
- With 255 sector blocks, a cylinder can only hold 14 blocks.

Why?

- A cylinder consists of 3872 sectors and 48 sectors are used for the cylinder indexes.
  - 3824 254 = 15.055 or 15 blocks
  - 3824 255 = 14.996 or 14 blocks
  - 15 x 254 = 3810 sectors of a cylinder are utilized or 99.6%
  - 14 x 255 = 3570 sectors of a cylinder are utilized or only 93.4%
- Assume an empty staging table and using FastLoad to load data into the table. With 255 sector blocks, the table will use 6% more cylinders.

What about 127 and 128 sector blocks?

- With 127 sector blocks, a cylinder can hold 30 blocks – utilize 99.6% of cylinder
- With 128 sector blocks, a cylinder can hold 29 blocks – utilize 97.1% of cylinder

Therefore, 127 or 254 sector blocks are typically better choices for PermDBSize and/or data block sizes. A greater percentage of cylinder space can be utilized with these choices.

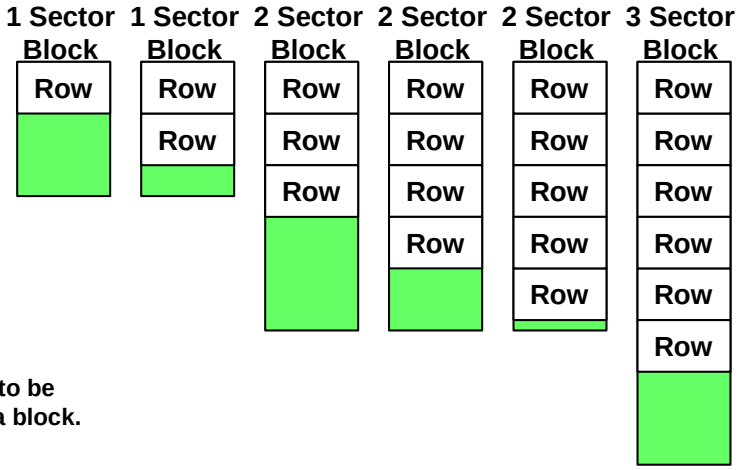
Variable Block Sizes

The Teradata RDBMS supports true variable block sizes. The illustration on the facing page shows how blocks can expand to accommodate additional rows as they are INSERTed. As rows are INSERTed, the Reference Array Pointers are placed into Row ID sequence.

– REMEMBER –
Large rows require more disk space for
Transient Journal, Permanent Journal, and Spool files.

Variable Block Sizes

- When inserting rows (ad hoc SQL or TPump), the block expands as needed to accommodate them.
- The system maintains rows within the block in logical ROW ID sequence.
- Large rows take more disk space for Transient Journal, Permanent Journal, and Spool files.
- Blocks are expanded until they reach "Largest Block Size". At this point, a **Block Split** is attempted.



Note:

Rows do NOT have to be contiguous in a data block.

Block Splits (INSERT and UPDATE)

Block splits occur during INSERT and UPDATE operations. Normally, when a data block expands beyond the maximum multi-row block size (Largest Block), it splits into two approximately equal-sized blocks. This is shown in the upper illustration on the facing page.

- If an Oversize Row is INSERTed into a data block, it causes a three-way block split (as shown in the lower illustration). This type of block split may result in uneven block sizes.

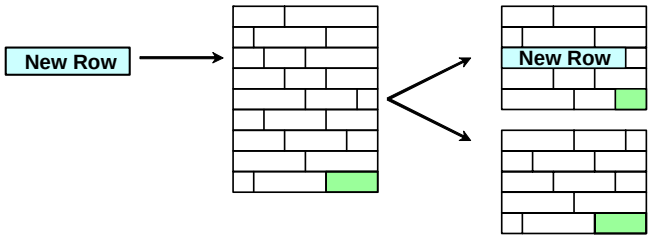
- With Teradata, block splits cost only one additional I/O per extra block created. There is little impact on OLTP and OLCP performance.

- Block splits automatically reclaim any contiguous, unused space greater than 511 bytes.

Block Splits (INSERT and UPDATE)

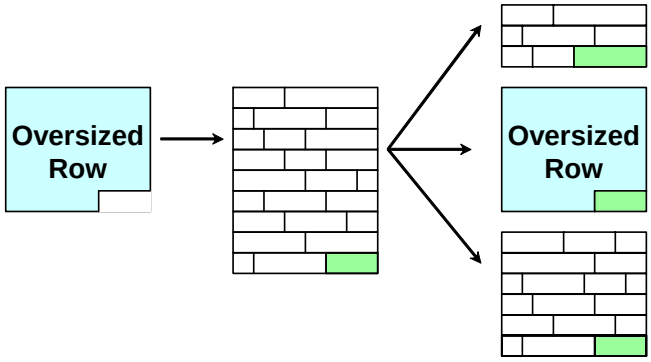
Two-Way Block Splits

- When a Data Block expands beyond Largest Block, it splits into two, fairly-equal blocks.
- This is the normal case.



Three-Way Block Splits

- An oversize row gets its own Data Block. The existing Data Block splits at the row's logical point of existence.
- This may result in uneven block sizes.



Notes:

- Block splits automatically reclaim any unused space over 511 bytes.
- While it is not typical to increment blocks by one 512 sector, it is tunable as to how many sectors are acquired at a time for the system.

Space Fragmentation

Space fragmentation is not an issue in the Teradata database because the system collects free blocks as a normal part of routine table maintenance. If a block of sectors is freed up and is adjacent to already free sectors in the cylinder, these are combined into one entry on the free block list.

As previously described, when an actual data block has to grow, it does not grow into adjacent free blocks – a new block is assigned from the free block list. The freed up data block (set of sectors) is placed on the free block (or segment) list. If there is already an entry on the free block list representing adjacent free blocks, then the freed up data block is combined with adjacent free sectors and only one entry is placed on the free block list.

Using the example on the facing page, assume we are looking at a 40-sector portion of a cylinder. These sectors are physically adjacent to each other. The free block list would have 2 entries on it – one representing the 4 unused sectors and a second entry representing the 6 unused sectors.

We will now consider 4 situations.

First case – If the first 10-sector data block is freed up, software will not place an entry on the free block list for just these 10 sectors. Software will effectively combine these 10 sectors with the following adjacent free 4 sectors and place one entry representing the 14 free sectors on the free block list. For this 40-sector portion of a cylinder, there will be 2 entries on the free block list – one for the first 14 unused sectors and a second entry for the 6 unused sectors that are still there.

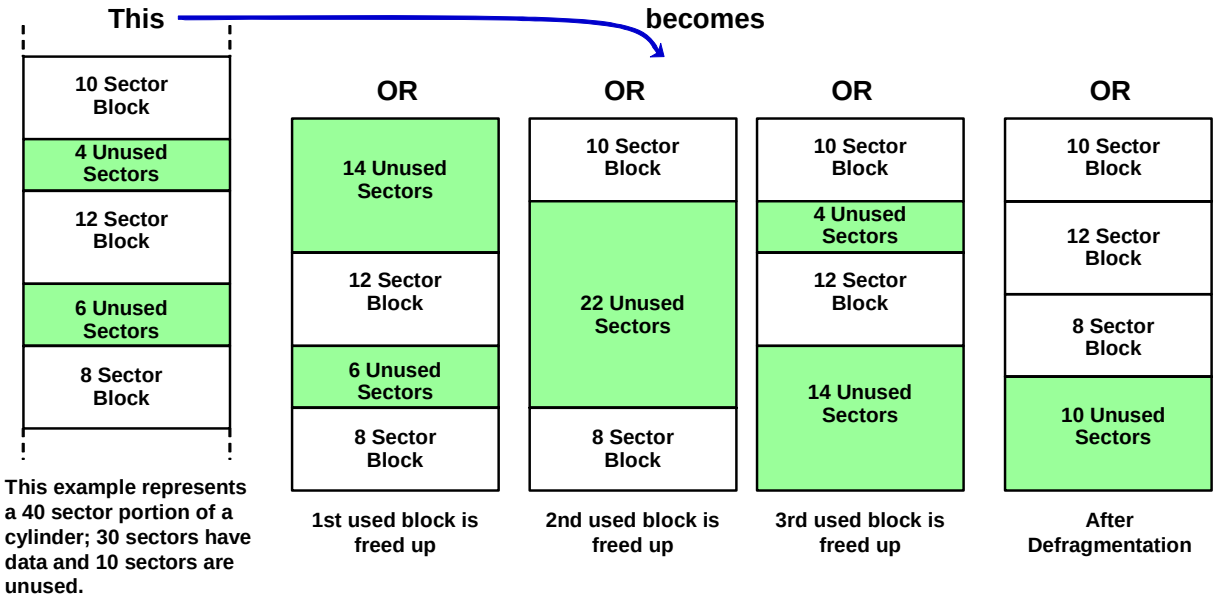
Second case – If the middle 12-sector data block is freed up, software will not place an entry on the free block list for just these 12 sectors, but will effectively combine these 12 sectors with the previous adjacent 4 free sectors and with the following 6 free adjacent sectors, effectively represented by one entry for 22 free sectors. For this 40-sector portion of a cylinder, there will be one entry on the free block list showing that 22 sectors that are free.

Third case – If the last 8-sector data block is freed up, software will not place an entry on the free block list for just these 8 sectors, but will effectively combine these 8 sectors with the previous adjacent 6 free sectors. One entry representing the 14 free sectors is placed on the free block list. For this 40-sector portion of a cylinder, there will be 2 entries on the free block list – one for the first 4 unused sectors and a second entry for the 14 unused sectors.

Fourth case – If there is no entry on the free block list large enough to meet a request for a new block, Teradata's file system software may choose to dynamically defragment the cylinder. In this case, all free sectors are combined together at the end of a new cylinder and one entry for the free space (sectors) is placed on the free block list. Defragmentation is actually done in the new cylinder and the existing cylinder is placed in the free cylinder list.

Space Fragmentation

- The system collects free blocks as a normal part of table maintenance.
- Smaller Free Blocks become larger when adjacent blocks become free, or when defragmentation is performed on the cylinder.



Cylinder Full

A **Cylinder Full** condition occurs when there is no block on the Free Block List that has enough sectors to accommodate additional data during an INSERT or UPDATE. If this condition occurs, the File System goes through the steps outlined on the facing page which results in a **Cylinder Migrate** to an existing adjacent cylinder or to a new cylinder**.** As part of this process, the file system software may also choose to perform a **Cylinder Defragmentation** or a **Mini Cylinder Pack (Mini-Cylpack)** operation.

- A **Mini-Cylpack** is a background process that occurs automatically when the number of free (or available) cylinders falls below a threshold. The mini-Cylpack process is the mechanism that Teradata uses to rearrange data blocks to free cylinders. This process involves moving data blocks from a data cylinder to the logically preceding data cylinder until a whole cylinder becomes empty.

- Mini-Cylpack is an indication that the system does not have enough free space to handle its current workload.

In the example at the bottom of the facing page, if Cylinder 37 became full, the File System would check Cylinder 204 and Cylinder 169 to see if they had enough room to perform a Cylinder Migrate. These two cylinders are logically adjacent to Cylinder 37 in the Master Index, but not necessarily physically adjacent on the disk.

During the Cylinder Migrate, if data blocks were moved to Cylinder 204, they would be taken from the top of Cylinder 37. If they were moved to Cylinder 169, they would be taken from the bottom of Cylinder 37.

Note:

Performance tests show that defragging can cause a significant performance hit. Therefore, the default tuning parameters that control how often you do this are set to only defragment cylinders if there are very few free cylinders left (<= 100) and the cylinder has quite a bit of free space that isn't usable (>= 25%). The latter indicates that, although there is significant free space on the cylinder, the free space is apparently so fragmented that a request for new sectors couldn't be satisfied. Otherwise, it's assumed that the cylinder is full and the overhead of defragging it wouldn't be worth it.

Cylinder Full

Cylinder Full means there is no block big enough on the Free Block List. The File System does either of the following:

- **Cylinder Migrate to an adjacent cylinder — checks logically adjacent cylinders for fullness. If it finds room, it moves a maximum of 10 data blocks from the full cylinder to an adjacent one.**
- **Cylinder Migrate to a new Cylinder — looks for a free cylinder, allocates one, and moves a maximum of 10 data blocks from the congested cylinder to a new one.**

While performing a Cylinder Migrate operation, the File System software may also do the following operations in the background.

- **Cylinder Defragmentation — if the total cylinder free space ≥ 25% of the cylinder size (25% is default), then the cylinder is defragmented. Defragmentation collects all free sectors at the end of a new cylinder by moving all the data blocks to the top of the new cylinder.**
- **Mini-Cylpack — if the number of free cylinders falls below a threshold (default is 10), then a "Mini-Cylpack" is performed to pack data together to free up a cylinder and place it on the free cylinder list.**

Master Index							Free Cylinder List	Free Cylinder List
Lowest			Highest			Pdisk and		
Table ID	Part #	Row ID	Table ID	Part #	Row Hash	Cylinder Number	Pdisk 0	Pdisk 1
:	:	:	:	:	:	:	:	:
078	0	58234, 2	095	0	72194	204	124	761
098	0	00107, 1	100	0	00676	037	125	780
100	0	00773, 3	100	0	01361	169	168	895
:	:	:	:	:	:	:	:	:

Mini-Cylpack

The **Mini-Cylpack** is the mechanism that Teradata uses to rearrange data blocks to free cylinders. The process involves moving data blocks from a data cylinder to the logically preceding data cylinder until a whole cylinder becomes empty.

- A Mini-Cylpack is an indication that the system does not have enough free space to handle its current workload.
- Excessive numbers of **Mini-Cylpacks** indicate too little disk space is available and/or too much spool is being utilized during data maintenance.
- Spool cylinders are never "Cylpacked".

Teradata has a **Free Space** (a percentage**)** parameter that can be set to control how much free space is left in a cylinder during loading and the use of the Ferret PackDisk utility. This parameter is not used with **mini-cylpacks**.

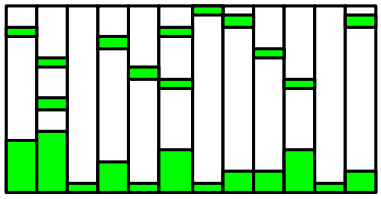
- This parameter should be set low (close to 0%) for systems which are used solely for Decision Support as there is no data maintenance involved.
- In cases where there is moderate data maintenance (batch or some OLTP), the **Free Space** parameter should be set at approximately 25%.
- If heavy data maintenance is to be done (OLTP), the **Free Space** parameter may have to be set at approximately 50% to prevent Cylpacks from affecting OLTP response times.

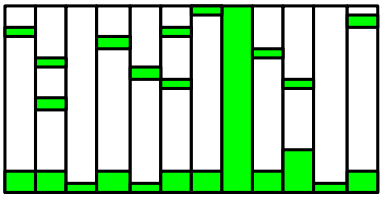
The **Free Space** parameter can be set at the system level, at a table level, and when executing the **Ferret PackDisk** utility.

- DBSControl – FREESPACEPERCENT (0% is the default)
- CREATE TABLE – FREESPACE = integer [PERCENT] (0 – 75)
- FERRET PACKDISK – FREESPACEPERCENT (or FSP) integer

The system administrator can specify a count of empty cylinders the system should attempt to maintain. Whenever a Cylinder Migrate to a new cylinder occurs, the system checks to see if the minimum number of empty cylinders still exists. If the system has dropped below the minimum, it starts a background task that begins packing cylinders. The task stops when either a cylinder is added to the Free Cylinder List or it has packed 10 cylinders. This process continues with every Cylinder Migrate to a new cylinder until the minimum count of empty cylinders is reached, or a full mini-cylpack is required.

Mini-Cylpack

BEFORE 

AFTER 

A Mini-Cylpack moves data blocks from the data cylinder(s) to logically preceding data cylinder(s) until a single cylinder is empty.

- **Spool cylinders are never cylpacked.**
- **Mini-Cylpacks indicate that the system does not have space to handle its current workload.**
- **Excessive Cylpacks indicate too little disk space and/or spool utilization during data maintenance.**

The *Free Space* parameter impacts how full a cylinder is filled with data loading and PackDisk.

- **DBSControl – FREESPACEPERCENT**
- **CREATE TABLE – FREESPACE**
- **FERRET PACKDISK – FREESPACEPERCENT (FSP)**

Space Utilization

The Teradata Database can use any particular cylinder to either store data or hold Spool files. A cylinder cannot be used for both data and Spool simultaneously. In sizing a system, you must make certain that you have enough cylinders to accommodate both requirements.

Limiting the number of rows and columns per query helps keep Spool requirements under control, as does keeping the number of columns per row to a minimum. Both can result from proper normalization.

Teradata 13.10 Auto Cylinder Pack Feature

One new background task in Teradata 13.10 is called AutoCylPack which attempts to combine adjacent, sparsely filled cylinders. These cylinder packs are typically executed when the system is idle.

AutoCylPack is particularly useful if a customer is using temperature-based BLC, because it cleans up post-compression cylinders that are no longer holding as much data. However, this feature works with compressed as well as uncompressed cylinders. Sometimes the activity of AutoCylPack can result in seeing a little bit of wait I/O (less than 5%).

File System Field 17 (DisableAutoCylPack) has a default value of FALSE, which means AutoCylPack is on and active all the time, unless you change this setting.

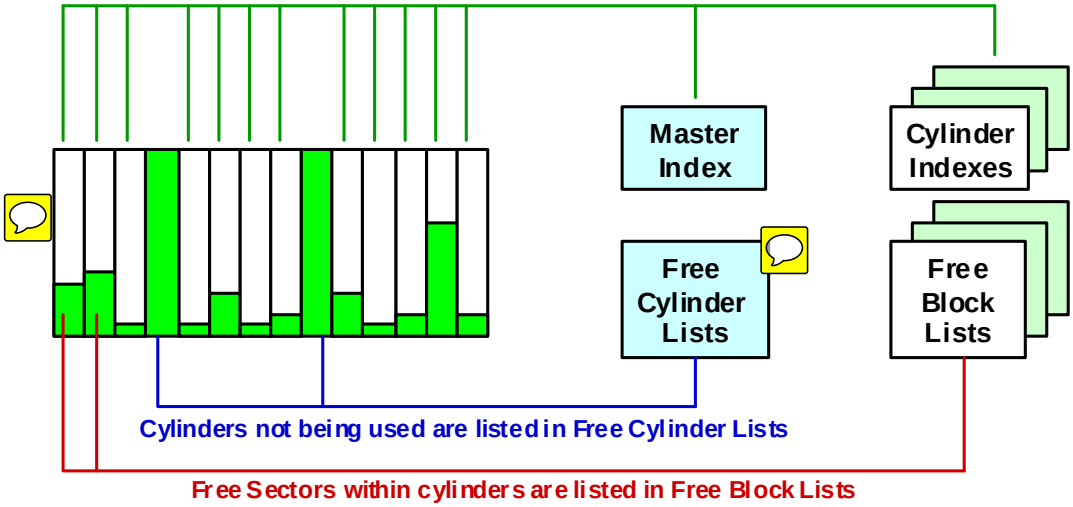
General notes:

There are a number of background tasks running in the Teradata database and AutoCylPack is another of these tasks. These tasks include deadlock detection, cylinder defragmentation, transient journal purging, periodic cache flushes, etc.

These tasks generally consume a small amount of system resources. However, you will tend to notice them more when the system is idle.

Space Utilization

Space being used is managed via Master Index and Cylinder Indexes



Cylinders contain Perm, Spool, Temporary, Permanent Journal, or WAL data, but NOT a combination.

BE SURE THAT YOU HAVE ENOUGH SPACE OF EACH.

Limiting the rows and columns per query reduces spool use.

Merge Datablocks (13.10 Feature)

This Teradata Database 13.10 feature automatically searches for “small” data blocks within a table and will combine (merge) these small datablocks into a single larger block. Over time, modifications to a table (especially with DELETEs of data rows) can result in a table having blocks that are less than 50% of the maximum datablock size. This File System feature combines these small blocks into a larger block.

The benefit is simply that future full table operations (full table scans and/or full table updates) will perform faster because fewer I/Os are performed. By having larger blocks in the Teradata file system, the selection of target rows can also be more efficient.

- Blocks that are 50% or greater of the maximum multi-row datablock size (63.5 KB in this example) are not considered to be small blocks. Small blocks are less than 50% of the maximum datablock size.

The merge of multiple small blocks into a larger block is limited by cylinder boundaries – does not occur between cylinders. A maximum of 7 logically adjacent preceding blocks can are merged together into a target block when the target block is updated. Therefore, a maximum of 8 total blocks can be merged together.

Why are logical following blocks NOT merged together?

- The File System software does not know if following blocks are going to be immediately updated.
- Reduces performance impact during dense sequential updates

How does a table get to the point of having many small blocks? DELETEs from this table can cause blocks to permanently shrink to a much smaller size unless a large amount of data is added again.

How have customers resolved this problem before Teradata 13.10?

- The ALTER TABLE command can be used to re-block a table. This technique can be time consuming and requires an exclusive table lock. This technique is still available with Teradata 13.10.

  – ALTER TABLE DATABLOCKSIZE = <value> IMMEDIATE

If this featured is enabled, the merge of small data blocks into a larger block runs automatically during full table SQL write operations. This feature can merge datablocks for the primary/fallback subtables and all of the index subtables. This feature runs automatically when the following SQL functions are executed.

- INSERT-SELECT, UPDATE-WHERE
- DELETE-WHERE (used on both permanent table and permanent journal datablocks)
- During the DELETE phase of Reconfig utility on source amps

Merge Datablocks (Teradata 13.10)

This Teradata Database 13.10 feature automatically searches for "small" data blocks within a table and will combine (merge) these small datablocks into a single larger block.

- **Over time, modifications to a table (especially with DELETEs of data rows) can result in a table having blocks that are less than 50% of the maximum datablock size.**
- **Up to 8 datablocks can be merged together.**

If enabled, the merge of small data blocks into a larger block runs automatically during full table SQL write operations. This feature can merge datablocks for the primary/fallback subtables and all of the index subtables.

- **INSERT-SELECT**
- **UPDATE-WHERE**
- **DELETE-WHERE**

How have customers resolved this problem before Teradata 13.10?

- **The ALTER TABLE command can be used to re-block a table. This technique can be time consuming and requires an exclusive table lock. This technique is still available with Teradata 13.10.**

```
ALTER TABLE DATABLOCKSIZE = <value> IMMEDIATE;
```

Merge Datablocks (Teradata 13.10) cont.

How to use this Feature

Defaults for this feature can be set at the system level via DBSControl settings and can be overridden with table level attributes. The CREATE TABLE and ALTER TABLE commands have options to enable or disable this feature for a specific table.

The key parameter that controls this feature is **MergeBlockRatio**. This parameter can be set at the system level and also as a table level attribute.

MergeBlockRatio has the following characteristics:

- Limits the resulting size of a merged block.
- Reduces the chances that a merged block will split again soon after it is merged, defeating the feature's purpose.
- Computed as a percentage of the maximum multi-row datablock size for the associated table.
- Candidate merged block must be smaller than this computed size after all target row updates are completed.
- Source blocks are counted up as eligible until the size limit is reached (zero to 8 blocks can be merged together).
- The default system level percentage is 60% and can be changed.

CREATE TABLE or ALTER TABLE options

- DEFAULT MERGEBLOCKRATIO
  - Default option on all CREATE TABLE statements
- MERGEBLOCKRATIO = integer [PERCENT]
  - Fixed MergeBlockRatio used for full table modification operations
  - Overrides the system default value
- NO MERGEBLOCKRATIO
  - Disables merges completely for the table

DBSControl FILESYS Group parameters

25. DisableMergeBlocks (TRUE/FALSE, default FALSE)
    - Disables feature completely across the system, even for tables with a defined MergeBlockRatio as a table level attribute.
    - Effective immediately – does not require a Teradata restart (tpareset)

26. MergeBlockRatio (1-100%, default 60%)
    - Default setting for any table – this can be overridden at the table level.
    - Ignored when DisableMergeBlocks is TRUE (FILESYS Flag #25)
    - This is not stored in or copied to table header
    - Effective immediately without a tpareset

Merge Datablocks (Teradata 13.10) cont.

This feature is automatically enabled for new Teradata 13.10 systems, but must be enabled for existing systems upgraded to 13.10.

Defaults for this feature are set via DBSControl settings.

- System defaults will work well for most tables.
- The CREATE TABLE and ALTER TABLE commands have options to enable/disable this feature for a specific table or change the default ratio.
- The key parameter is MergeBlockRatio.

MergeBlockRatio has the following characteristics:

- The default system level percentage is 60%.
- Computed as a percentage of the maximum multi-row datablock size for the associated table.
- Candidate merged block must be smaller than this computed size after all target row updates are completed.

CREATE TABLE or ALTER TABLE options

- DEFAULT MERGEBLOCKRATIO
  - Default option on all CREATE TABLE statements
- MERGEBLOCKRATIO = integer [PERCENT]
  - Fixed MergeBlockRatio used for full table modification operations
- NO MERGEBLOCKRATIO
  - Disables merges completely for the table

File System Write Summary

Regardless of how large your tables get, or how many SQL-based INSERTs, UPDATEs or DELETEs are executed, the process is the same. This module has discussed in some detail the sequence of steps that Teradata's file system software will attempt in order to complete the write operation.

The facing page summarizes some of the key topics discussed in this module.

File System Write Summary

Teradata's file system software automatically maintains the logical sequence of data rows within an AMP.

- **The logical sequence is based on tableId, partition #, and rowid.**

For write (INSERT, UPDATE, or DELETE) operations:

- **Read the Data Block if not present in memory.**
- **Place appropriate entries into the Transient Journal buffer (WAL buffer).**
- **Make the changes to the Data Block in memory and determine the new block's length.**
- **If the new block has changed size, allocate a new Data Block.**

Blocks will grow to the maximum data block size determined by the DATABLOCKSIZE table attribute, and then be split into smaller blocks.

- **Blocks will vary in size with Teradata.**
- **For a table that has been updated with "ad hoc" or TPump INSERTs, UPDATEs, or DELETEs, a typical block size for the table will be approximately 75% of the maximum data block size.**

If the Write operation fails, the file system does a rollback using the Transient Journal.

Module 14: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 14: Review Questions

1. When Teradata INSERTs a new row into a table, it first goes to the _____ to locate the proper cylinder for the new row.
   a. Cylinder Index
   b. Fallback AMP
   c. Free Cylinder List
   d. Master Index

2. When a new block is needed, the File System searches the Free Block List looking for the first Free Block whose size is equal to, or greater than the new block's requirement. It does not have to be an exact match.
   a. True
   b. False

3. Name the condition which occurs when there is no block on the Free Block List with enough sectors to accommodate the additional data during an INSERT or UPDATE.
   a. Mini Cylinder Pack
   b. Cylinder Migrate to a new cylinder
   c. Cylinder Migrate to an adjacent cylinder
   d. Cylinder Full

4. The _____ parameter can be set to control how completely cylinders are filled during loading and PackDisk.
   a. Free Space Percent
   b. DataBlockSize
   c. PermDBSize
   d. PermDBAllocUnit

Module 14: Review Questions (cont.)

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 14: Review Questions (cont.)

5. Number the following steps in sequence from 1 to 6 that the File System software will attempt to perform in order to insert a new row into an existing data block.

   ____ Perform a Cylinder Migrate operation to an adjacent cylinder

   ____ Simply insert the row into data block if enough contiguous free bytes in the block

   ____ Perform a Block split

   ____ Perform a Cylinder Migrate operation to a new cylinder

   ____ Defragment the block and insert the row

   ____ Expand or grow the block to hold the row

6. As part of a cylinder full condition, if the number of free sectors within a cylinder is greater than 25%, what operation will Teradata perform in the background? 

7. If the number of free cylinders falls below a minimum threshold, what operation will Teradata perform in the background? 

Notes

Module 15

Teradata SQL Assistant

After completing this module, you will be able to:

- **Define an ODBC data source for Teradata.**
- **Submit SQL using SQL Assistant.**
- **Utilize Explorer Tree to simplify creation of queries.**
- **Use SQL Assistant to import/export a LOB.**

Notes

Table of Contents

SQL Assistant 15-4
Defining a Data Source 15-6
    Compatibility 15-6
    Defining a Teradata .Net data source 15-6
Defining a Data Source (cont.) 15-8
  Defining an ODBC Data Source 15-8
Defining a Data Source (cont.) 15-10
  ODBC Driver Setup for LOBs 15-10
Connecting to a Data Source 15-12
Main Window 15-14
Database Explorer Tree 15-16
Creating and Executing a Query 15-18
    Creating statements (single and multi-queries) 15-18
Dragging Object Names to the Query Window 15-20
    Dragging Multiple Objects 15-20
Query Options 15-22
  To submit any part of any query 15-22
  Clearing the Query Window 15-22
  Formatting a Query 15-22
Viewing Query Results 15-24
  Sorting an Answerset Locally 15-24
Formatting Answersets 15-26
Using Query Builder 15-28
    Description of the Options 15-28
History Window 15-30
General Options 15-32
Connecting to Multiple Data Sources 15-34
Additional Options 15-36
Importing/Exporting Large Object Files 15-38
  Teradata SQL Assistant 12.0 Note 15-38
Importing/Exporting Large Object Files 15-40
To Import a LOB into Teradata 15-40
Selecting from a Table with a LOB 15-42
Displaying a JPG within SQL Assistant 15-44
Teradata SQL Assistant Summary 15-46
Module 15: Review Questions 15-48
Lab Exercise 15-1 15-50
  Lab Exercise 15-1 (cont.) 15-52
  Lab Exercise 15-1 (cont.) 15-56

SQL Assistant

Teradata SQL Assistant is an information discovery tool designed for the Windows operating system (e.g., Windows 7). Teradata SQL Assistant retrieves data from any ODBC-compliant database server. The data can then be manipulated and stored on the desktop PC.

Teradata SQL Assistant is a query tool written for relational database developers. It is intended for SQL-proficient developers who know how to formulate queries for processing on Teradata or other ODBC-compliant Databases. Used as a discovery tool, Teradata SQL Assistant catalogs submitted instructions to arrive at a derived answer. Teradata SQL Assistant stores the history of your SQL in a local Microsoft Access database table. This history is available in future executions of Teradata SQL Assistant.

Teradata SQL Assistant accepts standard Teradata SQL, DDL, and DML. In addition, Teradata SQL Assistant sends native SQL to any other database that provides an ODBC driver. If the driver supports the statements, they are processed correctly.

Key features of SQL Assistant include:

- Create reports from any Relational Database that provides an ODBC interface
- Export data from the database to a file on a PC
- Import data from a PC file directly to the database
- Use an import file to create many similar reports (query results or Answer sets).
- Send queries to any supported database or the same query to many different databases
- Create a historical record of the submitted SQL with timings and status information such as success or failure
- Use the Database Explorer Tree to easily view database objects
- Use a procedure builder that gives you a list of valid statements for building the logic of a stored procedure
- Limit data returned to prevent runaway queries

Teradata SQL Assistant also benefits database administrators by allowing them to directly issue SHOW statements to view text for CREATE or REPLACE commands. The DBA copies the text to the Query window, uses the Replace function to change a database name, and reissues the CREATE or REPLACE to define a new object with this new name. You can also display the CREATE text by going to the shortcut menu of the Database Explorer Tree and clicking Show Definition.

SQL Assistant Features

SQL Assistant is a Windows-based utility for submitting SQL to Teradata.

SQL Assistant has the following properties:

- Windows-based
- Two providers are available for Teradata connections:
  - Teradata ODBC Driver
  - Teradata .Net Data Provider
- Can be used to access other supported ODBC-compliant databases.
- Permits retrieval of previously used queries (History).
  - Saves information about previous query result sets.
- Supports DDL, DML and DCL commands.
  - Query Builder feature allows for easy creation of SQL statements.
- Provides both import and export capabilities to files on a PC.
- Provides a Database Explorer Tree to easily view database objects.
- Does not support non-ODBC compliant syntax such as WITH BY and FORMAT.
- Teradata Studio Express is a newer name for SQL Assistant Java Edition.
  - Targeted to Java developers who are familiar with Eclipse

Defining a Data Source

Before using Teradata SQL Assistant to access the Teradata database, you must first install the Teradata ODBC driver on your PC and the .Net Data Provider for Teradata. When connecting to a Teradata database, you can use either ODBC or the .Net Data Provider for Teradata.

Connection to any other database must be made through an ODBC connection. In order to use the ODBC connection, a vendor specific ODBC driver must be installed.

Before you can use Teradata SQL Assistant, you will need to define a "data source", namely the instance of the database you wish to work with.

Compatibility

Teradata SQL Assistant is certified to run with any Level 2 compliant 32-bit ODBC driver. The product also works with Level 1 compliant drivers, but may not provide full functionality. Consult the ODBC driver documentation to determine the driver's conformance level. Most commercially available ODBC drivers conform to Level 2.

Defining a Teradata .Net data source

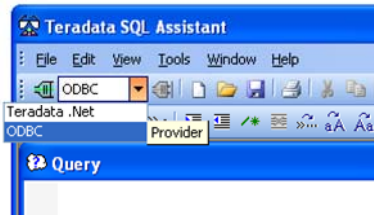
Use the Connection Information dialog to create, edit and delete data sources for .Net for Teradata. This dialog box is also used to connect to a .Net data source.

To define a Teradata .Net data source

1. Open Teradata SQL Assistant.
2. Select Teradata .Net from the provider drop down list.
3. Click the Connect icon or go to Tools > Connect.
4. Use the Connection Information dialog to choose a .Net data source.
5. Create a new data source by entering the name and server and other applicable information

Note: This module will illustrate the screens defining an ODBC data source. The specific screens defining a Teradata .Net data source are not provided in this module, but are similar

Defining a Data Source



You can define an ODBC data source in these ways:

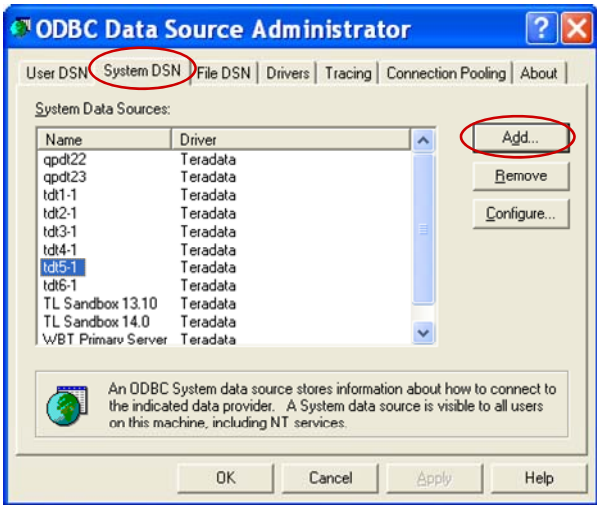
- **SQL Assistant (select Connect icon)**
- **Select Tools > Define ODBC Data Source**
  or
- **ODBC Data Source Administrator Program**

SQL Assistant has 2 provider options:

- **Teradata .Net Data Provider**
- **ODBC**

Select the System DSN tab and click on Add to create a new data source.

If using ODBC Administrator (not shown), select the Machine Data Source tab and click on Add to create a new data source.



Defining a Data Source (cont.)

When connecting to the Teradata database, use either the ODBC or the Teradata .Net Data Provider. Connection to any other database must be made through an ODBC connection.

Defining an ODBC Data Source

An ODBC-based application like Teradata SQL Assistant accesses the data in a database through an ODBC data source.

After installing Teradata SQL Assistant on a workstation or PC, start Teradata SQL Assistant. Next, define a data source for each database.

The Microsoft ODBC Data Source Administrator maintains ODBC data sources and drivers and can be used to add, modify, or remove ODBC drivers and configure data sources. An About Box for each installed ODBC driver provides author, version number, module size, and release date.

To define an ODBC data source, do one of the following:

- From the Windows desktop, select …
  Start > Control Panel > Administrative Tools > Data Sources (ODBC)

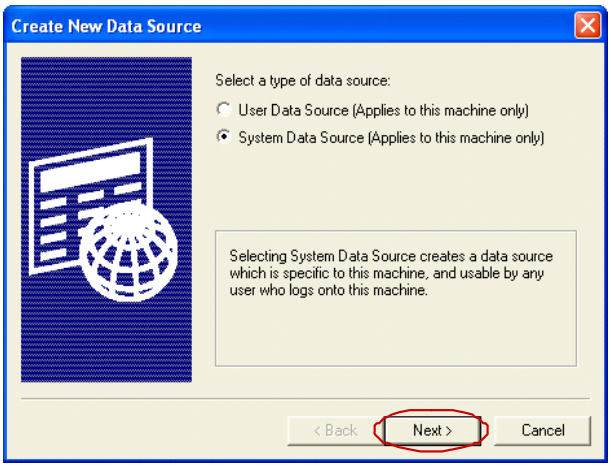
- From the Windows desktop, select
  Start > Programs > Teradata SQL Assistant
  After SQL Assistant launches, select Tools > Define Data Source

- Use the Connect icon from SQL Assistant and complete the dialog boxes.

In the "Define Data Source" dialog, decide what type of data source you wish to create:

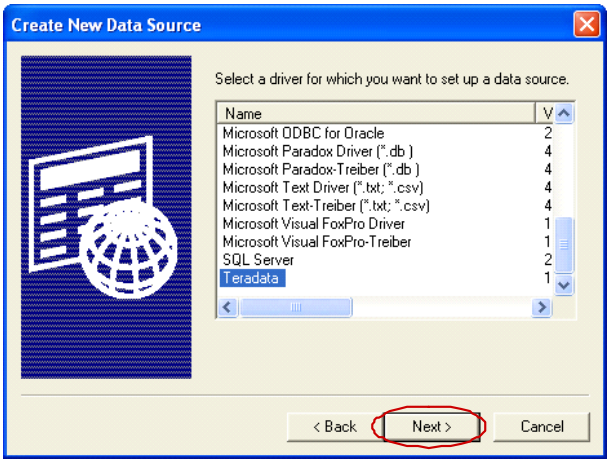
Data Source Description	Explanation
A **User Data Source** can be used only by the current Windows user	An ODBC user data source stores information about how to connect to the indicated data provider. A user data source is only visible to you.
A **System Data Source** can be used by any user defined on your PC.	An ODBC system data source stores information about how to connect to the indicated data provider. A system data source is visible to all users on this machine, including NT services.

Defining a Data Source (cont.)



If using ODBC Administrator, you will be given the user/system data source screen as shown to the left.

You will not get this display if defining your ODBC data source via SQL Assistant.



Select Teradata as the driver

and click Finish on the confirmation screen.

Defining a Data Source (cont.)

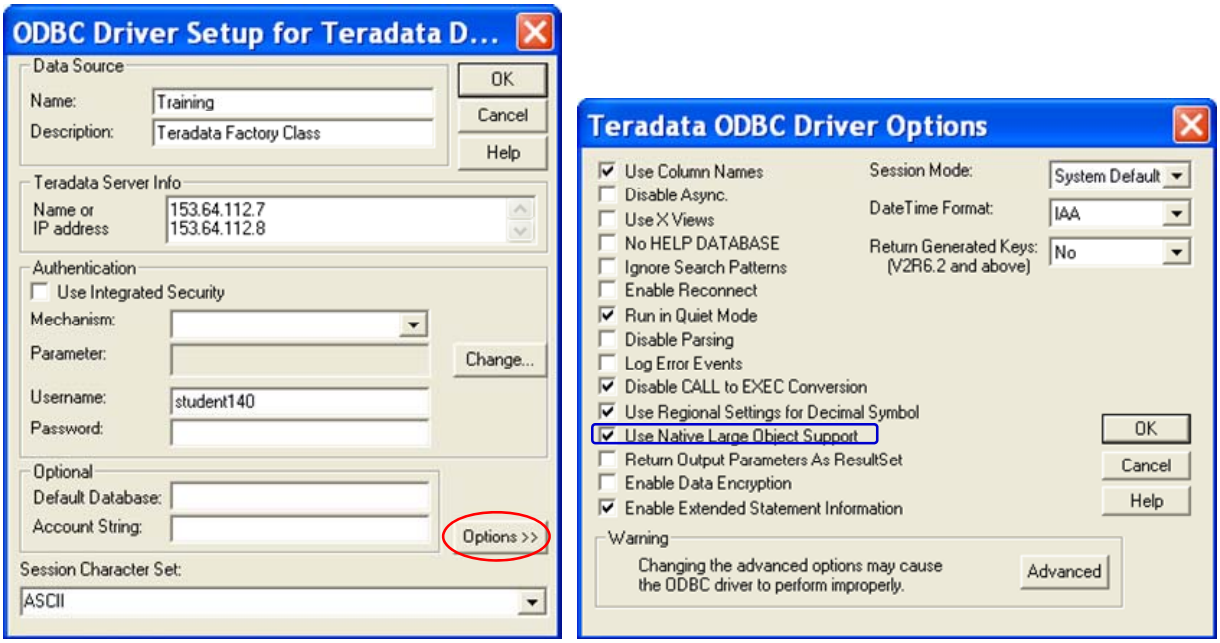
A dialog box (specific to Teradata) is used to define the Teradata system you wish to access.

Select This Field...	To...
Name	Enter a name that identifies this data source. You can also enter the name of the system or the logon you will be using.
Description	Enter a description. This is solely a comment field to describe the data source name you used.
Name(s) or IP address(es)	Enter the name(s) or IP address(es) of the Teradata Server of your Teradata system. Identify the host by either name (alias) or IP address. The setup routine automatically searches for other systems that have similar name aliases. Multiple server names may be entered by pulling the entries on separate lines within this box.
Do not resolve alias name to IP address	When this option is checked, setup routine does not attempt to resolve alias names entered into the "Name(s) and IP address(es)" box at setup time. Instead it will be resolved at connect time. When unchecked, the setup routine automatically appends COPn (where n = 1, 2, 3, ..., 128) for each alias name you enter.
Use Integrated Security	Select this option if will be logging on using integrated security measures.
Mechanism	Select from the list of mechanisms that automatically appear in this box. Leave this field blank to use the default mechanism.
Parameter	The authentication parameter is a password required by the selected mechanism.
Username	Enter a user name.
Password	Enter a password to be used for the connection if you intend to use Teradata SQL Assistant in an unattended (batch) mode. Entering a password here is not very secure and is normally not recommended.
Default Database	Enter the default database you want this logon to use. If the **Default Database** is not entered, the **Username** is used as the default.
Account String	You can optionally enter one of the accounts that assigned to your Username.
Session Character Set	Use the drop down menu to choose the character set. The default is ASCII.

ODBC Driver Setup for LOBs

When defining the ODBC Data Source, from the ODBC Driver Setup screen, use the Options button to display the Teradata ODBC Driver Options screen and verify that the option - **Use Native Large Object Support –** is checked**.**

Defining a Data Source (cont.)



To access LOBs with SQL Assistant, …

1) **Click on the Options button.**
2) **Verify that "Use Native Large Object Support" option box is checked.**

Connecting to a Data Source

Connecting to a data source is the equivalent of "logging on" with SQL Assistant. You may choose from any previously defined data source.

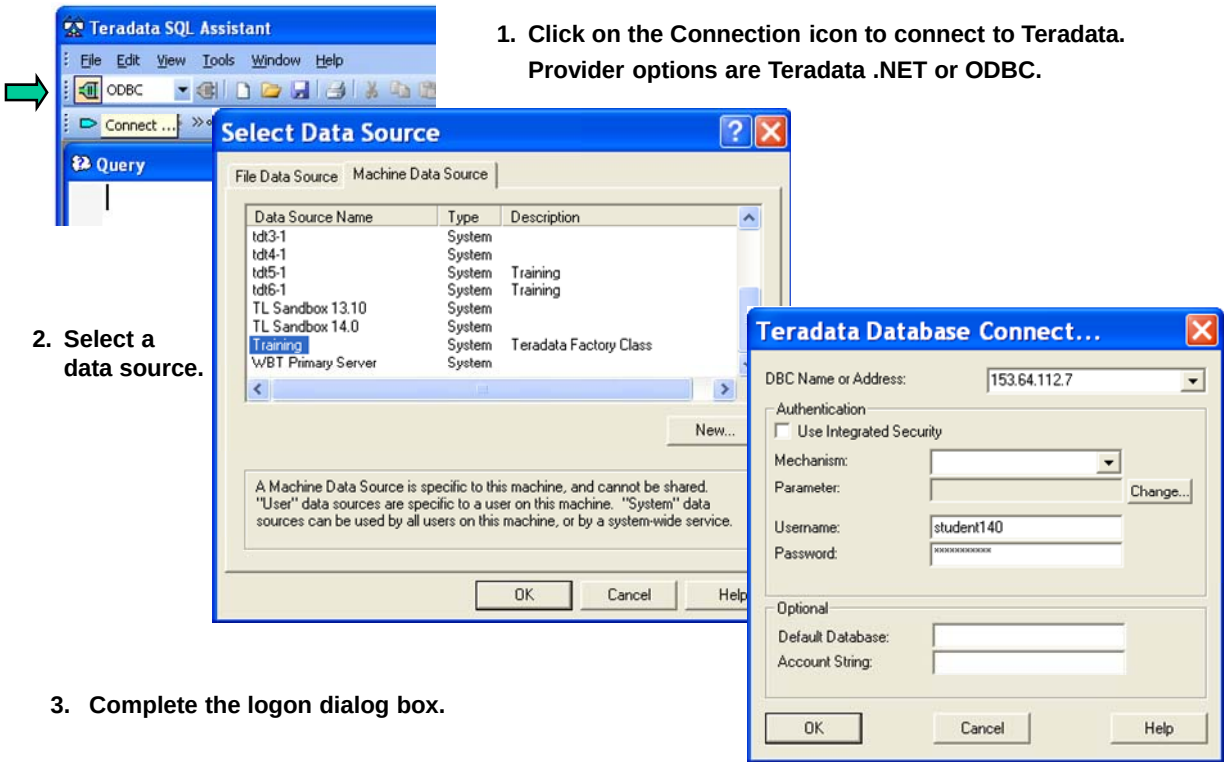
When the connection is complete, the **Connect** icon is disabled and the **Disconnect** icon, to its right, is enabled.

To connect to multiple data sources:

1. Go to the Tools > Options > General tab.
2. Click Allow connections to multiple data sources (Query windows),
3. Follow the procedure for connecting to a data source.

Each new data source appears in the Database Explorer Tree and opens a new query window with the data source name. To disconnect from one data source, click the Query window that is connected to the data source and click the disconnect icon.

Connecting to a Data Source



1. **Click on the Connection icon to connect to Teradata. Provider options are Teradata .NET or ODBC.**
2. **Select a data source.**
3. **Complete the logon dialog box.**

Main Window

The Query window is where you enter and execute a query. The results from your query are placed into one or more Answerset windows.

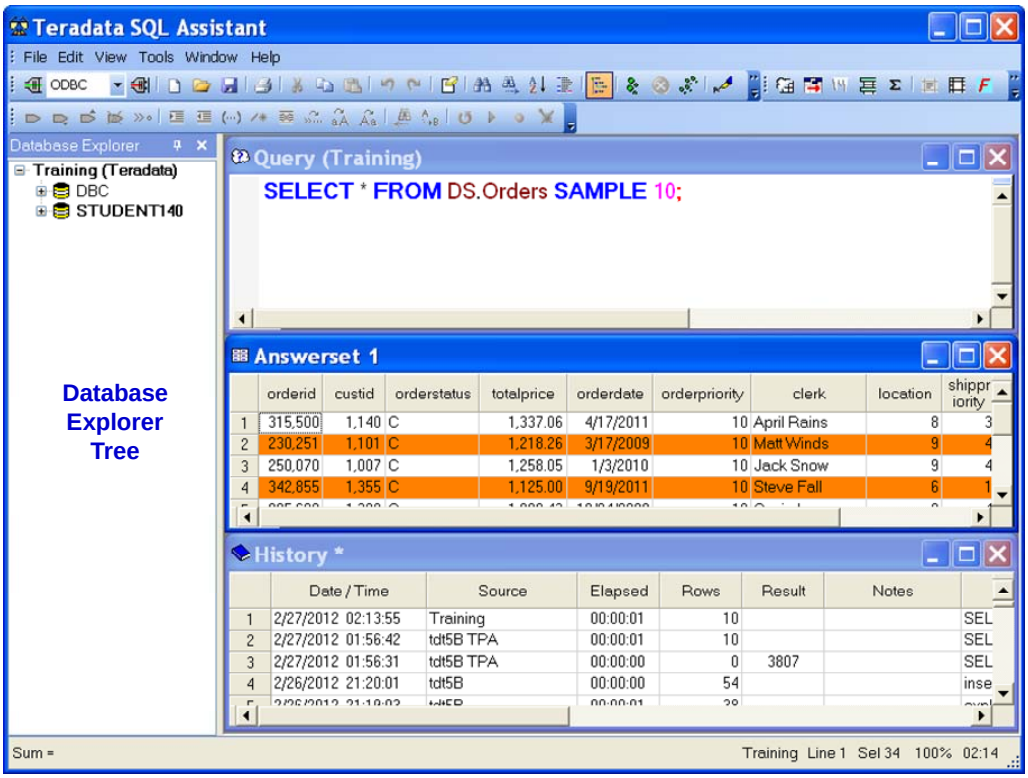
The Answerset window is a table Teradata SQL Assistant uses to display the output of a query.

The History window is a table that displays your past queries and related processing attributes. The past queries and processing attributes are stored locally in a Microsoft Access database. This gives you flexibility to work with previous SQL statements in the future.

The Database Explorer Tree displays on the left side of the main Teradata SQL Assistant window. It displays an alphabetical listing of databases and objects in the connected Teradata server. You can double-click on a database name to expand the tree display for that database.

You can use the Database Explorer Tree to reduce the time required to build a query and help reduce errors in object names. The Database Explorer Tree is optional so you can display or hide this window.

Main Window



Database Explorer Tree

Query Window

Answerset Window

History Window

Database Explorer Tree

The Database Explorer Tree feature of Teradata SQL Assistant displays an alphabetical listing of databases and objects of the connected user. It further permits drilldown on individual objects to view, column names, indexes and parameters as they apply. This is simply done by double-clicking on a database name to expand the tree display for that database.

The Database Explorer Tree displays on the left side of the main Teradata SQL Assistant window. You can use the Database Explorer Tree to reduce the time required to build a query and help reduce errors in object names. The Database Explorer Tree is optional so you can display or hide this window.

Initially, the following Teradata databases are loaded into the Database Explorer Tree:

- The User ID that was used to connect to the database
- The user's default database
- The database "DBC"

To add additional databases:

1. Do one of the following:
   - With the Database Explorer Tree active, press `Insert`.
   - Right-click anywhere in the Database Explorer Tree, then select **Add Database**.
2. Type the database name to be added.

3. If you want the database loaded only for the current session, clear the check box. By default, the check box is selected so the database will appear in the Database Explorer Tree in future sessions.

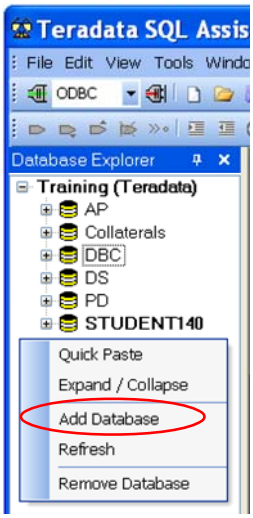
The Database Explorer Tree allows you to drill down to show:

- Columns and indexes of tables
- Columns of views
- Parameters of macros
- Parameters of stored procedures

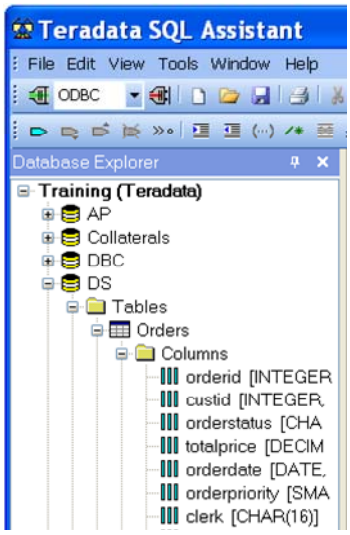
Explorer Tree Option

- The Database Explorer Tree displays an alphabetical listing of databases and objects of the connected user.
  - It is not a database hierarchy, but a list of databases and objects that the user needs to access.
- To refresh a database, right-click on the database/user name and select "Refresh".

To add another database to the Explorer Tree, right-click on the Explorer Tree.



To expand an item/object, click on the + sign or double-click on the object name.



Creating and Executing a Query

Queries are created by simply typing in the query text into the query window. It is not necessary to add a semi-colon at the end of a command, unless you are entering multiple commands in a single window. The query may be executed by clicking on the 'Execute' icon in the toolbar. This icon looks like a pair of footprints.

"Execute" actually executes the statements in the query one statement after the other and optionally stops if one of the statements returns an error. Function key F5 can also be used to execute queries serially.

"Execute Parallel" executes all statements at the same time - and is only valid if all the statements are Teradata SQL/DML statements. This submits the entire query as a single request, allowing the database to execute all the statements in parallel. Multiple answer sets are returned in the Answerset window. Function key F9 can also be used to execute queries in parallel.

Creating statements (single and multi-queries)

To allow multiple queries:

1. Select Tools > Options.
2. Select the General option.
3. Select the option "Allow Multiple Queries".

Once this option is selected, you may open additional tabs in the query window. Each tab can contain a separate query, and any of these queries can be executed. However, only one query can be executed at a time.

You can create queries consisting of one or more statements.

A semicolon is not required when you enter one statement at a time. However, a semicolon between the statements is required for two or more statements.

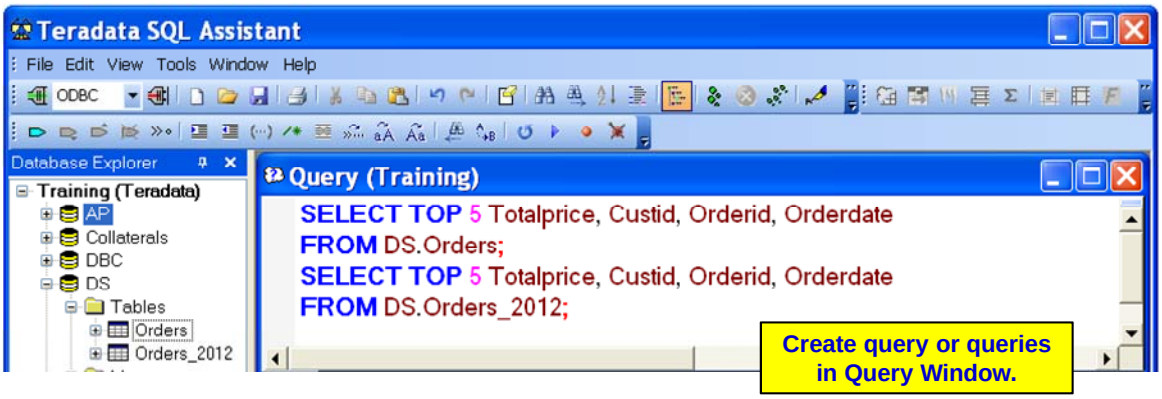
Each statement in the query is submitted separately to the database; therefore, your query may return more than one Answerset.

Creating and Executing a Query

1. Create a query in the Query Window.
2. To execute a query use either the "execute" or the "execute parallel" buttons.

   The "execute" button  (or F5) serially executes all statements in the query window.

   The "execute parallel" button  or (F9) executes all statements in the query window in a single multi-statement request. These queries are effectively executed in parallel.



Dragging Object Names to the Query Window

You can drag object names from the Database Explorer tree to the Query pane.

Click and drag the object from the Explorer tree to the Query pane. The name of the object appears in the Query window.

Teradata SQL Assistant includes an option (Tools > Options) that allows objects to automatically be qualified when dragging or pasting names from the Database Tree into the Query Window.

For example, if this option is checked, dragging the object "MyColumn" adds the parent object "MyTable", and appears as "MyTable.MyColumn" in the Query Window.

Use the Ctrl key to add a comma after the object name when it is dragged to the Query Window.

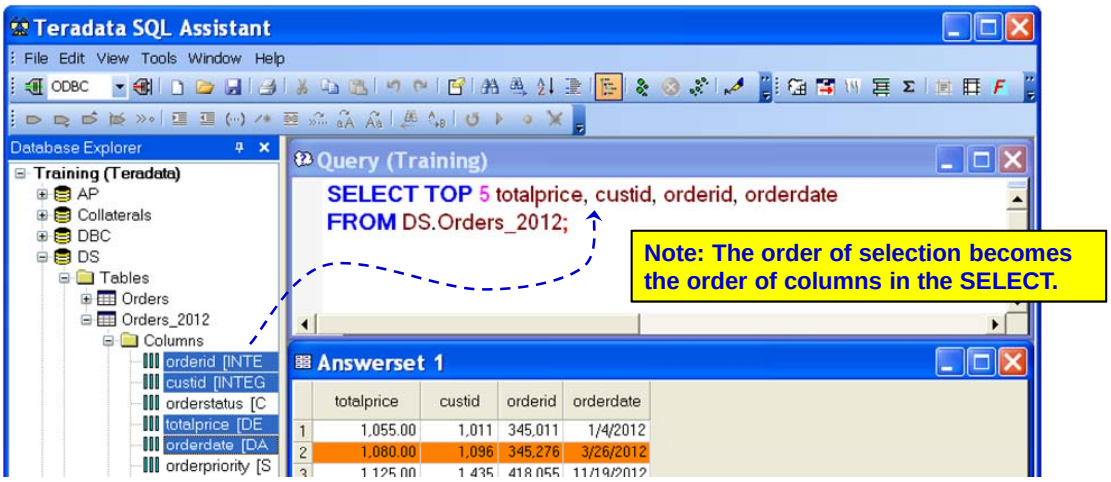
Dragging Multiple Objects

Use the Shift and Ctrl keys to select more than one object from the Database Explorer Tree that can be dragged to the Query window.

- Use the Ctrl key to select additional objects.
- Use the Shift key to select a range of objects.

Dragging Object Names to the Query Window

- Click and drag the object from the Database Explorer tree to the Query window. The name of the object appears in the Query window.
  - If the "Qualify names when dragged or pasted from the Database Tree" option (Tools > Options) is checked, then the parent name is automatically included.
  - Hold Ctrl key – causes a comma to be included after the object
- Selecting and dragging multiple objects
  - The Shift and Ctrl keys can also be used to select multiple objects in the Database Explorer tree for the purpose of dragging multiple objects to the Query Window.



Query Options

To submit any part of any query

1. Select Tools > Options.
2. Select the Query tab.
3. Check the option “Submit only the selected Query text, when highlighted”.
4. From the Query window, select the part of the query to submit by highlighting it.

Clearing the Query Window

The query window may be cleared using the “Clear Query” button on the tool bar.

Formatting a Query

The query formatting feature adds line breaks and indentation before certain keywords, making SQL that comes from automatic code generators or other sources more readable.

To Format a Query

1. Ensure a statement exists in the Query window.
2. Do one of the following:
   - From the Tool Bar, click the Format Query button.
   - Right-click in the Query window, then click **Format Query**
   - Press `Ctrl+Q`
   - Select **Edit** > **Format Query**

   Note: Some keywords will cause a line break and possibly cause the new line to be indented. If a keyword is found to already be the first word on a line and it is already prefixed by a tab character, then its indentation level will not change.

Indentation

When you press the `Enter` key, the new line will automatically indent to the same level as the line above.

If you highlight one or more lines in the query and press the `Tab` key, those lines are indented one level. If you press `Shift-Tab`, the highlighted lines are un-indented by one level.

This indentation of lines will only apply if the selected text includes a line feed character. For example, you must either select at least part of two lines, or if selecting only one line, then the cursor must be at the beginning of the next line. (Note that this is always the case when you use the margin to select a line.) If no line end is included in the selected text, or no text is selected, then a tab character will simply be inserted.

Query Options

To submit any part of a query:

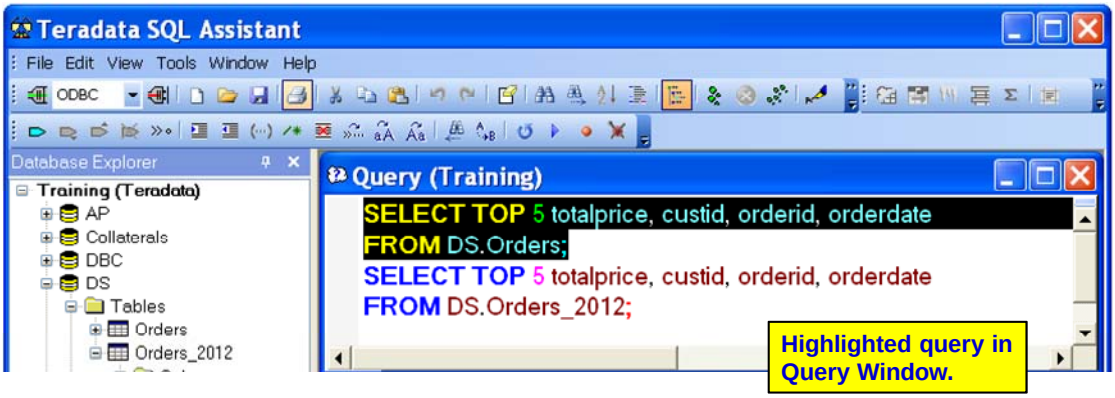
1. **Using Tools > Options > Query**

   Check the option "Submit only the selected Query text, when highlighted".

2. **Highlight the text in the query window and execute.** 

To clear the text in the query window, use the "Clear Query" button. 

To format a query, click on the "Format Query" button. 



Viewing Query Results

The results of a query execution may be seen in the Answer Set window. Large answer sets may be scrolled using the slide bars.

The Answerset window is a table that displays the results from a statement. You can sort the output in a number of ways and print as bitmaps in spreadsheet format. Individual cells, rows, columns, or blocks of columns may be formatted to change the background and foreground color as well as the font style, name, and size. You can make other modifications such as displaying or hiding gridlines and column headers.

The table may be resized by stretching the Answerset window using standard Windows sizing techniques. Individual columns, groups of columns, rows, or groups of rows may also be sized.

Output rows may be viewed as they are being retrieved from the database.

Sorting an Answerset Locally

There are two ways to sort an Answerset locally: **quick sort** or **full sort**. A quick sort sorts on a single column; a full sort allows sorting by data in multiple columns.

To sort an Answerset using quick sort:

- Right-click any column heading to sort the data by that column only. The data is initially sorted in ascending order. Right-click the same column header again reverses the sort order.

Note: The output from certain statements (e.g., `EXPLAIN`) cannot be sorted this way.

To sort an Answerset using a full sort:

- Do one of the following: From the Tool Bar, click the **sort** button, right-click in the Answerset window and select **Sort**, or use the Edit > Sort menu.

  In the Sort Answerset dialog box, all columns in the active window are presented in the **Available Columns** list box.

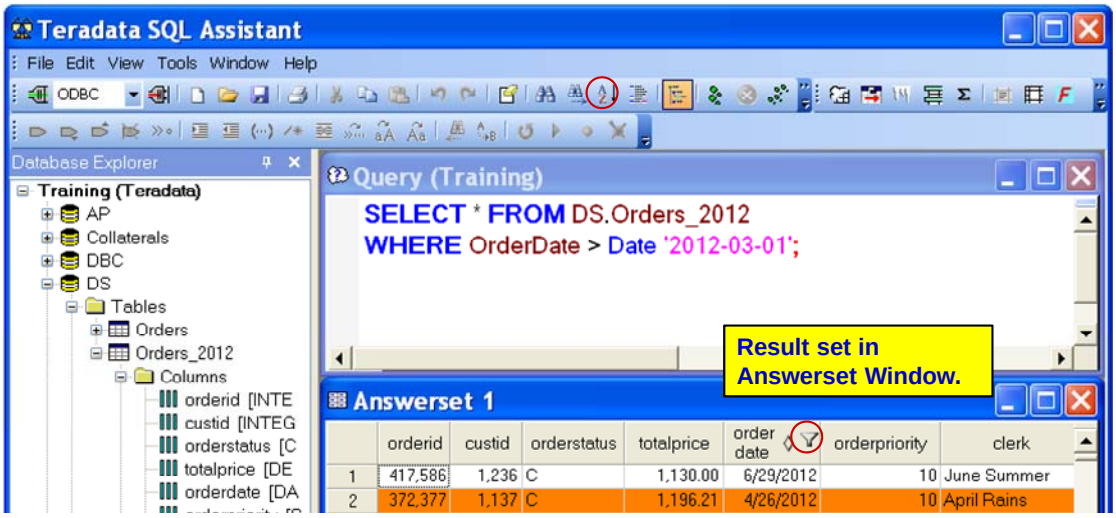
- Select the column name in the **Available Columns** list box, or use the up or down arrow keys to highlight the column name and press `Enter`.

  This moves the column name to the **Sort keys** list box. By default, the sort direction for this new sort column is ascending (Asc). If you click a column in the **Sort Keys** list box, or select the item using the arrow keys or mouse and press `Enter`, it reverses to descending sort order (Dsc).

To remove a sort column from the list, double-click the column name, or use the arrow keys to highlight the column and press `Delete`.

Viewing Query Results

- The Answerset window is a table that displays the results from a statement.
- The output can be sorted in different ways:
  - Quick sort (single column) – right click on the column heading
  - Full sort (1 or more columns) – use Edit > Sort menu or Sort button
- Data can be filtered using the funnel option at the column level.



Formatting Answersets

You can format the colors, font name, font style, and font size of a block of cells, individual cells, rows, columns, or the entire spreadsheet. You can also specify the number of decimal places displayed and if commas are displayed to mark thousand separators in numeric columns.

You can control the Answerset and the Answerset window by setting options. To set Answerset options, select Tools > Options > Answerset tab.

For example, to display Alternate Answerset Rows in Color, check and first option in the Answerset tab, and use the Choose button.

- Selecting this option makes it easier to see Answerset rows. The option applies the selected background color to alternating rows in the Answerset grid. The remaining rows use the standard 'Window Background' color.

- The **Choose** button displays the selected color. Clicking the **Choose** button allows you to change this color.

To format the colors, font name, font style, and font size of a block of specific cells individual cells, rows, columns, you right-click on the answer set cells. Some options are listed below.

To display commas:

1. Right-click in the Answerset cell you wish to change and select **Format Cells**.
2. Check **Display 1000 separators.**
3. Click **OK**.

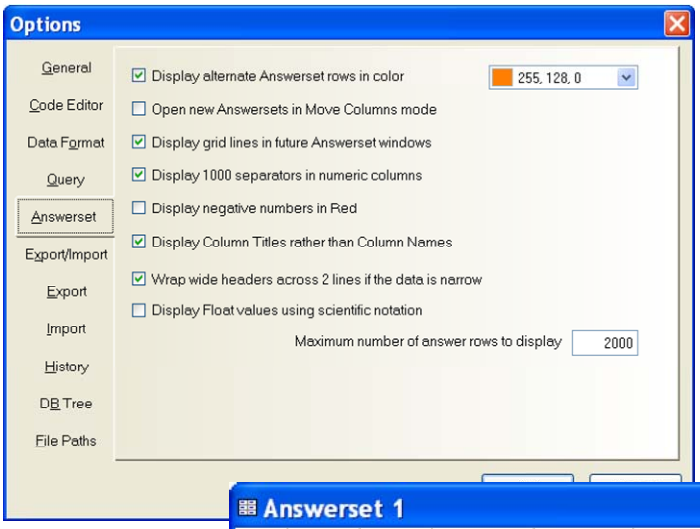
To display decimal places:

1. Right-click in the Answerset cell you wish to change and select **Decimal Places**.
2. Select a number between 0 and 4.

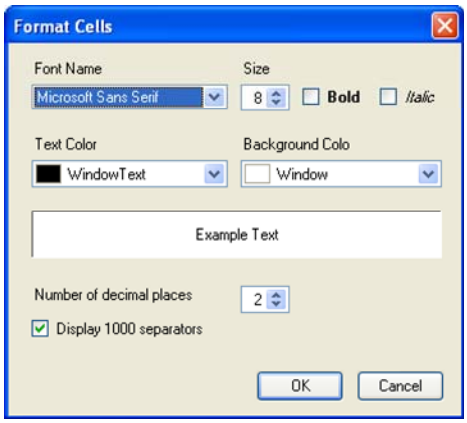
   To designate up to 14 decimal places:
   a. Right-click to bring up the Shortcut menu.
   b. Click **Format Cells** to bring up the Format Cells dialog.
   c. Under **Numerics**, select the desired number of decimal places.

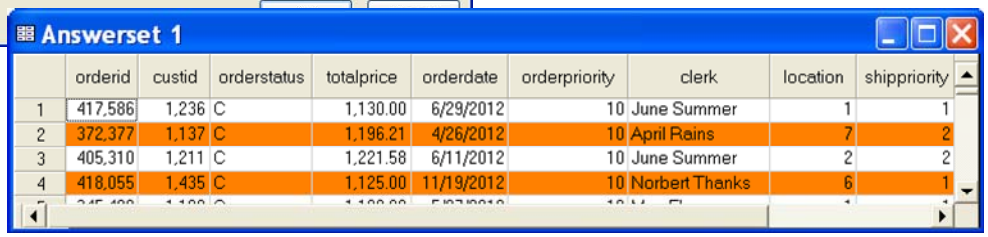
Formatting Answersets

To set defaults for Answersets,
use the Tools > Options > Answerset tab.



To format specific cells,
right-click on a cell or use the  **icon.**





	orderid	custid	orderstatus	totalprice	orderdate	orderpriority	clerk	location	shippriority
1	417,586	1,236	C	1,130.00	6/29/2012	10	June Summer	1	1
2	372,377	1,137	C	1,196.21	4/26/2012	10	April Rains	7	2
3	405,310	1,211	C	1,221.58	6/11/2012	10	June Summer	2	2
4	418,055	1,435	C	1,125.00	11/19/2012	10	Norbert Thanks	6	1

Using Query Builder

Query Builder provides the user with the ability to use 'templates' for SQL commands, which may then be modified by the user. This is a convenient way to create commands whose syntax is complex or not easily remembered. Simply find the appropriate command, then drag and drop it into the query window where it may then be customized.

The Query Builder window is a floating window you can leave open when you are working within the main Teradata SQL Assistant window.

To access the Query Builder tool, do one of the following:

- Press F2.
- Select **Help** > **Query Builder**.
- Right-click in the Query window and select **Query Builder** from the shortcut menu.

From the drop-down list in the upper left corner, choose one of the following options.

SQL Statements	Select a command from the statement list in the left pane to display an example of its syntax in the right pane.
Procedure Builder	Select a stored procedure statement from the list in the left pane to display an example of its syntax in the right pane.
<User Defined>	If you create a *custom.syn* file, this option appears in the drop-down list. The name will be the name you specified in the first line of the *custom.syn* file. Select this option and the queries you defined in this file will display.

Description of the Options

SQL Statements

When you choose the SQL Statements option, the statement list in the left pane shows each of the statement types available on the current data source. These syntax examples reflect the SQL syntax of the data source you are currently connected. For example, the Teradata syntax file is *Teradata.syn*.

Procedure Builder

When you choose the Procedure Builder option, the left pane shows a list of statements that are valid only when used in a CREATE or REPLACE procedure statement.

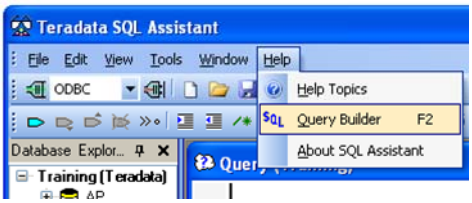
<User Defined>

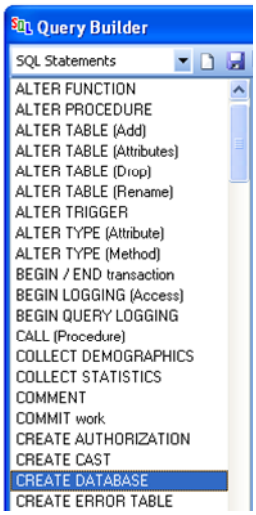
You can create a user-defined syntax file using any text editor such as Notepad or Microsoft Word. The name of the file must be *custom.syn*. The format of this file is the same as the other syntax files except it has an additional line at the start of the file containing the name you wish to see in the dropdown list in the Query Builder dialog.

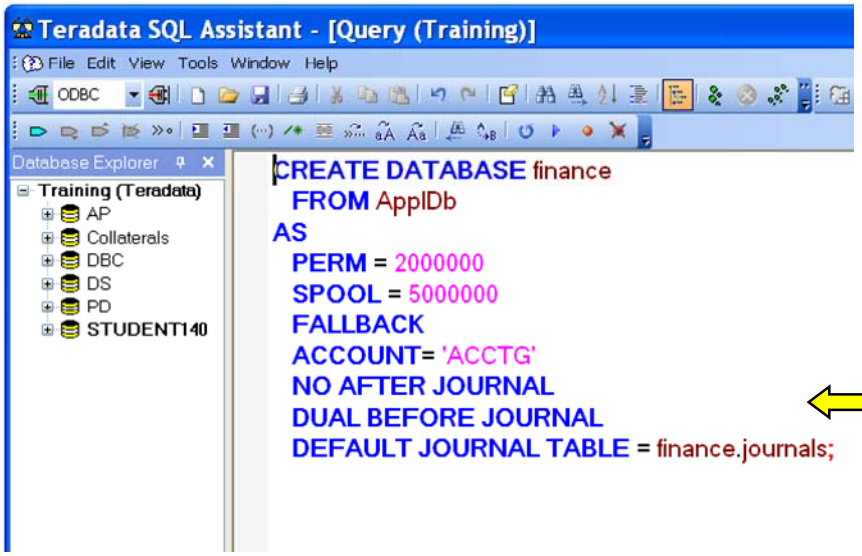
Query Builder

Query Builder provides the user with the ability to use 'templates' for SQL commands.

1. **Select Query Builder from the Help menu or use F2.**







2. **Double-click on SQL statement to place sample query in Query Window.**

History Window

The **History** window is a table that displays your past queries and related processing attributes. The past queries and processing attributes are stored locally in a Microsoft Access 2000 database. This allows the flexibility to work with previous SQL statements in the future.

Clicking any cell in the SQL Statement column in the History window copies the SQL to the Query Window. It may then be optionally modified and then resubmitted.

You can display or hide the History window at any time.

With Teradata SQL Assistant 13, all history rows are now stored in a single History database. The History Filter dialog allows you to specify a set of filters to be applied to the history rows. The operators include >, <, =, and **LIKE**. The filter applies to the entire history table. When you click in the fields or boxes in the **Filter** dialog, the possible operators and proper format are displayed at the bottom of the dialog.

You can filter your history on the following options:

- Date
- Data source
- User Name
- Statement Type – for example, SELECT or CREATE TABLE
- Statement Count – show only those queries tat contain this many statements
- Row Count
- Elapsed Time
- Show successful queries only

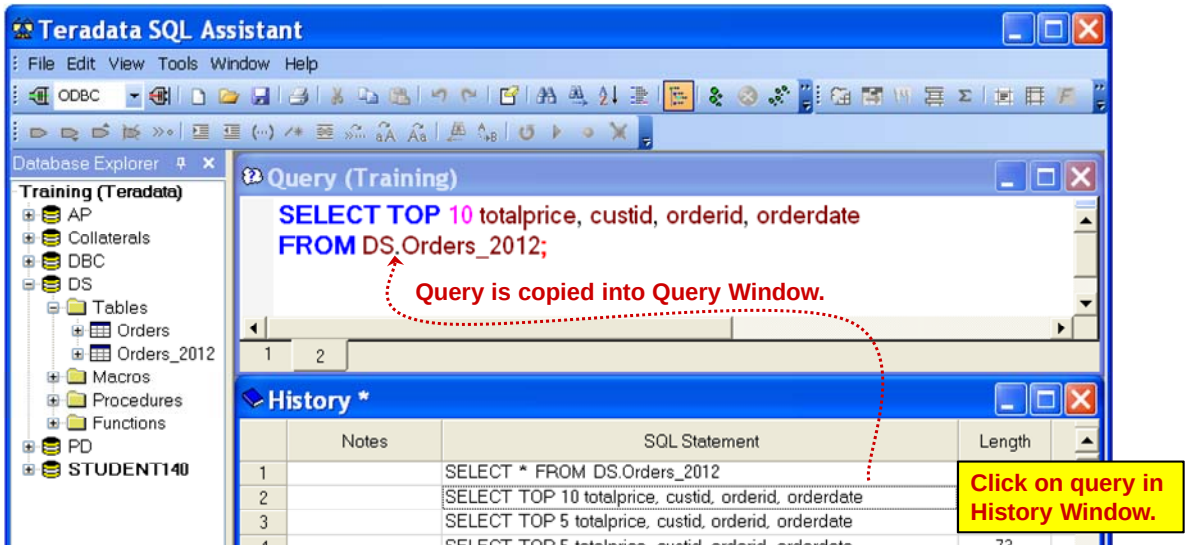
By default, Teradata SQL Assistant records all queries that are submitted. You may change this option so Teradata SQL Assistant records only those statements that are successful, or turn off history recording altogether.

The most recently executed statement appears as the first row in the History window. The data may be sorted locally after it has been loaded into the History window. New entries are added as the first row of history no matter what sort order has been applied.

History Window

A history of recently submitted queries may be recalled by activating the 'Show History' feature. Key options available with the History window are:

- **All history rows are now stored in a single History database. The History Filter dialog allows you to specify a set of filters to be applied to the history rows.**
- **You can choose to display all queries (successful or not), use a history filter to only display successful queries, or turn off history recording altogether.**



General Options

To set general program preferences:

1. Select Tools > Options.
2. Click the General tab.
3. Choose from the following options:

   - **Allow multiple Queries** - allows you to have multiple query windows open simultaneously. With this option selected, the New Query command opens a new tab in the Query window. The default for this setting is unchecked.

   - **Display this string for Null data fields** - enter the string you want displayed in place of Null data fields in your reports and imported/exported files. The default for this setting is "?".

   - Use a separate Answer window for

     – **Each Resultset** - opens a new Answer window for each new result set
     – **Each Query** - opens a new Answer window for each new query, but uses tabs within this window if the query returns multiple result sets. This is the default setting.
     – **Never** - directs all query results to display in a single tabbed Answer window

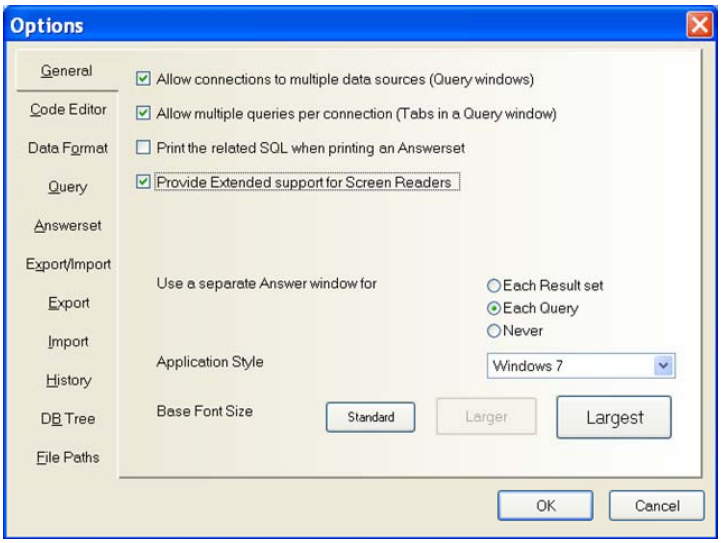
General Options

General options (Tools > Options > General tab) that are available include:

- **Allow connections to multiple data sources.**
- **Allow multiple queries per connection – allows you to have multiple query windows open simultaneously. New Queries are opened in new tabs.**

Data Format options include:

- **Date format**
- **Display of NULL data values**
- **Decimal places to displace**



Connecting to Multiple Data Sources

You can connect to multiple data sources. The “Allow connections to multiple data sources” option must be checked with the General Options.

Each new data source appears in the Database Tree and opens a new query window with the data source name. To disconnect from one data source, click the Query window that is connected to the data source and click the disconnect icon.

The example on the facing page show two connects to two different systems (tdt5-1 and tdt6-1).

Connecting to Multiple Data Sources

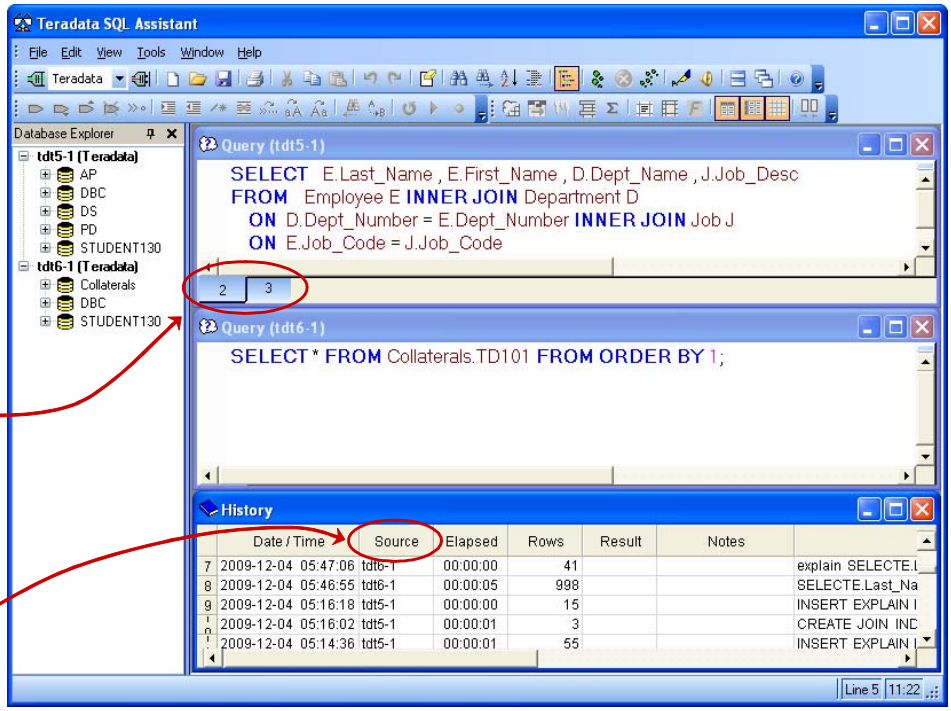
A separate query window is opened for each data source connection.

Connections have been made to two systems:

- **tdt5-1**
- **tdt6-1**

Multiple queries for tdt5-1 are shown via tabs.

History includes the Source name for queries.



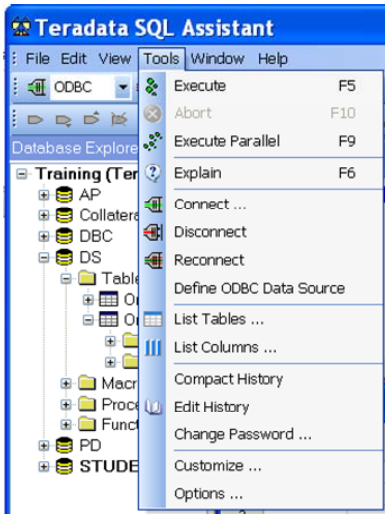
Additional Options

Teradata SQL Assistant provides many other tools and options, some of which are briefly noted on the facing page.

Additional Options

Additional Tools menu options include:

- **Explain – runs an Explain function on the SQL statements in the Query window and display the results in the Answerset window.**
- **List Tables – displays the Table List dialog box where you can enter the name of the database and the resulting list of tables or view displays in an Answerset window.**
- **List Columns – displays the Column List dialog box where you can list the columns in a particular table/view and the resulting list of columns displays in an Answerset window.**
- **Disconnect – disconnects from the current data source.**
- **Change Password – change your Teradata password.**
- **Compact History – reclaim space that may have been lost when history rows were deleted.**
- **Options – establish various options for queries, answersets, import/export operations, etc.**



Importing/Exporting Large Object Files

To import and/or export LOB (Large Object) files with SQL Assistant, you need to first make sure the **"Use Native Large Object Support"** option is set when defining ODBC driver for Teradata. This option was discussed earlier in this module.

This option is automatically selected starting with SQL Assistant 14.0.

Teradata SQL Assistant 12.0 Note

The following information is not needed with Teradata SQL Assistant 13 and later.

With Teradata SQL Assistant 12.0 and prior versions, to import a file larger than 10 MB into a BLOB or CLOB column, you need to enable this capability within SQL Assistant.

To enable importing of files larger than 10 MB into BLOB or CLOB columns:

1. Select **Tools > Options**, then select the **Export/Import** tab.
2. Select the **Import** tab.
3. Click in the **Maximum Size of an Imported data file** field.
4. Press the **Esc** key, then set the value to the size of the largest file you wish to load, up to a maximum of 9 digits.

   If you do not press the Esc key before entering the data, you will be limited to a maximum of 7 digits in this field.
5. Click **OK**.

Note: This will be a temporary change. The next time you click **OK** on the Options screen the value will be reset to the first 7 digits of the number you had last set - for example, 50 MB (50,000,000) will become 5 MB (5,000,000).

Importing/Exporting Large Object Files

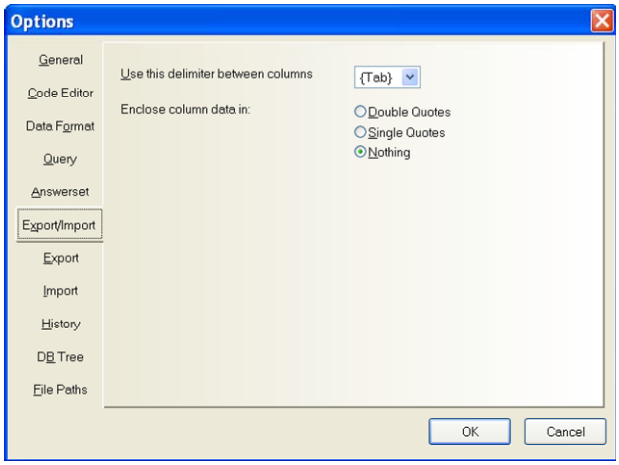
To import and/or export LOB (Large Object) files with SQL Assistant, you need to first make sure the "Use Native Large Object Support" option is set with the data source.

Teradata SQL Assistant supports Large Objects. Large objects come in two types:

- **Binary – these columns may contain Pictures, Music, Word documents, PDF files, etc.**
- **Text – these columns contain text data such as Text, HTML, XML or Rich Text (RTF).**

SQL Assistant > Tools > Options

- **To import a LOB, create a data file that contains the names of the Large Objects.**
- **Use the Export/Import Options dialog to specify the field delimiter.**
- **The example in this module assumes the fields in the imported file are TAB separated.**



Importing/Exporting Large Object Files

To import and/or export LOB (Large Object) files with SQL Assistant, you need to first make sure the **"Use Native Large Object Support"** option is set when defining ODBC driver for Teradata. This option was discussed earlier in this module.

To Import a LOB into Teradata

First, create a data file that contains the names of the LOB(s) to be imported. By default, the data file needs to be located in the same folder as the LOB.

Assume the data file to import from contains 4 fields that are <tab> separated.

Second, select the IMPORT DATA function and execute an Insert statement.

Example: **`INSERT INTO TF VALUES (?,?,?,?B);`**

The parameter markers in this example are:

- ? The data for this parameter is read from the Import file. It is always a character string, and will be converted to a numeric value if necessary.

- ?B The data for this parameter resides in a file that is in the same directory as the Import file. The import file contains only the name of the file to be imported. The contents of the file are loaded as a binary image (e.g., BLOB). You can also use ?? in place of ?B.

- ?C The data for this parameter resides in a file that is in the same directory as the import file. The import file contains only the name of the file to be imported. Use this marker to load a text file into a CHAR or CLOB column.

Importing a LOB into Teradata

1. Create a data file that contains the name(s) of the LOB(s). This data file needs to be located in the same folder as the LOB.

   TF Manual LOB.txt

1	TF Student Manual	PDF	TF v1400 Student Manual.pdf
2	TF Lab Workbook	PDF	TF v1400 Lab Workbook.pdf

   The various fields are separated by tabs.

2. Within SQL Assistant, from the File menu, select the "Import Data" option to turn on the Import Data function.

3. Enter an INSERT statement within the Query window.

   ```
   INSERT INTO TF VALUES (?, ?, ?, ?B);
   ```

4. In the dialog box that is displayed, choose the name of the file to import.

   For example, enter or choose "TF Manual LOB.txt".

5. From the File menu, select the "Import Data" option to turn off the Import Data function.

Selecting from a Table with a LOB

To select from a table with a LOB, simply execute a SELECT statement. If LOB column is projected, then a dialog box is displayed to enter the file name for the LOB.

Note that multiple files that are exported will have sequential numbers added to the file name.

In the example on the facing page, the file name was specified as TF_Manual. Therefore, the two manuals that will be created are named:

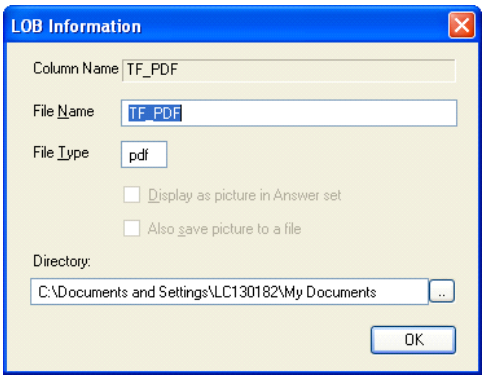
TF_PDF001.pdf
TF_PDF002.pdf

Selecting from a Table with a LOB

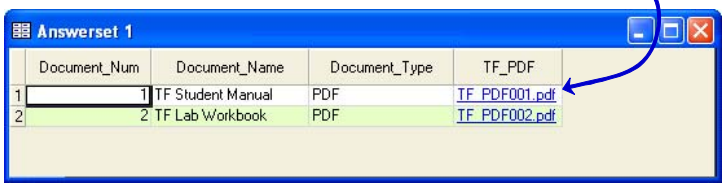
With SQL Assistant, enter the following query:

```
SELECT * FROM TF ORDER BY 1;
```

The following dialog box is displayed to represent the data files to export the LOBs into. Also specify the "File Type" as a known Microsoft file type extension.



The answer set window will include a link to exported data files.



Displaying a JPG within SQL Assistant

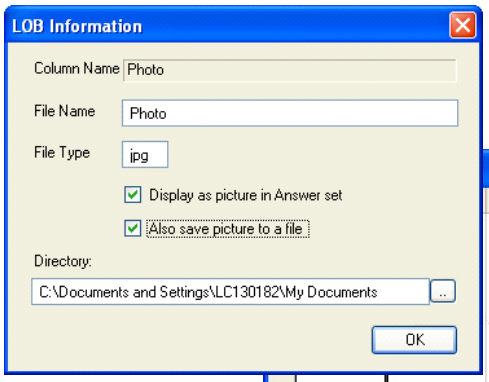
The “Display as picture …” can be selected to display a JPG file within the answer set.

Optionally, the “Also save picture to a file” can be selected.

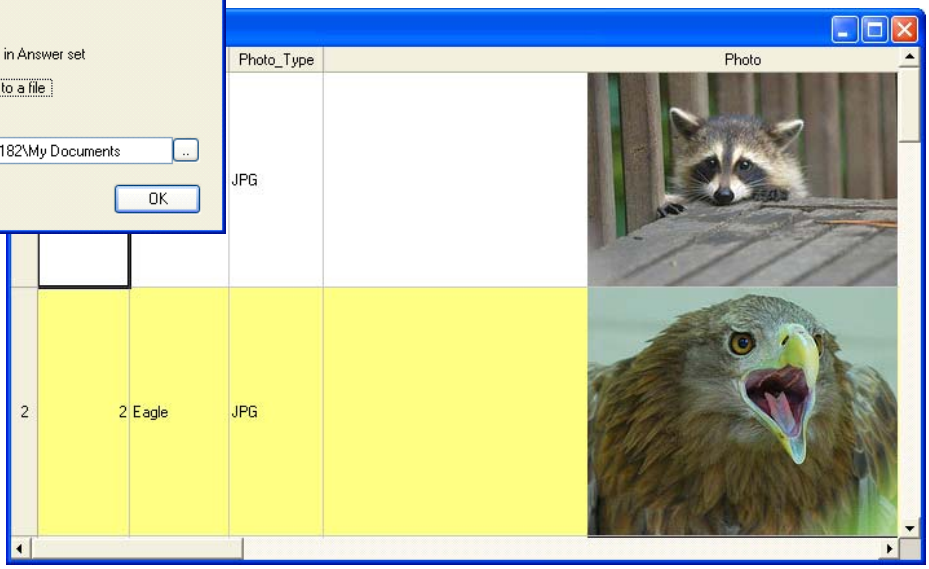
Note that large JPG files with display very large within the answer set window.

Displaying a JPG within SQL Assistant

SELECT * FROM Photos ORDER BY 1;



Optionally, the "Display as picture …" can be selected to display a JPG file within the answer set.



Teradata SQL Assistant Summary

The Teradata SQL Assistant utility can be of great value to you. The facing page summarizes some of the key features discussed in this module.

Teradata SQL Assistant Summary

Characteristics of Teradata SQL Assistant include:

- Windows-based utility that can be used to submit SQL queries to the Teradata database.
- Provides the retrieval of previously used queries (History).
- Saves information about previous query result sets.
- Supports DDL, DML and DCL commands.
  - Query Builder feature allows for easy creation of SQL statements.
- Provides both import and export capabilities to files on a PC.
- Provides a Database Explorer Tree to easily view database objects.

Module 15: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 15: Review Questions

1. Which two data interfaces are available with Teradata SQL Assistant?
   a. CLIv2
   b. JDBC
   c. ODBC
   d. Teradata .Net

2. Separate history database files are needed to maintain queries for different data sources.
   a. True
   b. False

3. Which piece of query information is not available in the History Window?
   a. User name
   b. Query band
   c. Elapsed time
   d. Data source name
   e. Number of rows returned

4. What are two techniques to execute multiple statements as a multi-statement request?

   _____ _____

Lab Exercise 15-1

Check your understanding of the concepts discussed in this module by completing the lab exercise as directed by your instructor.

Lab Exercise 15-1

Lab Exercise 15-1

Purpose

In this lab, you will use Teradata SQL Assistant to define a data source and execute some simple SQL commands.

What you need

Teradata SQL Assistant installed on the laptop or PC

Tasks

1. Define either an ODBC data source or a .NET data source using the following instructions.

   Complete the dialog box with the following information:

   Name – TFClass
   Description – Teradata Training for your name
   Name or IP Address – _____ (supplied by instructor)
   Username – _____ (supplied by instructor)
   Password – do not fill in your password (initially needed for a .NET connection)

   Verify the following options are set properly.

   Session Character Set – ASCII
   Options – Session Mode – System Default
   Use Native Large Object Support option is checked (not needed with a .NET connection)

Lab Exercise 15-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercise as directed by your instructor.

Lab Exercise 15-1 (cont.)

2. Connect to the data source your just created (TFClass) and logon with your username and password.

3. Using the Tools > Options tabs, ensure the following options are set as indicated:

   General – Check – Allow connections to multiple data sources
   General – Check – Allow multiple queries per connection
   Query – Check – Submit only the selected Query text, when highlighted
   Answerset – Check – Display alternate Answerset rows in color – choose a color
   Answerset – Check – Display Column Titles rather than Column Names
   History – Check – Display SQL text on a single line
   History – Check – Do not save duplicate queries in history

4. If the Explorer Tree pane is not visible, use the View > Explorer option to display the Explorer Tree.

   Add the following databases to the Explorer Tree: AP, DS, PD, Collaterals

   (Hint: Right-click on the Explorer Tree pane to use the "Add Database …" option.

5. Using the Explorer Tree, view the table objects in your database.

6. Using the Query Window, execute the following query.

   ```
   CREATE TABLE Old_Orders AS Orders WITH NO DATA;
   ```

   Does the new table object appear in the table object list? _____ If not, "refresh" the database.

Lab Exercise 15-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercise as directed by your instructor.

Use the following SQL to determine a count of rows in a table.

SELECT COUNT(*) FROM *tablename*;

Step 8 Hint: Your Old_Orders table should have 2400 rows. If not, check the dates you used in your queries.

Lab Exercise 15-1 (cont.)

7. **Using the Query window, execute the following query.**

```
INSERT INTO Old_Orders SELECT * FROM DS.Orders
WHERE orderdate BETWEEN '2008-07-01' AND '2008-09-30';
```

   Use the "Format Query" option to format the query.

   How many rows are in the Old_Orders table? _____

8. **Using the History window, recall the query from step #7 and modify it to add orders from '2008-10-01' through '2008-12-31'.**

   How many rows are in the Old_Orders table? _____

9. **Execute the following query by using the drag and drop object feature of SQL Assistant.**

```
SELECT custid, SUM (totalprice)
FROM   Old_Orders
GROUP  BY 1
ORDER  BY 1;
```

   Use the "Add Totals" feature to automatically generate a total sum for all of the orders.

   What is the sum of the orders using this feature? _____

Lab Exercise 15-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercise as directed by your instructor.

Use the following SQL to create a view.

```
CREATE    VIEW viewname
AS
SELECT    column1, column2
FROM      table_or_view_name
[WHERE condition];
```

Use the following SQL to create a simple macro.

```
CREATE    MACRO macroname
AS        (SELECT * FROM table_or_view_name ; ) ;
```

Use the following SQL to execute a simple macro.

```
EXEC macroname;
```

Lab Exercise 15-1 (cont.)

10. **Format only the cells of sum of the ordertotal to be in *italics* and green.**

11. **Using the Query Builder feature, create a view named "Old_Orders_v" for the Old_Orders table that includes the following columns and only includes orders for December, 2008.**

    orderid, custid, totalprice, orderdate

    SELECT all of the rows from the view named "Old_Orders_v".

    How many rows are displayed from this view? _____

12. **Using the Query Builder feature, create a simple macro named "Old_Orders_m" which selects all of the orders from the view named "Old_Orders_v".**

    Execute the macro "Old_Orders_m".

    What is the sum of the orders for December using this "Add Totals" feature? _____

13. **(Optional) Use the Collaterals database to access the Photos table to display various JPG files.**

    Execute the following: SELECT * FROM Collaterals.Photos ORDER BY 1;

    Note: Set the file type to JPG and check the option "Display as picture in Answerset".

Notes

Module 16

Analyze Primary Index Criteria

After completing this module, you will be able to:

- **Identify Primary Index choice criteria.**
- **Describe uniqueness and how it affects space utilization.**
- **Explain row access, selection, and selectivity.**
- **Choose between single and multiple-column Primary Indexes.**
- **Describe why a table might be created without a primary index.**
- **Specify the syntax to create a table without a primary index.**

Notes

Table of Contents

Primary Index Choice Criteria 16-4
Primary Index Defaults 16-6
CREATE TABLE – Indexing Rules 16-8
Order of Preference Exercise 16-10
Primary Index Characteristics 16-12
Multi-Column Primary Indexes 16-14
Primary Index Considerations 16-16
PKs and Duplicate Rows 16-18
NUPI Duplicate Row Check 16-20
Primary Index Demographics 16-22
Column Distribution Demographics for a PI Candidate 16-24
SQL to View Data Demographics 16-26
Example of Using Data Demographic SQL 16-28
TableSize View 16-32
SQL to View Data Distribution 16-34
E-R Diagram for Exercises 16-36
Exercise 2 – Sample 16-38
Exercise 2 – Choosing PI Candidates 16-40
What is a NoPI Table? 16-52
Reasons to Consider Using NoPI Tables 16-54
Creating a Table without a PI 16-56
How is a NoPI Table Implemented? 16-58
NoPI Random Generator 16-60
The Row ID for a NoPI Table 16-62
Multiple NoPI Tables at the AMP Level 16-66
Loading Data into a NoPI Table 16-68
NoPI Options 16-70
Summary 16-72
Module 16: Review Questions 16-74
Module 16: Review Questions (cont.) 16-76
Lab Exercise 16-1 16-78
Lab Exercise 16-2 16-82

Primary Index Choice Criteria

There are three Primary Index Choice Criteria: **Access Demographics**, **Distribution Demographics**, and **Volatility**.

- Access demographics are the first of three Primary Index Choice Criteria. Access columns are those that would appear (with a value) in a WHERE clause in an SQL statement. Choose the column most frequently used for access to maximize the number of one-AMP operations.

- Distribution demographics are the second of the Primary Index Choice Criteria. The more unique the index, the better the distribution. Optimizing distribution optimizes parallel processing.

- In choosing a Primary Index, there is a trade-off between the issues of access and distribution. The most desirable situation is to find a PI candidate that has good access and good distribution. Many times, however, index candidates offer great access and poor distribution or vice versa. When this occurs, the physical designer must balance these two qualities to make the best choice for the index.

- The third of the Primary Index Choice Criteria is volatility, or how often the data values will change. The Primary Index should not be very volatile. Any changes to Primary Index values may result in heavy I/O overhead, as the rows themselves may have to be moved from one AMP to another. Choose a column with stable data values.

Degree of Uniqueness and Space Utilization

The degree of uniqueness of a Primary Index has a direct influence on the space utilization. The more unique the index, the better the space is used.

Fewer Distinct PI Values than Amps

For larger tables, it is not a good idea to choose a Primary Index with fewer distinct values than the number of AMPs in the system when other columns are available. At best, one index value would be hashed to each AMP and the remaining AMPs would carry no data.

Non-Unique PIs

Choosing a Non-Unique PI (NUPI) with some very non-unique values can cause “spikes” in the distribution.

Unique (or Nearly-Unique) PIs

The designer should choose an index which is unique or nearly unique to optimize the use of disk space. Remember that the PERM limit of a database (or user) is divided by the number of AMPs in the system to yield a threshold that cannot be exceeded on any AMP.

Primary Index Choice Criteria

ACCESS	**Maximize one-AMP operations:** **Choose the column(s) most frequently used for access.** **Consider both join and value access.**
DISTRIBUTION	**Optimize parallel processing:** **Choose the column(s) that provides good distribution.**
VOLATILITY	**Reduce maintenance resource overhead (I/O):** **Choose the column(s) with stable data values.**

Note: Data distribution has to be balanced with Access usage in choosing a PI.

General Notes:

- **A good logical model identifies the Primary Key for each table or entity.**
  - **Do not assume that the Primary Key will become the Primary Index.**
  - **It is common for many tables in a database to have a Primary Index that is different than the Primary Key.**
  - **This module will first cover PI tables then cover details of the NO PRIMARY INDEX option. The general assumption in this course is that tables will have a PI.**

Primary Index Defaults

1. If the NO PRIMARY INDEX clause is specified, then the table is created without a primary index. If this clause is used, you cannot specify a primary index for the table.

   There are a number of limitations associated with a NoPI table that will be listed later.

2. If the PRIMARY INDEX, NO PRIMARY INDEX, PRIMARY KEY, or UNIQUE options are NOT specified in the CREATE TABLE DDL, then:

   Table to be created with or without a primary index will be based on a new DBSControl General flag ***Primay Index Default***. The default setting is "D" which effectively means the default is to create a table with the first column as a NUPI.

   D - This is the default setting. This setting works the same as the P setting.

   P - The first column in the table will be selected as the non-unique primary index. This setting works the same as that in the past when PRIMARY INDEX was not specified.

   N – The table will be created without a primary index (NoPI table).

3. With the NoPI Table feature, the system default setting essentially remains the same as that in previous Teradata releases where the first column was selected as the non-unique primary index when the user did not specify a PRIMARY INDEX or a PRIMARY KEY or a UNIQUE Constraint.

   Users can change the default setting for PrimaryIndexDefault to P or N and not rely on the system default setting which might be changed in a future release.

Primary Index Defaults

A Teradata 13.0 DBSControl flag determines if a PI or NoPI table is created when a CREATE TABLE DDL does NOT have any of the following explicitly specified:

- **PRIMARY INDEX clause**
- **NO PRIMARY INDEX clause**
- **PRIMARY KEY or UNIQUE constraints**

Values for DBS Control General field #53 "Primary Index Default":

D – "Teradata Default" (effectively same as option P)
P – "First Column is NUPI" – create tables with first column as a NUPI
N – "No Primary Index" – create tables without a primary index (NoPI)

The PRIMARY INDEX and NO PRIMARY INDEX clauses have precedence over PRIMARY KEY and UNIQUE constraints.

If the NO PRIMARY INDEX clause is specified AND if PRIMARY KEY or UNIQUE constraints are also defined, these will be implemented as Unique Secondary Indexes.

- **It may be unusual to create a NoPI table with these additional indexes.**

CREATE TABLE – Indexing Rules

The **primary index** may be explicitly specified at table create time. If not, a primary index choice will be made based on other choices made. Primary key and uniqueness constraints are always implemented by Teradata as unique indexes, either primary or secondary.

This chart assumes the system default is to create tables with a Primary Index.

The index implementation schedule is as follows:

Is a PI specified?

No	PK specified?	PK = UPI
	PK specified and UNIQUE constraints specified?	PK = UPI UNIQUE constraints = USI(s)
	UNIQUE column level constraints only specified?	1st UNIQUE column level constraint = UPI Other UNIQUE constraints = USI(s)
	UNIQUE column level constraints and table level UNIQUE constraints specified?	1st UNIQUE column level constraint = UPI Other UNIQUE constraints = USI(s)
	UNIQUE table level constraints only specified?	1st UNIQUE table level constraint = UPI Other table level UNIQUE constraints = USI(s)
	Neither specified?	1st column = NUPI
Yes	PK specified?	PK = USI
	PK specified and UNIQUE constraints specified?	PK = USI UNIQUE constraints = USI(s)
	UNIQUE constraints only specified?	UNIQUE constraints = USI(s)

CREATE TABLE – Indexing Rules

Unspecified Primary Index option – assuming system default is "Primary Index"

If	PRIMARY KEY specified	PK	= UPI
else	1st UNIQUE column level constraint specified	column	= UPI
else	1st UNIQUE table level constraint specified	column(s)	= UPI
else	1st column specified*	column	= NUPI

* If system default is "No Primary Index" AND none of the following have specified (Primary Index, PK, or UNIQUE), then table is created as a NoPI table.

Specified PRIMARY INDEX or NO PRIMARY INDEX

If	PRIMARY KEY is also specified	PK	= USI
and	any UNIQUE constraint (column or table level)	column(s)	= USI

Every PK or UNIQUE constraint is always implemented as a unique index.

Order of Preference Exercise

Complete the exercise on the facing page. Answers will be provided by your instructor.

Some additional examples include:

If table_5 was created as follows:

```
CREATE TABLE table_5
      (col1  INTEGER  NOT NULL
      ,col2  INTEGER  NOT NULL
      ,col3  INTEGER  NOT NULL
      ,CONSTRAINT uniq1 UNIQUE (col1,col2)
      ,CONSTRAINT uniq2 UNIQUE (col3));
```

Then, the indexes are a UPI on (col1,col2) and a USI on (col3).

If table_5 was created as follows:

```
CREATE TABLE table_5
      (col1  INTEGER  NOT NULL
      ,col2  INTEGER  NOT NULL
      ,col3  INTEGER  NOT NULL
      ,CONSTRAINT uniq1 UNIQUE (col3)
      ,CONSTRAINT uniq2 UNIQUE (col1,col2));
```

Then, the indexes are a UPI on (col3) and a USI on (col1,col2).

Notes:

- Recommendation: Specify Primary Index when creating a table.
- Table level constraints are typically used to specify a PK or UNIQUE constraint for multiple columns.

Order of Preference Exercise

Assuming the system default is "Primary Index", show the indexes that are created as a result of the DDL.

```
CREATE TABLE table_1
   (col1  INTEGER  NOT NULL  UNIQUE                col1 =
   ,col2  INTEGER  NOT NULL  PRIMARY KEY);         col2 =

CREATE TABLE table_2
   (col1  INTEGER  NOT NULL  PRIMARY KEY           col1 =
   ,col2  INTEGER)                                 col2 =
PRIMARY INDEX (col2);

CREATE TABLE table_3
   (col1  INTEGER                                  col1 =
   ,col2  INTEGER  NOT NULL);                      col2 =

CREATE TABLE table_4
   (col1  INTEGER  NOT NULL                        col1 =
   ,col2  INTEGER  NOT NULL                        col2 =
   ,col3  INTEGER  NOT NULL  UNIQUE                col3 =
   ,CONSTRAINT pk1 PRIMARY KEY (col1,col2));       (col1,col2) =

CREATE TABLE table_5
   (col1  INTEGER  NOT NULL                        col1 =
   ,col2  INTEGER  NOT NULL                        col2 =
   ,col3  INTEGER  NOT NULL  UNIQUE                col3 =
   ,CONSTRAINT uniq1 UNIQUE (col1,col2));          (col1,col2) =
```

UPI =	Unique Primary Index
NUPI =	Non Unique Primary Index
USI =	Unique Secondary Index

Primary Index Characteristics

Each table has one and only one Primary Index. A Primary Index may be different than a Primary Key.

UPI = Best Performance, Best Distribution

- UPIs offer the best performance possible for several reasons. They are:
- A Unique Primary Index involves a single base table row at most
- No Spool file is ever required
- Single value access via the Primary Index is a one-AMP operation and uses only one I/O

NUPI = Good Performance, Good Distribution

- NUPI performance differs from UPI performance because:
- Non-Unique Primary Indexes may involve multiple table rows.
- Duplicate values go to the same AMP and the same data block, if possible.
- Multiple I/Os are required if the rows do not fit in a single data block.
- Spool files are used when necessary.
- A duplicate row check is required on INSERT and UPDATE for a SET table.

Primary Index Characteristics

Primary Indexes (UPI and NUPI)

- A Primary Index may be different than a Primary Key.
- Every table has only one Primary Index.
- A Primary Index may contain null(s).
- Single-value access uses ONE AMP and, typically, one I/O.

Unique Primary Index (UPI)

- Involves a single base table row at most.
- No spool file is ever required.
- The system automatically enforces uniqueness on the index value.

Non-Unique Primary Index (NUPI)

- May involve multiple base table rows.
- A spool file is created when needed.
- Duplicate values go to the same AMP and the same data block.
- Only one I/O is needed if all the rows fit in a single data block.
- Duplicate row check is required for a Set table.

Multi-Column Primary Indexes

In practice, Primary Indexes are sometimes composed of several columns. Such composite indexes are known as multi-column Primary Indexes. They are used quite commonly and you can probably think of several existing applications that utilize them.

Increased Uniqueness

There are both advantages and disadvantages to using multi-column PIs. Perhaps the most important **advantage** is that by combining several columns, you can produce an index that is much more unique than any of the component columns. This **increased uniqueness** will result in better data distribution, among other benefits.

For example:

- PI = Lastname
- PI = Lastname + Firstname
- PI = Lastname + Firstname + MI

The above example points out how better data distribution occurs. Notice that each succeeding Primary Index is more unique than the one preceding it. That is, there are far less individuals with identical last and first names then there are with the same last name, and so on.

Increasing uniqueness means that as the number of columns increases:

- The number of distinct values increases.
- The number of rows per value decreases.
- The selectivity increases.

Trade-off

The **disadvantage** involved with multi-column indexes is that as the number of columns increases, the index becomes less usable.

A multi-column index can only be accessed when values for all columns are specified in the SQL statement. If a single value is omitted, the Primary Index cannot be used.

It is important for the physical designer to balance these factors and use multi-column indexes that have just enough columns. This will result in optimum uniqueness while reducing unnecessary full table scans.

Multi-Column Primary Indexes

Advantage

More columns = more uniqueness

- **Number of distinct values increase.**
- **Rows/value decreases.**
- **Selectivity increases.**

Disadvantage

More columns = less usability

- **PI can only be used when values for all PI columns are provided in SQL statement.**
- **Partial values cannot be hashed.**

Primary Index Considerations

The facing page summarizes the concepts you have seen throughout this module and provides a list of the most important considerations when choosing a Primary Index.

The first three considerations summarize the three types of demographics: **Access**, **Distribution**, and **Volatility**.

You should choose a column with good distribution to maximize parallel processing. A good rule-of-thumb is to base your Primary Index on the column(s) most often used for access (if you don't have too many rows per value) to maximize one-AMP operations. Finally, Primary Index values should be stable to reduce maintenance resource overhead.

Make sure that the number of distinct values for a PI is greater than the number of AMPs in the system, whenever possible, or some AMPs will have no rows.

Duplicate values hash to the same AMP and are stored in the same data block. If the index is very non-unique, multiple data blocks are used and incur multiple I/Os.

Very non-unique PIs may skew space usage among AMPs and cause Database Full conditions on AMPs where excessive numbers of rows are stored.

Primary Index Considerations

- **Base PI on the column(s) most often used for access, provided that the values are unique or nearly unique.**
- **Choose a column (or columns) with good distribution and no spikes.**
  - **NULLs and zero (for numeric data types) hash to binary zeroes and to the same AMP.**
- **Distinct values distribute evenly across all AMPs.**
  - **For large tables, the number of Distinct Primary Index values should be much greater (at least 10X; 50X may be better guideline) than the number of AMPs.**
- **Duplicate values hash to the same AMP and are stored in the same data block when possible.**
  - **Very non-unique values use multiple data blocks and incur multiple I/Os.**
  - **Very non-unique values may skew space usage among AMPs and cause premature Database Full conditions.**
  - **A large number of NUPI duplicate values on a SET table can cause expensive duplicate row checks.**
- **Primary Index values should not be highly volatile.**

This is how u write it here...

PKs and Duplicate Rows

Each row in table or entity in a good logical model will be uniquely identified by the table's primary key.

- Every table must have a Primary Key.
- Primary Keys (PKs) must be unique.
- Primary Keys cannot be changed.

In Set tables, the Teradata Database does not allow duplicate rows. When a table has a Unique Primary Index (UPI), the UPI enforces uniqueness. When a table has a Non-Unique Primary Index (NUPI), the matter can become more complicated.

In the case of a NUPI (without a USI defined), the file system must compare data values byte-by-byte within a Row Hash in order to ensure uniqueness. Many NUPI duplicates result in lots of duplicate row checks, which can be quite expensive in terms of system resources.

The way to avoid such a situation is to define a USI on the table whenever you have a NUPI. The USI does the job of enforcing uniqueness and thus save you the cost of doing duplicate row checks. Often, the best column(s) to use when defining such a USI is the PK.

Specifying a UNIQUE constraint on a column(s) other than the Primary Index also causes the creation of a Unique Secondary Index.

An exception to the above is found when using the load utilities, such as FastLoad and MultiLoad. These utilities do not allow the use of Secondary Indexes to enforce uniqueness. Therefore, a full row comparison is still necessary.

PKs and Duplicate Rows

Rule: Primary Keys Must be UNIQUE and NOT NULL.

- This rule of Relational Theory eliminates duplicate rows, which have plagued the industry for decades.
- With Set tables (the default in Teradata transaction mode), the Teradata database *does not allow* duplicate rows.
- With Multiset tables, the Teradata database *allows* duplicate rows.
  - All indexes must be non-unique indexes (NUPI and NUSI) in order to allow duplicate values.
  - A unique index (UPI or USI) will prevent duplicate index values, and therefore, duplicate rows (even if the table is created as Multiset).
- If no unique index exists for a SET table, the file system compares data values byte by byte within a Row Hash to ensure row uniqueness in a table.
  - Many NUPI duplicates result in expensive duplicate row checks.
  - To avoid these duplicate row checks, use a Multiset table.

NUPI Duplicate Row Check

Set tables (the default) do not allow duplicate rows. When a new row is inserted into a Set table with a Non-Unique Primary Index, the system must perform a **NUPI Duplicate Row Check**.

The table on the facing page illustrates the number of logical reads that must occur when this happens. The middle column is the number of logical reads required before that one row can be inserted. The right hand column shows how many cumulative logical reads would be required to insert all the rows up to and including that one.

As you can see, when you have a NUPI with excessive rows per value, the number of logical reads becomes prohibitively high. It is very important to limit the NUPI rows per value whenever possible.

The best way to avoid NUPI Duplicate row checks is to create the table as a MULTISET table.

> Note: USIs should be used for access or uniqueness enforcement. They should not be used just to avoid duplicate row checking, since sometimes they may be used and at other times they will not be used. The overhead of a USI does not justify the cost of trying to avoid the duplicate row check and they don't avoid the cost in most cases.

As a suggestion, keep the number of NUPI rows per value within the number of rows which will fit into you largest block. This will allow the system to satisfy a single-value NUPI access with one or two data block I/Os.

NUPI Duplicate Row Check

Limit NUPI rows per value to rows per block whenever possible.

To avoid NUPI duplicate row checks, create the table as a MULTISET table.

This chart illustrates the additional I/O overhead.

Row Number to be inserted	Number of Rows that must be logically read first	Cumulative Number of logical row reads
1	0	0
2	1	1
3	2	3
4	3	6
5	4	10
6	5	15
7	6	21
8	7	28
9	8	36
10	9	45
20	19	190
50	49	1225
100	99	4950
200	199	19900
500	499	124750
1000	999	499500

this is how we can write it.....

Primary Index Demographics

As you have seen, the three types of demographics important to choosing a Primary Index are: **Access** demographics, **Distribution** demographics and **Volatility** demographics. To make proper PI selections, you must have accurate demographics. Accurate demographics serve to quantify all three-index selection determinants.

Access Demographics

Access demographics identify index candidates that maximize one-AMP operations. Both Value Access and Join Access are important to PI selection. The higher the value, the more often the column is used for access.

Distribution Demographics

Distribution demographics identify index candidates that optimize parallel processing. Choose the column(s) that provides the best distribution.

Volatility

Volatility demographics identify table columns that are UPDATEd. This item does not refer to INSERT or DELETE operations.

Volatility demographics identify index candidates that reduce maintenance I/O. You want to have columns with stable data values as your PI candidates.

In this module, you will see how to use Distribution demographics to select PI candidates. Access and Volatility demographics will be presented in a later module.

Primary Index Demographics

Access Demographics

- Identify index candidates that maximize one-AMP operations.
- Columns most frequently used for access (Value and Join).

Distribution Demographics

- Identify index candidates that optimize parallel processing.
- Columns that provide good distribution.

Volatility Demographics

- Identify index candidates with low maintenance I/O.

Without accurate demographics, index choices are unsubstantiated.

Demographics quantify all 3 index selection determinants.

Column Distribution Demographics for a PI Candidate

Column Distribution demographics are expressed in four ways: **Distinct Values**, **Maximum Rows per Value**, **Maximum Rows NULL** and **Typical Rows per Value**. These items are defined below:

- **Distinct Values** is the total number of different values a column contains. For PI selection, the higher the Distinct Values (in comparison with the table row count), the better. Distinct Values should be greater than the number of AMPs in the system, whenever possible. We would prefer that all AMPs have rows from each TABLE.

- **Maximum Rows per Value** is the number of rows in the most common value for the column or columns. When selecting a PI, the lower this number is, the better the candidate. For a column or columns to qualify as a UPI, Maximum Rows per Value must be 1.

- **Maximum Rows NULL** should be treated the same as Maximum Rows Per Value when being considered as a PI candidate.

- **Typical Rows per Value** gives you an idea of the overall distribution which the column or columns would give you. The lower this number is, the better the candidate. Like Maximum Rows per Value, Typical Rows per Value should be small enough to fit on one data block.

The illustration at the bottom of the facing page shows a distribution graph for a column whose values are states. Note in the graph that 30K = Maximum Rows NULL, and 15K = Maximum Rows per Value (CA). Typical Rows per Value is approximately 30.

You should monitor all demographics periodically as they change over time.

Column Distribution Demographics for a PI Candidate

Distinct Values

- The more the better (compared to table row count).
- Should have enough values to allow for distribution to all AMPs.

Maximum Row Per Value 15K

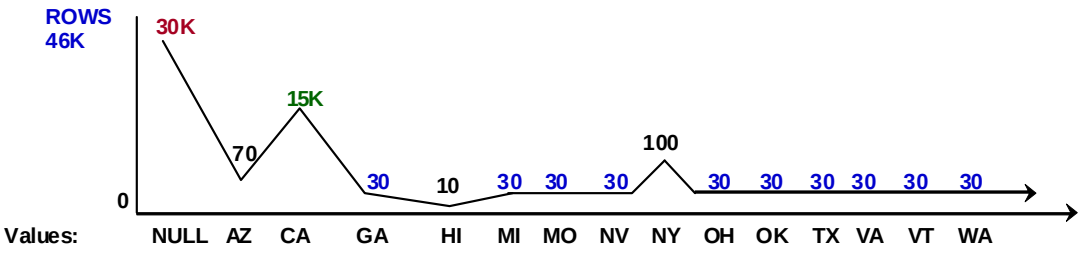
- The fewer the better.

Maximum Rows Null 30K

- The fewer the better.
- A very large number indicates a very large distribution spike.
- Large spikes can cause serious space consumption problems.

Typical Rows Per Value

- The fewer the better.
- Monitor periodically as it changes over time.



SQL to View Data Demographics

The facing page contains simple examples of SQL that can be used to determine data demographics for a column.

The Average Rows per value and Typical Rows per value can be thought of as the Mean and Median of a data set.

SQL to View Data Demographics

of Distinct Values for a column:

```
SELECT    COUNT(DISTINCT(column_name))    FROM  tablename;
```

Max Rows per Value for all values in a column:

```
SELECT    column_name, COUNT(*)           FROM  tablename
GROUP BY 1   ORDER BY 2 DESC;
```

Max Rows per Value for 5 most frequent values:

```
SELECT    TOP 5  t_colvalue, t_count
FROM      (SELECT    column_name, COUNT(*)
           FROM      tablename      GROUP BY 1)
                     t1 (t_colvalue, t_count)
           ORDER BY  t_count DESC;
```

Average Rows per Value for a column (mean value):

```
SELECT    COUNT(*) / COUNT(DISTINCT(col_name))  FROM  tablename;
```

Typical Rows per Value for a column (median value):

```
SELECT    t_count      AS "Typical Rows per Value"
FROM      (SELECT      col_name, COUNT(*)  FROM  tablename GROUP BY 1)
                       t1 (t_colvalue, t_count),
          (SELECT      COUNT(DISTINCT(col_name)) FROM  tablename)
                       t2 (num_rows)
QUALIFY ROW_NUMBER () OVER (ORDER BY t1.t_colvalue) = t2.num_rows /2 ;
```

Example of Using Data Demographic SQL

The facing page contains simple examples of SQL that can be used to determine data demographics for a column.

Example of Using Data Demographic SQL

of Distinct Values for a column:

```
SELECT   COUNT(DISTINCT(Last_name))
           AS "# Values"
FROM     Customer;
```

Values
464

Max Rows per Value for all values:

```
SELECT     Last_name, COUNT(*)
FROM       Customer
GROUP BY   1
ORDER BY   2 DESC;
```

Last_name	Count(*)
Smith	52
Jones	41
Wilson	38
White	36
Lee	36
:	:

Max Rows per Value for 3 most frequent values:

```
SELECT     t_colvalue, t_count
FROM       (SELECT     Last_name,  COUNT(*)
            FROM       Customer    GROUP BY 1)
                       t_table (t_colvalue, t_count)
QUALIFY RANK (t_count) <= 3;
```

t_colvalue	t_count(*)
Smith	52
Jones	41
Wilson	38

Example of Data Demographic SQL (cont.)

The facing page contains simple examples of SQL that can be used to determine data demographics for a column.

Example of Data Demographic SQL (cont.)

Average Rows per Value for a column (mean):

```
SELECT   'Last_name' AS "Column Name"
         ,COUNT(*) / COUNT(DISTINCT(Last_name))
                AS "Average Rows"
FROM     Customer;
```

Column Name	Average Rows
Last_name	15

Typical Rows per Value for a column (median):

```
SELECT   'Last_name' AS "Column Name"
         ,t_count      AS "Typical Rows"
FROM     (SELECT       Last_name, COUNT(*)
          FROM         Customer GROUP BY 1)
                       t_table (t_colvalue, t_count),
         (SELECT       COUNT(DISTINCT(Last_name))
          FROM         Customer)
                       t_table2 (t_distinct_count)
QUALIFY RANK (t_colvalue) = (t_distinct_count / 2);
```

Column Name	Typical Rows
Last_name	11

TableSize View

The TableSize[V][X] views are Data Dictionary views that provides AMP Vproc information about disk space usage at a table level, optionally for tables the current User owns or has SELECT privileges on.

Example

The SELECT statement on the facing page looks for poorly distributed tables by displaying the CurrentPerm figures for a single table on all AMP vprocs.

The result displays one table, **table2**, which is evenly distributed across all AMP vprocs in the system. The CurrentPerm figure is nearly identical across all vprocs. The other table, **table2_nupi**, is poorly distributed. The CurrentPerm figures range from 9,216 bytes to 71,680 bytes on different AMP vprocs.

TableSize View

Provides AMP Vproc disk space usage at table level.

DBC.TableSize[V][X]

Vproc	DatabaseName	AccountName
TableName	CurrentPerm	PeakPerm

Example: Display table distribution across AMPs.

```
SELECT    Vproc
          ,CAST (TableName   AS CHAR(20))
          ,CurrentPerm
          ,PeakPerm
FROM      DBC.TableSizeV
WHERE     DatabaseName = USER
ORDER BY  TableName, Vproc ;
```

Result:

Vproc	TableName	CurrentPerm	PeakPerm
0	table2	41,472	53,760
1	table2	41,472	53,760
2	table2	40,960	52,736
3	table2	40,960	52,736
4	table2	40,960	53,760
5	table2	40,960	53,760
6	table2	40,960	54,272
7	table2	40,960	54,272
0	table2_nupi	22,528	22,528
1	table2_nupi	22,528	22,528
2	table2_nupi	71,680	71,680
3	table2_nupi	71,680	71,680
4	table2_nupi	9,216	9,216
5	table2_nupi	9,216	9,216
6	table2_nupi	59,392	59,392
7	table2_nupi	59,392	59,392

SQL to View Data Distribution

The facing page contains simple examples of SQL that can be used to determine actual data distribution for a table.

SQL to View Data Distribution

Ex: Display the distribution of Customer by AMP space usage.

```
SELECT      Vproc
           ,TableName (CHAR(15))
           ,CurrentPerm
FROM        DBC.TableSizeV
WHERE       DatabaseName = DATABASE
AND         TableName = 'Customer'
ORDER BY    1 ;
```

Vproc	TableName	CurrentPerm
0	Customer	127488
1	Customer	127488
2	Customer	127488
3	Customer	127488
4	Customer	128000
5	Customer	128000
6	Customer	126976
7	Customer	126976

Ex: Display the distribution of Customer by AMP row counts.

```
SELECT      HASHAMP (HASHBUCKET
            (HASHROW (Customer_number)))  AS "AMP #"
           ,COUNT(*)
FROM        Customer
GROUP BY    1
ORDER BY    1 ;
```

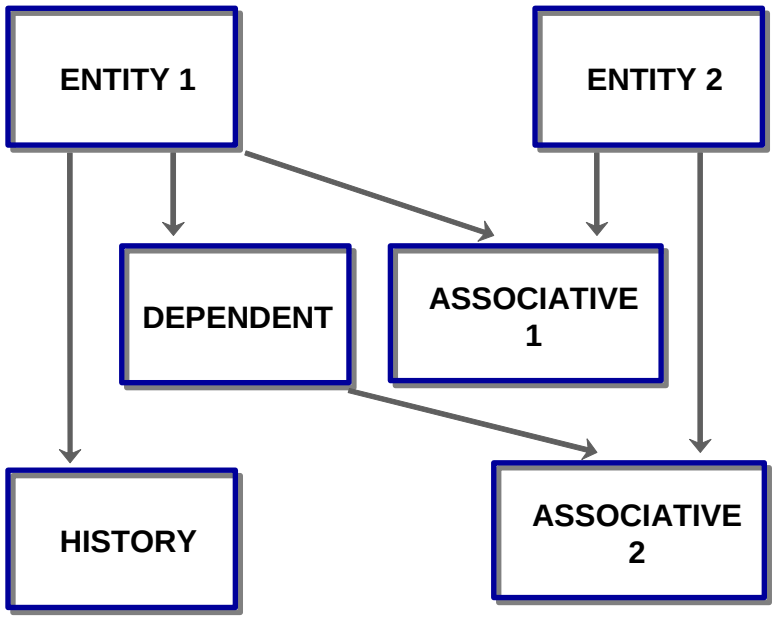
AMP #	Count(*)
0	867
1	886
2	877
3	870
4	881
5	878
6	879
7	862

The Row Hash functions can be used to predict the distribution of data rows for any column in a table.

E-R Diagram for Exercises

The E-R diagram on the facing page depicts the tables used in the exercises. Though the names of the tables and their columns are generic, the model is properly normalized to Third Normal Form (3NF).

E-R Diagram for Exercises



Note:

The exercise table and column names are generic so that index selections are not influenced by names.

Exercise 2 – Sample

The facing page has an example of how to use Distribution demographics to identify PI candidates. On the following pages, you will be asked to identify PI candidates in a similar manner.

Use the Primary Index Candidate Guidelines below to identify the PI candidates. Indicate whether they are UPI or NUPI candidates. Indicate borderline candidates with a ?

In later exercises, you will make the final index choices for these tables.

Primary Index Candidate Guidelines:

- ALL Unique Columns are PI candidates. These columns will be identified with the abbreviation ND for No Duplicates.
- The Primary Key (PK) is a UPI candidate.
- Any single column with high Distinct Values (maybe at least 10 times the number of AMPs), low Maximum Rows NULL, and with a Typical Rows per Value that is relatively close to the Maximum Rows per Value is a PI candidate.

Exercise 2 – Sample

On the following pages, there are sample tables with distribution demographics.

- Indicate ALL possible Primary Index candidates (UPI and NUPI).
- Later exercises will guide your final choices.

Primary Index Candidate Guidelines:

- PK and UNIQUE COLUMNS (ND)
- Any single column with:
  - High Distinct values (at least 10X)
  - Low Maximums for NULLs or a Value
  - Typical Rows that is close to Max Rows

Example 60,000,000 Rows	A	B	C	D	E	F	G	H
PK/FK	PK,SA		FK,NN	NN,ND				
Distinct Values	60M	7M	1.5M	60M	8	15M	15M	70
Max Rows/Value	1	12	500	1	8M	9	725K	90K
Max Rows/NULL	0	5	0	0	0	725K	5	10K
Typical Rows/Value	1	7	35	1	7M	3	3	80K
PI/SI	UPI	NUPI	NUPI?	UPI				
Collect Statistics (Y/N)								

Exercise 2 – Choosing PI Candidates

Use the Primary Index Candidate Guidelines to identify the PI candidates. Indicate whether they are UPI or NUPI candidates. Indicate borderline candidates with a question mark (?).

Primary Index Candidate Guidelines:

- ALL Unique Columns are PI candidates and will be identified with the abbreviation ND for No Duplicates.
- The Primary Key (PK) is a UPI candidate.
- Any single column with high Distinct Values (at least 100% greater than the number of AMPs), low Maximum Rows NULL, and with a Typical Rows per Value that is relatively close to the Maximum Rows per Value is a PI candidate.

Exercise 2 – Choosing PI Candidates

ENTITY 1

100,000,000 Rows	A	B	C	D	E	F
PK/FK	PK,UA					
Distinct Values	100M	95M	300K	250K	40M	1M
Max Rows/Value	1	2	400	350	3	110
Max Rows/NULL	0	0	0	0	1.5M	0
Typical Rows/Value	1	1	325	300	2	90
PI/SI						
Collect Statistics (Y/N)						

Exercise 2 – Choosing PI Candidates (cont.)

Use the Primary Index Candidate Guidelines to identify the PI candidates. Indicate whether they are UPI or NUPI candidates. Indicate borderline candidates with a question mark (?).

Primary Index Candidate Guidelines:

- ALL Unique Columns are PI candidates and will be identified with the abbreviation ND for No Duplicates.
- The Primary Key (PK) is a UPI candidate.
- Any single column with high Distinct Values (at least 100% greater than the number of AMPs), low Maximum Rows NULL, and with a Typical Rows per Value that is relatively close to the Maximum Rows per Value is a PI candidate.

Exercise 2 – Choosing PI Candidates (cont.)

ENTITY 2

10,000,000 Rows	G	H	I	J	K	L
PK/FK	PK,SA					
Distinct Values	10M	100K	9M	12	50	180K
Max Rows/Value	1	200	2	1M	240K	60
Max Rows/NULL	0	0	100K	0	0	0
Typical Rows/Value	1	100	1	800K	190K	50
PI/SI						
Collect Statistics (Y/N)						

Exercise 2 – Choosing PI Candidates (cont.)

Use the Primary Index Candidate Guidelines to identify the PI candidates. Indicate whether they are UPI or NUPI candidates. Indicate borderline candidates with a question mark (?).

Primary Index Candidate Guidelines:

- ALL Unique Columns are PI candidates and will be identified with the abbreviation ND for No Duplicates.
- The Primary Key (PK) is a UPI candidate.
- Any single column with high Distinct Values (at least 100% greater than the number of AMPs), low Maximum Rows NULL, and with a Typical Rows per Value that is relatively close to the Maximum Rows per Value is a PI candidate.

Exercise 2 – Choosing PI Candidates (cont.)

	DEPENDENT					
5,000,000 Rows	A	M	N	O	P	Q
PK/FK	PK				NN, ND	
	FK	SA				
Distinct Values	2M	50	90K	3M	5M	2M
Max Rows/Value	4	200K	75	2	1	5
Max Rows/NULL	0	0	0	390K	0	1M
Typical Rows/Value	1	60K	50	1	1	1
PI/SI						
Collect Statistics (Y/N)						

Exercise 2 – Choosing PI Candidates (cont.)

Use the Primary Index Candidate Guidelines to identify the PI candidates. Indicate whether they are UPI or NUPI candidates. Indicate borderline candidates with a question mark (?).

Primary Index Candidate Guidelines:

- ALL Unique Columns are PI candidates and will be identified with the abbreviation ND for No Duplicates.
- The Primary Key (PK) is a UPI candidate.
- Any single column with high Distinct Values (at least 100% greater than the number of AMPs), low Maximum Rows NULL, and with a Typical Rows per Value that is relatively close to the Maximum Rows per Value is a PI candidate.

Exercise 2 – Choosing PI Candidates (cont.)

ASSOCIATIVE 1

300,000,000 Rows	A	G	R	S
PK/FK	PK			
	FK	FK,SA		
Distinct Values	100M	10M	15K	800K
Max Rows/Value	5	50	21K	400
Max Rows/NULL	0	0	0	0
Typical Rows/Value	3	30	19K	350
PI/SI				
Collect Statistics (Y/N)				

Exercise 2 – Choosing PI Candidates (cont.)

Use the Primary Index Candidate Guidelines to identify the PI candidates. Indicate whether they are UPI or NUPI candidates. Indicate borderline candidates with a question mark (?).

Primary Index Candidate Guidelines:

- ALL Unique Columns are PI candidates and will be identified with the abbreviation ND for No Duplicates.
- The Primary Key (PK) is a UPI candidate.
- Any single column with high Distinct Values (at least 100% greater than the number of AMPs), low Maximum Rows NULL, and with a Typical Rows per Value that is relatively close to the Maximum Rows per Value is a PI candidate.

Exercise 2 – Choosing PI Candidates (cont.)

ASSOCIATIVE 2

100,000,000 Rows	A	M	G	T	U	
PK/FK	PK					
	FK		FK			
Distinct Values	50M		10M	560K	750	
Max Rows/Value	3		150	180	135K	
Max Rows/NULL	0		0	0	0	
Typical Rows/Value	1		8	170	100K	
PI/SI						
Collect Statistics (Y/N)						

Exercise 2 – Choosing PI Candidates (cont.)

Use the Primary Index Candidate Guidelines to identify the PI candidates. Indicate whether they are UPI or NUPI candidates. Indicate borderline candidates with a question mark (?).

Primary Index Candidate Guidelines:

- ALL Unique Columns are PI candidates and will be identified with the abbreviation ND for No Duplicates.
- The Primary Key (PK) is a UPI candidate.
- Any single column with high Distinct Values (at least 100% greater than the number of AMPs), low Maximum Rows NULL, and with a Typical Rows per Value that is relatively close to the Maximum Rows per Value is a PI candidate.

Exercise 2 – Choosing PI Candidates (cont.)

HISTORY

730,000,000 Rows	A	DATE	D	E	F	
PK/FK	PK					
	FK	SA				
Distinct Values	100M	730	N/A	N/A	N/A	
Max Rows/Value	18	1100K	N/A	N/A	N/A	
Max Rows/NULL	0	0	N/A	N/A	N/A	
Typical Rows/Value	3	900K	N/A	N/A	N/A	
PI/SI						
Collect Statistics (Y/N)						

What is a NoPI Table?

A NoPI Table is simply a table without a primary index.

Prior to Teradata Database 13.0, Teradata tables required a primary index. The primary index was primarily used to hash and distribute rows to the AMPs according to hash ownership. The objective was to divide data as evenly as possible among the AMPs to make use of Teradata's parallel processing. Each row stored in a table has a RowID which includes the row hash that is generated by hashing the primary index value. For example, the optimizer can choose an efficient single-AMP execution plan for SQL requests that specify values for the columns of the primary index.

Starting with Teradata Database 13.0, a table can be defined without a primary index. This feature is referred to as the NoPI Table feature. NoPI stands for No Primary Index.

Without a PI, the hash value as well as AMP ownership of a row is arbitrary. Within the AMP, there are no row-ordering constraints and therefore rows can be appended to the end of the table as if it were a spool table. Each row in a NoPI table has a hash bucket value that is internally generated. A NoPI table is internally treated as a hashed table; it is just that typically all the rows on one AMP will have the same hash bucket value.

What is a NoPI Table?

What is a No Primary Index (NoPI) Table?

- It is simply a table without a primary index – a Teradata 13.0 feature.
- As rows are inserted into a NoPI table, rows are always appended at the end of the table and never inserted in a middle of a hash sequence.
  - Organizing/sorting rows based on row hash is therefore avoided.

Basic Concepts

- Rows will still be distributed between AMPs. New code (Random Generator) will determine which AMP will receive rows or blocks of rows.
- Within an AMP, rows are simply appended to the end of the table. Rows will have a unique RowID – the Uniqueness Value is incremented.

Benefits

- A NoPI table will reduce skew in intermediate ETL tables which have no natural Primary Index.
- Loads (FastLoad and TPump Array Insert) into a NoPI staging table are faster.

Reasons to Consider Using NoPI Tables

The facing page identifies various reasons to consider using NoPI tables.

Why is a NoPI table useful?

- A NoPI can be very useful in those situations when the default primary index (first column) causes skewing of data between AMPs and performance degradation.
- This type of table provides a performance advantage in that data can be loaded and stored quickly into a NoPI table using FastLoad or TPump Array INSERT.

Reasons to Consider Using NoPI Tables

Reasons to consider using a NoPI Table

- Utilize NoPI tables instead of arbitrarily defaulting to first table column or creating an unnatural Primary Index from many columns.
- Some ETL tools generate intermediate tables to store data without a known distribution of values.

  If the first column is used (defaults) as the primary index (NUPI), this may lead to skewed data and performance issues.
  - The system default can be set to create tables without a primary index.
- As a staging table to be used with the mini-batch loading technique.
- A NoPI table can be used as a Sandbox table (or any table) where data can be inserted until an appropriate indexing method is determined.
- A NoPI table can be used as a Log file.
- As a Column Partitioned (columnar) table – Teradata 14.0 feature.

Creating a Table without a PI

The facing page identifies the syntax to create a table without a primary index.

If you attempt to include the key word SET (set table) and NO PRIMARY INDEX in the same CREATE TABLE statement, you will receive a syntax error.

Creating a Table without a PI

To create a NoPI table, specify the *NO PRIMARY INDEX* clause in the CREATE TABLE statement.

```
CREATE TABLE <table_name> (<column1>   <column1_datatype>,
                           <column2>   <column2_datatype>,
                           ... )
NO PRIMARY INDEX;
```

Considerations:

- **When a table is created with no primary index, the TableKind column is set to 'O' instead of 'T' and appears in the DBC.TVM table.**
- **If PRIMARY KEY or UNIQUE constraints are also defined, these will be implemented as Unique Secondary Indexes.**
- **A NoPI table is automatically created as a MULTISET table.**

How is a NoPI Table Implemented?

The NoPI Table feature is another step toward extending or supporting Mini-Batch. By allowing a table with no primary index acting as a staging table, data can be loaded into the table a lot more efficiently and in turn faster. All of the rows in a data request, after being received by Teradata and converted into proper internal format, can be appended to a NoPI table without having to be redistributed to their hash-owning AMPs. Rows in a NoPI table are not hashed based on the primary index because there isn't one.

The hash values are all internally controlled and generated and therefore the rows can be stored in any particular order and in any AMP. That means sorting of the rows is avoided.

The performance advantage, especially for FastLoad, from using a NoPI table is most significant for applications that currently load data into a staging table to be transformed or standardized before being stored into another staging table or the target table. For those applications, using a NoPI table can avoid the unnecessary row redistribution and sorting work. Another advantage for FastLoad is that users can quickly load data into a NoPI table and be done with the acquisition phase freeing up Client resources for other work.

For TPump, the performance advantage can be much bigger especially for applications that were not able to pack many rows into the same AMP step in a traditional PI table. On a NoPI table, all rows in a data request are packed into the same AMP step independently from the system configuration and the clustering of data. This will generally lead to big reductions in CPU and IO usage.

How is a NoPI Table Implemented?

Rows are distributed between AMPs using a random generator. Within an AMP, rows are simply added to a table in sequential order.

- **The random generator is designed in such as way that data will be balanced out between the AMPs.**
- **Although there is no primary index in a NoPI table, rows will still have a valid 64-bit RowID.**

The first part of the RowID is based on a hash bucket value (16 or 20 bits) that is internally generated and controlled by the AMP.

- **Typically, all the rows in a table on one AMP will have the same hash bucket value, but will have different uniqueness values.**

There are two separate steps used with a NoPI table.

1. **A new internal function (e.g., random generator) is used to choose a hash bucket which effectively determines which AMP the row(s) are sent to.**
2. **The AMP internally selects a hash bucket value that the AMP owns and uses it as the first part (16 or 20 bits) of the RowID.**

NoPI Random Generator

For SQL-based functions, the PE uses the following technique for the random generator.

The DBQL Query ID is used by the random generator to select a random row hash. The approach is to generate a random row hash in such a way that for a new request, data will generally be sent to a different AMP from the one that the previous request sent data to. The goal is to balance out the data as much as possible without the use of the primary index. The DBQL Query ID is selected for this purpose because it uses the PE vproc ID in its high digits and a counter-based value in its low digits.

There are two cases for INSERT; one is when only one single data row is processed and the other is when multiple data rows are processed with an Array INSERT request. In the case of an Array INSERT request, rows are sorted by their hash-owning AMPs so that the rows going to the same AMP can easily be grouped together into the same step. This random row hash will be generated once per request so that in the case of Array INSERT, the same random row hash is used for all of the rows. This means they all will be sent to the same AMP and usually in the same step.

FastLoad sends blocks of data to the AMPs. Each AMP (that receives blocks of data) uses random generator code **to distribute blocks of data between all of the AMPs in a round robin fashion.**

NoPI Random Generator

How is the AMP selected that will receive the row (or block of rows)?

- **The random generator can be executed at the PE or at the AMP level depending on the type of request (e.g., SQL versus FastLoad).**

For SQL-based functions, the PE uses the random generator.

- **The DBQL Query ID is used by the random generator to select a random hash value.**
  - **The approach is to generate a random hash bucket value in such a way that for a new request, data will generally be sent to a different AMP from the one that the previous request sent data to.**
  - **In the case of an Array INSERT request, this random hash bucket value will be generated once per request so that in the case of Array INSERT, the same random hash bucket value is used for all of the rows.**

For FastLoad-based functions, the AMP uses random generator code to distribute blocks of data between the AMPs in a round robin fashion.

The Row ID for a NoPI Table

For a NoPI table, the AMP will assign a RowID (64 bits) for a row or a set of rows using a hash bucket that the AMP owns. For a NoPI table, the RowID will consist of a 20-bit hash bucket followed by 44 bits that are used for the uniqueness part of the RowID. Only the first 20 bits (hash bucket) are used.

As more rows are added to the table, the uniqueness value is sequentially incremented.

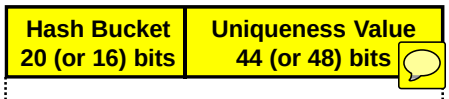
For systems using a 16-bit hash buckets, the RowID for a NoPI table will have 16 bits for the hash bucket value and 48 bits for the uniqueness id.

The Row ID for a NoPI Table

The RowID will still be 64 bits, but it is utilized a little differently in a NoPI table.

- The first 20 bits represents the hash bucket that is internally selected by the AMP.
- Remaining 44 bits are used for the uniqueness value of rows in a NoPI table.
- Note: Systems may be configured to use 16 bits for the hash bucket numbers – if so, then the uniqueness value will utilize 48 bits of the RowID.

Row ID for NoPI table



Each row still has a Row ID as a prefix.

Rows are logically maintained in Row ID sequence.

Row ID		Row Data		
Hash Bucket	Uniqueness	Cust_No	Last_Name	First_Name
000E7	00000000001	001018	Reynolds	Jane
000E7	00000000002	001020	Davidson	Evan
000E7	00000000003	001031	Green	Jason
000E7	00000000004	001014	Jacobs	Paul
000E7	00000000005	001012	Garcia	Jose
000E7	00000000006	001021	Carnet	Jean
:	:	:	:	:

The Row ID for a NoPI Table (cont.)

For a NoPI table, the AMP will assign a RowID (64 bits) for a row or a set of rows using a hash bucket that the AMP owns. This 64-bit RowID can be used by secondary and join indexes.

What is a different about the RowID for a NoPI table is that the uniqueness id is 44 bits long instead of 32 bits. The additional 12 bits available in the row hash are added to the 32-bit uniqueness. This gives a total of 44 bits to use for the uniqueness part of the RowID. For each hash bucket, there can be up to 17 trillion rows per AMP (approximately).

For systems using a 16-bit hash buckets, the RowID for a NoPI table will have 16 bits for the hash bucket value and 48 bits for the uniqueness id.

The RowID is still 64 bits long and a unique identifier of a row within a table.

The Row ID for a NoPI Table (cont.)

The RowID is 64 bits and can be referenced by secondary and join indexes.

- The first 20 (or 16) bits represent the hash bucket value which is internally chosen by and controlled by the AMP.
- Remaining 44 (or 48) bits are used for the uniqueness value of rows in a NoPI table. This module assumes that 20-bit hash bucket numbers are used.
  - The uniqueness value starts from 1 and will be sequentially incremented.
  - With 44 bits, there can be approximately 17 trillion rows on an AMP.
- Normally, all rows in a NoPI table on an AMP will have the same hash bucket value (first 20 bits) and the 44-bit uniqueness value will start at 1 and be sequentially incremented.
- Each row in a NoPI table will have a RowID with a hash bucket value that is actually owned by the AMP storing the row.

Fallback and index maintenance work the same as if the table is a primary index table.

As always, the RowID is transparent to the end-user.

Multiple NoPI Tables at the AMP Level

The facing page illustrates an example of two NoPI tables in a 27-AMP system.

Other NoPI considerations include:

Archive/Recovery Issues

Archive/Restore will be supported for NoPI table. Archiving a table or a database and restoring or copying that to the same system or a different system should work out fine with the existing scheme for NoPI table when no data redistribution takes place (same number of AMPs). Data redistribution takes place when there is a difference in configuration or hash function between the source system and the target system. In the case of a difference in configuration, each row in a table will be looked at and if its hash bucket belongs to some other AMP using the new configuration, that row will be redistributed to its hash-owning AMP.

Since one hash bucket is normally enough to use to assign RowID to all of the rows on each AMP, when we restore or copy data to a different configuration with more AMPs, there will be AMPs that will not have any data at all. This means that data in a NoPI table can be skewed after a Restore or Copy.

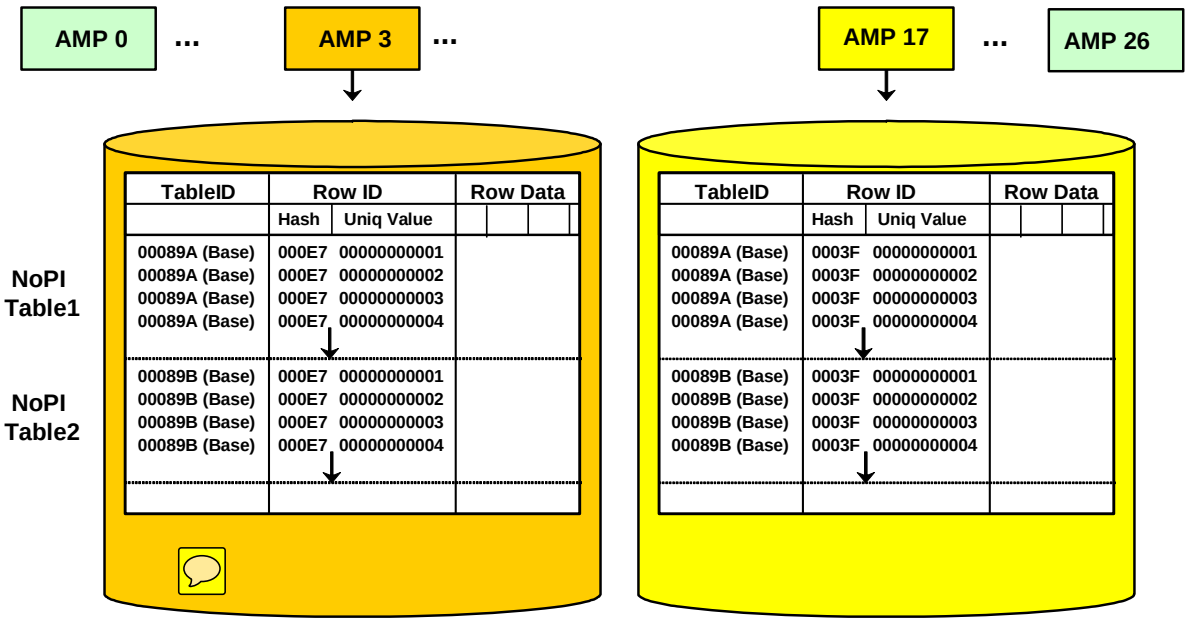
This is because permanent space is divided equally among the AMPs whether or not any of them get any data. As some AMPs not getting any data from a Restore or Copy, some other AMPs will get more data compared to what it was in the source system and this will require more space allocated overall.

However, as a staging table, NoPI table is not intended to stay around for too long so it is not expected to have many NoPI tables being restored or copied.

Reconfig Issues

Reconfig will be supported for NoPI table. The issue with Reconfig is very similar to that of Restore or Copy to a different configuration. Although rows in a NoPI table are not hashed based on the primary index and the AMPs where they reside are arbitrary, but each row does have a RowID with a hash bucket that is owned by the AMP storing that row. Redistributing rows in a NoPI table via Reconfig can be done by sending each row to the AMP that owns the hash bucket in that row based on the new configuration map. As with Restore and Copy, Reconfig can make a NoPI table skewed by going to a configuration with more AMPs.

Multiple NoPI Tables at the AMP Level



Data within an AMP is logically stored in Table ID / Row ID sequence.

Loading Data into a NoPI Table

The facing page summarizes various techniques of getting data inserted into a NoPI table.

Loading Data into a NoPI Table

Simple INSERTs

- For a simple INSERT, the PE selects a random AMP where the row is sent to. That AMP then turns the row into proper internal format and appends it to the end of the NoPI table.

INSERT–SELECT

- When inserting data from a source PI (or NoPI) table into a NoPI target table, data from the source table will NOT be redistributed and will be locally appended into the target table.

  INSERT-SELECT to a target NoPI table can result in a skewed NoPI table if the source table is skewed.

FastLoad

- Blocks of data are sent to the AMP load sessions and the AMP random generator code randomly distributes the blocks between the AMPs usually resulting in even distribution of the data between AMPs.

TPump

- With TPump Array INSERT, rows are packed together in a request and distributed to an AMP and then appended to the NoPI table on that AMP. Different requests are distributed to different AMPs by the PE. This will usually result in even distribution of the data between the AMPs.

NoPI Options

The following options are available to a NoPI table:

- FALLBACK
- Secondary indexes – USI and NUSI
- Join and reference indexes
- Primary Key and Foreign Key constraints are allowed on a NoPI table.
- LOBs are allowed on a NoPI table.
- INSERT and DELETE trigger actions are allowed on a NoPI table.
  - UPDATE trigger actions will be allowed starting with Teradata 13.00.00.03.
- NoPI table can be a Global Temporary or Volatile table.
- COLLECT/DROP STATISTICS are allowed on a NoPI table.
- FastLoad – note that duplicate rows are loaded and not deleted with a NoPI table

The following limitations apply to a NoPI table:

- SET is not allowed. Default is MULTISET for both Teradata and ANSI mode.
- No columns are allowed to be specified for the primary index.
- Partitioned primary index is not allowed.
- Permanent journaling is not allowed.
- Identity column is not allowed.
- Cannot be created as a queue or as an error table.
- Hash index is not allowed on a NoPI table.
- MultiLoad cannot be used to load a NoPI table.
- UPDATE, UPSERT, and MERGE-INTO operations are using the NoPI table as the target table.
  - UPDATE will be available with Teradata 13.00.00.03

NoPI Table Options

Options available with NoPI tables

- FALLBACK
- Secondary indexes – USI and NUSI
- Join and reference indexes
- Primary Key and Foreign Key constraints are allowed.
- LOBs are allowed on a NoPI table.
- INSERT and DELETE trigger actions are allowed on a NoPI table.
  - UPDATE trigger actions will be allowed starting with Teradata 13.00.00.03.
- Can be a Global Temporary or Volatile table.
- COLLECT/DROP STATISTICS are allowed.
- FastLoad – note that duplicate rows are loaded and not deleted with a NoPI table

Limitations of NoPI tables

- SET tables are not allowed.
- Partitioned primary index is not allowed.
- Permanent journaling is not allowed.
- Identity column is not allowed.
- Cannot be a queue or as an error table.
- Hash index is not allowed on a NoPI table.
- MultiLoad cannot be used on a NoPI table.
- UPDATE, UPSERT, and MERGE-INTO operations using the NoPI table as the target table are not allowed.
  - UPDATE will be available with Teradata 13.00.00.03

Summary

The facing page summarizes some of the key concepts covered in this module.

Summary

Tables with a Primary Index:

- **Base PI on the column(s) most often used for access, provided that the values are unique or nearly unique.**
- **Duplicate values hash to the same AMP and are stored in the same data block when possible.**
- **PRIMARY KEY and/or UNIQUE constraints are always implemented as a unique index (either a UPI or a USI.**

Tables without a Primary Index:

- **Although there is no primary index in a NoPI table, rows do have a valid row ID with both hash and uniqueness.**
  - **Hash value is internally selected in the AMP**
- **Rows in a NoPI table will be even distributed between the AMPs based upon a new code (i.e., random generator).**

Module 16: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 16: Review Questions

1. Which trade-off must be balanced to make the best choice for a primary index? ____
   a. Access and volatility
   b. Access and block size
   c. Block size and volatility
   d. Access and distribution

2. When volatility is considered as one of the Primary Index choice criteria, what is analyzed? ____
   a. Degree of uniqueness
   b. How often the data values will change
   c. How often the fixed length rows will change
   d. How frequently the column is used for access

3. To optimize the use of disk space, the designer should choose a primary index that _____.
   a. is non-unique
   b. consists of one column
   c. is unique or nearly unique
   d. consists of multiple columns
   e. has fewer distinct values than AMPs

Module 16: Review Questions (cont.)

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 16: Review Questions (cont.)

4. For NoPI tables, what are 2 ways in which the Random Generator is executed?
   a. At the AMP level with FastLoad
   b. At the PE level for ad hoc SQL requests
   c. At the TPump client level for array insert operations
   d. At the AMP level for INSERT-SELECT into an empty NoPI table

5. Assume DBSControl flag #53 (Primary Index Default) is set to N (No Primary Index), which two indexes are created for TableX given the following DDL command?

```
CREATE TABLE TableX
        (col1    INTEGER NOT NULL UNIQUE
        ,col2    CHAR(10) NOT NULL PRIMARY KEY
        ,col3    CHAR(80));
```

   a. col1 will be a UPI
   b. col1 will be a USI
   c. col2 will be a UPI
   d. col2 will be a USI

6. Which two options are permitted for NoPI tables?
   a. Fallback
   b. MultiLoad
   c. Hash Index
   d. BLOBs and CLOBs

Lab Exercise 16-1

Check your understanding of the concepts discussed in this module by completing the lab exercise as directed by your instructor.

Lab Exercise 16-1

Lab Exercise 16-1

Purpose

In this lab, you will use Teradata SQL Assistant to evaluate various columns of table as primary index candidates.

What you need

Populated PD.Employee table; your empty Employee table

Tasks

1. **INSERT/SELECT all rows from the populated PD.Employee table to your "Employee" table. Verify the number of rows in your table.**

   INSERT INTO Employee SELECT * FROM PD.Employee;

   SELECT COUNT(*) FROM Employee; Count = _____

Lab Exercise 16-1 (cont.)

Use the following SQL to determine the column metrics for this Lab.

# of Distinct Values for a column:

```
SELECT    COUNT(DISTINCT(column_name))
FROM      tablename;
```

Max Rows per Value for all values in a column:

```
SELECT    column_name, COUNT(*)
FROM      tablename      GROUP BY 1
ORDER BY 2 DESC;
```

Max Rows with NULL in a column:

```
SELECT    COUNT(*)
FROM      tablename
WHERE     column_name  IS NULL;
```

Average Rows per Value for a column (mean value):

```
SELECT    COUNT(*) / COUNT(DISTINCT(col_name))
FROM      tablename;
```

Typical Rows per Value for a column (median value):

```
SELECT    t_count   AS "Typical Rows per Value"
FROM      (SELECT col_name, COUNT(*)
           FROM    tablename GROUP BY 1)
                   t1 (t_colvalue, t_count),
          (SELECT COUNT(DISTINCT(col_name))
          FROM     tablename)
                   t2 (num_rows)
QUALIFY ROW_NUMBER () OVER
          (ORDER BY t1.t_colvalue) = t2.num_rows /2 ;
```

Lab Exercise 16-1 (cont.)

2. **Collect column demographics for each of these columns in Employee and determine if the column would be a primary index candidate or not.**

   By using the SHOW TABLE Employee command, you should be able to complete the Employee_number information without executing any SQL.

	Distinct Values	Max Rows for a Value	Max Rows NULL	Avg Rows per Value	Candidate for PI (Y/N)
Employee_Number					
Dept_Number					
Job_Code					
Last_name					

Lab Exercise 16-2

Distribution of table space by AMP:

```
SELECT      Vproc, TableName (CHAR(15)), CurrentPerm
FROM        DBC.TableSizeV
WHERE       DatabaseName = DATABASE
AND         TableName = 'tablename'
ORDER BY    1 ;
```

Lab Exercise 16-2

Lab Exercise 16-2

Purpose

In this lab, you will use the DBC.TableSizeV view to determine space distribution on a per AMP basis.

What you need

Your populated Employee table.

Tasks

1. Use SHOW TABLE command to determine which column is the Primary Index. PI = _____

   Determine the AMP space usage of your Employee table using DBC.TableSizeV.

   AMP #_____ has the least amount of permanent space – amount _____
   AMP #_____ has the greatest amount of permanent space – amount _____

2. Create a new table named Employee_2 with the same columns as Employee except specify Last_name as the Primary Index.

   Use INSERT/SELECT to populate Employee_2 from Employee.

   Determine the AMP space usage of your Employee_2 table using DBC.TableSizeV.

   AMP #_____ has the least amount of permanent space – amount _____
   AMP #_____ has the greatest amount of permanent space – amount _____

Notes

Module 17

Partitioned Primary Indexes

After completing this module, you will be able to:

- Describe the components that comprise a Row ID in a partitioned table.
- List two advantages of partitioning a table.
- List two potential disadvantages of partitioning a table.
- Create single-level and multi-level partitioned tables.
- Use the PARTITION key word to display partition information.

Notes

Table of Contents

Partitioning a Table 17-4
How is Partitioning Implemented? 17-6
Logical Example of NPPI versus PPI 17-8
Primary Index Access (NPPI) 17-10
Primary Index Access (PPI) 17-12
Why Partition a Table? 17-14
Advantages/Disadvantages of Partitioning 17-16
  Disadvantages of Partitioning 17-16
PPI Considerations 17-18
  Access of Tables with a PPI 17-18
How to Define a PPI 17-20
Partitioning with CASE_N and RANGE_N 17-22
Partitioning with RANGE_N – Example 1 17-24
  Access using Partitioned Data – Example 1 (cont.) 17-26
  Access Using Primary Index – Example 1 (cont.) 17-28
  Place a USI on NUPI – Example 1 (cont.) 17-30
  Place a NUSI on NUPI – Example 1 (cont.) 17-32
Partitioning with RANGE_N – Example 2 17-34
Partitioning – Example 3 17-36
Special Partitions with CASE_N and RANGE_N 17-38
Special Partition Examples 17-40
Partitioning with CASE_N – Example 4 17-42
  Additional examples: 17-42
SQL Use of PARTITION Key Word 17-44
SQL Use of CASE_N 17-46
Using ALTER TABLE with PPI Tables 17-48
ALTER TABLE – Example 5 17-50
ALTER TABLE – Example 5 (cont.) 17-52
ALTER TABLE TO CURRENT 17-54
ALTER TABLE TO CURRENT – Example 6 17-56
PPI Enhancements 17-58
Multi-level PPI Concepts 17-60
Multi-level PPI Concepts (cont.) 17-62
Multi-level Partitioning – Example 7 17-64
Multi-level Partitioning – Example 7 (cont.) 17-66
How is the MLPPI Partition # Calculated? 17-68
Character PPI 17-70
Character PPI – Example 8 17-72
Summary 17-74
Module 17: Review Questions 17-76
Lab Exercise 17-1 17-80

Partitioning a Table

As part of implementing a physical design, Teradata provides numerous indexing options that can improve performance for different types of queries and workloads. For example, secondary indexes, join indexes, or hash indexes may be utilized to improve performance for known queries. Teradata provides additional new indexing options to provide even more flexibility in implementing a Teradata database. One of these new indexing options is the Partitioned Primary Index (PPI). Key characteristics of Partitioned Primary Indexes are listed on the facing page.

Primary indexes can be partitioned or non-partitioned. A non-partitioned primary index (NPPI) is the traditional primary index by which rows are assigned to AMPs. Apart from maintaining their storage in row hash order, no additional assignment processing of rows is performed once they are hashed to an AMP.

A partitioned primary index (PPI) permits rows to be assigned to user-defined data partitions on the AMPs, enabling enhanced performance for range queries that are predicated on primary index values.

The Partitioned Primary Index (PPI) feature allows a class of queries to access a portion of a large table, instead of the whole table. The traditional uses of the Primary Index (PI) for data placement and rapid access of the data when the PI values are specified are retained.

Some common business queries generally require a full-table scan of a large table, even though it's predictable that a fairly small percentage of the rows will qualify. One example of such a query is a trend analysis application that compares current month sales to the previous month, or to the same month of the previous year, using a table with several years of sales detail. Another example is an application that compares customer behavior in one (fairly small) geographic region to another region.

Acronyms:

PI	– Primary Index
PPI	– Partitioned Primary Index
NPPI	– Non-Partitioned Primary Index

Partitioning a Table

What is a "Partitioned Primary Index" or PPI?

- A indexing mechanism in Teradata for use in physical database design.
- Data rows are grouped into partitions at the AMP level – partitioning is simply an ordering of the rows within a table on an AMP.

What advantages does partitioning provide?

- Increases the available options to improve the performance of certain types of queries – specifically range-constrained queries.
- Only the rows of the qualified partitions in a query need to be accessed – avoid full table scans.

How is a PPI created and managed?

- A PPI is easy to create and manage.
  - The CREATE TABLE and ALTER TABLE statements contain options to create and/or alter partitions.
- As always, data is distributed among AMPs and automatically placed within partitions.

How is Partitioning Implemented?

The PRIMARY INDEX clause (part of the CREATE TABLE statement) has been extended to include a PARTITION BY clause. This new partition expression definition is the only thing that needs to be done to create a partitioned table. Advantages to this approach are:

- No separate partition layout
- No disk layout for partitions
- No definition of location in the system for partition
- No need to define/manage separate tables per segment of the table that needs to be accessed
- Even data distribution and even processing of a logical partition is automatic due to the PI distribution of the rows

No query has to be modified to take advantage of a PPI table.

For tables with a PPI, Teradata utilizes a 3-level scheme to distribute and later locate the data. The 3 levels are:

- Rows are distributed across all AMPs (and accessed via the Primary Index) based upon HBN (Hash Bucket Number) portion of the Row Hash.

- At the AMP level, rows are first ordered by their partition number.

- Within the partition, data rows are logically stored in Row ID sequence.

A new term is associated with PPI tables. The **Row Key** is a combination of the Partition # and the Row Hash. The term Row Key will appear in EXPLAIN reports.

How is Partitioning Implemented?

Provides an additional level of data distribution and ordering.

- Rows are distributed across all AMPs (via Primary Index) based upon HBN portion of the Row Hash.
- Rows are first ordered by their partition number within the AMP.
- Within the partition, data rows are logically stored in Row ID sequence.

If a table is partitioned, rows are placed into partitions.

- Teradata 13.10 (and before) – partitions are numbered 1 to 65,535.
- Teradata 14.0 – maximum combined partitions is increased to 9.223 Quintillion.
  - If combined partitions is <= 65,535, then 2-byte partition numbers are used.
  - If combined partitions is > 65,535, then 8-byte partition numbers are used.

In a partitioned table, each row is uniquely identified by the following:

- Row ID = Partition # + Row Hash + Uniqueness Value
- Row Key = Partition # + Row Hash (e.g., Row Key will appear in Explain plans)
  - In a partitioned table, data rows will have the Partition # included as part of the data row.

To help understand how partitioning is implemented, this module will include examples of data access using tables defined with NPPI and PPI.

Logical Example of NPPI versus PPI

The facing page provides a logical example of an Orders table implemented with a NPPI (Non-Partitioned Primary Index) and the same table implemented with a PPI (Partitioned Primary Index). Only the Order_Number and a portion (YY/MM) of the Order_Date are shown in the example.

The column headings in this example represent the following:

RH – Row Hash – the two-digit row hash is used for simplification purposes. A true table would contain a Row ID for each row (Row Hash + Uniqueness Value). Note that as just in a real implementation, two different order numbers happen to hash to the same row hash value. Order numbers 1012 and 1043 on AMP 2 both hash to '36'.

O_# – Order Number – this example assumes that Order Number is the Primary Index and the data rows are hash distributed based on this value.

O_Date – Order Date – another column in the table. This example only contains orders for 4 months – from January, 2012 through April, 2012. For example, an order date, such as 12/01, represents January of 2012 (or 2012/01).

Important points to understand from this example:

- All of the rows in the NPPI table are stored in logical Row ID sequence (row hash + uniqueness value) within each AMP.
- The rows in the PPI table are first ordered by Partition Number, and then by Row Hash (actually Row ID) sequence within the Partition.
- This example illustrates 4 partitions – one for each of the 4 months shown in the example.
- A query that requests "order information" (with a WHERE condition that specifies a range of dates) will result in a full table scan of the NPPI table.
- The same query will only have to access the required partitions in the PPI table.

Logical Example of NPPI versus PPI

4 AMPs with Orders Table defined with Non-Partitioned Primary Index (NPPI).

RH	O_#	O_Date
'01'	1028	12/03
'03'	1016	12/02
'12'	1031	12/03
'14'	1001	12/01
'17'	1013	12/02
'23'	1040	12/04
'28'	1032	12/03
'30'	1038	12/04
'35'	1007	12/01
'39'	1011	12/01
'42'	1047	12/04
'48'	1023	12/02

RH	O_#	O_Date
'06'	1009	12/01
'07'	1017	12/02
'10'	1034	12/03
'13'	1037	12/04
'16'	1021	12/02
'21'	1045	12/04
'26'	1002	12/01
'29'	1033	12/03
'34'	1029	12/03
'36'	1012	12/01
'36'	1043	12/04
'45'	1015	12/02

RH	O_#	O_Date
'04'	1008	12/01
'05'	1048	12/04
'09'	1018	12/02
'15'	1042	12/04
'19'	1025	12/03
'24'	1004	12/01
'27'	1014	12/02
'32'	1003	12/01
'33'	1039	12/04
'40'	1035	12/03
'44'	1022	12/02
'47'	1027	12/03

RH	O_#	O_Date
'02'	1024	12/02
'08'	1006	12/01
'11'	1019	12/02
'18'	1041	12/04
'20'	1005	12/01
'22'	1020	12/02
'25'	1036	12/03
'31'	1026	12/03
'38'	1046	12/04
'41'	1044	12/04
'43'	1010	12/01
'46'	1030	12/03

4 AMPs with Orders Table defined with PPI on O_Date.

```
SELECT ...
WHERE O_Date
  BETWEEN '2012-03-01'
              AND
  '2012-03-31';
```

RH	O_#	O_Date
'14'	1001	12/01
'35'	1007	12/01
'39'	1011	12/01
'03'	1016	12/02
'17'	1013	12/02
'48'	1023	12/02
'01'	1028	12/03
'12'	1031	12/03
'28'	1032	12/03
'23'	1040	12/04
'30'	1038	12/04
'42'	1047	12/04

RH	O_#	O_Date
'06'	1009	12/01
'26'	1002	12/01
'36'	1012	12/01
'07'	1017	12/02
'16'	1021	12/02
'45'	1015	12/02
'10'	1034	12/03
'29'	1033	12/03
'34'	1029	12/03
'13'	1037	12/04
'21'	1045	12/04
'36'	1043	12/04

RH	O_#	O_Date
'04'	1008	12/01
'24'	1004	12/01
'32'	1003	12/01
'09'	1018	12/02
'27'	1014	12/02
'44'	1022	12/02
'19'	1025	12/03
'40'	1035	12/03
'47'	1027	12/03
'05'	1048	12/04
'15'	1042	12/04
'33'	1039	12/04

RH	O_#	O_Date
'08'	1006	12/01
'20'	1005	12/01
'43'	1010	12/01
'02'	1024	12/02
'11'	1019	12/02
'22'	1020	12/02
'25'	1036	12/03
'31'	1026	12/03
'46'	1030	12/03
'18'	1041	12/04
'38'	1046	12/04
'41'	1044	12/04

Primary Index Access (NPPI)

A non-partitioned table (NPPI) has a traditional primary index by which rows are assigned to AMPs. Apart from maintaining their storage in row hash order, no additional assignment processing of rows is performed once they are hashed to an AMP.

With a NPPI table, the PARSER will include Partition Number 0 in the request. For a table with a NPPI, all of the rows are assumed to be part of one partition (Partition 0).

Assuming that an SQL statement (e.g., SELECT) provides equality value(s) to the column(s) of a Primary Index, the TD Database software retrieves the row or rows from a single AMP as described below.

The Parsing Engine (PE) creates a four-part message composed of the Table ID, Partition #0, the Row Hash, and Primary Index value(s). The 48-bit Table ID is located via the Data Dictionary, the 32 bit Row Hash value is generated by the Hashing Algorithm, and the Primary Index value(s) come from the SQL request. The Parsing Engine (via the Data Dictionary) knows if a table has a NPPI and sets the Partition Number to 0.

The Message Passing Layer uses a portion of the Row Hash to determine to which AMP to send the request. The Message Passing Layer uses the HBN portion of the Row Hash (first 16 or 20 bits of the Row Hash) to locate a bucket in the **Hash Map(s)**. This bucket identifies to which AMP the PE will send the request. The Hash Maps are part of the Message Passing Layer interface.

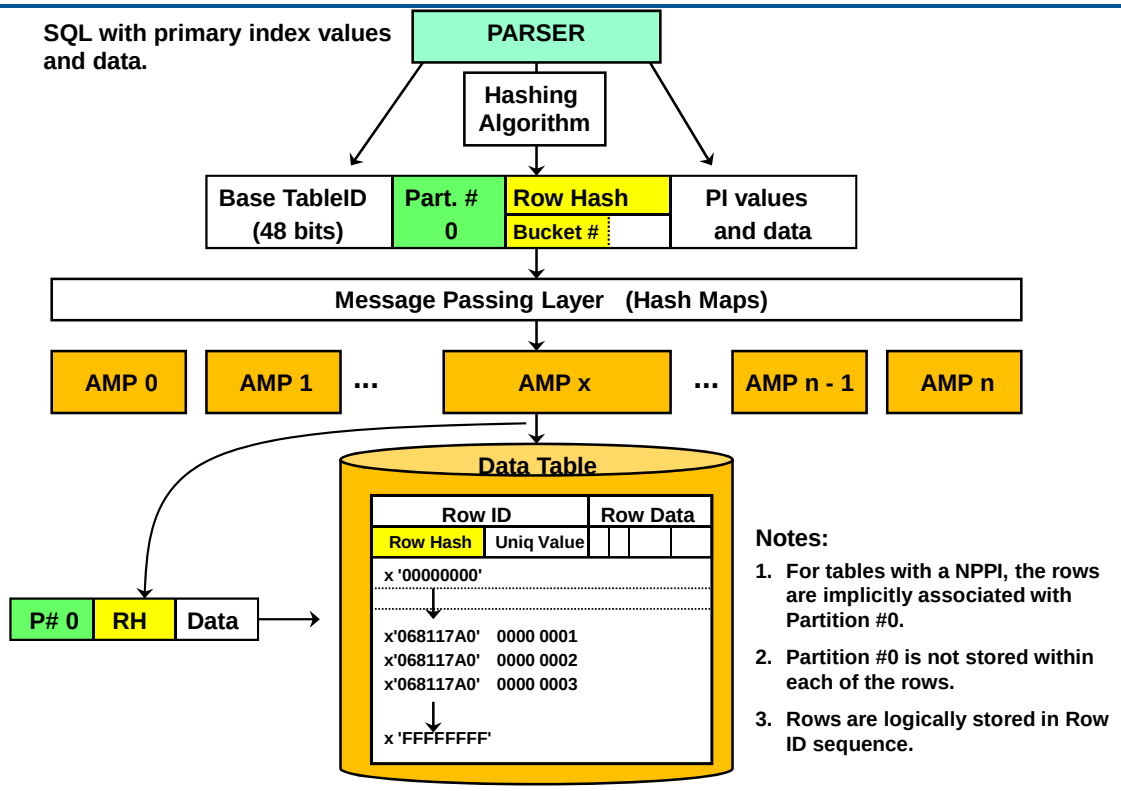
The AMP uses the Table ID and Row Hash to identify and locate the proper data block, then uses the Row Hash and PI value to locate the specific row(s). The PI value is required to distinguish between Hash Synonyms. The AMP implicitly assumes the rows are part of partition #0.

Note: The Partition Number (effectively 0) is not stored within the data rows for a table with a NPPI. The FLAG or SPARE byte (within the row overhead) has a bit set to zero for a NPPI row and it is set to one for a PPI row.

Acronyms:

- HBN – Hash Bucket Number
- PPI – Partitioned Primary Index
- NPPI – Non-Partitioned Primary Index

Primary Index Access (NPPI)



Notes:

1. For tables with a NPPI, the rows are implicitly associated with Partition #0.
2. Partition #0 is not stored within each of the rows.
3. Rows are logically stored in Row ID sequence.

Primary Index Access (PPI)

The process to locate a data row(s) via a PPI is similar to the process in retrieving data rows with a table defined with a NPPI – a process described earlier. If the SQL request provides data about columns associated with the partitions, then the PARSER will include specific partition information in the request.

- The key to remember is that a specific Row Hash value can be found in different partitions on the AMP. The Partition Number, Row Hash, and Uniqueness Value are needed to uniquely identify a row in a PPI-based table.

- A Row Hash and Uniqueness Value combination is only unique within a partition of a PPI table. The same Row Hash and Uniqueness Value combination can be present in different partitions (e.g., x'068117A0').

Assuming that an SQL statement (e.g., SELECT) provides equality value(s) to the Primary Index, then Teradata software retrieves the row(s) from a single AMP.

- If the SQL request also provides data for partition columns, then the AMP will only have to access the partition(s) identified in the request sent to it by the PE.

- If the SQL request only provides Primary Index values and the partitioning columns are outside of the Primary Index (and partitioning information is not included in the SQL request), the AMP will check each of the Partitions for the associated Row Hash.

The Parsing Engine (PE) creates a four-part message composed of the Table ID, Partition Information, the Row Hash, and Primary Index value(s). The 48-bit Table ID is located via the Data Dictionary, the 32-bit Row Hash value is generated by the Hashing Algorithm, and the Partition information and Primary Index value(s) come from the SQL request. The Parsing Engine (via the Data Dictionary) knows if a table has a PPI and determines the Partitions to include in the request based on the SQL request.

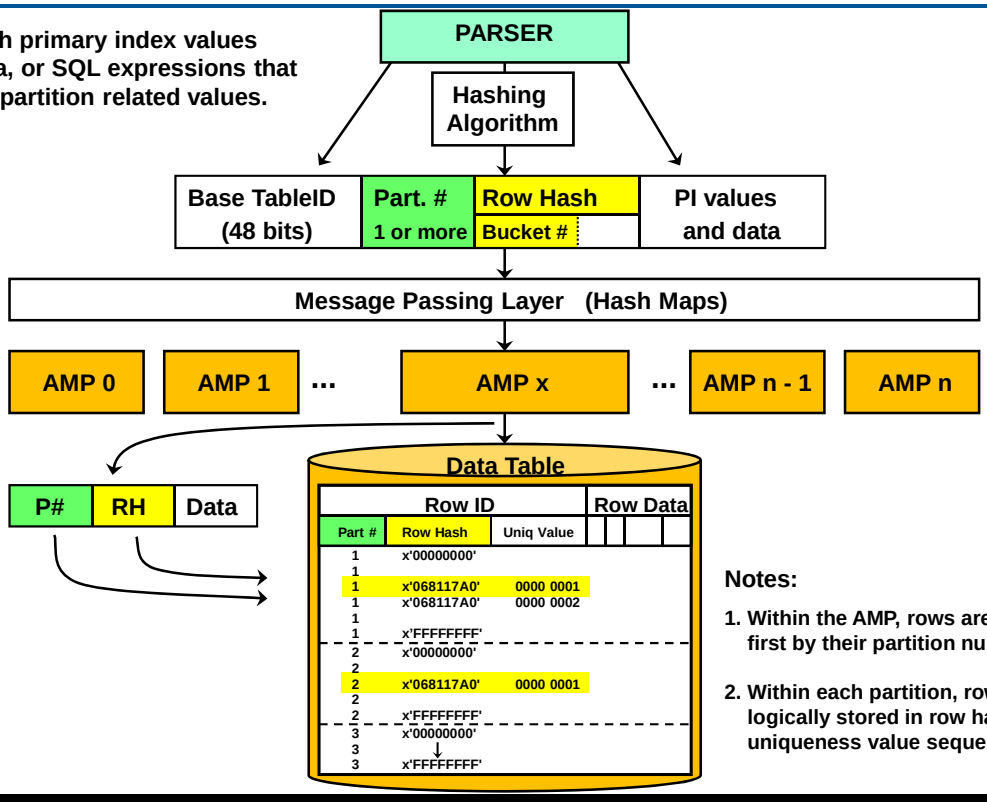
The Message Passing Layer uses a portion of the Row Hash to determine to which AMP to send the request. The Message Passing Layer uses the DSW portion of the Row Hash (first 16 or 20 bits of the Row Hash) to locate a bucket in the **Hash Map(s)**. This bucket identifies to which AMP the PE will send the request.

The AMP uses the Table ID, Partition Number(s), and Row Hash to identify and locate the proper data block(s). The AMP then uses the Row Hash and PI value to locate the specific row(s). The PI value is required to distinguish between Hash Synonyms. Each data row will have the Partition Number stored within it.

In the general case, there can be up to 65,535 partitions, numbered from one. As rows are inserted into the table, the partitioning expression is evaluated to determine the proper partition placement for that row. The two-byte partition number is embedded in the row, as part of the row identifier, making PPI rows two bytes wider than they would be if the table wasn't partitioned.

Primary Index Access (PPI)

SQL with primary index values and data, or SQL expressions that include partition related values.



Notes:

1. Within the AMP, rows are ordered first by their partition number.
2. Within each partition, rows are logically stored in row hash and uniqueness value sequence.

Why Partition a Table?

The decision to define a Partitioned Primary Index (PPI) for a table depends on how its rows are most frequently accessed. PPI tables are designed to optimize range queries while also providing efficient primary index join strategies. For range queries, only rows of the qualified partitions need to be accessed.

- One of the reasons to define a PPI on a table is to increase query efficiency by avoiding full table scans without the overhead and maintenance costs of secondary indexes.

The facing page provides one example using a sales data table that has 5 years of sales history. A PPI is placed on this table which partitions the data into 60 partitions (one for each month of the 5 years).

Queries that request a subset of the data (some number of months) only need to access the required partitions instead of the entire table. For example, a query that requests two months of sales data only needs to read 2 partitions of the data from each AMP. This is about 1/30 of the table. Without a PPI or any secondary indexes, this query has to perform a full table scan. Even with a secondary index, a full table scan would probably be done for 1/30 or 3% of the table.

The more partitions there are, the greater the potential benefit.

Some of the performance opportunities available by using the PPI feature include:

- Get more efficiency in querying against a subset of large volumes of transactional detail data as well as to manage this data more effectively.
  - Businesses have recognized the analytic value of detailed transactions and are storing larger and larger volumes of this data.
  - Increase query efficiency by avoiding full table scans without the overhead and maintenance costs of secondary indexes.
  - As the retention volume of detailed transactions increases, the percent of transactions that an “average” query requires for execution decreases.
- Allow “instantaneous” dropping of “old” data and simple addition of “new” data.
  - Support a “rolling n periods” methodology for transactional data.

The term “**partition elimination**” refers to an automatic optimization in which the optimizer determines, based on query conditions, that some partitions can't contain qualifying rows, and causes those partitions to be skipped. Partitions that are skipped for a particular query are called excluded partitions. Generally, the greatest benefit of a PPI table is obtained from partition elimination.

Why Partition a Table?

- **Increase query efficiency by avoiding full table scans without the overhead and maintenance costs of secondary indexes.**
  - **Partition Elimination – the key advantage to partitioning a table is that the optimizer can eliminate partitions for queries.**
- **For example, assume a sales data table has 5 years of sales history.**
  - **A PPI is placed on this table which partitions the data into 60 partitions (one for each month of the 5 years).**
  - **Assume a query only needs to read 2 months of the data from each AMP.**
    - **Only 1/30 (2 partitions) of the table has to be read.**
    - **With a NPPI, this query has to perform a full table scan.**
  - **A Valued-Ordered NUSI may be used to help performance for this type of query.**
    - **However, there is NUSI subtable permanent space and maintenance overhead.**
- **Deleting large volumes of rows in entire partitions can be extremely fast.**
  - **ALTER TABLE … DROP RANGE … ;**
  - **Disclaimer: Fast deletes assume that the table doesn't have a NO RANGE partition defined and has no secondary indexes, join indexes, or hash indexes.**

Advantages/Disadvantages of Partitioning

The main advantage of a PPI table is the automatic optimization that occurs for queries that specify a restrictive condition on the partitioning column. For example, a query which examines two months of sales data in a table with two years of sales history can read about one-twelfth of the table, instead of the entire table. The more partitions there are, the greater the potential benefit.

Disadvantages of Partitioning

The two main potential disadvantages of using a PPI table occur with PI access and direct PI-based joins. The PI access potential disadvantage occurs only when the partitioning column is not part of the PI. In this situation, a query specifying a PI value, but no value for the partitioning column, must look in each partition for that value, instead of positioning directly to the first row for the PI value.

The direct join potential disadvantage occurs when another table with the same PI is joined with an equality condition on every PI column. For two non-PPI tables, the rows of the two tables will be ordered the same, and the join can be performed directly. If one of the tables is partitioned, the rows won't be ordered the same, and the task, in effect, becomes a set of sub-joins, one for each partition of the PPI table.

In both of these situations, the disadvantage is proportional to the number of partitions, with fewer partitions being better than more partitions.

With the Aligned Row Format (Linux 64-bit), the two-byte partition number is embedded in the row, as part of the row identifier, plus an additional 2 bytes for a total of 4 additional bytes per data row. With the Packed64 Row Format (Linux 64-bit 13.10 new install), the overhead within in row for a PPI table is only 2 bytes for the partition number. Secondary Indexes referencing PPI tables use the 10-byte row identifier, making those subtable rows 2 bytes wider as well. Join Indexes always use a 10-byte row identifier regardless if the base tables are partitioned or not.

When the primary index is unique (but can't be defined as unique because of the partitioning), a USI or NUSI can be defined on the same columns as the primary index. Access via the secondary index won't be as fast as non-partitioned access via the primary index, but is fast enough for most applications.

Why can't a Primary Index be defined as Unique unless the partitioning expression columns are part of the PI column(s)?

It's because of the difficulty of performing the duplicate PI check for inserts. If there was already a row with that PI, it could be in any partition, so every partition would have to be checked to determine whether the duplicate PI exists. There can be thousands of partitions. An insert-select could take a very long time in such a situation. It's more efficient to check uniqueness (and it also provides an efficient access path) to define a unique secondary index (USI) on the same columns as the PI in this case.

Advantages/Disadvantages of Partitioning

Advantages:

- The partition expression definition is the only thing that needs to be done by the DBA or the database designer. No separate partition layout – no disk layout for partitions.
  - For example, the last row in one partition and the first row in the next partition will usually be in the same data block.
  - No definition of location in the system for partitions.
- Even data distribution and even processing of a logical partition is automatic.
  - Due to the PI distribution of the rows
- No modifications of queries required.

Potential disadvantages:

- PPI rows are 2 or 8 bytes longer. Table uses more PERM space.
  - Secondary index subtable rows are also increased in size.
- A PI access may be degraded if the partitioning column is not part of the PI.
  - A query specifying only a PI value must look in each partition for that value.
- Joins to non-partitioned tables with the same PI may be degraded.
- The PI can't be defined as unique when the partitioning column is not part of the PI.

PPI Considerations

Starting with Teradata V2R6.1, base tables, global temporary tables, and volatile temporary tables can be partitioned. This restriction doesn't mean that a PPI table can't have secondary indexes, or can't be referenced in the definition of a Join Index or Hash Index. It merely means that the PARTITION BY clause is not available on a CREATE JOIN INDEX or CREATE HASH INDEX statement.

In Teradata Database V2R6.2, Partitioned Primary Indexes (PPIs) are supported for non-compressed join indexes.

In the general case, there can be up to 65,535 partitions, numbered from one. The two-byte partition number is embedded in the data row, as part of the row identifier. Secondary Indexes and Join Indexes referencing PPI tables also use the wider row identifier. Except for the embedded partition number, PPI rows have the same format as non-PPI rows. A data block can contain rows from more than one partition. There are no new control structures needed to implement the partitioning scheme.

Access of Tables with a PPI

Some of the issues associated with accessing a table that has a defined PPI are listed below:

- If the SELECT statement does not provide values for any of the partitioning columns, then all of the partitions may be probed to find row(s) with the hash value.

- If the SELECT statement provides values for some of the partitioning columns, then partition elimination may reduce the number of the partitions that will be probed to find row(s) with the hash value.

  A common situation is with SQL specifying a range of values for partitioning columns. This allows some partitions to be excluded.

- If the SELECT statement provides values for all of the partitioning columns, then partition elimination will cause a single partition to be probed to find row(s) with the hash value.

In summary, a NUPI access of a PPI table will take longer when a query specifies the PI column values, but doesn't include the partitioning column(s). In this situation, each partition must be probed for the appropriate PI value. In the worst case, the number of disk reads could increase by a factor equal to the number of partitions. While probing a partition is a fast operation, a table with thousands of partitions might not provide acceptable performance for PI accesses for some applications.

PPI Considerations

PPI considerations include …

- **Base tables are partitioned, secondary indexes are not.**
- **However, a PPI table can have secondary indexes which reference rows in a PPI table via a RowID in the SI subtable.**
  - **Global and Volatile Temporary Tables can also be partitioned.**
  - **Non-Compressed Join Indexes can also be partitioned.**
- **A join or hash index can also reference rows in a PPI table.**

A table has a max of 65,535 (or 9.223 Quintillion) partitions.

- **Partitioning columns do not have to be columns in the primary index.**
- **There are numerous options for partitioning.**

As rows are inserted into the table, the partitioning expression is evaluated to determine the proper partition placement for that row.

How to Define a PPI

Primary indexes can be partitioned or non-partitioned. A primary index is defined as part of the CREATE TABLE statement. The PRIMARY INDEX definition has a new option to create partitioned primary indexes.

PARTITION BY <*partitioning expression*>

A partitioned primary index (PPI) permits rows to be assigned to user-defined data partitions on the AMPs, enabling enhanced performance for range queries that are predicated on partitioning columns(s) values. The <*partitioning_expression*> is evaluated and Teradata determines the appropriate partition number or assignment.

The <*partitioning-expression*> is a general expression, allowing wide flexibility in tailoring the partitioning scheme to the unique characteristics of the table. Two functions, CASE_N and RANGE_N, are provided to simplify the creation of common partitioning schemes. You can write any valid SQL expression as a partitioning expression with a few exceptions. The reference manual has details on SQL expressions that are not permitted in the <*partitioning expression*>.

Limitations on PARTITION BY option include:

- Partitioning expression must be a scalar expression that is INTEGER or can be cast to INTEGER.
- Multiple columns from the table may be specified in the expression
  - These are called the partitioning columns.
- Before Teradata 13.10, expression must not require character/graphic comparison in order to be evaluated.
  - Expression must not contain aggregate/ordered-analytic/statistical functions, DATE/, TIME, ACCOUNT, RANDOM, HASH, etc. functions.
- PARTITION BY clause not allowed for global temporary tables, volatile tables, join indexes, hash indexes, and secondary indexes in the first release of PPI.
- UNIQUE only allowed if all partitioning columns are included in the PI.
- Partitioning expression limited to approximately 8100 characters.
  - Stored as an implicit check constraint in DBC.TableConstraints

One or more columns can make up the partitioning expression, although it is anticipated that for most tables one column will be specified. The partitioning column(s) can be part of the primary index, but are not required to be. The result of the partitioning expression must be a scalar value that is INTEGER or can be cast to INTEGER. Most deterministic functions can be used within the expression. The expression must not require character or graphic comparisons, although character or graphic columns can be used in some circumstances.

How to Define a PPI

The PRIMARY INDEX definition portion of a CREATE TABLE statement has a optional PARTITION BY option.

```
CREATE TABLE …
    [UNIQUE] PRIMARY INDEX (col1, col2, …)
            PARTITION BY <partitioning-expression>
```

Options for the *<partitioning-expression>* include:

- **Range partitioning**
- **Conditional partitioning, modulo partitioning, and general expression partitioning.**
- **Partitioning columns do not have to be columns in the primary index. If they aren't, then the primary index cannot be unique.**

Column(s) included in the partitioning expression are called the "partitioning column(s)".

- **Two functions, CASE_N and RANGE_N, are provided to simplify the creation of common partitioning schemes.**

Partitioning with CASE_N and RANGE_N

For many tables, there is no suitable column that lends itself to direct usage as a partitioning column. For these situations, the CASE_N and RANGE_N functions can be used to concisely define partitioning expressions. When CASE_N or RANGE_N is used, two partitions are reserved for specific uses, leaving a maximum of 65,533 user-defined partitions. Note that the table still has a total of 65,535 available partitions.

The PARTITION BY phrase requires a partitioning expression that determines the partition assignment of a row. You can use the CASE_N function to construct a partitioning expression such that a row with any value or NULL for the partitioning column is assigned to a partition.

The CASE_N function is patterned after the SQL CASE expression. It evaluates a list of conditions and returns the position of the first condition that evaluates to TRUE, provided that no prior condition in the list evaluates to UNKNOWN. The returned value will map directly into a partition number.

Another option is to use the RANGE_N function to construct a partitioning expression with a list of ranges such that a row with any value or NULL for the partitioning column is assigned to a partition.

If CASE_N or RANGE_N is used in a partitioning expression in a CREATE TABLE or ALTER TABLE statement, it:

- Must not involve character or graphic comparisons
- Can specify a maximum of 65,533 user-defined partitions. The table can have a total of 65,535 partitions including the NO CASE (NO RANGE) and UNKNOWN partitions.

Partitioning with CASE_N and RANGE_N

The <partitioning expression> may use one of the following functions to help define partitions.

- CASE_N
- RANGE_N

Use of CASE_N results in the following:

- Evaluates a list of conditions and returns the position of the first condition that evaluates to TRUE.
- Result is the data row being placed into a partition associated with that condition.
- Note: Patterned after SQL CASE expression.

Use of RANGE_N results in the following:

- The expression is evaluated and is mapped into one of a list of specified ranges.
- Ranges are listed in increasing order and must not overlap with each other.
- Result is the data row being placed into a partition associated with that range.

NO CASE, NO RANGE, and UNKNOWN options are also available.

Partitioning with RANGE_N – Example 1

One of most common partitioning expression is to use RANGE_N partitioning to partition the table based on a group of dates (e.g., month partitions). A range is defined by a starting boundary and an optional ending boundary. If an ending boundary is not specified, the range is defined by its starting boundary, inclusively, up to but not including the starting boundary of the next range.

The list of ranges must specify ranges in increasing order, where the ending boundary of a range is less than the starting boundary of the next range.

RANGE_N Limitations include:

- Multiple test values are not allowed in a RANGE_N function.
- Test value in RANGE_N function must be INTEGER, BYTEINT, SMALLINT, or DATE.
- Range value and range size in a RANGE_N function must be constant.
- Ascending ranges only and ranges must not overlap with other.

For example, the following CREATE TABLE statement can be used to establish the monthly partitioning. This example does not have the NO RANGE partition defined.

```
CREATE SET TABLE Claim
    (claim_id        INTEGER     NOT NULL
    ,cust_id         INTEGER     NOT NULL
    ,claim_date      DATE        NOT NULL
        :
PRIMARY INDEX (claim_id)
      PARTITION BY RANGE_N (claim_date BETWEEN
          DATE '2003-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' MONTH);
```

To maintain uniqueness on the claim_id, you can include a USI on claim_id by including the following option.

```
    UNIQUE INDEX (claim_id)
```

If the claim_date column for an attempted INSERT or UPDATE has a date outside of the partitioning range or NULL, then an error will be returned and the row won't be inserted or updated.

Notes:

- UPI not allowed because partitioning column is not included in the PI.
- Unique Secondary Index is allowed on PI to enforce uniqueness.

The facing page contains examples of inserting data rows into a table partitioned by month and how the date is evaluated into the appropriate partition.

Partitioning with RANGE_N – Example 1

For example, partition the Claim table by "Claim Date".

```
CREATE TABLE Claim
      ( claim_id      INTEGER          NOT NULL
       ,cust_id       INTEGER          NOT NULL
       ,claim_date    DATE             NOT NULL
        … )
PRIMARY INDEX (claim_id)
    PARTITION BY RANGE_N
      (claim_date BETWEEN DATE '2003-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' MONTH,
       NO RANGE);
```

The following INSERTs place new rows into the Claim table. The date is evaluated and the rows are placed into the appropriate partitions.

INSERT INTO Claim VALUES (100039,1009, '2003-01-13', …); → placed in partition #1
INSERT INTO Claim VALUES (260221,1020, '2012-01-07', …); → placed in partition #109
INSERT INTO Claim VALUES (350221,1020, '2013-01-01', …); → placed in no range partition (#121)
INSERT INTO Claim VALUES (100039, 1009, NULL, …); → Error 3811 – NOT NULL violation

If the table did not have the NO RANGE partition defined, then the following error occurs:

INSERT INTO Claim VALUES (100039, 1009, '2013-01-01', …); (5728 – Partitioning violation)

Note: claim_id must be defined as a NUPI because claim_date is not part of PI.

Access using Partitioned Data – Example 1 (cont.)

The EXPLAIN text for these queries is shown below.

```
EXPLAIN   SELECT    *
          FROM      Claim_PPI
          WHERE     claim_date
          BETWEEN   DATE '2012-01-01' AND DATE '2012-01-31';
```

1) First, we lock a distinct DS."pseudo table" for read on a RowHash to prevent global deadlock for DS.Claim_PPI.
2) Next, we lock DS.Claim_PPI for read.
3) We do an **all-AMPs RETRIEVE step from a single partition** of DS.Claim_PPI with a condition of ("(DS.Claim_PPI.claim_date <= DATE '2012-01-31') AND (DS.Claim_PPI.claim_date >= DATE '2012-01-01')") into Spool 1 (group_amps), which is built locally on the AMPs. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with high confidence to be 21,100 rows (2,869,600 bytes). The estimated time for this step is 0.44 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.44 seconds.

The table named Claim_NPPI is similar to Claim_PPI except it does not have a Partitioned Primary Index, but does have “claim_id” as a UPI.

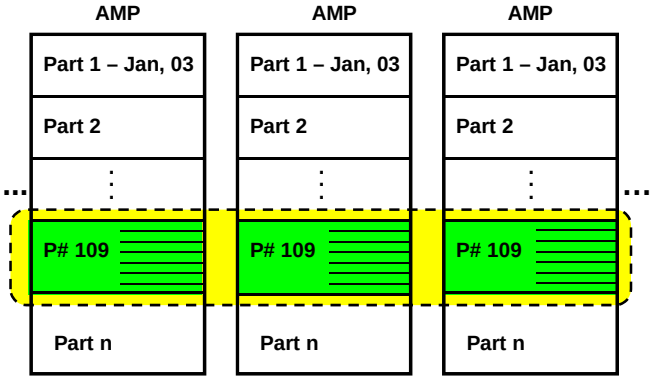
```
EXPLAIN   SELECT    *
          FROM      Claim_NPPI
          WHERE     claim_date
          BETWEEN   DATE '2011-01-01' AND DATE '2011-01-31';
```

1) First, we lock a distinct DS."pseudo table" for read on a RowHash to prevent global deadlock for DS.Claim_NPPI.
2) Next, we lock DS.Claim_NPPI for read.
3) We do an **all-AMPs RETRIEVE step** from DS.Claim_NPPI **by way of an all-rows scan** with a condition of ("(DS.Claim_NPPI.claim_date <= DATE '2012-01-31') AND (DS.Claim_NPPI.claim_date >= DATE '2012-01-01')") into Spool 1 (group_amps), which is built locally on the AMPs. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with high confidence to be 21,100 rows (2,827,400 bytes). The estimated time for this step is 49.10 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 49.10 seconds.

Note: Statistics were collected on the claim_id, cust_id, and claim_date of both tables. The Claim table has 1,440,000 rows.

Access using Partitioned Data – Example 1



QUERY – PPI

```
SELECT  *
FROM    Claim_PPI
WHERE   claim_date BETWEEN
        DATE '2012-01-01' AND
        DATE '2012-01-31' ;
```

PLAN – PPI

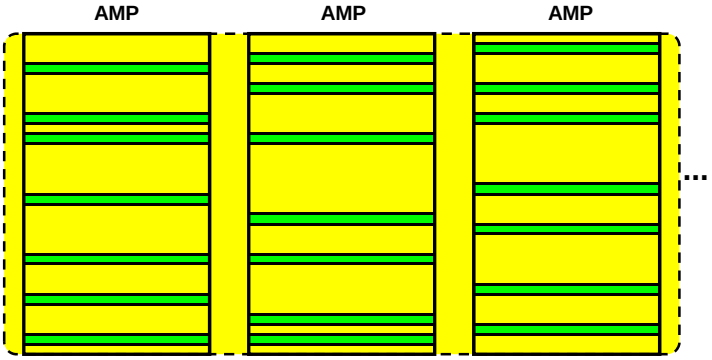
ALL-AMPs – Single Partition Scan
EXPLAIN estimated cost – 0.44 sec.

QUERY – NPPI

```
SELECT  *
FROM    Claim_NPPI
WHERE   claim_date BETWEEN
        DATE '2012-01-01' AND
        DATE '2012-01-31' ;
```

PLAN – NPPI

ALL-AMPs – Full Table Scan
EXPLAIN estimated cost – 49.10 sec.



Access Using Primary Index – Example 1 (cont.)

The EXPLAIN text for these queries is shown below.

```
EXPLAIN   SELECT    *
          FROM      Claim_PPI
          WHERE     claim_id = 260221;
```

1) First, we do a **single-AMP RETRIEVE step from all partitions** of DS.Claim_PPI by way of the primary index "DS.Claim_PPI.claim_id = 260221" with a residual condition of ("DS.Claim_PPI.claim_id = 260221") into Spool 1 (one-amp), which is built locally on that AMP. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 (136 bytes) is estimated with high confidence to be 1 row. The estimated time for this step is 0.09 seconds.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. **The total estimated time is 0.09 seconds.**

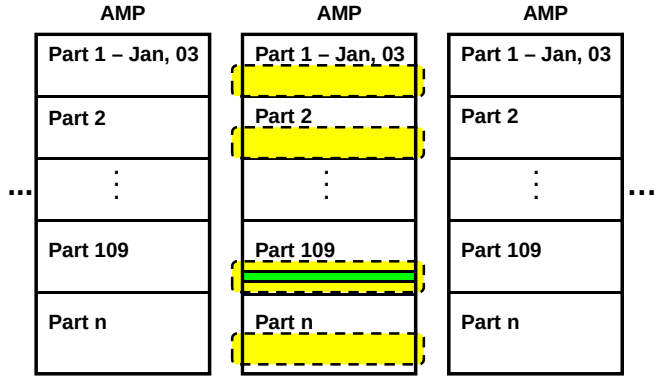
The table named Claim_NPPI is similar to Claim_PPI except it does not have a Partitioned Primary Index, but does have “claim_id” as a UPI.

```
EXPLAIN   SELECT    *
          FROM      Claim_NPPI
          WHERE     claim_id = 260221;
```

1) First, we do a **single-AMP RETRIEVE step** from DS.Claim_NPPI by way of the unique primary index "DS.Claim_NPPI.claim_id = 260221" with no residual conditions. The estimated time for this step is 0.00 seconds.

-> The row is sent directly back to the user as the result of statement 1. **The total estimated time is 0.00 seconds.**

Access Using Primary Index – Example 1 (cont.)



QUERY – PPI

```
SELECT   *
FROM     Claim_PPI
WHERE    claim_id =  260221;
```

PLAN – PPI

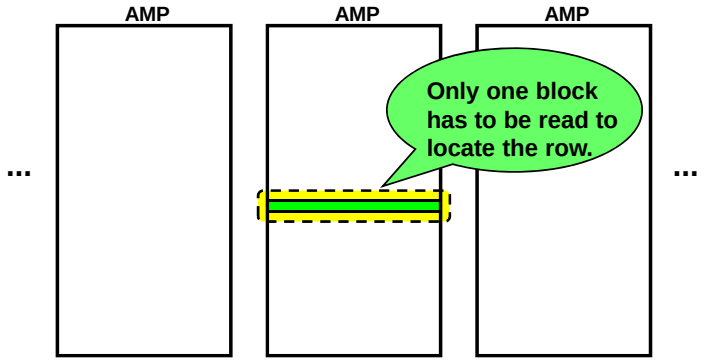
One AMP – All Partitions are probed
EXPLAIN estimated cost – 0.09 sec.

QUERY – NPPI

```
SELECT  *
FROM    Claim_NPPI
WHERE   claim_id = 260221;
```

PLAN – NPPI

One AMP – UPI Access
EXPLAIN estimated cost – 0.00 sec.



Place a USI on NUPI – Example 1 (cont.)

If the partitioning columns are not part of the Primary Index, the Primary Index cannot be unique (e.g., claim_date). To maintain uniqueness on the Primary Index, you can create a USI on the PI (e.g., Claim ID or claim_id).

Reasons for this may include:

- USI access to specific rows may be faster than scanning multiple partitions on a single AMP.
- Establish the USI as a referenced parent in Referential Integrity.

```
CREATE UNIQUE INDEX (claim_id) ON Claim_PPI;

EXPLAIN   SELECT    *
          FROM      Claim_PPI
          WHERE     claim_id =  260221;
```

1) First, we do a **two-AMP RETRIEVE** step from DS.Claim_PPI by way of **unique index # 4** "DS.Claim_PPI.claim_id = 260221" with no residual conditions. The estimated time for this step is 0.00 seconds.
-> The row is sent directly back to the user as the result of statement 1. The total estimated time is 0.00 seconds.

As an alternative, the SELECT can include the Primary Index values and the partitioning information. This allows the PE to build a request that has the AMP scan a specific partition. However, in this example, the user may not know the claim date in order to include it in the query.

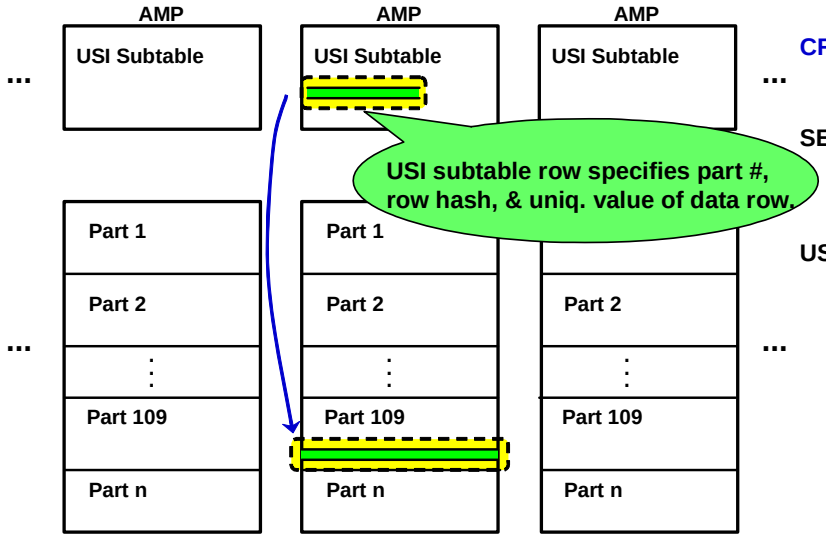
```
EXPLAIN   SELECT *
          FROM      Claim_PPI
          WHERE     claim_id = 260221
          AND       claim_date = DATE '2012-01-11';
```

1) First, we do a single-AMP RETRIEVE step from DS.Claim_PPI by way of the primary index "DS.Claim_PPI.claim_id = 260221, DS.Claim_PPI.claim_date = DATE '2012-01-11'" with a residual condition of ("(DS.Claim_PPI.claim_date = DATE '2012-01-11') AND (DS.Claim_PPI.claim_id = 260221)") into Spool 1 (one-amp), which is built locally on that AMP. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 (136 bytes) is estimated with high confidence to be 1 row. The estimated time for this step is 0.00 seconds.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.00 seconds.

Place a USI on NUPI – Example 1 (cont.)

Notes:

- **If the partitioning column(s) are not part of the Primary Index, the Primary Index cannot be unique (e.g., Claim Date is not part of the PI).**
- **To maintain uniqueness on the Primary Index, you can create a USI on the PI (e.g., Claim ID). This is a two-AMP operation.**




```
CREATE UNIQUE INDEX
   (claim_id) ON Claim_PPI ;

SELECT    *
  FROM    Claim_PPI
  WHERE   claim_id = 260221;
```

USI Considerations:

- **Eliminate partition probing**
- **Row-hash locks**
- **2-AMP operation**
- **Can only be used if values in PI column(s) are unique**
- **Will maintain uniqueness**
- **USI on NUPI only supported on PPI tables**

Place a NUSI on NUPI – Example 1 (cont.)

If the partitioning columns are not part of the Primary Index, the Primary Index cannot be unique (e.g., Claim ID). You can use a NUSI on the same columns that make up the PI and actually get a single-AMP access operation. This feature only applies to a NUSI created on the same columns as a PI on PPI table. Additionally, instead of table level locks (typical NUSI), row hash locks will be used.

Reasons to choose a NUSI for your PI may include:

- The primary index is non-unique (can't use a USI) and you need faster access than scanning or probing multiple partitions on a single AMP.
- MultiLoad can be used to load a table with a NUSI, not a USI.
- The access time for a USI and NUSI will be similar (each will access a subtable block) – however, the USI is a 2-AMP operation and requires BYNET message passing. The amount of space for a USI and NUSI subtable in this case will be similar. A typical NUSI with duplicate values will have multiple row ids (keys) in a subtable row and will save space per subtable row. However, a NUSI used as an index for columns with unique values will use approximately the same amount of subtable space as a USI. This is because each NUSI subtable row only contains 1 row id.

```
CREATE INDEX (claim_id) ON Claim_PPI;


EXPLAIN   SELECT *
          FROM    Claim_PPI
          WHERE   claim_id = 260221;
```

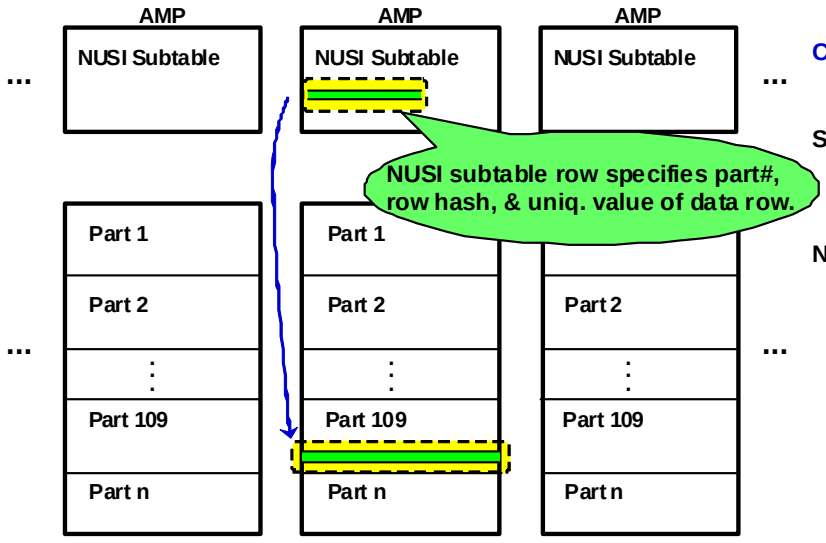
1) First, we do a **single-AMP RETRIEVE step** from DS.Claim_PPI by way of **index # 4** "DS.Claim_PPI.claim_id = 260221" with no residual conditions into Spool 1 (group_amps), which is built locally on that AMP. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 (136 bytes) is estimated with high confidence to be 1 row. The estimated time for this step is 0.00 seconds.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.00 seconds.

Place a NUSI on NUPI – Example 1 (cont.)

Notes:

- You can optionally create a NUSI on the same columns as the Primary Index (e.g., Claim ID). The PI may be unique or not.
- Optimizer generates a plan for a single-AMP NUSI access with row-hash locking (instead of table-level locking).




```
CREATE INDEX (claim_id)
   ON Claim_PPI;

SELECT    *
  FROM    Claim_PPI
  WHERE   claim_id = 260221;
```

NUSI Considerations:

- Eliminate partition probing
- Row-hash locks
- 1-AMP operation
- Can be used with unique or non-unique PI columns
- Must be equality condition
- NUSI Single-AMP operation only supported on PPI tables
- Use MultiLoad to load table

Partitioning with RANGE_N – Example 2

This example illustrates that a table can be partitioned with different size intervals. The current Sales data and Sales History data are placed in the same table. It typically is not practical to create a partitioning expression as shown in example #2, but the example is included to show the flexibility that you have with the partitioning expression.

For example, you may decide to partition the Sales History by month and the current sales data by day. You may want to do this if users frequently access the Sales History data with range constraints, resulting in full table scans. It may be that users access the current year data frequently looking at data for a specific day. The example on the facing page partitions the years 2002 to 2011 by month and the year 2011 by day.

One option may be to partition by week as follows:

```
PARTITION BY RANGE_N (sales_date
    BETWEEN DATE '2003-01-01' AND DATE '2003-12-31' EACH INTERVAL '7' DAY,
            DATE '2004-01-01' AND DATE '2004-12-31' EACH INTERVAL '7' DAY,
                      :                       :
            DATE '2012-01-01' AND DATE '2012-12-31' EACH INTERVAL '7' DAY);
```

One may think that a simple partitioning scheme to partition by week would be as follows:

```
PARTITION BY RANGE_N (sales_date
     BETWEEN DATE '2003-01-01' AND DATE '2012-12-31' EACH INTERVAL '7' DAY);
```

This is a simpler PARTITION expression to initially code, but may require more work or thought later. There is a minor drawback to partitioning by weeks because a 7-day partition usually spans one year into the next. Assume that a year from now, you wish to ALTER this table and DROP the partitions for the year 2003. The ALTER TABLE DROP RANGE option has to specify a range of dates that actually represent a complete partition or partitions in the table. A complete partition ends on 2003-12-19, not 2003-12-31. The ALTER TABLE command will be described later in this module.

If daily partitions are desired for all of the years, the following partitioning expression can be used to create a partitioned table with daily partitions.

```
PARTITION BY RANGE_N (
    sales_date  BETWEEN
        DATE '2003-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' DAY);
```

Performance Note: Daily partitions for ten years creates 3653 partitions (10 x 365 plus three leap days) and may not useful in many situations. Try to avoid daily partitions over a long period of time.

Partitioning with RANGE_N – Example 2

Notes:

- **This example places current and history sales data into one table.**
- **Current year data is partitioned on a more granular basis (daily) while historical sales data is placed into monthly partitions.**
- **Partitions of varying intervals can be created on the same PPI for a table.**

```
CREATE TABLE Sales_and_SalesHistory
    ( store_id                  INTEGER NOT NULL,
      item_id                   INTEGER NOT NULL,
      sales_date                DATE FORMAT 'YYYY-MM-DD',
      total_revenue             DECIMAL(9,2),
      total_sold                INTEGER,
      note                      VARCHAR(256))
PRIMARY INDEX  (store_id, item_id)
PARTITION BY RANGE_N (
      sales_date  BETWEEN
          DATE '2003-01-01' AND DATE '2011-12-31' EACH INTERVAL '1' MONTH,
          DATE '2012-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' DAY);
```

To partition by week, the following partitioning can be used.

```
PARTITION BY RANGE_N (sales_date BETWEEN
                  DATE '2003-01-01' AND DATE '2003-12-31' EACH INTERVAL '7' DAY,
                  DATE '2004-01-01' AND DATE '2004-12-31' EACH INTERVAL '7' DAY,
                        :                         :
```

Partitioning – Example 3

This example partitions by Store Id (store number). Prior to Teradata 14.0, a table has a maximum limit of 65,535 partitions. Therefore, the partitioning expression value from Store Id or an expression involving Store Id must be between 1 and 65,535.

If a company had a small number of stores, you could use the RANGE_N expression to limit the number of possible partitions. The alternative partitioning (that is shown on facing page) expression allows for ten partitions instead of 65,535. The optimizer may be able to more accurately cost join plans when the maximum number of partitions is known and small, making this a better choice than using the column directly.

Assume that a company has 1000 stores, and the store numbers (store_id) are from 100001 to 101001. To utilize 1000 partitions, the following partitioning expression could be defined.

```
... PRIMARY INDEX (store_id, item_id, sales_date)
    PARTITION BY store_id – 100000;
```

If a company has a small number of stores and a small number of products, another option may be to partition by a combination of Store Id and Item Id.

Assume the following:

Store numbers – 100001 to 100065 - less than 65 stores
Item numbers – 5000 to 5999 - less than 1000 item ids

Basically, the table has three-digit item_id codes and less than 65 stores.

This table could be partitioned as follows:

```
... PRIMARY INDEX (store_id, item_id, sales_date)
    PARTITION BY ((store_id – 100000) * 1000 + (item_id – 5000));
```

Assume that the store_id is 100009 and the item_id is 5025. This row would be placed in partition # 9025.

If many queries specify both a Store Id and an Item Id, this might be a useful partitioning scheme. Even if it wouldn't be useful, it demonstrates that the physical designers and/or database administrators have wide latitude in defining generalized partitioning schemes to meet the needs of individual tables.

Partitioning – Example 3

Notes:

- The simplest partitioning expression uses one column from the row without modification. Before Teradata 14.0, the column values must be between 1 and 65,535.
- Assume the store_id is a value between 100001 and 101001. Therefore, a simple calculation can be performed.
- This example will partition by data by store_id and effectively utilize 1000 partitions.

```
CREATE TABLE Store_Sales
   (  store_id          INTEGER NOT NULL,
      item_id           INTEGER NOT NULL,
      sales_date        DATE FORMAT 'YYYY-MM-DD',
      total_revenue     DECIMAL(9,2),
      total_sold        INTEGER,
      note              VARCHAR(256))
UNIQUE PRIMARY INDEX (store_id, item_id, sales_date)
      PARTITION BY store_id - 100000;
```

Alternative Definition:

- Assume the customer wishes to group these 1000 stores into 100 partitions.
- The RANGE_N expression can be used to identify the number of partitions and group multiple stores into the same partition.

```
PARTITION BY RANGE_N ( (store_id - 100000) BETWEEN 1 AND 1000 EACH 10);
```

Special Partitions with CASE_N and RANGE_N

The keywords, NO CASE (or NO RANGE) [OR UNKNOWN] and UNKNOWN are used to define the specific-use partitions.

Even if these options are not specified with the CASE_N (or RANGE_N) expressions, these two specific-use partitions are still reserved in the event the ALTER TABLE command is later used to add these options.

If it is necessary to test a CASE_N condition directly as NULL, it needs to be the first condition listed. This following example is correct. NULLs will be placed in partition #1.

```
PARTITION BY CASE_N
     (col3  IS NULL,
      col3  <  10,
      col3  <  100,
      NO CASE OR UNKNOWN)

INSERT INTO PPI_TabA VALUES (1, 'A', NULL, DATE);
INSERT INTO PPI_TabA VALUES (2, 'B', 5, DATE);
INSERT INTO PPI_TabA VALUES (3, 'C', 50, DATE);
INSERT INTO PPI_TabA VALUES (4, 'D', 500, DATE);
INSERT INTO PPI_TabA VALUES (5, 'E', NULL, DATE);

SELECT PARTITION AS "Part #", COUNT(*) FROM PPI_TabA
GROUP BY 1 ORDER BY 1;
```

Part #	Count(*)
1	2
2	1
3	1
4	1

Although you can code an example as follows, it should not be coded this way and will provide inconsistent results. NULLs will be placed in partition #4.

```
PARTITION BY CASE_N
     (col3  <  10,
      col3  IS NULL,
      col3  <  100,
      NO CASE OR UNKNOWN)

SELECT PARTITION AS "Part #", COUNT(*) FROM PPI_TabA
GROUP BY 1 ORDER BY 1;
```

Part #	Count(*)
1	1
3	1
4	3

Special Partitions with CASE_N and RANGE_N

The CASE_N and RANGE_N can place rows into specific-use partitions when ...

- the expression doesn't meet any of the CASE and RANGE expressions.
- the expression evaluates to UNKNOWN.
- two partition numbers are reserved even if the above options are not used.

The PPI keywords used to define two specific-use partitions are:

- **NO CASE (or NO RANGE) [OR UNKNOWN]**
  - If this option is used, then a specific-use partition is used when the expression isn't true for any case (or is out of range).
  - If OR UNKNOWN is included with the NO CASE (or NO RANGE), then UNKNOWN expressions are also placed in this partition.
- **UNKNOWN**
  - If this option is specified, a different specific-use partition is used for unknowns.
- **NO CASE (or NO RANGE), UNKNOWN**
  - If this option is used, then two separate specific-use partitions are used when the expression isn't true for any case (or is out of range) and different special partition is used for NULLs.

Special Partition Examples

This example assumes the following CASE_N expression.

```
PARTITION BY CASE_N (
    col3 <  10 ,
    col3 <  100 ,
    col3 <  1000 ,
    NO CASE OR UNKNOWN)
```

This statement creates four partitions, conceptually numbered (*Note) from one to four in the order they are defined. The first partition is when col3 is less than 10, the second partition is when col3 is at least 10 but less than 100, and the third partition is when col3 is at least 100 but less than 1,000.

The NO CASE OR UNKNOWN partition is for any value which isn't true for any previous CASE_N expression. In this case, it would be when col3 is equal to or greater than 1,000 or when col3 is NULL.

This partition is also used for values for which it isn't possible to determine the truth of the previous CASE_N expressions. Usually, this is a case where col3 is NULL or unknown.

Internally, UNKNOWN (option by itself) rows are assigned to partition #1. NOCASE (NO RANGE) OR UNKNOWN rows are physically assigned to partition #2. Internally, the first user-defined partition is actually partition #3.

The physical implementation in the file system is:

col3 < 10	– partition #1 (internally, rows placed in partition # 3)
col3 < 100	– partition #2 (internally, rows placed in partition # 4)
col3 < 1000	– partition #3 (internally, rows placed in partition # 5)
NO CASE or UNKNOWN	– partition #4 (internally, rows placed in partition # 2)

It is **NOT** syntactically possible to code a partitioning expression that has both NO CASE OR UNKNOWN, and UNKNOWN in the same expression. UNKNOWN expressions will either be placed in the partition with NO CASE or in a partition of their own. The following SQL is **NOT** permitted.

```
PARTITION BY CASE_N (
    col3 <  10 ,
       :
    NO CASE OR UNKNOWN,
    UNKNOWN)                    - causes an error
```

Special Partition Examples

The following examples illustrate the use of NO CASE and UNKNOWN options.

```
Ex. 1 PARTITION BY CASE_N (
        col3 <  10 ,
        col3 <  100 ,
        col3 <  1000 ,
        NO CASE OR UNKNOWN)
```

If col3 = 5, row is assigned to Partition #1.
If col3 = 50, row is assigned to Partition #2.
If col3 = 500, row is assigned to Partition #3.
If col3 = 5000, row is assigned to Partition #4.
If col3 = NULL, row is assigned to Partition #4.

In summary, NO CASE and UNKNOWN rows are placed into the same partition.

```
Ex. 2 PARTITION BY CASE_N (
        col3 <  10 ,
        col3 <  100 ,
        col3 <  1000 ,
        NO CASE,
        UNKNOWN)
```

If col3 = 5, row is placed in Partition #1.
If col3 = 50, row is placed in Partition #2.
If col3 = 500, row is placed in Partition #3.
If col3 = 5000, row is placed in Partition #4.
If col3 = NULL, row is placed in Partition #5.

In summary, NO CASE and UNKNOWN rows are placed into separate partitions.

Note: RANGE_N works in a similar manner.

Partitioning with CASE_N – Example 4

This example illustrates the capability of partitioning based upon conditions (CASE_N).

For example, assume a table has a total revenue column, defined as decimal. The table could be partitioned on that column, so that low revenue products are separated from high revenue products. The partitioning expression could be written as shown on the facing page. In this example, 8 partitions are defined for total revenue values up to 100,000. Two additional partitions are defined – one for revenues greater than 100,000 and another for unknown revenues (e.g., NULL).

Teradata 13.10 Note: Teradata 13.10 allows CURRENT_DATE and/or CURRENT_TIMESTAMP with partitioning expressions. However, it is recommended to NOT use these in a CASE expression for a partitioned primary index (PPI). Why? In this case, all rows are scanned during reconciliation.

Additional examples:

The following examples illustrate the use of the NO CASE option by itself or the UNKNOWN option by itself.

```
Ex.1    PARTITION BY CASE_N (
            col3 <  10 ,
            col3 <  100 ,
            col3 <  1000 ,
            NO CASE)

        If col3 = 5, row is assigned to Partition #1.
        If col3 = 50, row is assigned to Partition #2.
        If col3 = 500, row is assigned to Partition #3.
        If col3 = 5000, row is assigned to Partition #4.
        If col3 = NULL, Error 5728

        5728: Partitioning violation for table DBname.Tablename.

Ex. 2   PARTITION BY CASE_N (
            col3 <  10 ,
            col3 <  100 ,
            col3 <  1000 ,
            UNKNOWN)

        If col3 = 5,        row is assigned to Partition #1.
        If col3 = 50,       row is assigned to Partition #2.
        If col3 = 500,      row is assigned to Partition #3.
        If col3 = 5000,     Error 5728
        If col3 = NULL,     row is assigned to Partition #4.

        5728: Partitioning violation for table DBname.Tablename.
```

Partitioning with CASE_N – Example 4

Notes:

- **Partition the data based on total revenue for the products.**
- **The NO CASE and UNKNOWN options allow for total_revenue >=100,000 or "unknown revenue".**
- **A UPI is NOT allowed because the partitioning columns are NOT part of the PI.**

```
CREATE TABLE Sales_Revenue
   ( store_id          INTEGER NOT NULL,
     item_id           INTEGER NOT NULL,
     sales_date        DATE FORMAT 'YYYY-MM-DD',
     total_revenue     DECIMAL(9,2),
     total_sold        INTEGER,
     note              VARCHAR(256))
PRIMARY INDEX (store_id, item_id, sales_date)
PARTITION BY CASE_N
            ( total_revenue  <        2000 ,
              total_revenue  <        4000 ,
              total_revenue  <        6000 ,
              total_revenue  <        8000 ,
              total_revenue  <       10000 ,
              total_revenue  <       20000 ,
              total_revenue  <       50000 ,
              total_revenue  <      100000 ,
              NO CASE ,
              UNKNOWN );
```

SQL Use of PARTITION Key Word

The facing page contains an example of using the key word PARTITION to determine the number of rows there are in physical partitions. This example is based on the Sales_Revenue table is defined on the previous page.

The following table shows the same result as the facing page, but also identifies the internal partition #'s as allocated.

Part #	Row Count	
1	169690	internally mapped to partition #3
2	163810	internally mapped to partition #4
3	68440	internally mapped to partition #5
4	33490	internally mapped to partition #6
5	18640	internally mapped to partition #7
6	27520	internally mapped to partition #8
7	1760	internally mapped to partition #9

Note that this table does not have any rows with a total_revenue value greater than 50,000 and less than 100,000. Partition #8 was not assigned. Also, there are no rows with a total_revenue >=100,000 or NULL because the NO CASE and UNKNOWN partitions are not used.

Assume the following three SQL INSERT commands are executed:

```
INSERT INTO Sales_Revenue
    VALUES (1003, 5051, CURRENT_DATE, 51000, 45, NULL);
INSERT INTO Sales_Revenue
    VALUES (1003, 5052, CURRENT_DATE, 102000, 113, NULL);
INSERT INTO Sales_Revenue
    VALUES (1003, 5053, CURRENT_DATE, NULL, NULL, NULL);
```

The result of executing the SQL statement again would now be as follows:

Part #	Row Count	
1	169690	internally mapped to partition #3
2	163810	internally mapped to partition #4
3	68440	internally mapped to partition #5
4	33490	internally mapped to partition #6
5	18640	internally mapped to partition #7
6	27520	internally mapped to partition #8
7	1760	internally mapped to partition #9
8	1	internally mapped to partition # 10
9	1	internally mapped to partition # 2 (NO CASE)
10	1	internally mapped to partition # 1 (UNKNOWN)

SQL Use of PARTITION Key Word

The PARTITION SQL key word can be used to return partition numbers that have rows and a count of rows that are currently located in partitions of a table.

SQL:

```
SELECT    PARTITION   AS "Part #",
          COUNT(*)    AS "Row Count"
FROM      Sales_Revenue
GROUP BY  1
ORDER BY  1;
```

Result:

Part #	Row Count	
1	169690	total_revenue < 2,000
2	163810	total_revenue < 4,000
3	68440	total_revenue < 6,000
4	33490	total_revenue < 8,000
5	18640	total_revenue < 10,000
6	27520	total_revenue < 20,000
7	1760	total_revenue < 50,000

SQL - insert two rows:

```
INSERT INTO Sales_Revenue VALUES (1003, 5052, CURRENT_DATE, 102000, 113, NULL);
INSERT INTO Sales_Revenue VALUES (1003, 5053, CURRENT_DATE, NULL, NULL, NULL);
```

SQL (same as above):

```
SELECT    PARTITION   AS "Part #",
          COUNT(*)    AS "Row Count"
FROM      Sales_Revenue
GROUP BY  1
ORDER BY  1;
```

Result:

Part #	Row Count	
1	169690	total_revenue < 2,000
2	163810	total_revenue < 4,000
:	:	:
7	1760	total_revenue < 50,000
9	1	NO CASE
10	1	UNKNOWN

SQL Use of CASE_N

The facing page contains an example of using the CASE_N expression with SQL. You may wish to use this function to determine/forecast how rows will be mapped to various partitions in a table. The Sales_Revenue table was created as follows:

```
CREATE TABLE Sales_Revenue
   (  store_id              INTEGER NOT NULL,
      item_id               INTEGER NOT NULL,
      sales_date            DATE FORMAT 'YYYY-MM-DD',
      total_revenue         DECIMAL(9,2),
      total_sold            INTEGER,
      note                  VARCHAR(256))
PRIMARY INDEX (store_id, item_id, sales_date)
PARTITION BY CASE_N
      ( total_revenue  <   2000, total_revenue  <     4000,
        total_revenue  <   6000, total_revenue  <     8000,
        total_revenue  < 10000, total_revenue  <    20000,
        total_revenue  < 50000, total_revenue  <   100000,
        NO CASE, UNKNOWN);
```

The CASE_N expression in the query on the facing page is simply an SQL statement that shows how the rows would be partitioned.

SQL Use of RANGE_N

An example of using the RANGE_N expression with SQL is:

```
SELECT  RANGE_N ( Calendar_Date BETWEEN
           DATE '2004-11-28' AND DATE '2004-12-31' EACH INTERVAL '7' DAY,
           DATE '2005-01-01' AND DATE '2005-01-09' EACH INTERVAL '7' DAY)
                                    AS "Part #",
        MIN (Calendar_Date)         AS "Minimum Date",
        MAX (Calendar_Date)         AS "Maximum Date"
        FROM        Sys_Calendar.Calendar
        WHERE       Calendar_Date
        BETWEEN     DATE '2004-11-28' AND DATE '2005-01-09'
        GROUP BY    "Part #"
        ORDER BY    "Part #";
```

Output from this SQL is:

Part #	Minimum Date	Maximum Date
1	2004-11-28	2004-12-04
2	2004-12-05	2004-12-11
3	2004-12-12	2004-12-18
4	2004-12-19	2004-12-25
5	2004-12-26	2004-12-31
6	2005-01-01	2005-01-07
7	2005-01-08	2005-01-09

SQL Use of CASE_N

The CASE_N (and RANGE_N) expressions can be used with SQL to forecast the number of rows that will be placed into partitions.

This example uses a different partitioning scheme than the table actually has to determine how many rows would be placed into various partitions.

```
SELECT    CASE_N ( total_revenue  <    1500 ,
                   total_revenue  <    2000 ,
                   total_revenue  <    3000 ,
                   total_revenue  <    5000 ,
                   total_revenue  <    8000 ,
                   total_revenue  <   12000 ,
                   total_revenue  <   20000 ,
                   total_revenue  <   50000 ,
                   NO CASE,
                   UNKNOWN )      AS "Case #",
          count(*)                AS "Row Count"
FROM      Sales_Revenue
GROUP BY  1
ORDER BY  1;
```

Result:

Case #	Row Count
1	81540
2	88150
3	97640
4	103230
5	64870
6	31290
7	14870
8	1760

Notes:

- Currently, in this table, there are no rows with total_revenue >= 50,000 or NULL.
- The Case # would become the Partition # if the table was partitioned in this way.

Using ALTER TABLE with PPI Tables

The ALTER TABLE statement has been extended in support of PPI. For empty tables, the primary index and partitioning expression may be re-specified. For tables with rows, the partitioning expression may be modified only in ways that don't require existing rows to be re-evaluated.

The permitted changes for populated tables are to drop ranges at the ends or to add ranges at the ends. For example, a common use of this capability would be to drop ranges for the oldest dates, and to prepare additional ranges for future dates, among other things.

Limitations with ALTER TABLE:

- Primary Index of a non-empty table may not be altered
- Partitioning of a non-empty table is generally limited to altering the "ends".
- If a table has **Delete triggers**, they must be disabled if the WITH DELETE option is specified.
- If a save table has **Insert triggers**, they must be disabled if the WITH INSERT option is specified.

For empty tables with a PPI, the ALTER TABLE statement can be used to do the following:

- Remove partitioning for a partitioned table
- Establish partitions for a table (adds or replaces)
- Change the columns that comprise the primary index
- Change a unique primary index to non-unique.
- Change a non-unique primary index to unique.

For empty or non-empty tables, the ALTER TABLE statement can also be used to name an unnamed primary index or drop the name of a named primary index.

- To name an unnamed primary index or change the existing name of a primary index to something else, specify
  … MODIFY PRIMARY INDEX *index_name*;

- To drop the name of a named index, specify
  … MODIFY PRIMARY INDEX NOT NAMED;

Assume you have a populated data table (and the table is quite large) defined with a "non-unique partitioned primary index" and all of the partitioning columns are part of the PI. You realize that the table should have been defined with a "unique partitioned primary index", but the table is already loaded with data. Here is a technique to convert this NUPI into a UPI without copying or reloading the data.

- CREATE a USI on the columns making up the PI. ALTER the table, effectively changing the NUPI to a UPI, and the software will automatically drop the USI.

Using ALTER TABLE with PPI Tables

The ALTER TABLE statement has enhancements for a partitioned table to modify the partitioning properties of the primary index for a table.

For populated tables, ...

- **You are permitted to drop and/or add ranges at the "ends" of existing partitions on a range-partitioned table.**
  - **ALTER TABLE includes ADD / DROP RANGE options.**
  - **You can also add or drop special partitions (NO RANGE or UNKNOWN).**
  - **You cannot drop all the ranges.**
- **Possible use – drop ranges for the oldest dates and prepare additional ranges for future dates.**
- **The set of primary index columns cannot be altered for a populated table.**

Teradata 13.10 Feature

- **ALTER TABLE has a new option to resolve partitioned table definitions with DATE, CURRENT_DATE, and CURRENT_TIMESTAMP to their current values.**
  - **This feature only applies to partitioned tables and join indexes.**

To use ALTER TABLE for any purpose other than the above situations, the table must be empty.

ALTER TABLE – Example 5

The DROP RANGE option is used to drop a range set from the RANGE_N function on which the partitioning expression for the table is based. You can only drop ranges if the partitioning expression for the table is derived only from a RANGE_N function. You can drop empty partitions without specifying the WITH DELETE or WITH INSERT option. Some of the ALTER TABLE statement options include:

DROP RANGE WHERE *conditional_expression* – a conditional partitioning expression used to drop a range set from the RANGE_N function on which the partitioning expression for the table is based.

You can only drop ranges if the partitioning expression for the table is derived only from a RANGE_N function.

You must base *conditional_partitioning_expression* on the system-derived PARTITION column.

DROP RANGE BETWEEN … [NO RANGE [OR UNKNOWN]] – used to drop a set of ranges from the RANGE_N function on which the partitioning expression for the table is based.

You can also drop NO RANGE OR UNKNOWN and UNKNOWN specifications from the definition for the RANGE_N function.

You can only drop ranges if the partitioning expression for the table is derived exclusively from a RANGE_N function.

Ranges must be specified in ascending order.

ADD RANGE BETWEEN … [NO RANGE [OR UNKNOWN]] – used to add a set of ranges to the RANGE_N function on which the partitioning expression for the table is based.

You can also add NO RANGE OR UNKNOWN and UNKNOWN specifications to the definition for the RANGE_N function.

You can only add ranges if the partitioning expression for the table is derived exclusively from a RANGE_N function.

DROP Does NOT Mean DELETE

If a table does not have the NO RANGE partition, then partitions are dropped from the table without using the Transient Journal and the rows are either deleted or are copied (WITH INSERT) into a user-specified table.

If a table has a NO RANGE partition, rows are copied from dropped partition into the NO RANGE partition.

ALTER TABLE – Example 5

To drop/add partitions and NOT COPY the old data to another table:

```
ALTER TABLE Sales MODIFY PRIMARY INDEX
   DROP RANGE  BETWEEN DATE '2003-01-01' AND DATE '2003-12-31'  EACH INTERVAL '1' MONTH
   ADD RANGE   BETWEEN DATE '2013-01-01' AND DATE '2013-12-31'  EACH INTERVAL '1' MONTH
   WITH DELETE;
```

To drop/add partitions and COPY the old data to another table:

```
ALTER TABLE Sales MODIFY PRIMARY INDEX
   DROP RANGE  BETWEEN DATE '2003-01-01' AND DATE '2003-12-31'  EACH INTERVAL '1' MONTH
   ADD RANGE   BETWEEN DATE '2013-01-01' AND DATE '2013-12-31'  EACH INTERVAL '1' MONTH
   WITH INSERT INTO SalesHistory;
```

Notes:

- Ranges are dropped and/or added to the "ends".
- DROP does NOT necessarily mean DELETE!
  - If a table has a NO RANGE partition, rows are moved from the dropped partitions into the NO RANGE partition. This can be time consuming.
- The SalesHistory table must exist before using the WITH INSERT option.
- The Sales table was partitioned as follows:

```
PARTITION BY RANGE_N (sales_date  BETWEEN
        DATE '2003-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' MONTH );
```

ALTER TABLE – Example 5 (cont.)

This page contains notes on the internal implementation. The important point is to understand that dropping or adding partitions (to the "ends" of an already partitioned table with data) does not cause changes to the internal partitioning numbers that are currently implemented. The logical partition numbers change, but the internal partition numbers do not. For this reason, dropping or adding partitions does not cause an undue amount of work.

The following table shows the same result as the facing page, but also identifies the internal partition #'s as allocated.

PARTITION	Count(*)	
1	10850	internally mapped to partition #3
2	10150	internally mapped to partition #4
:	:	:
13	12400	internally mapped to partition #15
14	11200	internally mapped to partition #16
:	:	:
119	14800	internally mapped to partition #121
120	14950	internally mapped to partition #122

In the example on the facing page, 12 partitions were dropped for the year 2003 and 12 partitions were added for the year 2013. The partitions for 2013 don't appear because they are empty.

The following table shows the same result as the facing page, but also identifies the internal partition #'s as allocated after the partitions for the year 2003 were dropped.

PARTITION		
1	12400	internally mapped to partition #15
2	11200	internally mapped to partition #16
:	:	:
107	14800	internally mapped to partition #121
108	14950	internally mapped to partition #122

You can add the NO RANGE and/or UNKNOWN partitions to an already partitioned table.

```
ALTER TABLE Sales MODIFY PRIMARY INDEX
      ADD RANGE NO RANGE OR UNKNOWN;
```

If this table had NO RANGE partition defined and the 12 partitions were dropped (as in this example), the data rows from the dropped partitions are moved to the NO RANGE partition. To remove the special partitions and delete the data, use the following command:

```
ALTER TABLE Sales MODIFY PRIMARY INDEX
      DROP RANGE NO RANGE OR UNKNOWN
      WITH DELETE;
```

ALTER TABLE – Example 5 (cont.)

Partitions may only be dropped or added from/to the "ends" of a populated table.

SQL:

```
SELECT    PARTITION,
          COUNT(*)
FROM      Sales
GROUP BY  1
ORDER BY  1;
```

Result:	PARTITION	COUNT(*)	
	1	10850	Part #1 - January 2003
	2	10150	
	:	:	
	119	14800	
	120	14950	Part #120 - December 2012

```
ALTER TABLE Sales MODIFY PRIMARY INDEX
    DROP RANGE   BETWEEN
                 DATE '2003-01-01' AND DATE '2003-12-31'  EACH INTERVAL '1' MONTH
    ADD RANGE    BETWEEN
                 DATE '2013-01-01' AND DATE '2013-12-31'  EACH INTERVAL '1' MONTH
    WITH DELETE;
```

SQL:

```
SELECT    PARTITION,
          COUNT(*)
FROM      Sales
GROUP BY  1
ORDER BY  1;
```

Result:	PARTITION	COUNT(*)	
	1	12400	Part #1 - January 2004
	2	11200	
	:	:	
	107	14800	
	108	14950	Part #108 - December 2012

ALTER TABLE TO CURRENT

Staring with Teradata 13.10, you can now specify CURRENT_DATE and CURRENT_TIMESTAMP functions in a partitioned primary index for base tables and join indexes.

Also starting with Teradata 13.10, Teradata provides a new option with the ALTER TABLE statement to modify a partitioned table that has been defined with a moving CURRENT_DATE (or DATE) or moving CURRENT_TIMESTAMP. This new option is called ALTER TABLE TO CURRENT.

When you specify CURRENT_DATE and CURRENT_TIMESTAMP as part of a partitioning expression for a partitioned table, these functions resolve to the date and timestamp when you define the PPI. To partition on a new CURRENT_DATE or CURRENT_TIMESTAMP value, submit an ALTER TABLE TO CURRENT request.

The ALTER TABLE TO CURRENT syntax is shown on the facing page.

The WITH DELETE option is used to delete any row whose partition number evaluates to a value outside the valid range of partitions.

The WITH INSERT [INTO] *save_table* option is used to insert any row whose partition number evaluates to a value outside the valid range of partitions into the table specified by *save_table*.

The WITH DELETE or INSERT INTO *save_table* clause is sometimes referred to as a null partition handler. You cannot specify a null partition handler for a join index.

Save_table and the table being altered must be different tables with different names.

ALTER TABLE TO CURRENT

This Teradata 13.10 option allows you to periodically resolve the CURRENT_DATE (or DATE) and CURRENT_TIMESTAMP of a partitioned table to their current values.

Benefits include:

- **You do not have to change the partitioning expression to update the value for CURRENT_DATE or CURRENT_TIMESTAMP.**
- **To partition on a new CURRENT_DATE or CURRENT_TIMESTAMP value, simply submit an ALTER TABLE TO CURRENT request.**

Considerations:

- **The ALTER TABLE TO CURRENT request causes the CURRENT_DATE and/or CURRENT_TIMESTAMP to effectively repartition the rows in the table.**
- **If RANGE_N specifies CURRENT_DATE or CURRENT_TIMESTAMP in a partitioning expression, you cannot use ALTER TABLE to add or drop ranges for the table. You must use the ALTER TABLE TO CURRENT statement to achieve this function.**

ALTER TABLE TO CURRENT – Example 6

The ALTER TABLE TO CURRENT option allows you to periodically modify the partitioning. This option resolves the CURRENT_DATE (or DATE) and CURRENT_TIMESTAMP to their current values.

The example on the facing page assumes partitioning begins on a year boundary. Using this example, consideration for the two options are:

- With hard-coded dates in the CREATE TABLE statement, you must compute the new dates and specify them explicitly in the ADD RANGE clause of the request. This requires manual intervention every year you submit the request.

- With CURRENT_DATE in the CREATE TABLE statement, you can schedule the ALTER TABLE TO CURRENT request to be submitted annually or simply execute the next year. This request rolls the partition window forward by efficiently dropping and adding partitions.

  As a result of executing the ALTER TABLE TO CURRENT WITH DELETE, Teradata deletes the rows from the table because they are no longer needed.

Considerations:

- You should evaluate how a DATE, CURRENT_DATE, or CURRENT_TIMESTAMP function will require reconciliation in a partitioning expression before you define such expressions on a table or join index.

- If you specify multiple ranges using a DATE or CURRENT_DATE function in one of the ranges, and then later reconcile the partitioning the range specified using CURRENT_DATE might overlap one of the existing ranges. If so, reconciliation aborts the request and returns an error to the requestor. If this happens, you must recreate the table with a new partitioning expression based on DATE or CURRENT_DATE. Because of this, you should design a partitioning expression that uses a DATE or CURRENT_DATE function in one of its ranges with care.

DATE, CURRENT_DATE, and CURRENT_TIMESTAMP functions in a partitioning expression are most appropriate when the data must be partitioned as one or more Current partitions and one or more History partitions, where the terms Current and History are defined with respect to the resolved DATE, CURRENT_DATE, or CURRENT_TIMESTAMP values in the partitioning expression.

This enables you to reconcile a table or join index periodically to move older data from the current partition into one or more history partitions using an ALTER TABLE TO CURRENT request instead of redefining the partitioning using explicit dates that must be determined each time you alter a table using ALTER TABLE requests to ADD or DROP ranges.

ALTER TABLE TO CURRENT – Example 6

This example creates a partitioning expression to maintain the last 8 years of historical data, data for the current year, and data for one future year for a total of 10 years.

```
CREATE TABLE Sales
    ( store_id                INTEGER NOT NULL,
      item_id                 INTEGER NOT NULL,
      sales_date              DATE FORMAT 'YYYY-MM-DD',
       :
PRIMARY INDEX  (store_id, item_id)
PARTITION BY RANGE_N
(sales_date  BETWEEN DATE '2004-01-01' AND DATE '2013-12-31' EACH INTERVAL '1' MONTH);
```

Assuming the current year is 2012, an equivalent definition using CURRENT_DATE is:

```
PRIMARY INDEX  (store_id, item_id)
PARTITION BY RANGE_N
(sales_date BETWEEN
      CAST(((EXTRACT(YEAR FROM CURRENT_DATE) - 8 - 1900) * 10000 + 0101) AS DATE) AND
      CAST(((EXTRACT(YEAR FROM CURRENT_DATE) +1 - 1900) * 10000 + 1231) AS DATE)
      EACH INTERVAL '1' MONTH);
```

In 2013, execute ALTER TABLE Sales TO CURRENT WITH DELETE;

- Teradata deletes the rows from 2004 because they are no longer needed.
- To view the date when the table was last resolved, then DBC.IndexConstraintsV provides new columns named "ResolvedCurrent_Date" and "ResolvedCurrent_TimeStamp".

PPI Enhancements

The facing page identifies various enhancements with different Teradata releases.

PPI Enhancements

Teradata V2R6.0

- Selected Partition Archive, Restore, and Copy
- Dynamic partition elimination for merge join
- Single-AMP NUSI access when NUSI on same columns as NUPI;
- Partition elimination on RowIDs referenced by NUSI

Teradata V2R6.1

- PPI for global temporary tables and volatile tables
- Collect statistics on system-derived column PARTITION

Teradata V2R6.2

- PPI for non-compressed join indexes

Teradata 12.0

- Multi-level partitioning

Teradata 13.10

- Tables and non-compressed join indexes can now include partitioning on a character column.
- PPI tables allow a test value (e.g., RANGE_N) to have a TIMESTAMP(n) data type.
- ALTER TABLE *tablename* TO CURRENT …;

Teradata 14.0

- Increased partition limit to 9.223 quintillion
- New data types for RANGE_N – BIGINT and TIMESTAMP
- ADD option for a partitioning level

Multi-level PPI Concepts

The facing page identifies the basic concepts of using a multi-level PPI.

Multi-level partitioning allows each partition at a given level to be further partitioned into sub-partitions. Each partition for a level is sub-partitioned the same per a partitioning expression defined for the next lower level. The system hash orders the rows within the lowest partition levels. A multilevel PPI (MLPPI) undertakes efficient searches by using partition elimination at the various levels or combinations of levels.

Notes associated with multilevel partitioning:

- Note that the number of levels of partitioning cannot exceed 15. Each level must define at least two partitions. The number of levels of partitioning may be further restricted by other limits such as the maximum size of the table header, data dictionary entry sizes, etc.

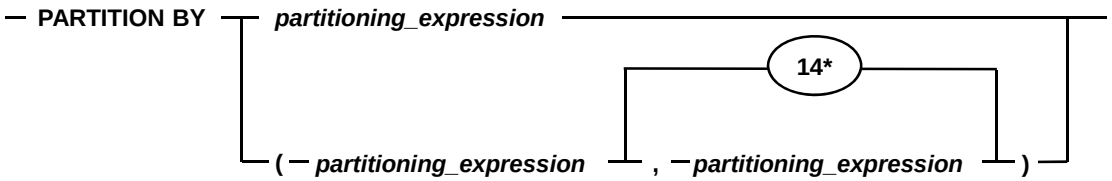
- The number of partitions in a table cannot exceed 65,535 partitions. The number of partitions in an MLPPI is determined by multiplying the number of partitions at the different levels (d1 * d2 * d3 * …).

- The specification order of partitioning expressions can be important for multi-level partitioning. The system maps multi-level partitioning expressions into a single-level combined partitioning expression. It then maps the resulting combined partition number 1-to-1 to an internal partition number.

- A usage implication - you can alter only the highest partition level, which by definition is always level 1, to change the number of partitions at that level when the table is populated with rows.

Multi-level PPI Concepts

- **Allows multiple partitioning expressions instead of only one for a table or a non-compressed join index.**
- **Multilevel partitioning allows each partition at a level to be sub-partitioned.**
  - **Each partitioning level is defined independently using a RANGE_N or CASE_N expression.**
- **A multi-level PPI allows efficient searches by using partition elimination at the various levels or combination of levels.**
- **Allows more flexibility in which partitioning expression to use when there are multiple choices for the partitioning expressions.**
- **Teradata 14 allows for a maximum of 9.223 quintillion partitions and 62 levels.**
- **Syntax:**



14* – Teradata 13.10 limit.

Multi-level PPI Concepts (cont.)

The facing page contains an example showing the benefit of using a multi-level PPI.

You can use a multilevel PPI to improve query performance via partition elimination, either at each of the partition levels or by combining all of them. An MLPPI provides multiple access paths to the rows in the base table. As with other indexes, the Optimizer determines if the index is usable for a query and, if usable, whether its use provides the estimated least costly plan for executing the query.

The following list describes the various access methods that are available when a multilevel PPI is defined for a table:

- If there is an equality constraint on the primary index and there are constraints on the partitioning columns such that access is limited to a single partition at each level, access is as efficient as with an NPPI.
- This is a single-AMP, single-hash access in a single sub-partition at the lowest level of the partition hierarchy.
- With constraints defined on the partitioning columns, performance of a primary index access can approach the performance of an NPPI depending on the extent of partition elimination that can be achieved.
- This is a single-AMP, single-hash access in multiple (but not all) sub-partitions at the lowest level of the partition hierarchy.
- Access by means of equality constraints on the primary index columns that does not also include all the partitioning columns, and without constraints defined on the partitioning columns, might not be as efficient as access with an NPPI. The efficiency of the access depends on the number of non-empty sub-partitions at the lowest level of the partition hierarchy.
- This is a single-AMP, single-hash access in all sub-partitions at the lowest level of the partition hierarchy.
- With constraints on the partitioning columns of a partitioning expression such that access is limited to a subset of, say *n* percent, of the partitions for that level, the scan of the data is reduced to about *n* percent of the time required by a full-table scan.
- This is an all-AMP scan of only the non-eliminated partitions for that level. This allows multiple access paths to a subset of the data: one for each partitioning expression. If constraints are defined on partitioning columns for more than one of the partitioning expressions in the MLPPI definition, partition elimination can lead to even less of the data needing to be scanned.

Multi-level PPI Concepts (cont.)

Query – Compare District 25 revenue for Week 6 vs. same period last year?

Multi-level Partitioning – Example 7

You create an MLPPI by specifying two or more partitioning expressions, where each expression must be defined using either a RANGE_N function or a CASE_N function exclusively. The system combines the individual partitioning expressions internally into a single partitioning expression that defines how the data is partitioned on an AMP.

The first partitioning expression is the highest level partitioning. Within each of those partitions, the second partitioning expression defines how each of the highest-level partitions is sub-partitioned. Within each of those second-level partitions, the third-level partitioning expression defines how each of the second level partitions is sub-partitioned. Within each of these lowest level partitions, rows are ordered by the row hash value of their primary index and their assigned uniqueness value.

You define the ordering of the partitioning expressions in your CREATE TABLE SQL text, and that ordering implies the logically ordering by RowID. Because the partitions at each level are distributed among the partitions of the next higher level in the hierarchy, scanning a partition at a certain level requires skipping some internal partitions.

Partition expression order does *not* affect the ability to eliminate partitions, but *does* affect the efficiency of a partition scan. As a general rule, this should not be a concern if there are many rows, which implies multiple data blocks, in each of the partitions.

The facing page contains an example of creating a multi-level PPI.

There are two levels of partitioning defined in this example. The first level defines 120 partitions and the second defines 75 partitions. Therefore, the total number of partitions for the combined partitioning expression is the product of 120 * 75, or 9000.

Multi-level Partitioning – Example 7

For example, partition Claim table by "Claim Date" and "State ID".

```
CREATE TABLE Claim
      ( claim_id          INTEGER  NOT NULL
       ,cust_id           INTEGER  NOT NULL
       ,claim_date        DATE     NOT NULL
       ,state_id          BYTEINT  NOT NULL
       , ... )
PRIMARY INDEX (claim_id)
      PARTITION BY (
      /* First level of partitioning */
            RANGE_N (claim_date BETWEEN
            DATE '2003-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' MONTH ),
      /* Second level of partitioning */
            RANGE_N (state_id        BETWEEN 1 AND 75 EACH 1) )
UNIQUE INDEX (claim_id);
```

Notes:

- **For multi-level PPI, the set of partitioning expressions must be enclosed in parentheses.**
- **Each level must define at least two partitions for a multi-level PPI.**
- **The number of defined partitions in this example is (120 * 75) or 9000.**

Multi-level Partitioning – Example 7 (cont.)

The facing page continues the example of using a multi-level PPI. This example assumes that the query has conditions where only claims for a specific month and for a specific state need to be returned. Teradata only needs to scan the data blocks associated with the specified criteria.

Multi-level PPI Example 7 (cont.)

Assume

- **Eliminating all but one month out of many years of claims history would facilitate scanning less than 2% of the claims history.**
- **Similarly, eliminating all but the California claims out of the many states would facilitate scanning less than 4% of the claims history.**

Then, combining both of these predicates for partition elimination would facilitate scanning less than 0.08% of the claims history for satisfying the following query.

```
SELECT    …
FROM      Claim C, States S
WHERE     C.state_id = S.state_id
AND       S.state_name = 'California'
AND       C.claim_date BETWEEN DATE '2012-01-01' AND DATE '2012-01-31';
```

How is the MLPPI Partition # Calculated?

The facing page shows the calculation that is used to determine the partition number for a MLPPI table.

How is the MLPPI Partition # Calculated?

Multilevel partitioning is rewritten internally to single-level partitioning to generate a combined partition number as follows:

$$(p_1 - 1) * dd_1 + (p_2 - 1) * dd_2 + ... + (p_{n-1} - 1) * dd_{n-1} + p_n$$

where n is the number of partitioning expressions

p_i is the value of the partitioning expression for level i

d_i is the number of partitions for level i

dd_i is the product of d_{i+1} through d_n

$dd = d_1 * d_2 * ... * d_n <= 65535$

dd is the total number of combined partitions

Example:

Assume January, 2012 is the 109th first level partition and California is the 6th state code for the second level partition. Also assume that the first level has 120 partitions and the second level has 75 partitions.

$$(109 - 1) * 75 + 6 = 8106$$

is the logical partition number for claims in California for January of 2012.

Character PPI

This Teradata 13.10 feature extends the capabilities and options when defining a PPI for a table or a non-compressed join index. Tables and non-compressed join indexes can now include partitioning on a character column. This feature provides the improved query performance benefits of partitioning when queries have conditions on columns with character (alphanumeric) data.

- Before Teradata 13.10, customers were limited to creating partitioning on tables that did **not** involve comparison of character data. Partitioning expressions were limited to numeric or date type data.

The Partitioned Primary Index (PPI) feature of Teradata has always allowed a class of queries to access a portion of a large table, instead of the entire table. This capability has simply been extended to include character data. The traditional uses of the Primary Index (PI) for data placement and rapid access of the data when the PI values are specified are still retained.

When creating a table or a join index, the **PARTITION BY** clause (part of PRIMARY INDEX) can now include partitioning on a character column. This allows the comparison of character data.

This feature allows a partitioning expression to involve comparison of character data (CHAR, VARCHAR, GRAPHIC, VARGRAPHIC) types. A comparison may involve a predicate (=, >, <, >=, <=, <>, BETWEEN, LIKE) or a string function.

- The use of a character expression is a PPI table is referred to as CPPI (Character PPI).

The most common partitioning expressions utilize RANGE_N or CASE_N expressions.

Prior to Teradata 13.10, both the CASE_N and RANGE_N functions did not allow the PPI definition of character data. This limited the useful partitioning that could be done using character columns as a standard ordering (collation) of the character data is not preserved.

Both the RANGE_N and CASE_N functions support the definition of character data in Teradata 13.10. The term "character or char" will be used to represent CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC data types.

The test value of a RANGE_N function should be a simple column reference, involving no other functions or expressions. For example, if SUBSTR is added, then static partition elimination will not occur. Keep the partitioning expressions as simple as possible.

RANGE_N (**SUBSTR (state_code, 1, 1)** BETWEEN 'AK' and 'CA', …

This definition will not allow static partition elimination.

Character PPI

Tables and non-compressed join indexes can now include partitioning on a character column. This feature is referred to as CPPI (Character PPI).

- **Prior to Teradata 13.10, partitioning expressions (RANGE_N and CASE_N) are limited to numeric or date type data.**

This feature allows a partitioning expression to involve comparison of character data (CHAR, VARCHAR, GRAPHIC, VARGRAPHIC) types. A comparison may involve a predicate (=, >, <, >=, <=, <>, BETWEEN, LIKE) or a string function.

Collation and case sensitivity considerations:

- **The session collation in effect when the character PPI is created determines the ordering of data used to evaluate the partitioning expression.**
- **The ascending order of ranges in a character PPI RANGE_N expression is defined by the session collation in effect when the PPI is created or altered, as well as the case sensitivity of the column or expression in the test value.**
- **The default case sensitivity of character data for the session transaction semantics in effect when the PPI is created will also determine case sensitivity of comparison unless overridden with an explicit CAST to a specific case sensitivity.**

Character PPI – Example 8

In this example, the Claim table is first partitoned by claim_date (monthly intervals). Claim_date is then sub-partitioned by state codes. State codes are then sub-partitioned by the first two letters of a city name. The special partitions of NO RANGE and UNKNOWN are defined at the claim_date, state_code, and city levels.

Why is the facing page partitioning example defined with intervals of 1 month for claim_date?

- Teradata 13.10 has a maximum limit of 65,535 partitions in a table and defining 8 years of day partitioning with two levels of sub-partitioning cause this limit to be exceeded.

The following queries will benefit from this type of partitioning.

```
SELECT * FROM Claim_MLPPI2
          WHERE  state_code = 'GA' AND city LIKE 'a%';

SELECT * FROM Claim_MLPPI2
          WHERE  claim_date = '2012-08-24' AND city LIKE 'a%';
```

The session mode when these tables were created and when these queries were executed was Teradata mode (BTET). Teradata mode defaults to "not case specific". The session collation in effect when the character PPI is created determines the ordering of data used to evaluate the partitioning expression.

The ascending order of ranges in a character PPI RANGE_N expression is defined by the session collation in effect when the PPI is created or altered, as well as the case sensitivity of the column or expression in the test value. The default case sensitivity of character data for the session transaction semantics in effect when the PPI is created will also determine case sensitivity of comparison unless overridden with an explicit CAST to a specific case sensitivity.

The default case sensitivity in effect when the character PPI is created will also affect the ordering of character data for the PPI.

- Default case sensitivity of comparisons involving character constants is influenced by the **session mode**. String literals have a different default CASESPECIFIC attribute depending on the session mode.
  - Teradata Mode (BTET) is NOT CASESPECIFIC
  - ANSI mode is CASESPECIFIC
- If any expression in the comparison is case specific, then the comparison is case sensitive.

Character PPI – Example 8

In this example, 3 levels of partitioning are defined.

```
CREATE TABLE Claim_MLPPI2
    (claim_id          INTEGER          NOT NULL,
     cust_id           INTEGER          NOT NULL,
     claim_date        DATE             NOT NULL,
     city              VARCHAR(30)      NOT NULL,
     state_code        CHAR(2)          NOT NULL,
     claim_info        VARCHAR (256))
PRIMARY INDEX (claim_id)
  PARTITION BY
  ( RANGE_N
       (claim_date     BETWEEN          DATE '2005-01-01' and DATE '2012-12-31'
                                        EACH INTERVAL '1' MONTH, NO RANGE),
    RANGE_N
       (state_code     BETWEEN          'A', 'D', 'I', 'N', 'T' AND 'ZZ', NO RANGE),
    RANGE_N
       (city           BETWEEN          'A', 'C', 'E', 'G', 'I', 'K', 'M', 'O',
                                        'Q', 'S', 'U', 'W' AND 'ZZ', NO RANGE) )
UNIQUE INDEX (claim_id);
```

The following queries will benefit from this type of partitioning.

- **SELECT * FROM Claim_MLPPI2 WHERE state_code = 'OH';**
- **SELECT * FROM Claim_MLPPI2 WHERE state_code = 'GA' AND city LIKE 'a%';**
- **SELECT * FROM Claim_MLPPI2 WHERE claim_date = DATE '2012-08-24' AND city LIKE 'a%';**

Summary

The customer (e.g., DBA, Database Designer, etc.) has a flexible and powerful tool to structure tables to allow automatic optimization of frequently used queries. This tool is the Partitioned Primary Index (PPI) feature. This feature allows tables to be partitioned on columns of interest, while retaining the traditional use of the primary index (PI) for data distribution and efficient access when the PI values are specified in the query.

The facing page contains a summary of the key customer benefits that can be obtained by using Partitioned Primary Indexes.

Whether and how to partition a table is a physical design choice.

A well-chosen partitioning scheme may be able to turn many frequently run queries into partial-table scans instead of full-table scans, with much improved performance.

However, understand that there are trade-off considerations that must be understood and carefully considered to get the most benefit from the PPI feature.

Summary

- **Improves the performance of queries that use range constraints on the range partitioning column(s) by allowing for range/partition elimination.**
  - **Allows primary index access and range access without a secondary index.**
- **General Recommendations**
  - **Collect statistics on system-derived column PARTITION.**
  - **Do not define or name a column PARTITION in a PPI table – you won't be able to reference the system-derived column PARTITION for the table.**
  - **If possible, avoid use of NO RANGE, NO RANGE OR UNKNOWN, or UNKNOWN options with RANGE_N partitioning for DATE columns.**
  - **Consider only having as many date ranges as needed currently plus some for the future – helps the optimizer cost plans better, especially when partitioning column is not included in the Primary Index.**
- **Note (as with all partitioning/indexing schemes) there are tradeoffs due to performance impacts on table access, joins, maintenance, and other operations.**

Module 17: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 17: Review Questions

1. In a PPI table, every row is uniquely identified by its Partition # + Row Hash + Uniqueness Value.
2. The Row Key consists of the Partition # + Row Hash.
3. In an NPPI table, the partition number defaults to 0.
4. True or False. For a PPI table, the partition number and the Row Hash are both used by the Message Passing Layer to determine which AMP(s) should receive the request.

   FALSE (it only cares about the first part of the row hash)
5. Which two options apply to the RANGE_N expression in a partitioning expression? ____ ____
   a. Ranges can be listed in descending order
   b. Allows use of NO RANGE OR UNKNOWN option
   c. Partitioning column must be part of the Primary Index
   d. Has a maximum of 65,535 partitions with Teradata Release 13.10
6. With a populated table, select 2 actions that are allowed with the ALTER TABLE command. ____ ____
   a. Drop all of the ranges
   b. Add or drop ranges from the partition “ends”
   c. Change the columns that comprise the primary index
   d. Add or drop special partitions (NO RANGE, UNKNOWN)
7. Which 2 choices are advantages of partitioning a table? ____ ____
   a. Fast delete of rows in partitions
   b. Fewer AMPs are involved when accessing data
   c. Faster access (than an NPPI table) if the table has a UPI
   d. Range queries can be executed without a secondary index

Module 17: Review Questions (cont.)

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 17: Review Questions (cont.)

Given this CREATE TABLE statement, answer the following questions.

```
CREATE TABLE     Orders
   (Order_id          INTEGER NOT NULL,
    Cust_id           INTEGER NOT NULL,
    Order_status      CHAR(1),
    Total_price       DECIMAL(9,2) NOT NULL,
    Order_date        DATE FORMAT 'YYYY-MM-DD' NOT NULL,
    Order_priority    SMALLINT,
    Clerk             CHAR(16),
    Ship_priority     SMALLINT,
    Order_Comment     VARCHAR(80) )
PRIMARY INDEX    (Order_id)
PARTITION BY RANGE_N (Order_date
      BETWEEN DATE '2003-01-01' AND DATE '2012-12-31'
      EACH INTERVAL '1' MONTH)
UNIQUE INDEX     (Order_id);
```

8. What is the name of partitioning column? Order_date
9. What is the time period for each partition? 1 Month
10. Why is there a Unique Secondary Index specified instead of defining Order_id as a UPI? _____
    a. This is a coding style choice.
    b. You cannot have a UPI when using a partitioned primary index.
    c. You cannot have a UPI if the partitioning column is not part of the primary index.
    d. This is a mistake. You cannot have a secondary and a primary index on the same column(s).

Lab Exercise 17-1

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

SQL hints:

INSERT INTO *table_1* SELECT * FROM *table_2*;

SELECT COUNT(*) FROM *table_name*;

SHOW TABLE *table_name*;

A count of rows in the Orders table is 31,200.
A count of rows in the Orders_2012 table is 12,000.

Lab Exercise 17-1

Lab Exercise 17-1

Purpose

In this lab, use Teradata SQL Assistant to create tables with primary indexes partitioned in various ways.

What you need

Populated DS tables and empty tables in your database

Tasks

1. Using INSERT/SELECT, populate your Orders and Orders_2012 tables from the DS.Orders and DS.Orders_2012 tables, respectively. Your Orders table will have data for the years 2003 to 2011 and the Orders_2012 table will have data for 2012. Verify the number of rows in your tables.

   SELECT COUNT(*) FROM Orders; Count = _____ (should be 31,200)
   SELECT COUNT(*) FROM Orders_2012; Count = _____ (should be 12,000)

2. Use the SHOW TABLE for Orders to help create a new, similar table (same column names and definitions, etc.) named "Orders_PPI" that has a PPI based on the following:

   Primary Index – orderid
   Partitioning column – orderdate
   - From '2003-01-01' through '2012-12-31', partition by month
   - Include the NO RANGE option (the UNKNOWN option is not needed for orderdate)
   - Do not create any secondary indexes for this table

   How many partitions are logically defined for the Orders_PPI table? _____

Lab Exercise 17-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

SQL hints:

```
INSERT INTO table_1 SELECT * FROM table_2;

SELECT      COUNT(*)
FROM        table_name;

SELECT      PARTITION, COUNT(*)
FROM        table_name
GROUP BY    1
ORDER BY    1;

SELECT      PARTITION, COUNT(*)
FROM        table_name
WHERE       orderdate BETWEEN '2012-01-01' AND '2012-12-31'
GROUP BY    1
ORDER BY    1;

SELECT COUNT(DISTINCT(PARTITION)) FROM table_name;
```

Lab Exercise 17-1 (cont.)

3. **INSERT/SELECT all of the rows from your Orders table into the Orders_PPI table. Verify the number of rows in your table. Count = _____**

   How many partitions would you estimate have data at this time? _____

4. **Use the PARTITION key word to list the partitions and number of rows in various partitions.**

   How many partitions actually have data? _____

   How many rows are in each partition for the year 2003? _____

   How many rows are in each partition for the year 2011? _____

5. **Use INSERT/SELECT to add the rows from the Orders_2012 table to your Orders_PPI table. Verify the number of rows in your table. Count = _____**

   Use the PARTITION key word to determine the number of partitions used and the number of rows in various partitions.

   How many partitions actually have data? _____

   How many rows are in each partition for the year 2012? _____

Lab Exercise 17-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

SQL hints:

```
INSERT INTO table_1  SELECT * FROM table_2;

SELECT      COUNT(*) FROM table_name;

SELECT      COUNT(DISTINCT(PARTITION))
FROM        table_name
WHERE       orderdate … ;

SELECT      MAX (PARTITION)
FROM        table_name;

SELECT      PARTITION, COUNT(*)
FROM        table_name
GROUP BY    1
ORDER BY    1;

SELECT COUNT(DISTINCT(PARTITION)) FROM table_name;
```

The PARTITION key word only returns partition numbers of partitions that contain rows. The following “canned” SQL can be used to return a list of partitions that are not used between the first and last used partitions.

```
SELECT p + 1                AS "The missing partitions are:"
FROM
   (SELECT   p1 - p2        AS p,
             PARTITION      AS p1,
             MDIFF(PARTITION, 1, PARTITION) AS p2
    FROM     table_name
    QUALIFY  p2 > 1)
    AS temp;
```

Lab Exercise 17-1 (cont.)

6. **INSERT the following row (using these values) into the Orders_PPI table.**

   (100000, 1000, 'C', 1000, '2000-12-31', 10, 'your name', 5, 20, 'old order')

   How many partitions are now in Orders_PPI? ____

   What is the partition number (highest partition #) of the NO RANGE OR UNKNOWN partition? ____

7. **(Optional) Create a new table named "Orders_PPI_ML" that has a Multi-level PPI based on the following:**

   Primary Index – orderid
   First Level of Partitioning column – orderdate (use month ranges for all 10 years)
   Include the NO RANGE option for orderdate
   Second Level of Partitioning column – location (10 different order locations (1 through 10)
   Place the NO RANGE and UNKNOWN rows into the same special partition for location
   Unique secondary index – orderid

8. **(Optional) Populate the Orders_PPI_ML table from the Orders and Orders_2012 tables using INSERT/SELECT. Verify the number of rows in Orders_PPI_ML. Count = _____**

Lab Exercise 17-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

SQL hints:

INSERT INTO *table_1* VALUES (*value1, value2, …*);

INSERT INTO *table_1* SELECT * FROM *table_2*;

SELECT COUNT(*) FROM *table_name*;

SELECT COUNT(DISTINCT(PARTITION)) FROM *table_name*;

Lab Exercise 17-1 (cont.)

9. (Optional) For the Orders_PPI_ML table, use the PARTITION key word to answer the following questions.

   How many partitions actually have data? _____

   What is the highest partition number? _____

   What is the partition number for orders in January, 2012 and location 1? _____

   What is the partition number for orders in February, 2012 and location 1? _____

   Is there a difference of 11 partitions between these two months? _____

   Why or why not? _____

10. (Optional) Before altering the table, verify the number of rows in Orders_PPI. Count = _____

    Use the ALTER TABLE command on Orders_PPI to do the following.

    - DROP RANGE (with DELETE) for year 2003
    - ADD RANGE for orders that will be placed in year 2013 with an interval of 1 month

    Use SHOW TABLE on Orders_PPI to view the altered partitioning.

    Use the PARTITION key word to list the partitions and the number of rows in various partitions.

    How many partitions currently have data rows? _____
    How many rows now exist in the table? _____ Has the row count changed? ___

    If the row count did not change, why not? _____

Notes

Module 18

Teradata Columnar

After completing this module, you will be able to:

- **Describe the components that comprise a Row ID in a column partitioned table.**
- **Identify two advantages of creating a column partitioned table.**
- **Identify two disadvantages of creating a column partitioned table.**
- **Identify the recommended way to populate a column partitioned table.**
- **Specify how rows are deleted in a column partitioned table.**

Notes

Table of Contents

Teradata Columnar 18-4
Teradata Columnar Benefits 18-6
  Columnar Join Indexes 18-6
No Primary Index Table DDL 18-8
The No Primary Index Table 18-10
Column Partition Table DDL (without Auto-Compression) 18-12
  Characteristics of a Columnar Table 18-12
Column Partition Container (No Automatic Compression) 18-14
The Column Partition Table (without Auto-Compression) 18-16
CP Table Query #1 (without Auto-Compression) 18-18
CP Table Query #1 (without Auto-Compression) 18-20
Column Partition Table DDL (with Auto-Compression) 18-22
Auto-Compression for CP Tables 18-24
Auto-Compression Techniques for CP Tables 18-26
User-Defined Compression Techniques 18-28
Column Partition Container (Automatic Compression) 18-30
The Column Partition Table (with Auto-Compression) 18-32
CP Table Query #2 (with Auto-Compression) 18-34
CP Table with Row Partitioning DDL 18-36
  Determining the Column Partition Level 18-36
The Column Partition Table (with Row Partitioning) 18-38
CP Table with Multi-Column Container DDL 18-40
The CP Table with Multi-Column Container 18-42
CP Table Hybrid Row & Column Store DDL 18-44
  COLUMN Format Considerations 18-44
  ROW Format Considerations 18-44
The CP Table (with Hybrid Row & Column Store) 18-46
Populating a CP Table 18-48
  INSERT-SELECT 18-48
  Options 18-48
DELETE Considerations 18-50
    The Delete Column Partition 18-50
UPDATE Considerations 18-52
  USI Access 18-52
  NUSI Access 18-52
CP Table Restrictions 18-54
Summary 18-56
Module 18: Review Questions 18-58
Lab Exercise 18-1 18-60

Teradata Columnar

Teradata Column or Column Partitioning (CP) is a new physical database design implementation option (starting with Teradata 14.0) that allows single columns or sets of columns of a NoPI table to be stored in separate partitions. Column partitioning can also be applied to join indexes.

Columnar is simply a new approach for organizing the data of a user-defined table or join index on disk.

Teradata Columnar offers the ability to partition a table or join index by column. Teradata Columnar can be used alone or in combination with row partitioning in multilevel partitioning definitions. Column partitions may be stored using traditional 'ROW' storage or alternatively stored using the new 'COLUMN' storage option. In either case, columnar can automatically compress physical rows where appropriate.

The key benefit in defining row-partitioned (PPI) tables is when queries access a subset of rows based on constraints on one or more partitioning columns. The major advantage of using column partitioning is to improve the performance of queries that access a subset of the columns from a table, either for predicates (e.g., WHERE clause) or projections (i.e., SELECTed columns).

Because sets of one or more columns can be stored in separate column partitions, only the column partitions that contain the columns needed by the query need to be accessed. Just as row-partitioning can eliminate rows that need not be read, column partitioning eliminates columns that are not needed.

The advantages of both can be combined, meaning even less data moved and thus reduced I/O. Fewer data blocks need to be read since more data of interest is packed together into fewer data blocks.

Columnar makes more sense in CPU-rich environments because CPU cycles are needed to "glue" columns back together into rows, for compression and for different join strategies (mainly hash joins).

Teradata Columnar

- **Description**
  - Columnar (or Column Partitioning) is a new physical database design implementation option that allows sets of columns (including just a single column) of a table or join index to be stored in separate partitions.
  - This is effectively an I/O reduction feature to improve performance for suitable classes of workloads.
  - This allows the capability for a table or join index to be column (vertically) partitioned, row (horizontally) partitioned or both by using the already existing multilevel partitioning capability.

- **Considerations**
  - Note that column partitioning is a physical database design choice and may not be suitable for all workloads using that table/join index.
  - It is especially suitable if both a small number of rows are selected and a few columns are projected.
  - When individual rows are deleted, they are not physically deleted, but are marked as deleted.

Teradata Columnar Benefits

The facing page lists a number of Teradata Columnar benefits.

Columnar Join Indexes

A join index can also be created as column-partitioned for either a columnar table or a non-columnar table. Conversely, a join index can be created as non-columnar for either type of table as well.

Sometime within a mixed workload, some queries might perform better if data is not column partitioned and some where it is column partitioned. Or, perhaps some queries perform better with one type of partitioning on a table, whereas other queries do better with another type of partitioning. Join indexes allow creation of alternate physical layouts for the data with the optimizer automatically choosing whether to access the base table and/or one of its join indexes.

A column-partitioned join index must be a single-table, non-aggregate, non-compressed, join index with no primary index, and no value-ordering, and must include RowID of the base table. A column-partitioned join index may optionally be row partitioned. It may also be a sparse join index.

This module will only describe and include examples of base tables that utilize column partitioning.

Teradata Columnar Benefits

Benefits of using the Teradata Columnar feature include:

- **Improved query performance**

  Column partitioning can be used to improve query performance via column partition elimination. Column partition elimination reduces the need to access all the data in a row while row partition elimination reduces the need to access all the rows.

- **Reduced disk space**

  The feature also allows for the possibility of using a new auto-compression capability which allows data to be automatically (as applicable) compressed as physical rows are inserted into a column-partitioned table or join index.

- **Increased flexibility**

  Provides a new physical database design option to improve performance for suitable classes of workloads.

- **Reduced I/O**

  Allows fast and efficient access to selected data from column partitions, thus reducing query I/O.

- **Ease of use**

  Provides simple default syntax for the CREATE TABLE and CREATE JOIN INDEX statements. No change is needed to queries.

No Primary Index Table DDL

The facing page simply illustrates the DDL to create a NoPI table. This example will be as a basis for multiple examples of creating tables with various column partitioning options.

No Primary Index Table DDL

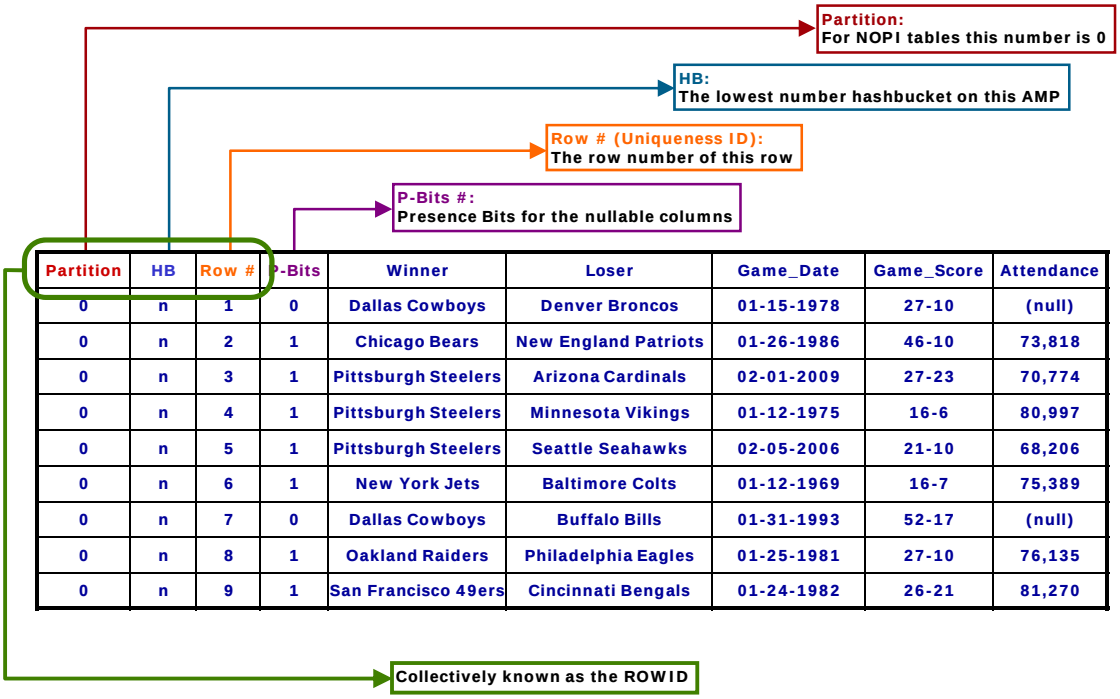
```
CREATE TABLE Super_Bowl
   (Winner          CHAR(25)     NOT NULL
   ,Loser           CHAR(25)     NOT NULL
   ,Game_Date       DATE         NOT NULL
   ,Game_Score      CHAR(7)      NOT NULL
   ,Attendance      INTEGER)
NO PRIMARY INDEX;
```

In this module, we will use a example of Super Bowl history information to simply demonstrate column partitioning.

The No Primary Index Table

The No Primary Index table is shown on the facing page.

The No Primary Index Table



Partition	HB	Row #	P-Bits	Winner	Loser	Game_Date	Game_Score	Attendance
0	n	1	0	Dallas Cowboys	Denver Broncos	01-15-1978	27-10	(null)
0	n	2	1	Chicago Bears	New England Patriots	01-26-1986	46-10	73,818
0	n	3	1	Pittsburgh Steelers	Arizona Cardinals	02-01-2009	27-23	70,774
0	n	4	1	Pittsburgh Steelers	Minnesota Vikings	01-12-1975	16-6	80,997
0	n	5	1	Pittsburgh Steelers	Seattle Seahawks	02-05-2006	21-10	68,206
0	n	6	1	New York Jets	Baltimore Colts	01-12-1969	16-7	75,389
0	n	7	0	Dallas Cowboys	Buffalo Bills	01-31-1993	52-17	(null)
0	n	8	1	Oakland Raiders	Philadelphia Eagles	01-25-1981	27-10	76,135
0	n	9	1	San Francisco 49ers	Cincinnati Bengals	01-24-1982	26-21	81,270

Column Partition Table DDL (without Auto-Compression)

With column partitioning, each column or specified group of columns in the table can become a partition containing the column partition values of that column partition. This is the simplest partitioning approach since there is no need to define partitioning expressions, as seen in the example on the facing page.

The clause PARTITION BY COLUMN specifies that the table has column partitioning. Each column of this table will have its own partition and will be (by default) in column storage since no explicit column grouping is specified.

Note that a primary index is not specified since this is NoPI table. A primary index may not be specified if the table is column partitioned.

Characteristics of a Columnar Table

A table or join index that is partitioned by column has several key characteristics:

- It does not have a primary index.
- Each column partition can be composed of single or multiple columns.
- Each column partition usually consists of multiple physical rows.
- A new physical row format COLUMN may be utilized for a column partition. Such a physical row is called a 'container' and it is used to implement columnar-storage for a column partition.
- Alternatively, a column partition may also have traditional physical rows with ROW format. Such a physical row for columnar partitions is called a subrow. This is used to implement row-storage for a column partition.
- Note that in subsequent discussions, when 'row storage' or 'row format' is stated, it is referring to columnar partitioning with the ROW storage option selected. This is not to be confused with row-partitioning which we associate with a PPI table.

In a table with multiple levels of partitioning, only one level may be column partitioned. All other levels must be row-partitioned (i.e., PPI).

Column Partition Table DDL
(without Auto-Compression)

```
CREATE TABLE Super_Bowl
 (Winner              CHAR(25)      NOT NULL
 ,Loser               CHAR(25)      NOT NULL
 ,Game_Date           DATE          NOT NULL
 ,Game_Score          CHAR(7)       NOT NULL
 ,Attendance          INTEGER)
NO PRIMARY INDEX
PARTITION BY COLUMN
  (Winner             NO AUTO COMPRESS
  ,Loser              NO AUTO COMPRESS
  ,Game_Date          NO AUTO COMPRESS
  ,Game_Score         NO AUTO COMPRESS
  ,Attendance         NO AUTO COMPRESS);
```

Defaults for a column partitioned table.

- Single-column partitions; options include multicolumn partitions.
- Auto compression is on; NO AUTO COMPRESS turns off auto-compression for the column.
- System-determined column-store for above column partitions; options include column-store (COLUMN) or row-store (ROW).

Column Partition Container (No Automatic Compression)

In order to support columnar-storage for a column partition, a new format, referred to as a COLUMN format in the syntax, is available for a physical row.

A physical row with this format is referred to as a ***container*** and each container holds a series of column partition values for a column partition.

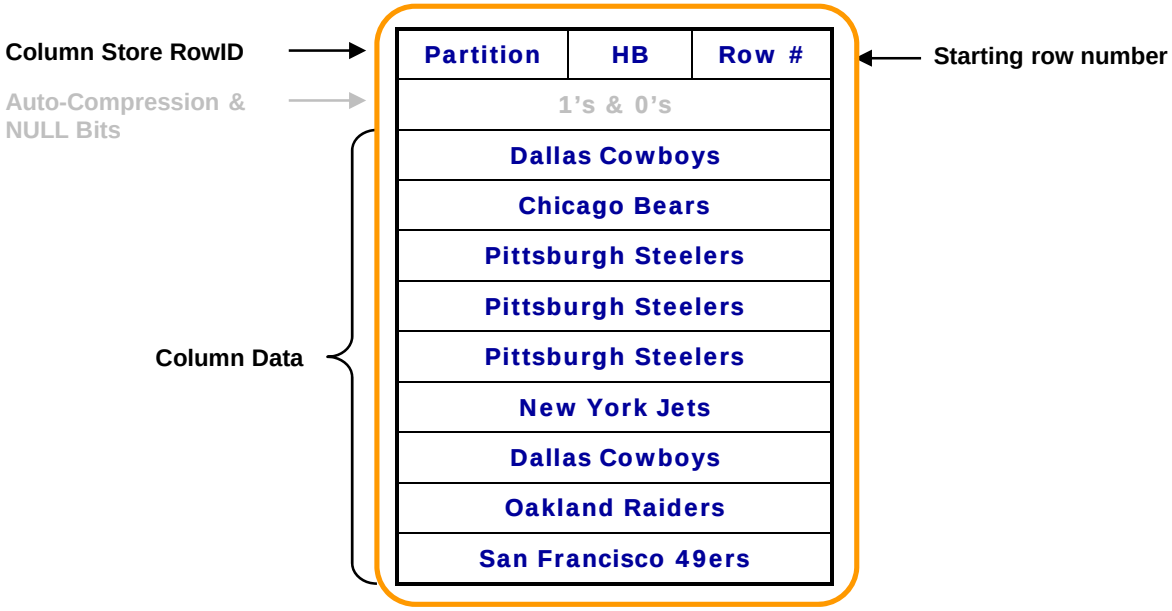
Each container is assigned a specific partition number which identifies the column or group of columns whose column partition values are held in the container. When a column partition is stored on disk, one or more containers may be needed to hold all the column partition values of the column partition. Since a container is a physical row, the size of a container is limited by the maximum physical row size.

The example on the facing page assumes that NO AUTO COMPRESS has been specified for the column.

Containers hold multiple values for the same column (or columns) of a table. For purposes of this explanation, the assumption is being made that each partition contains only a single column so a column partition value is the same as a column value. Recall that each column value belongs to a specific row and that each row is identified by a RowID consisting of a row-hash and uniqueness value. Since all of the rows on a single AMP of a NoPI table share the same row hash, the uniqueness value becomes the real differentiator. So the connection between a specific column value for a particular row on a given AMP and its uniqueness value is the key in locating the corresponding column value.

Assume that a given container holds 1000 values. The RowID of each container carries a hash bucket and uniqueness which represents the first column value entry in the container. The first value's hash bucket and uniqueness is explicit while the other values' hash bucket and uniqueness are implicit and are understood based on their position in their container. The exact location of a column partition value is known based on its relative position within the container. For example, if the uniqueness value in the container's RowID is 201 and a column partition value with a uniqueness value of 205 needs to be located, the 5th entry in that container is the corresponding column partition value.

Column Partition Container
(No Automatic Compression)



Column Container is effectively a row in the partition.

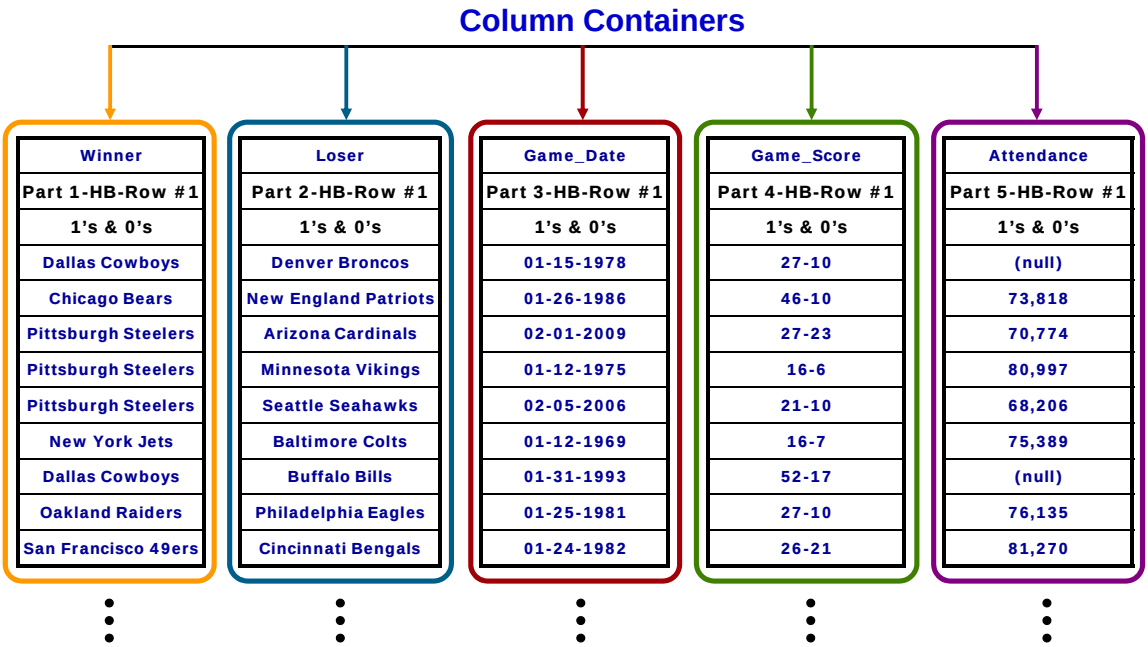
The Column Partition Table (without Auto-Compression)

The result of creating a column partitioned table (as shown previously) is shown on the facing page with some sample data.

The clause PARTITION BY COLUMN specifies that the table has column partitioning. Each column of this table will have its own partition and will be (by default) in column storage since no explicit column grouping is specified.

The default of auto-compression is overridden for each of the columns.

The Column Partition Table
(without Auto-Compression)



Column containers are effectively separate rows in a NoPI table.

CP Table Query #1 (without Auto-Compression)

One of the key advantages of column partitioning is opportunity for reduced I/O. This can be realized if only a subset of the columns in a table are read and if those column values are held in separate column partitions. Data is stored on disk by partition, so when partition elimination takes place, data blocks in the eliminated partitions are simply not read.

There are three ways to initiate read access to data within a column-partitioned table:

- A full column partition scan
- Indexed access (using a secondary, join index, or hash index),
- A RowID join.

Both unique and non-unique secondary indexes are allowed on column-partitioned tables, as are join indexes and hash indexes.

Queries best suited for scanning a column-partitioned table are queries that:

- Contain one or a few predicates that are very selective in combination.
- Require a small enough number of columns to be accessed that the caches required to support their consolidation can be held in memory.

CP Table Query #1 (without Auto-Compression)

Which teams have lost to the "Dallas Cowboys" in the Super Bowl?

Winner	Loser	Game_Date	Game_Score	Attendance
Part 1-HB-Row #1	Part 2-HB-Row #1	Part 3-HB-Row #1	Part 4-HB-Row #1	Part 5-HB-Row #1
1's & 0's	1's & 0's	1's & 0's	1's & 0's	1's & 0's
Dallas Cowboys	Denver Broncos	01-15-1978	27-10	(null)
Chicago Bears	New England Patriots	01-26-1986	46-10	73,818
Pittsburgh Steelers	Arizona Cardinals	02-01-2009	27-23	70,774
Pittsburgh Steelers	Minnesota Vikings	01-12-1975	16-6	80,997
Pittsburgh Steelers	Seattle Seahawks	02-05-2006	21-10	68,206
New York Jets	Baltimore Colts	01-12-1969	16-7	75,389
Dallas Cowboys	Buffalo Bills	01-31-1993	52-17	(null)
Oakland Raiders	Philadelphia Eagles	01-25-1981	27-10	76,135
San Francisco 49ers	Cincinnati Bengals	01-24-1982	26-21	81,270
⋮	⋮	⋮	⋮	⋮

Only the accessed columns are needed.

CP Table Query #1 (without Auto-Compression)

If indexing is not available, Teradata can access data in a CP table by scanning a column partition on all the AMPs in parallel.

In the example on the facing page, the “Winner” column containers are scanned for “Dallas Cowboys”.

The following describes the scanning of CP data:

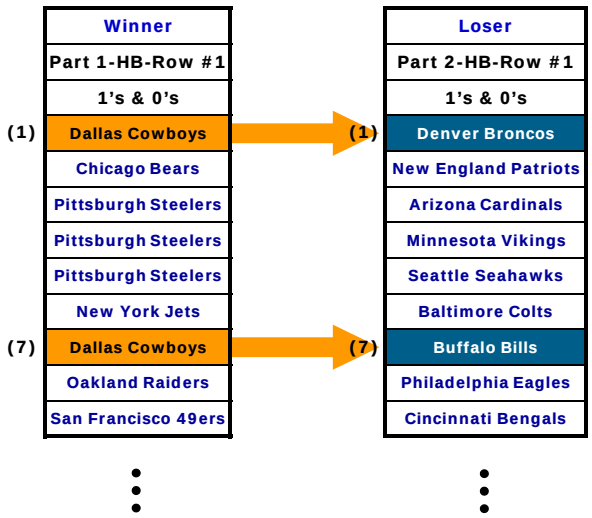
1. Columns within the table definition that aren’t referenced in the query are ignored due to partition elimination.
2. If there is a predicate column in the query, its column partition is read.
3. Values within the predicate column partition are examined and compared against the value passed in the query WHERE clause.
4. Each time a qualifying value is located, the next step is building up a row for the output spool.
5. All the column partition values for a logical row have the same RowID except for the column partition number. The RowID associated with each predicate column value that matches the constraint in the query becomes the link to other column partition values of the same logical row by simply modifying the column partition number of the RowID to the column partition number for each of these other column partition values.

If there is more than one predicate column in the query that can be used to disqualify rows, the column for one of these predicates is chosen and its column partition is scanned. Statistics, as well as other heuristics, are used by the optimizer to pick the most selective and least costly predicate. Once that decision has been made, only that single column partition needs to be scanned.

If there are no useful predicate columns in the query (for instance, OR’ed predicates), one column partition is chosen to be scanned and for each of its column partition values additional corresponding column partition values are accessed until either predicate evaluation disqualifies the logical row or all the projected column values have been retrieved and brought together to form rows for the output spool.

CP Table Query #1 (without Auto-Compression)

Which teams have lost to the "Dallas Cowboys" in the Super Bowl?



The relative row number in each container is used to access the data.

Column Partition Table DDL (with Auto-Compression)

The DDL to create a column partitioned table with auto-compression is shown on the facing page. Each column will be maintained in a separate partition.

The clause PARTITION BY COLUMN specifies that the table has column partitioning. Each column of this table will have its own partition and will be (by default) in column storage since no explicit column grouping is specified.

Column Partition Table DDL
(with Auto-Compression)

```
CREATE TABLE Super_Bowl
  (Winner            CHAR(25)      NOT NULL
  ,Loser             CHAR(25)      NOT NULL
  ,Game_Date         DATE          NOT NULL
  ,Game_Score        CHAR(7)       NOT NULL
  ,Attendance        INTEGER)
NO PRIMARY INDEX
PARTITION BY COLUMN;

Note: Auto Compression is on by Default.
```

Auto-Compression for CP Tables

Auto-compression is a completely transparent compression option for column partitions. It is applied to a container when a container is full after appending some number of column partition values without auto-compression by an INSERT or UPDATE statement. Each container is assessed separately to see how, and if, it can be compressed.

Several available compression techniques are considered for compressing a container but, unless there is some size reduction, no compression is performed. If a container is compressed, the needed data is automatically uncompressed as it is read.

Auto-compression happens automatically and is most effective when the column partition is based on a single column only, and less effectively as more columns are included in the column partition.

User-defined compression, such as multi-value or algorithmic compression that is already defined by the user is honored and carried forward if it helps compress the container. If block level compression is specified, it applies for data blocks holding the physical rows of the table independent of whether auto-compression is applied or not.

Auto-Compression for CP Tables

Auto Compression

- When a column partition is defined to have auto-compression (i.e., the NO AUTO COMPRESS option is not specified), data is compressed by the system as physical rows that are inserted into a column-partitioned table or join index.
- For some values, there is no applicable compression technique that reduces the size of the physical row and the system will determine not to compress the values for that physical row.
- The system decompresses any compressed column-partition values when they are retrieved.
- Auto-compression is most effective for a column partition with a single column and COLUMN format.
- There is overhead in determining whether or not a physical row is to be compressed and, if it is to be compressed, what compression techniques are to be used.
- This overhead can be eliminated by specifying the NO AUTO COMPRESS option for the column partition.

Auto-Compression Techniques for CP Tables

The facing page lists and briefly describes each of the auto-compression techniques that Teradata may utilize.

Auto-Compression Techniques for CP Tables

- **Run-Length Encoding**

  Each series of one or more column-partition values that are the same are compressed by having the column-partition value occur once with an associated count of the number of occurrences in the series.

- **Local Dictionary Compression**

  This is similar to a user-specified value-list compression for a column. Often occurring column-partition values within a physical row are placed in a value-list dictionary local to the physical row.

- **Trim Compression**

  Trim high-order zero bytes of numeric values and trailing pad bytes of character and byte values with bits to indicate how many bytes were trimmed or what the length is after trimming.

- **Null Compression**

  Similar to null compression (COMPRESS NULL) for a column except applied to a column-partition value. A single-column or multicolumn-partition value is a candidate for null compression if all the column values in the column-partition value are null (this also means all these columns must be nullable).

- **Delta on Mean Compression**

  Delta on Mean compression computes the mean/average of all the values in the column container. This mean value is saved and stored in the container. After Delta on Mean compression, the value that is stored for a row is the difference with the mean. So for instance, if the average is say 100 and the four values in four different rows are 99, 98, 101, and 102. Then the values stored are -1, -2, 1, and 2.

- **Unicode to UTF8 Compression**

  For a column defined with UNICODE character set but where the value consists of ASCII characters, compress the Unicode representation (2 bytes per character) to UTF8 (1 byte per character).

User-Defined Compression Techniques

User-defined compression, such as multi-value or algorithmic compression that is already defined by the user is honored and carried forward if it helps compress the container.

If block level compression is specified, it applies for data blocks holding the physical rows of the table independent of whether auto-compression is applied or not.

Note that auto-compression is applied locally to a container based on column partition values (which may be multicolumn) while user-specified MVC and ALC are applied globally for a column and are applicable to both containers and subrows.

Auto-compression is differentiated from block level compression in several key ways:

- Auto-compression requires no parameter setting, but rather is completely transparent to the user while block level compression is a result of the appropriate settings of parameters.
- Auto-compression acts on a container (a physical row) while block level compression acts on a data block (which consists of one or more physical rows).
- Decompressing a column partition value in a container has little overhead while software-based block level compression incurs noticeable decompression overhead.
- Only column partition values that are needed by the query are decompressed. BLC has to decompress the entire data block even if only one or a few values are needed from the data block.
- Determining the auto-compression to use for a container, compressing a container, and compressing additional values to be inserted into the container can cause an increase in the CPU needed for appending values to column partitions.

You can expect additional CPU to be required when inserting rows into a CP table that uses auto-compression. This is similarly to the increase in CPU if MVC or ALC compression is added to the columns.

User-Defined Compression Techniques

All the current compression techniques available in Teradata today, can be leveraged and used for column partitioned tables.

- **Dictionary-Based Compression:**

  Allows end-users to identify and target specific values that would be compressed in a given column. Also known as, Multi-Value Compression.

- **Algorithmic Compression:**

  Allows users the option of defining compression/decompression algorithms that would be implemented as UDFs and that would be specified and applied to data at the column level in a row. Teradata provides three compression/decompression algorithms that offer compression for UNICODE and LATIN data columns.

- **Block-Level Compression:**

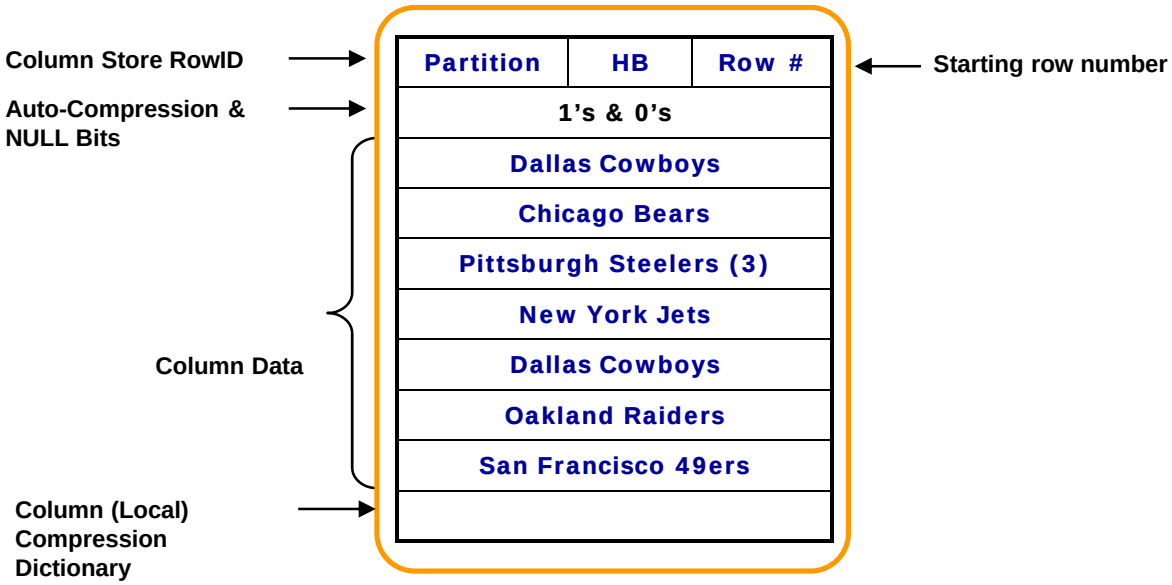
  Feature provides the capability to perform compression on whole data blocks at the file system level before the data blocks are actually written to storage.

Column Partition Container (Automatic Compression)

In order to support columnar-storage for a column partition, a new format, referred to as a COLUMN format in the syntax, is available for a physical row.

The example on the facing page assumes that automatic compression is on for the column.

Column Partition Container
(Automatic Compression)



Column Container is effectively a row in the partition.

The Column Partition Table (with Auto-Compression)

The result of creating a column partitioned table with auto-compression is shown on the facing page.

The Column Partition Table
(with Auto-Compression)

Winner
Part 1-HB-Row #1
1's & 0's
Dallas Cowboys
Chicago Bears
Pittsburgh Steelers
Pittsburgh Steelers
Pittsburgh Steelers
New York Jets
Dallas Cowboys
Oakland Raiders
San Francisco 49ers

Run-Length Encoding

Loser
Part 2-HB-Row #1
1's & 0's
Denver Broncos
New England Patriots
Arizona Cardinals
Minnesota Vikings
Seattle Seahawks
Baltimore Colts
Buffalo Bills
Philadelphia Eagles
Cincinnati Bengals

Trim Trailing Spaces

Game_Date
Part 3-HB-Row #1
1's & 0's
01-15-1978
01-26-1986
02-01-2009
01-12-1975
02-05-2006
01-12-1969
01-31-1993
01-25-1981
01-24-1982

No Compression

Game_Score
Part 4-HB-Row #1
1's & 0's
27-10
46-10
27-23
16-6
21-10
16-7
52-17
27-10
26-21

Local Dictionary Compression (27-10 is compressed)

Attendance
Part 5-HB-Row #1
1's & 0's
(Null)
73,818
70,774
80,997
68,206
75,389
(Null)
76,135
81,270

Null Compression

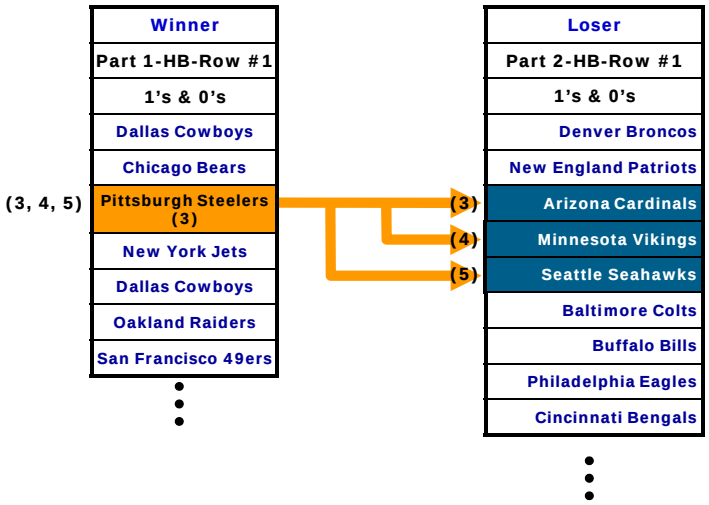
Columnar compression is based on each Container. Therefore, each Container may have different compression characteristics and even different compression methods.

CP Table Query #2 (with Auto-Compression)

The Pittsburgh Steelers team was compressed, but effectively represented 3 values in the container. These 3 values correspond to 3, 4, and 5 in the other container (Loser column).

CP Table Query #2
(with Auto-Compression)

Which teams have lost to the "Pittsburgh Steelers" in the Super Bowl?



CP Table with Row Partitioning DDL

Row partitioning can be combined with column partitioning on the same table. This allows queries to read only non-eliminated combined partitions. Such partitions are defined by the intersection of the columns referenced in the query and any partitioning column selection criteria.

There is usually an advantage to putting the column partitioning at level one of the combined partitioning scheme.

The DDL to create a column partitioned table with auto-compression and Row partitioning is shown on the facing page.

Determining the Column Partition Level

It is initially recommended that column partitioning either be defined as the first level or, if not as the first level, at least as the second level. When column partitioning is defined as the first level it is easier for the file system to locate related data that is from the same logical row of the table. When column partitioning is defined at a lower level, more boundary checks have to be made, possibly causing an impact on performance.

If you are inserting a new table row, it takes more effort if the column partitioning is not the first level. Values of columns from the newly-inserted table row need to be appended at the end of each column partition. If column-partitioning is not first, it is necessary to read through several combined partitions to find the correct container that represents the end point.

On the other hand, if you have row partitioning at the second or a lower level partitioning level so that column partitioning can be at the first level, this can be less efficient when row partition elimination based on something like a date range is taking place.

CP Table with Row Partitioning DDL

```
CREATE TABLE Super_Bowl
   (Winner           CHAR(25)      NOT NULL
   ,Loser            CHAR(25)      NOT NULL
   ,Game_Date        DATE          NOT NULL
   ,Game_Score       CHAR(7)       NOT NULL
   ,City             CHAR(40))
NO PRIMARY INDEX
PARTITION BY
(COLUMN
,RANGE_N(Game_Date BETWEEN
              DATE '1960-01-01' and DATE '2059-12-31'
              EACH INTERVAL '10' YEAR));

Note: Auto Compression is on by Default.
```

The Column Partition Table (with Row Partitioning)

The result of creating a column partitioned table with auto-compression and row partitioning is shown on the facing page.

The Column Partition Table
(with Row Partitioning)

- In the 1970s, which teams won Super Bowls, who were the losing teams , and what was the date the game was played?

Winner	Loser	Game_Date	Game_Score	City
Part 1-HB-Row #1	Part 2-HB-Row #1	Part 3-HB-Row #1	Part 4-HB-Row #1	Part 5-HB-Row #1
1's & 0's	1's & 0's	1's & 0's	1's & 0's	1's & 0's
New York Jets	Baltimore Colts	01-12-1969	16-7	Miami, FL

Winner	Loser	Game_Date	Game_Score	Attendance
Part 11-HB-Row #1	Part 12-HB-Row #1	Part 13-HB-Row #1	Part 14-HB-Row #1	Part 15-HB-Row #1
1's & 0's	1's & 0's	1's & 0's	1's & 0's	1's & 0's
Dallas Cowboys	Denver Broncos	01-15-1978	27-10	New Orleans, LA
Pittsburgh Steelers	Minnesota Vikings	01-12-1975	16-6	

Winner	Loser	Game_Date	Game_Score	Attendance
Part 41-HB-Row #1	Part 42-HB-Row #1	Part 43-HB-Row #1	Part 44-HB-Row #1	Part 45-HB-Row #1
1's & 0's	1's & 0's	1's & 0's	1's & 0's	1's & 0's
Pittsburgh Steelers	Seattle Seahawks	02-05-2006	21-10	68,206

CP Table with Multi-Column Container DDL

When a table is defined with column partitioning, by default each column becomes its own column partition. However, it is possible to group multiple columns into a single partition.

This has the result of fewer column partitions with more data held within each column partition.

Grouping columns into fewer column partitions may be appropriate in these situations:

- When the table has a large number of columns (having fewer column partitions may improve the performance of INSERT-SELECT and UPDATE statements).
- When access to the table often involves a large percentage of the columns and the access is not very selective.
- When a common subset of columns are frequently accessed together.
- When a multicolumn NUSI is created on a group of columns.
- There are too few available column partition contexts to access all the needed column partitions for queries.

Note that auto-compression will probably be less effective if columns are grouped together instead of being in their own column partitions.

CP Table with Multi-Column Container DDL

```
CREATE TABLE Super_Bowl
 ( Winner            CHAR(25)      NOT NULL
  ,Loser             CHAR(25)      NOT NULL
  ,Game_Date         DATE          NOT NULL
  ,Game_Score        CHAR(7)       NOT NULL
  ,Attendance        INTEGER)
NO PRIMARY INDEX
PARTITION BY COLUMN
  (Winner            NO AUTO COMPRESS
  ,Loser             NO AUTO COMPRESS
  ,(Game_Date
   ,Game_Score
   ,Attendance)      NO AUTO COMPRESS)
;
```

Recommendation:
The group of multiple columns should be less than 256 bytes.

Note that this example is without Auto-Compression.

Watch the difference between 'Projection' and 'Predicate'.

If you are always projecting three columns, it may make sense to group these columns into one Container. If, however, one of these columns is used in a WHERE Predicate, then it may be better to place this column into its own Container.

The CP Table with Multi-Column Container

The example on the facing page illustrates a CP table that has a multi-column container.

The CP Table with Multi-Column Container

Single Column Containers

Winner
Part 1-HB-Row #1
1's & 0's
Dallas Cowboys
Chicago Bears
Pittsburgh Steelers
Pittsburgh Steelers
Pittsburgh Steelers
New York Jets
Dallas Cowboys
Oakland Raiders
San Francisco 49ers

Loser
Part 2-HB-Row #1
1's & 0's
Denver Broncos
New England Patriots
Arizona Cardinals
Minnesota Vikings
Seattle Seahawks
Baltimore Colts
Buffalo Bills
Philadelphia Eagles
Cincinnati Bengals

Multi-Column Container

Game_Date	Game_Score	Attendance
Part 3-HB-Row #1		
1's & 0's		
01-15-1978	27-10	(null)
01-26-1986	46-10	73,818
02-01-2009	27-23	70,774
01-12-1975	16-6	80,997
02-05-2006	21-10	68,206
01-12-1969	16-7	75,389
01-31-1993	52-17	(null)
01-25-1981	27-10	76,135
01-24-1982	26-21	81,270

General recommendations:

- If you have a lot of columns in a table, then multi-column containers may be needed.
- Multi-column containers will not compress as well as single-column containers.
- If you select any column in a multi-column container you will get all of the other columns.

CP Table Hybrid Row & Column Store DDL

The example on the facing page illustrates the DDL to create a column partitioned table that has a combination of row and column storage.

COLUMN Format Considerations

The COLUMN format packs column partition values into a physical row, referred to as a container, up to a system-determined limit. Whether or not to change a column partition to use ROW format depends on the whether the benefit of row header compression and auto-compression can be realized or not.

A row header occurs once per container, with the RowID of the first column partition value becoming the RowID of the container itself. In a column-partitioned table, each column partition value is assigned its own RowID, but in a container these RowIDs are implicit except for the first one specified in the header. The uniqueness value can be determined from the position of a column partition value relative to the first column partition value. Thus the row id for each value in the container is implicitly available and an explicit RowID does not need be carried for each individual column value in the container.

If many column partition values can be packed into a container, this form of compression (referred to as row header compression) can reduce the space needed for a column-partitioned table compared to the table without column partitioning. If only a few column partition values (because they are wide) can be placed in a container, there can actually be an increase in the space needed for the table compared to the table without column partitioning. In this case, ROW format may be more appropriate.

ROW Format Considerations

A subrow, on the other hand, has a format that is the same as traditional row (except it only has the values of a subset of the columns). Subrows are appropriate when column partition values are very wide and you expect only one or a few values to fit in a columnar container.

In this case, auto-compression and row header compression using COLUMN format might not be very effective. ROW format provides quicker access to specific values because no search through a physical row is required to find only one value. Each column partition value is in it owns subrow with a row header. Subrows are not subject to auto-compression but may be in the future.

CP Table Hybrid Row & Column Store DDL

```
CREATE TABLE Super_Bowl
 (Winner      CHAR(25)      NOT NULL
 ,Loser       CHAR(25)      NOT NULL
 ,Game_Date   DATE          NOT NULL
 ,Game_Score    CHAR(7)     NOT NULL
 ,Attendance    INTEGER
 ,City          CHAR(40))
NO PRIMARY INDEX
PARTITION BY COLUMN
  (Winner      NO AUTO COMPRESS
  ,Loser       NO AUTO COMPRESS
  ,ROW (Game_Date
       ,Game_Score
       ,Attendance
       ,City) NO AUTO COMPRESS);
```

This example illustrates the syntax to create a row store, but in reality you would only define the row format if the set of columns was greater than 256 bytes.

General recommendation:

- A column (or set of columns) should be at least 256 bytes wide before considering ROW format.
- Row stores will take up more space, but act like a row in terms of retrieving data.
- Each row will have a row header and require more space.

The CP Table (with Hybrid Row & Column Store)

As an alternative to COLUMN format, column partition values may be held in a physical row using what is known in Teradata syntax as ROW format. The type of physical row supports row-storage for a column partition and is referred to as a subrow. Each subrow holds one column partition value for a column partition. A subrow has the same format as regular row except that it is generally a subset of the columns for a table row instead of all the columns. Just like a container, each subrow is assigned to a specific partition. One or more subrows may be needed to hold the entire column partition. Since a subrow is a physical row, the size of a subrow is limited by the maximum physical row size.

A column partition may have COLUMN format or ROW format but not a mix of both. However, different column partitions in column-partitioned table may have different formats.

The CP Table
(with Hybrid Row & Column Store)

Column and Row Store in "one" table.

Column Store Containers

Winner
Part 1-HB-Row #1
1's & 0's
Dallas Cowboys
Chicago Bears
Pittsburgh Steelers
Pittsburgh Steelers
Pittsburgh Steelers
New York Jets
Dallas Cowboys
Oakland Raiders
San Francisco 49ers

Loser
Part 2-HB-Row #1
1's & 0's
Denver Broncos
New England Patriots
Arizona Cardinals
Minnesota Vikings
Seattle Seahawks
Baltimore Colts
Buffalo Bills
Philadelphia Eagles
Cincinnati Bengals

Row Store

Partition	HB	Row #	Game_Date	Game_Score	Attendance	City
0	n	1	01-15-1978	27-10	(null)	New Orleans, LA
0	n	2	01-26-1986	46-10	73,818	New Orleans, LA
0	n	3	02-01-2009	27-23	70,774	Tampa, FL
0	n	4	01-12-1975	16-6	80,997	New Orleans, LA
0	n	5	02-05-2006	21-10	68,206	Detroit, MI
0	n	6	01-12-1969	16-7	75,389	Miami, FL
0	n	7	01-31-1993	52-17	(null)	Pasadena, CA
0	n	8	01-25-1981	27-10	76,135	New Orleans, LA
0	n	9	01-24-1982	26-21	81,270	Pontiac, MI

Populating a CP Table

INSERT-SELECT

INSERT-SELECT is the expected and most efficient method of loading data into a column-partitioned table. If the data originates from an external source, FastLoad can be used to load it into a staging table from which the INSERT-SELECT can take place.

If the source was a SELECT that included several joins and as a result skewed data was produced, options can be added to the INSERT-SELECT statement to avoid a skewed column-partitioned table and improve the effectiveness of auto-compression:

Options

HASH BY (RANDOM or hash_spec_list):
The selected rows are redistributed by the hash value of the expressions in the hash_spec_list. Alternatively, HASH BY RANDOM can be specified to have data blocks redistributed randomly. It is important that a column or columns be selected that distributes well if the HASH BY option is used.

LOCAL ORDER BY:
A local sort is done on each AMP before physically storing the rows. This could help auto-compression to be more effective by ensuring that like values of the sorting columns appear together.

During an INSERT-SELECT process, each source row is read, and its columns individually appended to the column partitions to which they belong. As many column partition values as can fit are built up simultaneously in memory, and written out to disk when the buffer is filled.

If the column-partitioned table being loaded has a large number of columns, additional passes of the source table may be required to append all of the columns to their respective column partitions.

Populating a CP Table

```
CREATE TABLE Super_Bowl_Staging
  (Winner         CHAR(25) NOT NULL
  ,Loser          CHAR(25) NOT NULL
  ,Game_Date      DATE     NOT NULL
  ,Game_Score     CHAR(7)  NOT NULL
  ,Attendance     INTEGER)
NO PRIMARY INDEX;
```

```
CREATE TABLE Super_Bowl
  (Winner         CHAR(25) NOT NULL
  ,Loser          CHAR(25) NOT NULL
  ,Game_Date      DATE     NOT NULL
  ,Game_Score     CHAR(7)  NOT NULL
  ,Attendance     INTEGER)
NO PRIMARY INDEX
PARTITION BY COLUMN;
```

1. **Load data into staging table.**
2. **INSERT INTO Super_Bowl ….. SELECT * FROM Super_Bowl_Staging …**

DELETE Considerations

Rows can be deleted from a column-partitioned table using the DELETE ALL, or selectively using DELETE. DELETE ALL uses the standard fast-path delete as would be done on a primary-indexed table. If a column-partitioned table also happens to include row partitioning, the same fast-path delete can be applied to one or more row partitions. Space is immediately reclaimed.

The selective DELETE, in which only one or a few rows of the table are deleted, requires a scan of a column partition or indexed access to the column-partitioned table. In this case, the row being deleted is not physically removed, but only flagged as having been deleted. The space taken by a row being deleted is scattered across multiple column partitions and is not reclaimed at the time of the deletion. This form of delete should only be used to delete a small percentage of rows.

During a delete operation, all large objects are immediately deleted, as are entries in secondary indexes. Join indexes are updated to reflect the change as it happens.

The Delete Column Partition

Each column-partitioned table has one delete column partition, in addition to the user-specified column partitions. It holds information about deleted rows so they do not get included in an answer set. When a single row delete takes place in a column-partitioned table, rather than removing each deleted value across all the column partitions, which would involve multiple physical row updates, a single action is performed. One bit in the delete column partition is set as an indication that the hash bucket and uniqueness associated with the table row has been deleted.

This delete column partition is accessed any time a query is made against a column-partitioned table without indexed access. At the time a column partition is scanned, the delete column partition is checked to make sure a table row being requested by the query has not been deleted (if it has, the value is skipped). This additional partition access can be observed in the EXPLAIN text.

DELETE Considerations

- DELETE ALL uses the standard fast-path delete as would be done on a primary-indexed table.
  - If a CP table also happens to include row partitioning, the same fast-path delete can be applied to one or more row partitions. Space is immediately reclaimed.
- The selective DELETE, in which only one or a few rows of the table are deleted, requires a scan of a column partition or indexed access to the column-partitioned table.
  - In this case, the row being deleted is not physically removed, but only flagged as having been deleted.
  - This form of delete should only be used to delete a small percentage of rows.
- The Delete Column Partition - each column-partitioned table has one delete column partition, in addition to the user-specified column partitions. It holds information about deleted rows so they do not get included in an answer set.
  - One bit in the delete column partition is set as an indication that the hash bucket and uniqueness associated with the table row has been deleted.

UPDATE Considerations

Updating rows in column partitioned table requires a delete and an insert operation. It involves marking the appropriate bit in the delete column partition, and then re-inserting columns for the new updated version of the table row. The cost of this update is less severe than a Primary Index update (also a delete plus insert) because in the column-partitioned table update, the deletion and re-insertion takes place on the same AMP.

The part of the update that re-inserts a new table row is essentially a re-append. The highest uniqueness value on that AMP is incremented by one, and all the column values for that updated row are appended to their corresponding column partitions. Because multiple I/Os are performed in doing this re-append, row-at-a-time updates on column-partitioned tables should be approached with caution. The space that is being used by the old row is not reclaimed, but a delete bit is turned on in the delete column partition, indicating that the old version of the row is obsolete.

An UPDATE statement should only be used to update a small percentage of rows.

USI Access

For example, consider a unique secondary index (USI) access. The USI subtable provides the specific RowID of the base table row. In the columnar case, the base table row is located on a specific AMP which can be stored in multiple containers. As it is for a PI or NoPI tables, the hash bucket in the RowID carried in the USI is used to locate the AMP that contains the base table row. The column values from the multiple containers are located the same as using a RowID retrieved from a NUSI which is described below.

NUSI Access

With non-unique secondary indexes (NUSIs), a row-id list is retrieved from the NUSI subtable. In the case of a column-partitioned table, the table row has been decomposed into columns that are located in different column partitions on disk. Several different internal partition numbers come into play in reconstructing the table row.

Rather than relying on the column partition number, it is only the hash bucket and uniqueness that is of importance in the NUSI subtable RowID list. The hash bucket and uniqueness identifies the table row on that AMP, while the column partition number plays no meaningful role.

Because column partition numbers in the NUSI subtable are not useful in the case of a column-partitioned table, all RowIDs in a NUSI carry a column partition number of 1. The hash bucket and uniqueness value are the important link between the NUSI subtable and the column values of interest for the query. Once the hash bucket and uniqueness value is known, a RowID is formulated using the internal partition number of the column of interest. This RowID is then used to read the containing physical row/container. A relative position is determined using the uniqueness value which is then used to access the appropriate column value. This process is repeated to locate the column value of each of the remaining columns for this row. These individual column values are brought together to formulate the row. This process occurs for each RowID in the NUSI subtable entry.

UPDATE and USI/NUSI Considerations

UPDATE Considerations

- Updating rows in column partitioned table requires a delete and an insert operation.
- It involves marking the appropriate bit in the delete column partition, and then re-inserting columns for the new updated version of the table row.
- An UPDATE statement should only be used to update a small percentage of rows.

USI/NUSI Considerations

- For a USI on a CP table, the base table row is located on a specific AMP which can be stored in multiple containers. The hash bucket in the RowID carried in the USI is used to locate the AMP that contains the base table row.
- For a NUSI on a CP table, the table row has been decomposed into columns that are located in different column partitions on disk. Several different internal partition numbers come into play in reconstructing the table row.
- Rather than relying on the column partition number, it is only the hash bucket and uniqueness that is of importance in the NUSI subtable RowID list.

CP Table Restrictions

The following limitations apply:

- Column partitioning for join indexes is restricted to single-table, non-aggregate, non-compressed join indexes with no PI and no ORDER BY clause
  - ROWID of base table must be included in a CP join index

- Column partitioning is not allowed for the following:
  - Primary index (PI) base tables
  - Global temporary, volatile, and queue tables
  - Secondary indexes

- Column partitioning is not applicable for the following:
  - Global temporary trace tables
  - Error tables
  - Compressed join indexes

- NoPI table with only row partitioning is not allowed

- A column cannot be specified to be in more than one column partition

- Column grouping cannot be specified in both the column definition list of CREATE TABLE statement and in the COLUMN clause.

- Column grouping cannot be specified in both the select list of a CREATE JOIN INDEX statement and in the COLUMN clause.

CP Table Restrictions

- **Column Partitioning is predicated on the NoPI table structure and as such the following restrictions apply:**
  - Set Tables
  - Queue tables
  - Global Temporary Tables
  - Volatile Tables
  - Derived Tables
  - Multi-table or Aggregate Join Index
  - Compressed Join Index
  - Hash Index
  - Secondary Index are not column partitioned
- **Column Partitioned tables cannot be loaded with either the FastLoad or the MultiLoad utilities.**
- **Merge-Into and UPSERT statements are not supported.**
- **Population of Column Partition tables will require an Insert-Select process after data has been loaded into a staging table.**
- **No synchronized scanning with Columnar Tables.**
- **Since Columnar Tables are NoPI Tables, Teradata cannot use Full Cylinder Reads.**

Summary

The facing page contains a summary of key concepts from this module.

Summary

When is column partitioning useful?

- Queries access varying subsets of the columns of table
  or
  Queries of the table are selective (Best if both occur for queries)
- For example, ad hoc, data analytics
- Data can be loaded with large INSERT-SELECTs
- There is no or little update/delete maintenance between refreshes or appends of the data for the table or for row partitions.

Do NOT use this feature when:

- Queries need to be run on current data that is changing (deletes and updates).
- Performing tactical queries or OLTP queries.
- Workload is CPU bound such that a trade-off of reduced I/O with increased CPU does not improve the throughput.
  - Column partitioning is *not* intended to be a CPU savings feature.

Module 18: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 18: Review Questions

1. Which two choices apply to Column Partitioning?
   a. SET table
   b. NoPI table
   c. Table with multi-level partitioning
   d. Table with existing row partitioning

2. What are two benefits of Column Partitioning?
   a. Reduced I/O
   b. Reduced CPU
   c. Reduced disk space usage
   d. Reduced tactical query response times

3. True or False? Deleting a row in a column partitioned table will reclaim table space.

4. True or False? In a multi-level partitioned table, only one level may be column partitioned.

5. True or False? The preferred way to load a columnar table is using INSERT/SELECT.

Lab Exercise 18-1

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

SQL hints:

INSERT INTO *table_1* SELECT * FROM *table_2*;

SELECT COUNT(*) FROM *table_name*;

SHOW TABLE *table_name*;

A count of rows in the Orders table is 31,200.
A count of rows in the Orders_2012 table is 12,000.

Lab Exercise 18-1

Lab Exercise 18-1

Purpose

In this lab, you will use Teradata SQL Assistant to create tables with column partitioning in various ways.

What you need

Populated DS tables and Orders and Orders_2012 tables in your database

Tasks

1. Use the SHOW TABLE for Orders to help create a new, similar table (same column names and definitions, etc.) that does NOT have a primary index and name this table "Orders_NoPI".

2. Populate the Orders_NoPI table (via INSERT/SELECT) with all of the rows from the DS.Orders and DS.Orders_2012 tables.

   Verify the number of rows in your table. Count = _____ (count should be 43,200)

3. Use the SHOW TABLE for Orders_NoPI to create a new column partitioned table named "Orders_CP" based on the following:

   - Each column is created as a separate partition
   - Utilize auto compression for every column

   Populate the Orders_CP table (via INSERT/SELECT) from the Orders_NoPI table.

Lab Exercise 18-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

SQL hints:

INSERT INTO *table_1* SELECT * FROM *table_2*;

SELECT COUNT(*) FROM *table_name*;

SELECT COUNT(DISTINCT(PARTITION)) FROM *table_name*;

Lab Exercise 18-1 (cont.)

4. Verify the number of rows in your table. Count = _____ (count should be 43,200)

   How many partitions actually have data? _____

   Note: The table only has 1 logical partition.

5. Use the SHOW TABLE for Order_CP to create a new column partitioned table named "Orders_CP_noAC" based on the following:

   - Each column is created as a separate partition
   - Turn off auto compression for every column

   Populate the Orders_CP_noAC table (via INSERT/SELECT) from the Orders_NoPI table.

Lab Exercise 18-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

SQL hints:

INSERT INTO *table_1* SELECT * FROM *table_2*;

SELECT COUNT(*) FROM *table_name*;

SELECT COUNT(DISTINCT(PARTITION)) FROM *table_name*;

```
SELECT      TableName, SUM(CurrentPerm)
FROM        DBC.TableSizeV
WHERE       DatabaseName = DATABASE
AND         TableName In ('tablename1', 'tablename2', …)
GROUP BY    1
ORDER BY    1;
```

Lab Exercise 18-1 (cont.)

6. **(Optional) Use the SHOW TABLE for Order_CP to create a new column partitioned table named "Orders_CP_TP based on the following:**

   - **Each column is created as a separate partition (COLUMN partitioning is the first level)**
   - **Utilize auto compression for every column**
   - **Incorporate table partitioning with orderdate as the partitioning column**
     - **From '2003-01-01' through '2012-12-31', partition by month**
     - **Do not use the NO RANGE or UNKNOWN options.**

   Populate the Orders_CP_TP table (via INSERT/SELECT) from the Orders_NoPI table.

7. **(Optional) Use the PARTITION key word to determine the number of partitions defined in the Orders_CP_TP.**

   How many partitions actually have data? _____

8. **(Optional) Determine the AMP space usage of the Orders_CP, Orders_CP_noAC, and Orders_CP_TP tables using DBC.TableSizeV.**

   CurrentPerm of Orders_CP _____

   CurrentPerm of Orders_CP_noAC _____

   CurrentPerm of Orders_CP_TP _____

Notes

Module 19

Secondary Index Usage

After completing this module, you will be able to:

- **Describe USI and NUSI implementations.**
- **Describe dual NUSI access.**
- **Describe NUSI bit mapping.**
- **Explain NUSI and Aggregate processing.**
- **Compare NUSI vs. full table scan (FTS).**

Notes

Table of Contents

Secondary Indexes 19-4
Defining Secondary Indexes 19-6
Secondary Index Subtables 19-8
  Primary Indexes (UPIs and NUPIs) 19-8
  Unique Secondary Indexes (USIs) 19-8
  Non-Unique Secondary Indexes (NUSIs) 19-8
USI Subtable General Row Layout 19-10
  USI Change for PPI 19-10
USI Hash Mapping 19-12
NUSI Subtable General Row Layout 19-14
  NUSI Change for PPI 19-14
NUSI Hash Mapping 19-16
Table Access – A Complete Example 19-18
Secondary Index Considerations 19-20
Single NUSI Access (Between, Less Than, or Greater Than) 19-22
Dual NUSI Access 19-24
  AND with Equality Conditions 19-24
  OR with Equality Conditions 19-24
NUSI Bit Mapping 19-26
  Example 19-26
Value-Ordered NUSIs 19-28
Value-Ordered NUSIs (cont.) 19-30
Covering Indexes 19-32
    Join Index Note: 19-32
    Example 19-32
Covering Indexes (cont.) 19-34
  NUSIs and Aggregate Processing 19-34
  Example 19-34
NUSI vs. Full Table Scan (FTS) 19-36
  Example 19-36
Full Table Scans – Sync Scans 19-38
Module 19: Review Questions 19-40

Secondary Indexes

Secondary Indexes are generally defined to provide faster set selection. The Teradata Database allows up to 32 Secondary Indexes per table. Teradata handles Unique Secondary Indexes (USIs) and Non-Unique Secondary Indexes (NUSIs) very differently.

The diagram illustrates how Secondary Index values are stored in subtables. Secondary Index values, like Primary Index values, are input to the Hashing Algorithm. As with Primary Indexes, the Hashing Algorithm takes the Secondary Index value and outputs a Row Hash. These Row Hash values point to a subtable which stores index rows containing the base table SI column values and Row IDs which point to the row(s) in the base table with the corresponding SI value.

The Teradata Database can determine the difference between a base table and a SI subtable by checking the Subtable ID, which is part of the Table ID.

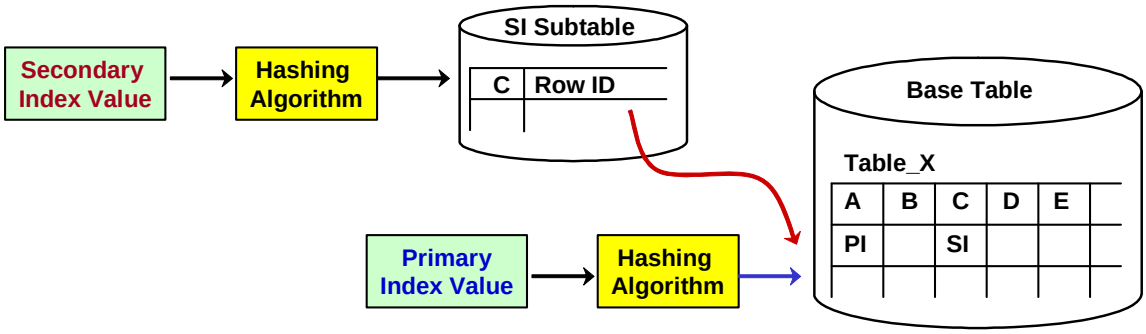
Secondary Indexes

Secondary indexes provide faster set selection.

- They may be unique (USI) or non-unique (NUSI).
- A USI may be used to maintain uniqueness on a column.
- The system maintains a separate subtable for each secondary index.
- A secondary index can consist of 1 to 64 columns.

Subtables keep base table secondary index row hash, column values, and RowID (which point to the row(s) in the base table with that value).

- The implementation of a USI is different than a NUSI.
- Users cannot access subtables directly.



Defining Secondary Indexes

Use the CREATE INDEX statement to create a secondary index on an existing table or join index. The index can be optionally named.

Notes on ORDER BY option:

- If the ORDER BY option is not used, the default is to order by hash.
- If the ORDER BY option is specified and neither of the keywords (HASH or VALUES) is specified, then the default is to order by values.

Recommendation: If the ORDER BY option is used, specify one of the keywords – HASH or VALUES.

Notes on the ALL option:

- The ALL option indicates that a NUSI should retain row ID pointers for each logical row of a **join index** (as opposed to only the compressed physical rows).
- ALL also ignores the NOT CASESPECIFIC attribute of data types so a NUSI can include case-specific values.
- ALL enables a NUSI to cover a join index, enhancing performance by eliminating the need to access a join index when all values needed by a query are in the secondary index. However, ALL might also require the use of additional index storage space.
- Use this keyword when a NUSI is being defined for a join index and you want to make it eligible for the Optimizer to select when covering reduces access plan cost. ALL can also be used for an index on a table, however.
- You cannot specify multiple indexes that differ only by the presence or absence of the ALL option.
- The use of the ALL option for a NUSI on a data table does not cause a syntax error.

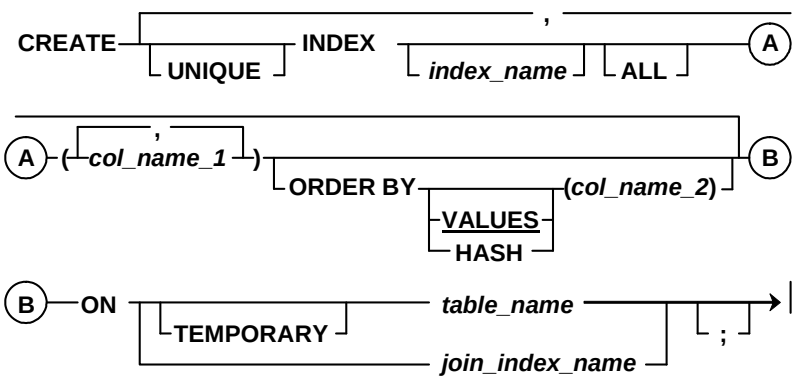
Additional notes:

- *column_name_2* specifies the sort order to be used. *column_name_2* is a column name that must appear in the *column_name_1* list.
- You can put two NUSI secondary indexes on the same column (or set of columns) if one of the indexes is ordered by hash and the other index is ordered by value.
- You cannot define a USI on a join index. Other secondary indexes are allowed.

Defining Secondary Indexes

Secondary indexes can be defined …

- **when a table is created (CREATE TABLE).**
- **for an existing table (CREATE INDEX).**



Examples:

Unnamed USI:

```
CREATE UNIQUE INDEX
        (item_id, store_id, sales_date)
ON Daily_Sales;
```

Named Value-Ordered NUSI:

```
CREATE INDEX ds_vo_nusi
     (sales_date)
ORDER BY VALUES ON Daily_Sales;
```

Secondary Index Subtables

The diagram on the facing page illustrates how the Teradata Database retrieves rows based upon their index values. It compares and contrasts examples of Primary (UPIs and NUPIs), Unique Secondary (USIs) and Non-Unique Secondary Indexes (NUSIs).

Primary Indexes (UPIs and NUPIs)

As you have seen previously, in the case of a Primary Index, the Teradata Database hashes the value and uses the Row Hash to find the desired row. This is always a one-AMP operation and is shown in the top diagram on the facing page.

Unique Secondary Indexes (USIs)

The middle diagram illustrates the process of a USI row retrieval. An index subtable contains index rows, which in turn point to base table rows matching the supplied index value. USI rows are globally hash- distributed across all AMPs, and are retrieved using the same procedure for Primary Index data row retrieval. Since the USI row is hash-distributed on different columns than the Primary Index of the base table, the USI row typically lands on an AMP other than the one containing the data row. Once the USI row is located, it "points" to the corresponding data row. This requires a second access and usually involves a different AMP. In effect, a USI retrieval is like two PI retrievals:

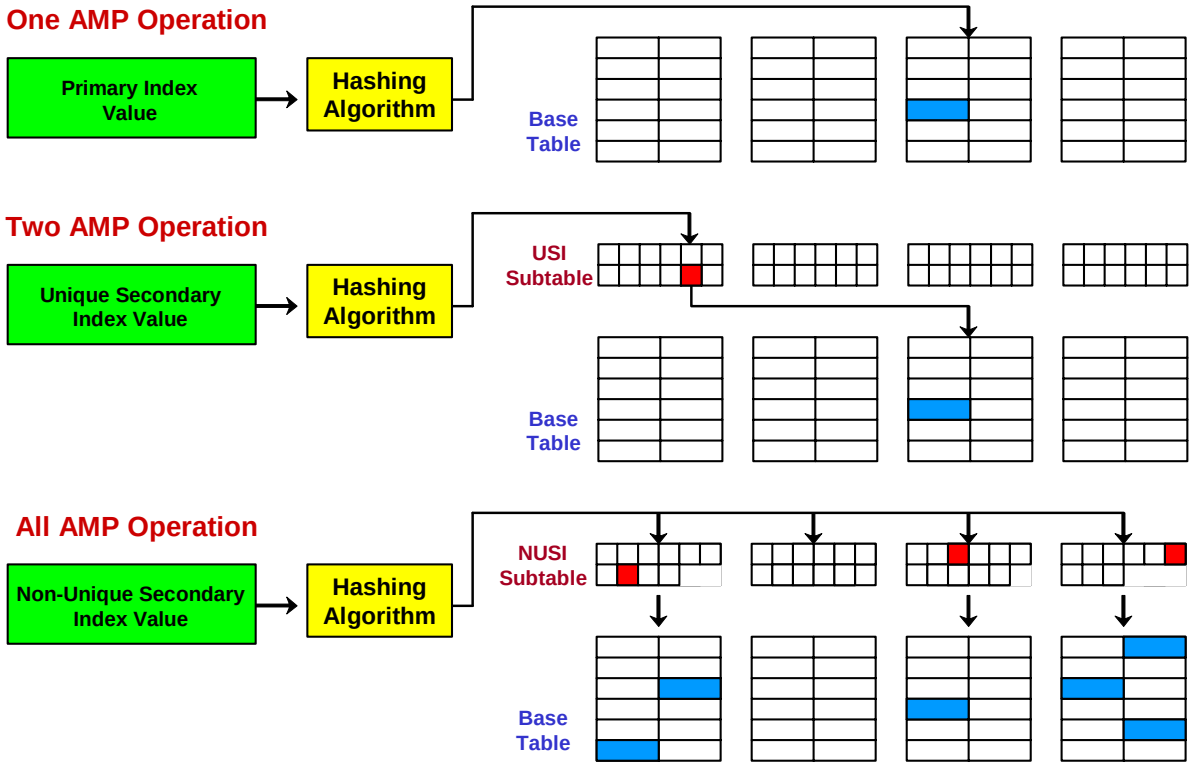
Master Index - Cylinder Index - Index Block
Master Index - Cylinder Index - Data Block

Non-Unique Secondary Indexes (NUSIs)

NUSIs are implemented on an AMP-local basis. Each AMP is responsible for maintaining only those NUSI subtable rows that correspond to base table rows located on that AMP. Since NUSIs allow duplicate index values and are based on different columns than the PI, data rows matching the supplied NUSI value could appear on any AMP.

In a NUSI retrieval (illustrated at the bottom of the facing page), a message is sent to all AMPs to see if they have an index row for the supplied value. Those that do use the "pointers" in the index row to access their corresponding base table rows. Any AMP that does not have an index row for the NUSI value will not access the base table to extract rows.

Secondary Index Subtables



USI Subtable General Row Layout

The layout of a USI subtable row is illustrated at the top of the facing page. It is composed of several sections:

- The first two bytes designate the **row length**.
- The next 8 bytes contain the **Row ID of the row**. Within this Row ID, there are 4 bytes of Row Hash and 4 bytes of Uniqueness Value.
- The following 2 bytes are **additional system bytes** that will be explained later as will be the 7 bytes for row offsets.
- The next section contains the **SI value**. This is the value that was used by the Hashing Algorithm to generate the Row Hash for this row. This section varies in length depending on the index.
- Following the SI value are 8 bytes containing the **Row ID of the base table row**. The base table Row ID tells the system where the row corresponding to this particular USI value is located.

  If the table is partitioned, then the USI subtable row needs 10 or 16 bytes to identify the **Row ID of the base table row.** The Row ID (of the base table row) is combination of the Partition Number, Row Hash, and Uniqueness Value.

- The last two bytes contain the **reference array pointer** at the end of the block.

The Teradata Database creates one index subtable row for each base table row.

USI Change for PPI

For tables defined with a PPI, a two-byte or optionally eight-byte (TD 14.0) partition number is embedded in the data row. Therefore, the unique row identifier is comprised of the Partition Number, the Row Hash, and the Uniqueness Value.

The USI subtable rows use the wider row identifier to identify the base table row, making these subtable rows wider as well. Except for the embedded partition number, USI subtable rows (for a PPI table) have the same format as non-PPI rows.

The facing page shows the row layout for USI subtable rows.

USI Subtable General Row Layout

USI Subtable Row Layout

Row Length	Row ID of USI				Secondary Index Value	Base Table Row Identifier			Ref. Array Pointer
	Row Hash	Uniq. Value				Part. #	Row Hash	Uniq. Value	
	4	4				2 or 8	4	4	

Notes:

- **USI subtable rows are distributed by the Row Hash, like any other row.**
- **The Row Hash is based on the unique secondary index value.**
- **The subtable row includes the secondary index value and a second Row ID which identifies a single base table row (usually on a different AMP).**
- **There is one index subtable row for each base table row.**
- **For PPI tables, a two-byte (or optionally eight-byte with Teradata 14.0) partition number is embedded in the base table row identifier.**
  - **Therefore, the base table row identifier is comprised of the Partition Number, Row Hash, and the Uniqueness Value.**

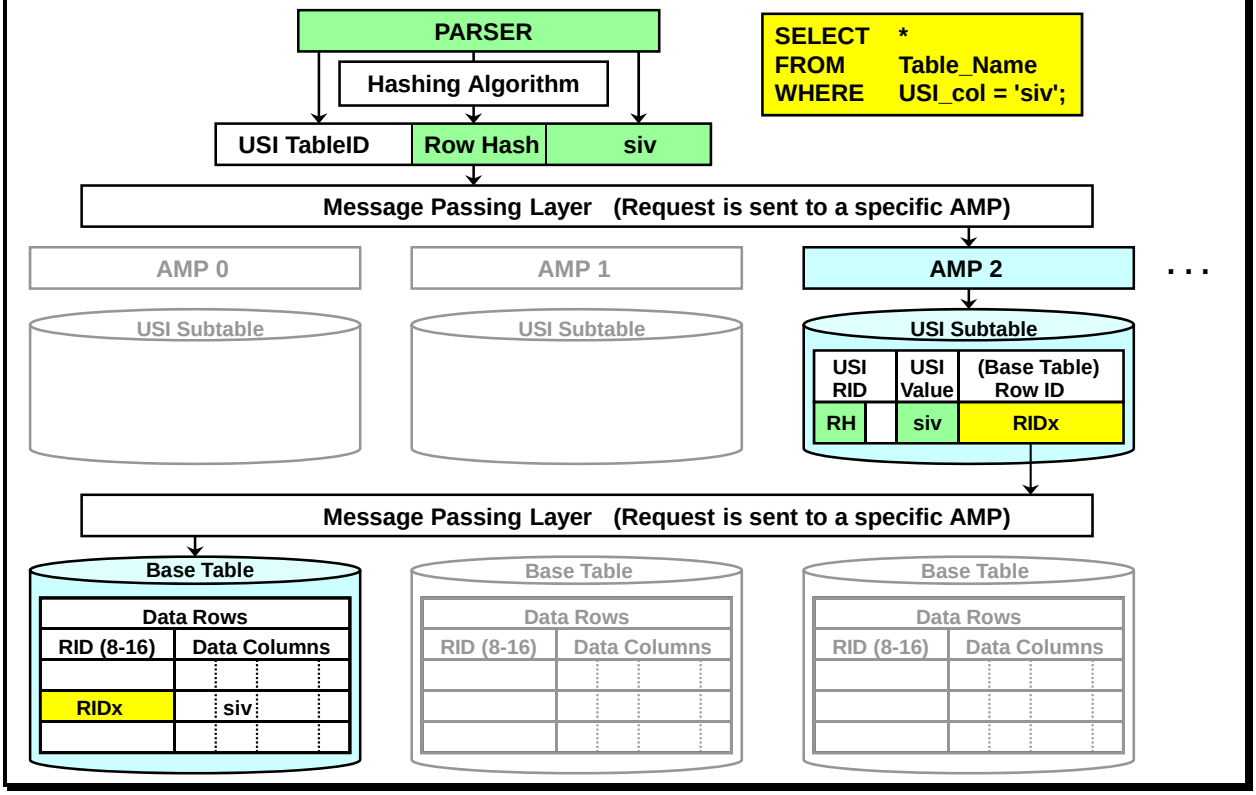
USI Hash Mapping

The example on the facing page shows the three-part message that is put onto the Message Passing Layer for USI access.

- The only difference between this and the three-part message used in PI access (previously discussed) is that the Subtable ID portion of the Table ID references the USI subtable not the data table. Using the DSW for the Row Hash, the Message Passing Layer (a.k.a., Communication Layer) directs the message to the correct AMP which uses the Table ID and Row Hash as a logical index block identifier and the Row Hash and USI value as the logical index row identifier. If the AMP succeeds in locating the index row, it extracts the base table Row ID ("pointer"). The Subtable ID portion of the Table ID is then modified to refer to the base table and a new three-part message is put onto the Communications Layer.

- Once again, the Message Passing Layer uses the DSW to identify the correct AMP. That AMP uses the Table ID and Row ID to locate the correct data block and then uses the Row ID to locate the correct row.

USI Hash Mapping



NUSI Subtable General Row Layout

The layout of a **NUSI subtable row** is illustrated on the facing page. It is almost identical to the layout of a USI subtable row. There are, however, two major differences:

- First, NUSI subtable rows are not distributed across the system via AMP number in the Hash Map. NUSI subtable rows are built from the base table rows found on that particular AMP and refer only to the base rows of that AMP.

- Second, NUSI rows may point to or reference more than one base table row. There can be many base table Row IDs (8, 10, or 16 bytes) in a NUSI subtable row. Because NUSIs are always AMP-local to the base table rows, it is possible to have the same NUSI value represented on multiple AMPs.

A NUSI subtable is just another table from the perspective of the file system.

NUSI Change for PPI

For tables defined with a PPI, the two-byte partition number is embedded in the data row. Therefore, the unique row identifier is comprised of the Partition Number, Row Hash, and Uniqueness Value. PPI data rows are two bytes wider than they would be if the table was not partitioned.

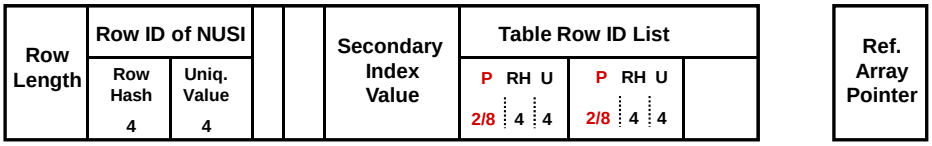
If the base table is partitioned, then the NUSI subtable row needs 10 or 16 bytes for each RowID entry to identify the **Row ID of the base table row.** The Row ID (of the base table row) is combination of the Partition Number, Row Hash, and Uniqueness Value.

The NUSI subtable rows use the wider row identifier to identify the base table row, making these subtable rows wider as well. Except for the embedded partition number, NUSI subtable rows (for a PPI table) have the same format as non-PPI rows.

The facing page shows the row layout for NUSI subtable rows.

NUSI Subtable General Row Layout

NUSI Subtable Row Layout



Notes:

- The Row Hash is based on the base table secondary index value.
- The NUSI subtable row contains Row IDs that identify the base table rows on this AMP that carry the Secondary Index Value.
- The Row IDs reference (or "point") to base table rows on this AMP only.
- There are one (or more) subtable rows for each secondary index value on the AMP.
  - One NUSI subtable row can hold approximately 4000 – 8000 Row IDs; assuming a NUSI data type less than 200 characters (CHAR(200)).
  - If an AMP has more than 4000 – 8000 rows with the same NUSI value, another NUSI subtable row is created for the same NUSI value.
- The maximum size of a single NUSI row is 64 KB.

NUSI Hash Mapping

The example on the facing page shows the standard, three-part Message Passing Layer row-access message. Because NUSIs are AMP-local indexes, this message gets broadcast to all AMPs. Each AMP uses the values to search the appropriate index block for a corresponding NUSI row. Only those AMPs with one or more of the desired rows use the base table Row IDs to access the proper data blocks and data rows.

In the example, the SELECT statement is designed to find those rows with a NUSI value of **'siv'**. Examination of the NUSI subtables on each AMP shows that AMPs 0, 2 and 3 (not shown) all have a subtable index row, and, therefore, base table rows satisfying this condition. These AMPs then participate in the base table access. The NUSI subtable on AMP 1, on the other hand, shows that there are no rows with a NUSI value of **'siv'** located on this AMP. AMP 1 does not participate in the base table access process.

If the table is not partitioned, the subtable rows will identify the 8-byte Row IDs of the base table rows.

If the table is partitioned with less than (or equal) 65,535 partitions, the subtable rows will identify the 10-byte Row IDs of the base table rows. This Row ID includes the Partition Number.

If the table is partitioned with more than 65,535 partitions, the subtable rows will identify the 16-byte Row IDs of the base table rows. This Row ID includes the Partition Number.

NUSI Hash Mapping

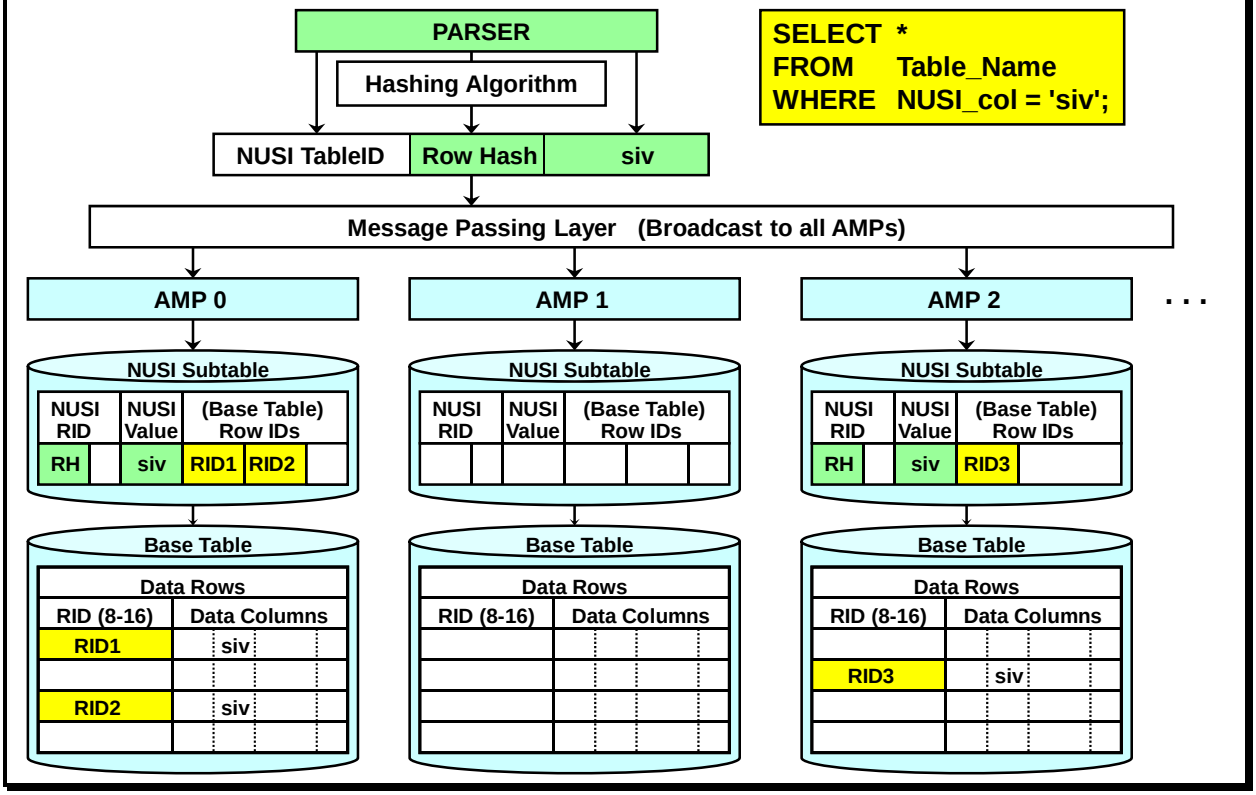


Table Access – A Complete Example

The example on the facing page shows a four-AMP configuration with Base Table Rows, NUSI Subtable rows, and USI Subtable Rows. The table and index can be used to answer the following queries without having to do a full table scan:

```
SELECT * FROM Customer  WHERE  Phone = '666-5555' ;

SELECT * FROM Customer  WHERE  Cust = 80;

SELECT * FROM Customer  WHERE  Name = 'Rice' ;
```

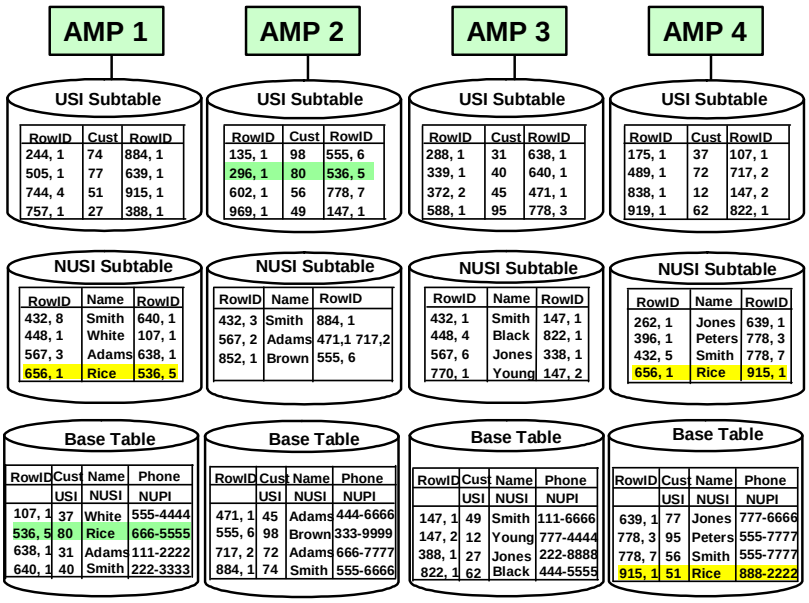
Table Access – A Complete Example

CUSTOMER

Cust	Name	Phone
USI	NUSI	NUPI
37	White	555-4444
98	Brown	333-9999
74	Smith	555-6666
95	Peters	555-7777
27	Jones	222-8888
56	Smith	555-7777
45	Adams	444-6666
31	Adams	111-2222
40	Smith	222-3333
72	Adams	666-7777
80	Rice	666-5555
49	Smith	111-6666
12	Young	777-7777
62	Black	444-5555
77	Jones	777-6666
51	Rice	888-2222

Example:

```
SELECT * FROM Customer  WHERE  Phone = '666-5555' ;
SELECT * FROM Customer  WHERE  Cust = 80;
SELECT * FROM Customer  WHERE  Name = 'Rice' ;
```



Secondary Index Considerations

As mentioned at the beginning of this module, a table may have up to 32 Secondary Indexes that can be created and dropped dynamically. It is probably not a good idea to create 32 SIs for each table just to speed up set selection because SIs consume the following extra resources:

- SIs require additional storage to hold their subtables. In the case of a Fallback table, the SI subtables are Fallback also. Twice the additional storage space is required.

- SIs require additional I/O to maintain these subtables.

When deciding whether or not to define a NUSI, there other considerations. The Optimizer may choose to do a Full Table Scan rather than utilize the NUSI in two cases:

- When the NUSI is not selective enough.

- When no COLLECTed STATISTICS are available.

As a guideline, choose only those rows having frequent access as NUSI candidates. After the table has been loaded, create the NUSI indexes, COLLECT STATISTICS on the indexes, and then do an EXPLAIN referencing each NUSI. If the Parser chooses a Full Table Scan over using the NUSI, drop the index.

Secondary Index Considerations

- A table may have up to 32 secondary indexes.
- Secondary indexes may be created and dropped dynamically.
  - They require additional storage space for their subtables.
  - They require additional I/Os to maintain their subtables.
- If the base table is Fallback, the secondary index subtable is Fallback as well.
- The Optimizer may, or may not, use a NUSI, depending on its selectivity.
- Without COLLECTed STATISTICS, the Optimizer often does a FTS.
- The following approach is recommended:
  - Create the index.
  - COLLECT STATISTICS on the index (or column).
  - Use EXPLAIN to see if the index is being used.

Single NUSI Access (Between, Less Than, or Greater Than)

The Teradata Database accesses data from a NUSI-defined column in three ways:

- If the NUSI is not ordered by value, utilize the NUSI and do a Full Table Scan (FTS) of the NUSI subtable. In this case, the Row IDs of the qualifying base table rows would be retrieved into spool. The Teradata Database would use those Row IDs in spool to access the base table rows themselves.

- If the NUSI is ordered by values, the NUSI subtable may be used to locate matching base table rows.

- Ignore the NUSI and do an FTS of the base table itself.

In order to make this decision, the Optimizer requires COLLECTed STATISTICS.

– REMEMBER –
The only way to determine for certain whether an index is being used is to utilize the EXPLAIN facility.

Single NUSI Access
(Between, Less Than, or Greater Than)

If the NUSI is not value-ordered, the system may do a FTS of the NUSI subtable.

- Retrieve Row IDs of qualifying base table rows into spool.
- Access the base table rows from the spooled Row IDs.

The Optimizer requires COLLECTed STATISTICS to make this choice.

- `CREATE INDEX (hire_date) ON Employee;`
- ```
  SELECT last_name, first_name, hire_date
  FROM Employee
  WHERE hire_date BETWEEN DATE '2012-01-01' AND DATE '2012-12-31';
  ```
- ```
  SELECT last_name, first_name, hire_date
  FROM Employee
  WHERE hire_date < DATE '2012-01-01';
  ```
- ```
  SELECT last_name, first_name, hire_date
  FROM Employee
  WHERE hire_date > DATE '1999-12-31';
  ```

If the NUSI is ordered by values, the NUSI subtable is much more likely be used to locate matching base table rows.

Use EXPLAIN to see if, and how, indexes are being used.
  ```

Dual NUSI Access

In the example on the facing page, two NUSIs are CREATEd on separate columns of the EMPLOYEE TABLE. The Teradata Database decides how to use these NUSIs based on their selectivity.

AND with Equality Conditions

- If one of the two indexes is strongly selective, the system uses it alone for access.
- If both indexes are weakly selective, but together they are strongly selective, the system does a bit-map intersection.
- If both indexes are weakly selective separately and together, the system does an FTS.

In any case, any conditions in the SQL statement not used for access (residual conditions) become row qualifiers.

OR with Equality Conditions

When accessing data with two NUSI equality conditions joined by the OR operator (as shown in the last example on the facing page), the Teradata Database may do one of the following:

- Do a FTS of the base table.
- If each of the NUSIs is strongly selective, it may use each of the NUSIs to return the appropriate rows.
- Do an FTS of the two NUSI subtables and do the following steps.
  - Retrieve Rows IDs of qualifying base table rows into two separate spools.
  - Eliminate duplicates from the two spools of Row IDs.
  - Access the base rows from the resulting spool of Row IDs.

If only one of the two columns joined by the OR is indexed, the Teradata Database always does an FTS of the base tables.

Dual NUSI Access

Each column is a separate NUSI:

```
CREATE INDEX  (department_number) ON Employee;
CREATE INDEX  (job_code)          ON Employee;
```

AND with Equality Conditions:

```
SELECT   last_name, first_name, …
FROM     Employee
WHERE    department_number = 500
AND      job_code = 2147;
```

Optimizer options with AND:

- Use one of the two indexes if it is strongly selective.
- If the two indexes together are strongly selective, optionally do a bit-map intersection.
- If both indexes are weakly selective separately and together, the system does a FTS.

OR with Equality Conditions:

```
SELECT   last_name, first_name, ...
FROM     Employee
WHERE    department_number = 500
OR       job_code = 2147;
```

Optimizer options with OR:

- Do a FTS of the base table.
- If each of the NUSIs is strongly selective, it may use each of the NUSIs to return the appropriate rows.
- Do a FTS of the two NUSI subtables and retrieve Rows IDs of qualifying rows into spool and eliminate duplicate Row IDs from spool.

NUSI Bit Mapping

NUSI Bit Mapping is a process that determines common Row IDs between multiple NUSI values by a process of intersection:

- It is much faster than copying, sorting and comparing the Row ID lists.
- It dramatically reduces the number of base table I/Os.

NUSI bit mapping can be used with conditions other than equality if all of the following conditions are satisfied:

- All conditions must be linked by the AND operator.
- At least two NUSI equality conditions must be specified.
- The Optimizer is more likely to consider if you have COLLECTed STATISTICS on the NUSIs.

Even when the above conditions are satisfied, the only way to be absolutely certain that NUSI bit mapping is occurring is to use the EXPLAIN facility.

Example

The SQL statement and diagram on the facing page show how NUSI bit-map intersections can narrow down the number of rows even though each condition is weakly selective.

In this example, the designer wants to access rows from the employee table. There are three NUSIs defined: salary_amount, country_code, and job_code. All three of these NUSIs are weakly selective. You can see that 7% of the employees earn more than $75,000 per year (>75000), 40% of the employees are located in the USA, and 12% of the employees have a job code of IT.

In this case, the bit map intersection of these three NUSIs has an aggregate selectivity of .3%. That is, only .3% of the employees satisfy all three conditions: earning over $75,000, USA based, and work in IT.

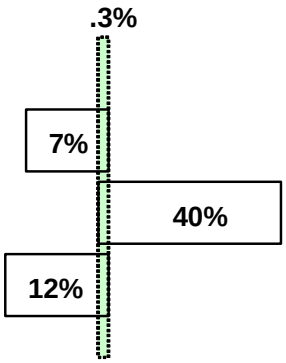
NUSI Bit Mapping

- Determines common Row IDs between multiple NUSI values.
- Faster than copying, sorting, and comparing the Row ID lists.
- Dramatically reduces the number of base table I/Os.
- All NUSI conditions must be linked by the AND operator.
- The Optimizer is much more likely to consider bit mapping if you COLLECT STATISTICS.
- Use EXPLAIN to see if bit mapping is being used.
- Requires at least 2 NUSI equality conditions.

```
SELECT  *
FROM    Employee
WHERE   salary_amount > 75000

AND     country_code = 'USA'

AND     job_code = 'IT';
```



Value-Ordered NUSIs

NUSIs are maintained as separate subtables on each AMP. Their index entries point to base table or Join Index rows residing on the same AMP as the index. The row hash for NUSI rows is based on the secondary index column(s). Unlike row hash values for base table rows, this row hash does not determine the distribution of subtable rows; only the local sort order of each subtable.

Enhancements have been made to support the user-specified option of sorting the index rows by data value rather than by hash code. This is referred to as "value ordered" indexes and is presented to the user in the form of new syntax options in the CREATE INDEX statement.

By using the “value-ordered” indexes feature, this option can be specified to sort the index rows by data value rather than by hash code.

The typical use of a **hash-ordered** NUSI is with an *equality* condition on the secondary index column(s). The specified secondary index value is hashed and then each NUSI subtable is probed for rows with the same row hash. For each matching NUSI entry, the corresponding Row IDs are used to access the base rows on the same AMP. Because the NUSI rows are stored in row hash order, searching the NUSI subtable for a particular row hash is very efficient.

Value-ordered NUSIs, on the other hand, are useful for processing range conditions and conditions with an inequality on the secondary index column set.

Although hash-ordered NUSIs can be selected by the Optimizer to access rows for range conditions, a far more common response is to specify a full table scan of the NUSI subtable to find the matching secondary key values. Therefore, depending on the size of the NUSI subtable, this might not be very efficient.

By sorting the NUSI rows by data value, it is possible to search only a portion of the index subtable for a given range of key values. The major advantage of a value-ordered NUSI is in the performance of range queries.

Value-ordered NUSIs have the following limitations.

The sort key is limited to a single numeric column.
The sort key column must be four or fewer bytes.

The following query is an example of the sort of SELECT statement for which value-ordered NUSIs were designed.

```
SELECT     *
FROM       Orders
WHERE      orderdate
BETWEEN    DATE '2012-02-01' AND DATE '2012-02-29';
```

Value-Ordered NUSIs

A Value-Ordered NUSI is limited to a single column numeric (4-byte) value.

Some benefits of using value-ordered NUSIs:

- **Index subtable rows are sorted (sequenced) by data value rather than hash value.**
- **Optimizer can search only a portion of the index subtable for a given range of values.**
- **Can provide major advantages in performance of range queries.**
- **Even with PPI, the Value-Ordered NUSI is still a valuable index selection for other columns in a table.**

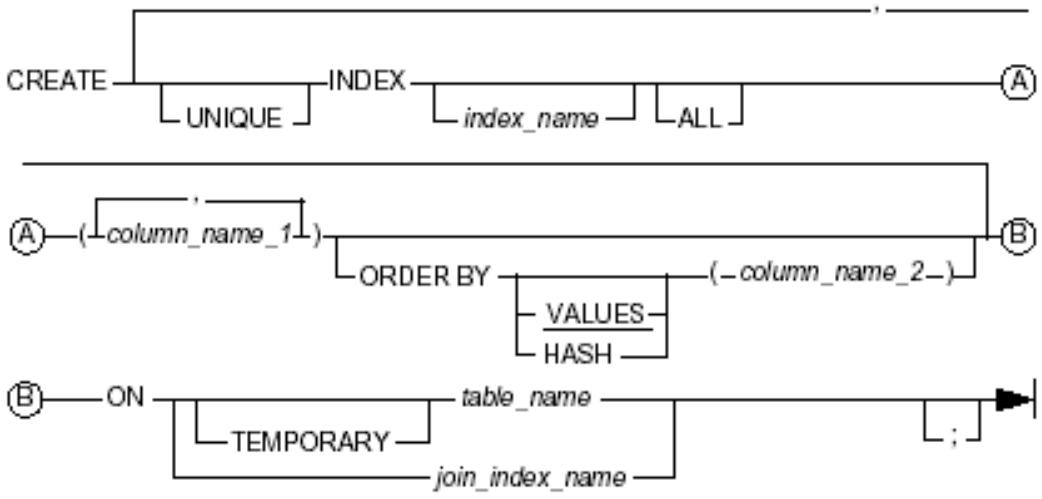
Example of creating a Value-Ordered NUSI:

```
CREATE INDEX    (sales_date)
                ORDER BY VALUES (sales_date)
                ON Daily_Sales;

SELECT           sales_date
                ,SUM (sales)
FROM            Daily_Sales
WHERE           sales_date
BETWEEN         DATE '2012-02-09' AND DATE '2012-02-15'
GROUP BY        1
ORDER BY        1 ;
```

→ The optimizer may choose to transverse the NUSI using a range constraint rather than do a FTS.

Value-Ordered NUSIs (cont.)



column_1_name	The names of one or more columns whose field values are to be indexed. You can specify up to 64 columns for the new index. The index is based on the combined values of each column. Unless you use the ORDER BY clause, all columns are hash-ordered. Multiple indexes can be defined on the same columns as long as each index differs in its ordering option (VALUES versus HASH).
ORDER BY	Row ordering on each AMP by a single NUSI column: either value-ordered or hash-ordered. Rules for using an ORDER BY clause are shown in the following table.
VALUES	Value-ordering for the ORDER BY column. Select VALUES to optimize queries that return a contiguous range of values, especially for a covered index or a nested join.
HASH	Hash-ordering for the ORDER BY column. Select HASH to limit hash-ordering to one column, rather than all columns (the default). Hash-ordering a multi-column NUSI on one of its columns allows the NUSI to participate in a nested join where join conditions involve only that ordering column.

Note: A Value-Ordered NUSI actually reserves two subtable IDs and this counts as 2 secondary indexes in the maximum count of 32 for a table.

Value-Ordered NUSIs (cont.)

- Option that increases the ability of a NUSI to “cover” SQL queries without having to access the base table.
- Value-Ordered is sorted by the ‘ORDER BY VALUES’ clause and the sort column is limited to a single numeric column that cannot exceed 4 bytes.
  - Value-Ordered is useful for range constraint queries.
- The ‘ORDER BY HASH’ clause provides the ability to create a multi-valued index, but have the NUSI hashed based on a single attribute within the index, not the entire composite value.
  - Hash-Ordered is useful for equality searches based on a single attribute.
  - Example: A NUSI may contain 10 columns for covering purposes and a single value 'ORDER BY HASH' for equality searches on that NUSI value.
- Optimizer is much more likely to use a value-ordered NUSI if you have collected statistics on the value-ordered NUSI.

Covering Indexes

If the query references only columns of that table that are fully contained within a given index, the index is said to "cover" the table in the query. In these cases, it is often more efficient to access only the index subtable and avoid accessing the base table rows altogether.

Covering will be considered for any table in the query that references only columns defined in a given NUSI. These columns can be specified anywhere in the query including the:

- SELECT list
- WHERE clause
- Aggregate functions
- GROUP BY expressions

The presence of a WHERE condition on each indexed column is not a prerequisite for using the index to cover the query. The optimizer will consider the legality and cost of covering versus other alternative access paths and choose the optimal plan. Many of the potential performance gains from index covering require no user intervention and will be transparent except for the execution plan returned by EXPLAIN.

Join Index Note:

This course hasn't covered Join Indexes to this point, but it is possible to create a NUSI on top of a Join Index. The CREATE INDEX has a special option of ALL which is required if these columns will be potentially used for covering.

The class of indexed data that will require user intervention to take advantage of covering is NUSIs, which may be defined on a Join Index. By default, a NUSI defined on a Join Index will maintain RowID pointers to only physical rows. In order to use the NUSI to cover the data stored in a Join Index, Row IDs must be kept for each associated logical row. As a result, when defining a potential covering NUSI on top of a Join Index, users should specify the ALL option to indicate the NUSI rows should point to logical rows.

Example

Defining a NUSI on top of a Join Index

```
CREATE JOIN INDEX OrdCustIdx as
        SELECT (custkey, custname), (orderstatus, orderdate, ordercomment)
        FROM    Orders O LEFT JOIN Customer C ON O.custkey = C.custkey
        ORDER BY custkey
        PRIMARY INDEX (custname);

CREATE INDEX idx_name_stat ALL (custname, orderstatus) on OrdCustIdx;
```

Covering Indexes

- The optimizer considers using a NUSI subtable to "cover" any query that references only columns defined in a given NUSI.
- These columns can be specified anywhere in the query including:
  - SELECT list
  - WHERE clause
  - Aggregate functions
  - GROUP BY clauses
  - Expressions
- Presence of a WHERE condition on each indexed column is not a prerequisite for using the index to cover the query.
- NUSIs (especially a covering NUSI) are considered by the optimizer in join plans and can be joined to other tables in the system.

Query considered for index covering:

```
CREATE INDEX IdxOrd
    (orderkey, orderdate, totalprice)
ON Orders ;
```

Query considered for index covering and ordering:

```
CREATE INDEX IdxOrd2
    (orderkey, orderdate, totalprice)
    ORDER BY VALUES (orderkey)
ON Orders ;
```

Query to access the table via the OrderKey.

```
SELECT    orderdate, AVG(totalprice)
FROM      Orders
WHERE     orderkey >1000
GROUP BY  orderdate ;
```

Covering Indexes (cont.)

NUSIs and Aggregate Processing

When aggregation is performed on a NUSI column, the Optimizer accesses the NUSI subtable that offers much better performance than accessing the base table rows. Better performance is achieved because there should be fewer index blocks and rows in the subtable than data blocks and rows in the base table, thus requiring less I/O.

Example

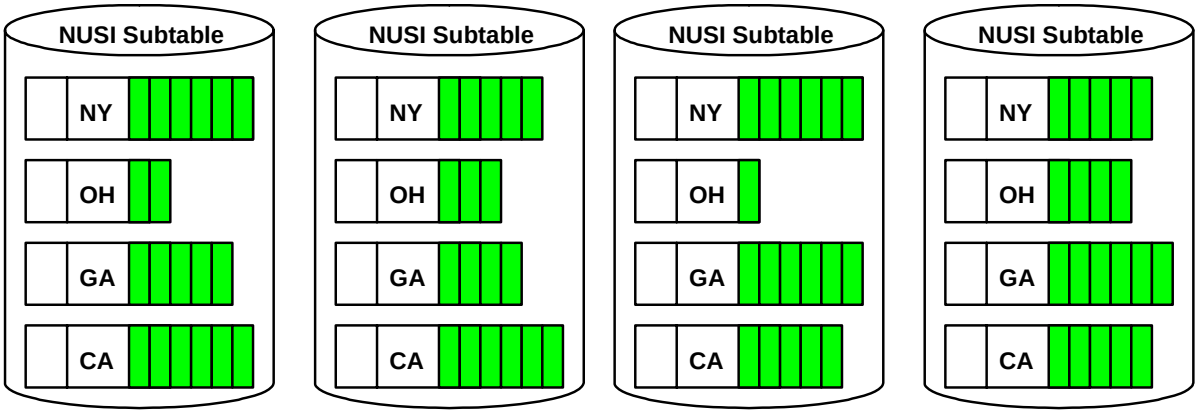
In the example on the facing page, there is a NUSI defined on the state column of the location table. Aggregate processing of this NUSI column produces much faster results for the SELECT statement, which counts the number of rows for each state.

Covering Indexes (cont.)

- The Optimizer uses NUSI subtables for aggregation when possible.
- If the aggregated column is a NUSI, subtable access may be sufficient.
- The system counts Row ID List entries for each AMP for each value.
- Also referred to as a "Covered NUSI".

```
SELECT    COUNT (*), state
FROM      Location
GROUP BY  state;
```

= subtable Row ID



NUSI vs. Full Table Scan (FTS)

The Optimizer generally chooses an FTS over a NUSI when one of the following occurs:

- Rows per value is greater than data blocks per AMP.
- It does not have COLLECTed STATISTICS on the NUSI.
- The index is too weakly selective. The Optimizer determines this by using COLLECTed STATISTICS.

Example

The table on the facing page shows how the access method chosen affects the number of physical I/Os per AMP.

In the case of a NUSI, there is ONE I/O necessary to read the Index Block on each AMP plus 0-ALL (where ALL = Number of Data Blocks) I/Os required to read the Data Blocks for a possible total ranging from the Number of AMPs - (Number of AMPs + ALL) I/Os.

In the case of a Full Table Scan, there are no I/Os required to read any Index Blocks, but the system reads ALL Data Blocks.

The only way to tell whether or not a NUSI is being used is by using EXPLAIN.

COLLECT STATISTICS on all NUSIs.
Use EXPLAIN to see whether a NUSI is being used.
Do not define NUSIs that will not be used.

NUSI vs. Full Table Scan (FTS)

The Optimizer generally chooses a FTS over a NUSI when:

- It does not have COLLECTed STATISTICS on the NUSI.
- The index is too weakly selective.
- Small tables.

Access Method	Physical I/Os per AMP	
NUSI	1 0 – Many	Index Subtable Block(s) Data Blocks
Full Table Scan	0 ALL	Index Subtable Blocks Data Blocks

General Rules:

- COLLECT STATISTICS on all NUSIs.
- USE EXPLAIN to see whether a NUSI is being used.
- Do not define NUSIs that will not be used.

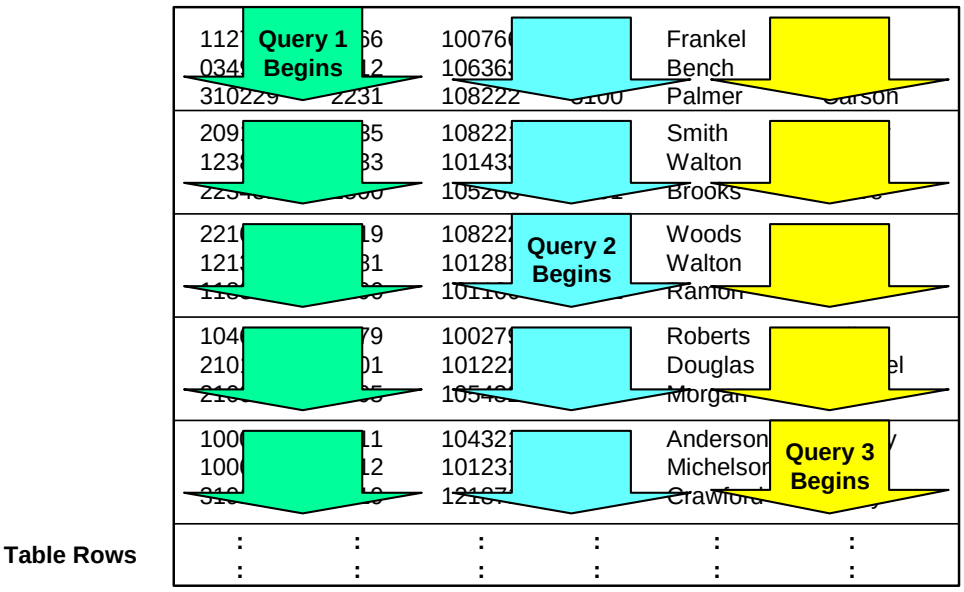
Full Table Scans – Sync Scans

In the case of multiple users that access the same table at the same time, the system can do a synchronized scan (sync scan) on the table.

Full Table Scans – Sync Scans

In the case of multiple users that access the same table at the same time, the system can do a synchronized scan (sync scan) on the table.

- **Multiple simultaneous scans share reads – this is a sync scan at the block level.**
- **New query joins scan at the current scan point.**



Module 19: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 19: Review Questions

1. Because the row is hash-distributed on different columns, the subtable row will typically land on an AMP other than the one containing the data row. This index would be:
   a. UPI or NUPI
   b. USI
   c. NUSI
   d. None of the above
2. The Teradata DBS hashes the value and uses the Row Hash to find the desired rows. This is always a one-AMP operation. This index would be:
   a. UPI or NUPI
   b. USI
   c. NUSI
   d. None of the above
3. _____ is a process that determines common Row IDs between multiple NUSI values by a process of intersection.
   a. NUSI Bit Mapping
   b. Dual NUSI Access
   c. Full Table Scan
   d. NUSI Read
4. If aggregation is performed on a NUSI column, the Optimizer accesses the NUSI subtable and returns the result without accessing the base table. This is referred to as:
   a. NUSI bit mapping
   b. Full table scan
   c. Dual NUSI access
   d. Covering Index

Notes

Module 20

Analyze Secondary Index Criteria

After completing this module, you will be able to:

- **Describe Composite Secondary Indexes.**
- **Choose columns as candidate Secondary Indexes.**
- **Analyze Change Rating, Value Access, and Range Access.**

Notes

Table of Contents

Accessing Rows 20-4
Row Selection 20-6
Secondary Index Considerations 20-8
Secondary Index Usage 20-10
Secondary Index Candidate Guidelines 20-12
Exercise 3 – Sample 20-14
  Secondary Index Candidate Guidelines 20-14
Exercise 3 – Choosing SI Candidates 20-16
  Exercise 3 – Choosing SI Candidates (cont.) 20-18
  Exercise 3 – Choosing SI Candidates (cont.) 20-20
  Exercise 3 – Choosing SI Candidates (cont.) 20-22
  Exercise 3 – Choosing SI Candidates (cont.) 20-24
  Exercise 3 – Choosing SI Candidates (cont.) 20-26
Change Rating 20-28
Value and Range Access 20-30
Exercise 4 – Sample 20-32
Exercise 4 – Eliminating Index Candidates 20-34
  Exercise 4 – Eliminating Index Candidates (cont.) 20-36
  Exercise 4 – Eliminating Index Candidates (cont.) 20-38
  Exercise 4 – Eliminating Index Candidates (cont.) 20-40
  Exercise 4 – Eliminating Index Candidates (cont.) 20-42
  Exercise 4 – Eliminating Index Candidates (cont.) 20-44
Module 20: Review Questions 20-46

Accessing Rows

Three SQL commands require that rows be physically read. They are **SELECT**, **UPDATE**, and **DELETE.** Their syntax and use are described below:

SELECT [expression] FROM tablename ...
UPDATE tablename SET col_name = [expression] ...
DELETE FROM tablename ...

- The **SELECT** command returns the value(s) from the table(s) for display or processing. Many people confuse the SQL SELECT statement with a READ command (e.g., COBOL). SELECT simply asks for the column values expressed in the project list to be returned for display or processing. The rows which have their values returned, deleted or updated are identified by the WHERE clause (when present). It is the WHERE clause that controls File System reads.

- The **UPDATE** command changes one or more column values to new values.

- The **DELETE** command removes rows from a table.

Any of the three SQL statements can be modified with a WHERE clause. Values specified in a WHERE clause tell Teradata which rows should be acted upon. Proper use of the WHERE clause will improve throughput by limiting the number of rows that must be handled.

Accessing Rows

SELECT {expression} FROM tablename…

- Returns the value(s) from the table(s) for display or processing.
- The row(s) must be physically read first.

UPDATE tablename SET columns = {expression}…

- Changes one or more column values to new values.
- The rows(s) must be physically located (read) first.

DELETE FROM tablename…

- Removes rows from a table.
- The row(s) must be physically located (read) first.

Any of the above SQL statements can contain a WHERE clause.

- Values in the WHERE cause tell Teradata what set of rows to act on.
- Without a WHERE clause, all rows participate in the operation.
- Limiting the number of rows Teradata must handle improves throughput.

Row Selection

When TERADATA processes an SQL statement with a WHERE clause, it examines the clause and builds an execution plan and access method to satisfy the clause conditions.

Certain conditions contained in the WHERE clause take advantage of indexing (assuming that the appropriate index is in place). These conditions are shown in the upper box on the facing page. Notice that these conditions all ask the RDBMS to locate a specific value or set of values. Application programmers should use these conditions whenever possible as they offer the best performance.

Other WHERE clause conditions are not able to take advantage of indexing and will always cause a Full Table Scan of either the Base Table or a SI subtable. Though they benefit from the Teradata distributed architecture, they are less desirable from a performance standpoint. These kind of conditions are listed in the middle box on the opposite page and do not focus on a specific value or set of values, thus forcing the system to do a Full Table Scan to find all the values to satisfy them.

Note that poor relational models severely limit physical design choices and generally force more Full Table Scans.

Maximum number of ORed conditions or IN list values per request can't exceed 1,048,576. There really no fixed limit on the number of entries in an IN list; however, the maximum SQL text size is 1MB and this places a request-specific upper bound on this number.

– NOTE –
The small box at the bottom of the facing page lists commands that operate on the answer sets generated by previous conditions, such as those shown in the boxes above.

Row Selection

WHERE clause conditions that may use indexing if available*:

colname = value colname IS NULL colname IN (subquery)	colname IN (explicit list of values) t1.col_x = t1.col_y t1.col_x = t2.col_x	condition1 AND condition2 condition1 OR condition2 colname = ANY, SOME or ALL

* Access methods for the above depend on whether the column(s) are indexed, type of index, and selectivity of the index.

WHERE clause conditions that typically cause a Full Table Scan:

non-equality comparisons colname IS NOT NULL colname NOT IN (explicit list of values) colname NOT IN (subquery) colname BETWEEN ... AND … Join condition1 OR Join condition2 t1.col_x [computation] = value t1.col_x [computation] = t1.col_y	INDEX (colname) SUBSTRING (colname) SUM MIN MAX AVG COUNT	DISTINCT ANY ALL NOT (condition1) col1 || col2 = value colname LIKE ... missing a WHERE clause

The following functions affect output only, not base row selection.

GROUP BY	ORDER BY
HAVING	UNION
WITH	INTERSECT
WITH … BY ...	EXCEPT

Poor relational models severely limit physical design choices and generally force more Full Table Scans.

Secondary Index Considerations

The facing page describes key considerations involved in decisions regarding the use of Secondary Indexes. It is important to weigh the costs of Secondary Indexes against the benefits.

- Some of these costs are increased use of disk space and increased I/O.
- The main benefit of Secondary Indexes is faster set selection. Choose them on frequently used set selections.

– REMEMBER –

Data demographics change over time.

Revisit all index choices regularly to make sure that they remain appropriate and serve you well.

Secondary Index Considerations

- Secondary Indexes consume disk space for their subtables.
- INSERTs, DELETEs, and UPDATEs (sometimes) cost double the I/Os.
- Choose Secondary Indexes on frequently used set selections.
  - Secondary Index use is typically based on an Equality search.
  - A NUSI may have multiple rows per value.
- The Optimizer may not use a NUSI if it is too weakly selective.
- Avoid choosing Secondary Indexes with volatile data values.
- Weigh the impact on Batch Maintenance and OLTP applications.
- USI changes are Transient Journaled. NUSI changes are not.
- Remove or drop NUSIs that are not used.

Data demographics change over time. Revisit ALL index (Primary and Secondary) choices regularly.

Make sure they are still serving you well.

Secondary Index Usage

The facing lists common usage for a USI and a NUSI.

Secondary Index Usage

Unique Secondary Index (USI) Usage

- A USI is used to maintain uniqueness in a column or columns.
- Usage is determined by specifying the USI value in an equality condition in the WHERE clause or ON clause.
- Unique Secondary Indexes support …
  - Nested Joins
  - Row-hash locking

Non-unique Secondary Index (NUSI) Usage

- Usage is determined by specifying the NUSI value in an equality condition in the WHERE clause or ON clause.
- Non-Unique Secondary Indexes support Nested Joins and Merge Joins
- Optimizer can choose to use bit-mapping for weakly selective (>10%) NUSIs which can alleviate limitations associated with composite NUSIs.
- In some cases, it may be better to use multiple single-column NUSIs (City, State) instead a single composite NUSI.
  - User has to balance the overhead of multiple NUSIs as compared to a single composite NUSI.
- Can be used to "cover" a query, avoiding base table access.
- Can significantly reduce base table I/O during value and join operations.

Secondary Index Candidate Guidelines

All Primary Index candidates are Secondary Index candidates.

Columns that are not Primary Index candidates have to also be considered as NUSI candidates. A NUSI will be used by the Optimizer to select data if it is strongly selective. A guideline to use in initially selecting NUSI candidates is the following:

The optimizer does not only look at selectively of a column to determine if a FTS or an indexed access will be used in a given plan. The decision is made based after comparing the total cost of both approaches, after considering multiple factors, including row size, block size, number of rows in the table, and also the I/O and CPU cost (based on the current hardware cost factors).

In this course, we are going to 5% as a guideline for NUSI selectivity.

> Example 1: Assume a table has 100M rows and a column has 50 distinct values that are evenly distributed (each value has the same number of rows). Therefore, each value has 2M rows and effectively represents 2% of the rows. The NUSI would be used.
>
> Example 2: Assume a table has 100M rows and a column has 20 distinct values that are evenly distributed (each value has the same number of rows). Therefore, each value has 5M rows and effectively represents 5% of the rows. The NUSI would be used.
>
> Example 3: Assume a table has 100M rows and a column has 10 distinct values that are evenly distributed (each value has the same number of rows). Therefore, each value has 10M rows and effectively represents 10% of the rows. The NUSI would not be used.

The greater the discrepancy between typical rows per value and max rows per value, the higher the probability the NUSI would not be used based on the max value used to qualify the rows.

Secondary Index Candidate Guidelines

- All Primary Index (PI) candidates are Secondary Index candidates.
  - A UPI is a USI candidate and a NUPI is a NUSI candidate.
- Columns that are not PI candidates should also be considered as NUSI candidates.
- A NUSI will be used depending on the percentage of table rows that will be accessed. For example:
  - If the number of rows accessed via a NUSI is ≤ 5%, the NUSI will be used.
  - If the number of rows accessed via a NUSI is 5 – 10%, the NUSI may or may not be used.
  - If the number of rows accessed via a NUSI is > 10%, the NUSI will not be used.
- If 5% is used as the guideline, then any column with 20 or more distinct values is considered as a NUSI candidate.
  - The optimizer (based on statistics) will decide to use (or not) the NUSI for specific values.
  - The greater the discrepancy between typical rows per value and max rows per value, the higher the probability the NUSI would not be used based on the max value used to qualify the rows.
- These are only guidelines for candidate selection. Validate (via Explain and testing) that the NUSI will be chosen AND that it will provide better performance.

Exercise 3 – Sample

In this exercise, you will work with the same tables you used to identify PI candidates in Exercise 2 in Module 17.

Use the Secondary Index Candidate Guidelines below to identify all USI and NUSI candidates. The table on the facing page provides you with an example of how to apply the Secondary Index Candidate Guidelines.

You will make further index choices for these tables in following exercises.

Note: These exercises do not provide row sizes. Therefore, assume that the rows could be as large as 960 bytes and assume a typical block size of 96 KB.

Secondary Index Candidate Guidelines

All Primary Index candidates are Secondary Index candidates.

Columns that are not Primary Index candidates have to also be considered as NUSI candidates. A NUSI will be used by the Optimizer to select data if it is strongly selective. A guideline to use in initially selecting NUSI candidates is the following:

If the number of distinct values ≥ 20, then the column is a NUSI candidate.

Exercise 3 – Sample

On the following pages, there are sample tables with typical rows per value demographics.

- Indicate ALL possible Secondary Index candidates (USI and NUSI).
- Later exercises will guide your final choices.

Secondary Index Guidelines

- All PI candidates are Secondary Index candidates.
- Other columns are NUSI candidates if typical rows/value is ≤ 5% or # of distinct values ≥ 20.

Example 60,000,000 Rows	A	B	C	D	E	F	G	H
PK/FK	PK,SA		FK,NN	NN,ND				
Distinct Values	60M	7M	1.5M	60M	8	15M	15M	700
Max Rows/Value	1	12	500	1	8M	9	725K	90K
Max Rows/NULL	0	5	0	0	0	725K	5	10K
Typical Rows/Value	1	7	35	1	7M	3	3	80K
PI/SI	UPI USI	NUPI NUSI	NUPI? NUSI	UPI USI		NUSI	NUSI	NUSI
Collect Statistics (Y/N)								

Exercise 3 – Choosing SI Candidates

In this exercise, you will work with the same tables you used to identify PI candidates in Exercise 2 in Module 17.

Use the Secondary Index Candidate Guidelines below to identify all USI and NUSI candidates.

- All Primary Index candidates are Secondary Index candidates.
- Columns that are not Primary Index candidates have to also be considered as NUSI candidates. A NUSI will be used by the Optimizer to select data if it is strongly selective. A guideline to use in initially selecting NUSI candidates is the following:

If the number of distinct values ≥ 20, then the column is a NUSI candidate.

Exercise 3 – Choosing SI Candidates

ENTITY 1

100,000,000 Rows	A	B	C	D	E	F
PK/FK	PK,UA					
Distinct Values	100M	95M	300K	250K	40M	1M
Max Rows/Value	1	2	400	350	3	110
Max Rows/NULL	0	0	0	0	1.5M	0
Typical Rows/Value	1	1	325	300	2	90
PI/SI	UPI	NUPI	NUPI	NUPI		NUPI
Collect Statistics (Y/N)						

Exercise 3 – Choosing SI Candidates (cont.)

Use the Secondary Index Candidate Guidelines below to identify all USI and NUSI candidates.

- All Primary Index candidates are Secondary Index candidates.
- Columns that are not Primary Index candidates have to also be considered as NUSI candidates. A NUSI will be used by the Optimizer to select data if it is strongly selective. A guideline to use in initially selecting NUSI candidates is the following:

If the number of distinct values ≥ 20, then the column is a NUSI candidate.

Exercise 3 – Choosing SI Candidates (cont.)

ENTITY 2

10,000,000 Rows	G	H	I	J	K	L
PK/FK	PK,SA					
Distinct Values	10M	100K	9M	12	50	180K
Max Rows/Value	1	200	2	1M	240K	60
Max Rows/NULL	0	0	100K	0	0	0
Typical Rows/Value	1	100	1	800K	190K	50
PI/SI	UPI	NUPI				NUPI
Collect Statistics (Y/N)						

Exercise 3 – Choosing SI Candidates (cont.)

Use the Secondary Index Candidate Guidelines below to identify all USI and NUSI candidates.

- All Primary Index candidates are Secondary Index candidates.
- Columns that are not Primary Index candidates have to also be considered as NUSI candidates. A NUSI will be used by the Optimizer to select data if it is strongly selective. A guideline to use in initially selecting NUSI candidates is the following:

If the number of distinct values ≥ 20, then the column is a NUSI candidate.

Exercise 3 – Choosing SI Candidates (cont.)

	DEPENDENT					
5,000,000 Rows	A	M	N	O	P	Q
PK/FK	PK				NN,ND	
	FK	SA				
Distinct Values	2M	50	90K	3M	5M	2M
Max Rows/Value	4	200K	75	2	1	5
Max Rows/NULL	0	0	0	390K	0	1M
Typical Rows/Value	1	60K	50	1	1	1
PI/SI	UPI				UPI	
	NUPI		NUPI			
Collect Statistics (Y/N)						

Exercise 3 – Choosing SI Candidates (cont.)

Use the Secondary Index Candidate Guidelines below to identify all USI and NUSI candidates.

- All Primary Index candidates are Secondary Index candidates.
- Columns that are not Primary Index candidates have to also be considered as NUSI candidates. A NUSI will be used by the Optimizer to select data if it is strongly selective. A guideline to use in initially selecting NUSI candidates is the following:

If the number of distinct values ≥ 20, then the column is a NUSI candidate.

Exercise 3 – Choosing SI Candidates (cont.)

ASSOCIATIVE 1

300,000,000 Rows	A	G	R	S
PK/FK	PK			
	FK	FK,SA		
Distinct Values	100M	10M	15K	800K
Max Rows/Value	5	50	21K	400
Max Rows/NULL	0	0	0	0
Typical Rows/Value	3	30	19K	350
	UPI			
PI/SI	NUPI	NUPI	NUPI?	NUPI
Collect Statistics (Y/N)				

Exercise 3 – Choosing SI Candidates (cont.)

Use the Secondary Index Candidate Guidelines below to identify all USI and NUSI candidates.

- All Primary Index candidates are Secondary Index candidates.
- Columns that are not Primary Index candidates have to also be considered as NUSI candidates. A NUSI will be used by the Optimizer to select data if it is strongly selective. A guideline to use in initially selecting NUSI candidates is the following:

If the number of distinct values ≥ 20, then the column is a NUSI candidate.

Exercise 3 – Choosing SI Candidates (cont.)

ASSOCIATIVE 2

100,000,000 Rows	A	M	G	T	U	
PK/FK	PK					
	FK		FK			
Distinct Values	50M		10M	560K	750	
Max Rows/Value	3		150	180	135K	
Max Rows/NULL	0		0	0	0	
Typical Rows/Value	1		8	170	100K	
	UPI					
PI/SI	NUPI		NUPI	NUPI		
Collect Statistics (Y/N)						

Exercise 3 – Choosing SI Candidates (cont.)

Use the Secondary Index Candidate Guidelines below to identify all USI and NUSI candidates.

- All Primary Index candidates are Secondary Index candidates.
- Columns that are not Primary Index candidates have to also be considered as NUSI candidates. A NUSI will be used by the Optimizer to select data if it is strongly selective. A guideline to use in initially selecting NUSI candidates is the following:

If the number of distinct values ≥ 20, then the column is a NUSI candidate.

Exercise 3 – Choosing SI Candidates (cont.)

HISTORY

730,000,000 Rows	A	DATE	D	E	F	
PK/FK	PK					
	FK	SA				
Distinct Values	100M	730	N/A	N/A	N/A	
Max Rows/Value	18	1100K	N/A	N/A	N/A	
Max Rows/NULL	0	0	N/A	N/A	N/A	
Typical Rows/Value	3	900K	N/A	N/A	N/A	
PI/SI	UPI					
	NUPI					
Collect Statistics (Y/N)						

Change Rating

Change Rating is a number that comes from **Application & Transaction Modeling (ATM)**.

- Change Rating indicates how often the values in a column, or columns, are updated.
- It is a value from 0 to 10, with 0 describing those columns which never change and 10 describing those columns which change with every write operation.
- The Change Rating values of various types of columns are shown on the facing page.

Change Rating has nothing to do with the SQL INSERT or DELETE statements. A table may be subject to frequent INSERTs and/or DELETEs, but the Change Ratings of columns will be low as long as the values within those columns remain stable throughout the lifetime of the row.

Change Rating is dependent only on the SQL UPDATE statement. Change Rating is affected when column values are UPDATEd.

Utilize Change Rating when choosing indexes. Primary Indexes must be based on columns with very stable data values. PI columns should never have Change Ratings higher than 1.

Secondary Indexes should be based on columns with at least fairly stable data values. You should not choose columns with Change Ratings higher than 3 for SIs.

Change Rating

Change Rating indicates how often values in a column are UPDATEd:

- 0 = column values never change.
- 10 = column changes with every write operation.

PK columns are always 0.

Historical data columns are always 0.

Data that does not normally change = 1.

Update tracking columns = 10.

All other columns are rated 2 - 9.

Base Primary Index choices on columns with very stable data values:

- A change rating of 0 - 1 is reasonable.

Base Secondary Index choices on columns with fairly stable data values:

- A change rating of 0 - 3 is reasonable.

Value and Range Access

Value Access Frequency is a numeric rating which tells you how many times all known transactions access the table in a given time interval (e.g., a one-year period). It measures how frequently a column, or columns, is accessed by SQL statements containing an equality value.

Range Access Frequency is a numeric rating which tells you how many times all known transactions access the table in a given time interval (e.g., a one-year period). It measures how frequently a column, or columns, is accessed by SQL statements that access a range of values such as a DATE range. These types of queries may contain inequality or BETWEEN expressions.

A Value Access or Range Access of 0 implies that there is no need to access the table through that column. Since NUSIs require system resources to maintain them (INSERTs and DELETEs require additional I/O to update the SI subtables), there is no point in having a NUSI if it is not used for access. All NUSI candidates with very low Value Access or Range Access Frequency should be eliminated.

Value and Range Access

Value Access:

- How often a column appears with an equality value. For example:

  WHERE column_name = hardcoded_value or substitutable_value

Range Access:

- How often a column is used to access a range of data values (e.g., range of dates). For example:

  WHERE column_name BETWEEN value AND value or WHERE column_name > value

Value Access or Range Access Frequency:

- How often in a given time interval (e.g., annually) all known transactions access rows from the table through this column either with an equality value or with a range of values.

Notes:

- The above demographics result from Activity Modeling.
- Low Value Access or Range Access Frequency:
  - Secondary Index overhead may cost more than doing the FTS.
- NUSIs may be considered by the optimizer for joins. In the following exercises, we are going to eliminate NUSIs with a value access of 0, but we may need to reconsider the NUSI as an index choice depending on join access (when given join metrics).
- EXPLAINs indicate if the index choices are utilized or not.

Exercise 4 – Sample

In this exercise, you will again work with the same tables that you used in Exercises 2 and 3.

In this exercise, you will look at three additional demographics to eliminate potential index candidates and to possibly choose Value-Ordered NUSI candidates. The three additional data demographics that you will look at are:

- Change Rating
- Value Access
- Range Access

Use the following Change Rating demographics guidelines to eliminate those candidates that do not fit the guidelines. The table on the right provides you with an example of how to apply these guidelines.

- PI candidates should have Change Ratings from 0 - 1.
- SI candidates should have Change Ratings from 0 - 3.

Also, eliminate those NUSI candidates which have Value Access = 0 and Range Access = 0.

If a Range Access is greater than 0, then consider the column as a possible Value-Ordered NUSI (VONUSI) candidate.

The table on the facing page provides you with an example of how to apply these guidelines.

You will make final index choices for these tables in Exercise 5 (later module).

Exercise 4 – Sample

On the following pages, there are sample tables with change row and value access demographics.

- Eliminate Index candidates based on change rating and value access.
- Identify any VONUSI candidates with a Range Access > 0
- Later exercises will guide your final choices.

Change Rating Guidelines:

- PI – change rating 0 - 1.
- SI – change rating 0 - 3.

Value Access Guideline:

- NUSI Value Access > 0
- VONUSI Range Access > 0

Example 60,000,000 Rows	A	B	C	D	E	F	G	H
PK/FK	PK,SA		FK,NN	NN,ND				
Value Access	5K	2.6K	0	500K	0	0	0	52
Range Access	12	0	0	0	0	0	0	4K
Distinct Values	60M	7M	1.5M	60M	8	15M	15M	700
Max Rows/Value	1	12	500	1	8M	9	725K	90K
Max Rows/NULL	0	5	0	0	0	725K	5	10K
Typical Rows/Value	1	7	35	1	7M	3	3	80K
Change Rating	0	1	5	3	0	4	4	9
PI/SI	UPI	NUPI	~~NUPI?~~	~~UPI~~				
	USI	NUSI	~~NUSI~~	USI		~~NUSI~~	~~NUSI~~	~~NUSI~~
Collect Statistics (Y/N)								

Exercise 4 – Eliminating Index Candidates

In this exercise, you will look at three additional demographics to eliminate potential index candidates and to possibly choose Value-Ordered NUSI candidates. The three additional data demographics that you will look at are:

- Change Rating
- Value Access
- Range Access

Use the following Change Rating demographics guidelines to eliminate those candidates that do not fit the guidelines.

- PI candidates should have Change Ratings from 0 - 1.
- SI candidates should have Change Ratings from 0 - 3.

Also, eliminate those NUSI candidates which have Value Access = 0 and Range Access = 0.

If a Range Access is greater than 0, then consider the column as a possible Value-Ordered NUSI (VONUSI) candidate.

Exercise 4 – Eliminating Index Candidates

ENTITY 1

100,000,000 Rows	A	B	C	D	E	F
PK/FK	PK,UA					
Value Access	50K	0	0	0	0	0
Range Access	0	0	0	0	0	0
Distinct Values	100M	95M	300K	250K	40M	1M
Max Rows/Value	1	2	400	350	3	110
Max Rows/NULL	0	0	0	0	1.5M	0
Typical Rows/Value	1	1	325	300	2	90
Change Rating	0	3	1	1	1	1
	UPI	NUPI	NUPI	NUPI		NUPI
PI/SI	USI	NUSI	NUSI	NUSI	NUSI	NUSI
Collect Statistics (Y/N)						

Exercise 4 – Eliminating Index Candidates (cont.)

In this exercise, you will look at three additional demographics to eliminate potential index candidates and to possibly choose Value-Ordered NUSI candidates. The three additional data demographics that you will look at are:

- Change Rating
- Value Access
- Range Access

Use the following Change Rating demographics guidelines to eliminate those candidates that do not fit the guidelines.

- PI candidates should have Change Ratings from 0 - 1.
- SI candidates should have Change Ratings from 0 - 3.

Also, eliminate those NUSI candidates which have Value Access = 0 and Range Access = 0.

If a Range Access is greater than 0, then consider the column as a possible Value-Ordered NUSI (VONUSI) candidate.

Exercise 4 – Eliminating Index Candidates (cont.)

ENTITY 2

10,000,000 Rows	G	H	I	J	K	L
PK/FK	PK,SA					
Value Access	5K	365	12	12	0	0
Range Access	12	0	0	0	0	260
Distinct Values	10M	100K	9M	12	50	180K
Max Rows/Value	1	200	2	1M	240K	60
Max Rows/NULL	0	0	100K	0	0	0
Typical Rows/Value	1	100	1	800K	190K	50
Change Rating	0	0	9	1	2	0
	UPI	NUPI				NUPI
PI/SI	USI	NUSI	NUSI		NUSI	NUSI
Collect Statistics (Y/N)						

Exercise 4 – Eliminating Index Candidates (cont.)

In this exercise, you will look at three additional demographics to eliminate potential index candidates and to possibly choose Value-Ordered NUSI candidates. The three additional data demographics that you will look at are:

- Change Rating
- Value Access
- Range Access

Use the following Change Rating demographics guidelines to eliminate those candidates that do not fit the guidelines.

- PI candidates should have Change Ratings from 0 - 1.
- SI candidates should have Change Ratings from 0 - 3.

Also, eliminate those NUSI candidates which have Value Access = 0 and Range Access = 0.

If a Range Access is greater than 0, then consider the column as a possible Value-Ordered NUSI (VONUSI) candidate.

Exercise 4 – Eliminating Index Candidates (cont.)

DEPENDENT

5,000,000 Rows	A	M	N	O	P	Q
PK/FK	PK				NN,ND	
	FK	SA				
Value Access	0	0	0	0	0	0
Range Access	0	0	0	0	0	0
Distinct Values	2M	50	90K	3M	5M	2M
Max Rows/Value	4	200K	75	2	1	5
Max Rows/NULL	0	0	0	390K	0	1M
Typical Rows/Value	1	60K	50	1	1	1
Change Rating	0	0	3	1	0	1
PI/SI	UPI				UPI	
	NUPI		NUPI			
	USI				USI	
	NUSI	NUSI	NUSI	NUSI		NUSI
Collect Statistics (Y/N)						

Exercise 4 – Eliminating Index Candidates (cont.)

In this exercise, you will look at three additional demographics to eliminate potential index candidates and to possibly choose Value-Ordered NUSI candidates. The three additional data demographics that you will look at are:

- Change Rating
- Value Access
- Range Access

Use the following Change Rating demographics guidelines to eliminate those candidates that do not fit the guidelines.

- PI candidates should have Change Ratings from 0 - 1.
- SI candidates should have Change Ratings from 0 - 3.

Also, eliminate those NUSI candidates which have Value Access = 0 and Range Access = 0.

If a Range Access is greater than 0, then consider the column as a possible Value-Ordered NUSI (VONUSI) candidate.

Exercise 4 – Eliminating Index Candidates (cont.)

ASSOCIATIVE 1

300,000,000 Rows	A	G	R	S	
PK/FK	PK				
	FK	FK,SA			
Value Access	260	0	0	0	
Range Access	0	0	0	0	
Distinct Values	100M	10M	15K	800K	
Max Rows/Value	5	50	21K	400	
Max Rows/NULL	0	0	0	0	
Typical Rows/Value	3	30	19K	350	
Change Rating	0	0	0	0	
	UPI				
PI/SI	NUPI	NUPI	NUPI?	NUPI	
	USI				
	NUSI	NUSI	NUSI	NUSI	
Collect Statistics (Y/N)					

Exercise 4 – Eliminating Index Candidates (cont.)

In this exercise, you will look at three additional demographics to eliminate potential index candidates and to possibly choose Value-Ordered NUSI candidates. The three additional data demographics that you will look at are:

- Change Rating
- Value Access
- Range Access

Use the following Change Rating demographics guidelines to eliminate those candidates that do not fit the guidelines.

- PI candidates should have Change Ratings from 0 - 1.
- SI candidates should have Change Ratings from 0 - 3.

Also, eliminate those NUSI candidates which have Value Access = 0 and Range Access = 0.

If a Range Access is greater than 0, then consider the column as a possible Value-Ordered NUSI (VONUSI) candidate.

Exercise 4 – Eliminating Index Candidates (cont.)

ASSOCIATIVE 2

100,000,000 Rows	A	M	G	T	U	
PK/FK	PK					
	FK		FK			
Value Access	0		0	0	0	
Range Access	0		0	0	0	
Distinct Values	50M		10M	560K	750	
Max Rows/Value	3		150	180	135K	
Max Rows/NULL	0		0	0	0	
Typical Rows/Value	1		8	170	100K	
Change Rating	0		0	0	0	
PI/SI	UPI					
	NUPI		NUPI	NUPI		
	USI					
	NUSI		NUSI	NUSI	NUSI	
Collect Statistics (Y/N)						

Exercise 4 – Eliminating Index Candidates (cont.)

In this exercise, you will look at three additional demographics to eliminate potential index candidates and to possibly choose Value-Ordered NUSI candidates. The three additional data demographics that you will look at are:

- Change Rating
- Value Access
- Range Access

Use the following Change Rating demographics guidelines to eliminate those candidates that do not fit the guidelines.

- PI candidates should have Change Ratings from 0 - 1.
- SI candidates should have Change Ratings from 0 - 3.

Also, eliminate those NUSI candidates which have Value Access = 0 and Range Access = 0.

If a Range Access is greater than 0, then consider the column as a possible Value-Ordered NUSI (VONUSI) candidate.

Exercise 4 – Eliminating Index Candidates (cont.)

HISTORY

730,000,000 Rows	A	DATE	D	E	F	
PK/FK	PK					
	FK	SA				
Value Access	10M	5K	0	0	0	
Range Access	0	20K	0	0	0	
Distinct Values	100M	730	N/A	N/A	N/A	
Max Rows/Value	18	1100K	N/A	N/A	N/A	
Max Rows/NULL	0	0	N/A	N/A	N/A	
Typical Rows/Value	3	900K	N/A	N/A	N/A	
Change Rating	0	0	N/A	N/A	N/A	
PI/SI	UPI					
	NUPI					
	USI					
	NUSI	NUSI				
Collect Statistics (Y/N)						

Module 20: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 20: Review Questions

1. With a NUPI, a technique to avoid a duplicate row check is to _____.
   a. use set tables
   b. use the NOT NULL constraint on the column
   c. create the table as a MULTISET table
   d. compare data values byte-by-byte within a Row Hash in order to ensure uniqueness

2. Which type of usage normally applies to a USI? ____
   a. Range access
   b. NOT condition
   c. Equality value access
   d. Inequality value access

3. Which two types of usage normally apply to a composite NUSI that is hash-ordered? ____ ____
   a. Covering index
   b. Equality value access
   c. Inequality value access
   d. Non-covering range access

Notes

Module 21

Access Considerations and Constraints

After completing this module, you will be able to:

- **Analyze Optimizer Access scenarios.**
- **Explain partial value searches and data conversions.**
- **Identify the effects of conflicting data types.**
- **Determine the cost of I/Os.**
- **Identify column level attributes and constraints.**
- **Identify table level attributes and constraints.**
- **Add, modify and drop constraints from tables.**
- **Explain how the Identity column allocates new numbers.**

Notes

Table of Contents

Access Method Comparison 21-4
Unique Primary Index (UPI) 21-4
Non-Unique Primary Index (NUPI) 21-4
Unique Secondary Index (USI) 21-4
Non-Unique Secondary Index (NUSI) 21-4
Full-Table Scan (FTS) 21-4
Optimizer Access Scenarios 21-6
Data Conversions 21-8
Storing Numeric Data 21-10
Data Conversion Example 21-12
Matching Data Types 21-14
Counting I/O Operations 21-16
Additional I/O 21-16
Transient Journal I/O 21-18
INSERT and DELETE Operations 21-20
UPDATE Operations 21-22
Primary Index Value UPDATE 21-24
Table Level Attributes 21-26
Example of Column and Table Level Constraints 21-28
Table Level Constraints 21-28
Example (13.0) – SHOW Department Table 21-30
Example (13.10) – SHOW Department Table 21-32
Altering Table Constraints 21-34
Identity Column – Overview 21-36
Business Value 21-36
Business Usage 21-36
Identity Column – Implementation 21-38
Performance 21-38
Process for Generating Identity Column Numbers 21-38
Identity Column – Example 1 21-40
Identity Column – Example 2 21-42
Identity Column – Considerations 21-44
Limited to DECIMAL(18,0) 21-44
Restrictions 21-44
Module 21: Review Questions 21-46

Access Method Comparison

We have seen in preceding modules that Teradata can access data (through indexes or Partition, or Full Table Scans). The facing page illustrates these various access methods in order of number of AMPs affected.

Unique Primary Index (UPI)

The UPI is the most efficient way to access data. Accessing data through a UPI is a one-AMP operation that leads directly to the single row with the desired UPI value. The system does not have to create a Spool file during a UPI access.

Non-Unique Primary Index (NUPI)

Accessing data through a NUPI is a one-AMP operation that may lead to multiple rows with the desired NUPI value. The system creates a spool file during a NUPI access, if needed. NUPI access is efficient if the number of physical block reads is small.

Unique Secondary Index (USI)

A USI is a very efficient way to access data. Data access through a USI is usually a two-AMP operation, which leads directly to the single row with the desired USI value. The system does not have to create a spool file during a USI access.

There are cases where a USI is actually more efficient than a NUPI. In these cases, the optimizer decides on a case-by-case basis which method is more efficient. Remember: the optimizer can only make informed decisions if it is provided with statistics.

Non-Unique Secondary Index (NUSI)

As we have seen, the non-unique secondary index (NUSI) is efficient only if the number of rows accessed is a small percentage of the total data rows in the table. NUSI access is an all-AMPs operation since the NUSI subtables must be scanned on each AMP. It is a multiple rows operation since there can be many rows per NUSI value. A spool file will be created if needed.

Full-Table Scan (FTS)

The Full-Table Scan is efficient in that each row is scanned only once. Although index access is generally preferred to a FTS, there are cases where they are the best way to access the data.

Like the situation with NUPIs and USIs, Full Table Scans can sometimes be more efficient than a NUSI. The optimizer decides on a case-by-case basis which is more efficient (assuming that it has been provided with statistics).

The Optimizer chooses what it thinks is the fastest access method.
COLLECT STATISTICS to help the Optimizer make good decisions.

Access Method Comparison

Unique Primary Index

- Very efficient
- One AMP, one row
- No spool file

Non-Unique Primary Index

- Efficient if the number of rows per value is reasonable and there are no severe spikes.
- One AMP, multiple rows
- Spool file if needed

No Primary Index

- Access is a full table scan without secondary indexes.

Unique Secondary Index

- Very efficient
- Two AMPs, one row
- No spool file

Non-Unique Secondary Index

- Efficient only if the number of rows accessed is a small percentage of the total data rows in the table.
- All AMPs, multiple rows
- Spool file if needed

Partition Scan

- Efficient since because of partition elimination.
- All AMPs; all rows in specific partitions

Full-Table Scan

- Efficient since each row is touched only once.
- All AMPs, all rows
- Spool file may equal the table in size

The Optimizer chooses the fastest access method.

COLLECT STATISTICS to help the Optimizer make good decisions.

Optimizer Access Scenarios

Given the SQL WHERE clause on the facing page, the Optimizer decides which column it will use to access the data. This decision is based upon what indexes have been defined on the two columns (**Col_1** and **Col_2**).

When you examine the table, you can see that the optimizer chooses the most efficient access method depending on the situation. Interesting cases to note are as follows:

If **Col_1** is a NUPI and **Col_2** is a USI, the Optimizer chooses **Col_1** (NUPI) if its selectivity is close to a UPI (nearly unique). Otherwise, it accesses via **Col_2** (USI) since only one row is involved, even though it is a two-AMP operation.

If both columns are NUSIs, the Optimizer must determine the how selective each of them is. Depending on the relative selectivity, the Optimizer may choose to access via **Col_1**, **Col_2**, NUSI Bit Mapping or a FTS.

If either one of the columns is a NUSI or the other column is not indexed, the Optimizer determines the selectivity of the NUSI. Depending on this selectivity, it chooses either to utilize the NUSI or to do a FTS.

Whenever one of the columns is used to access the data, the remaining condition is used as a row qualifier. This is known as a **residual condition**.

Optimizer Access Scenarios

SINGLE TABLE CASE

```
WHERE   Table_1.Col_1 = :value_1
AND     Table_1.Col_2 = :value_2 ;
```

Col_1 \ Col_2	USI	NUSI	NOT INDEXED
UPI	UPI	UPI	UPI
NUPI	NUPI or USI [1]	NUPI	NUPI
USI	USI	USI	USI
NUSI	USI	Either, Both, or FTS [2]	NUSI or FTS [3]
NOT INDEXED	USI	NUSI or FTS [3]	FTS

Column the Optimizer uses for access.

Notes:

1. **The Optimizer prefers Primary Indexes over Secondary Indexes.** It chooses the NUPI if only one I/O (block) is accessed.
   The Optimizer prefers Unique indexes over non-unique indexes. Only one row is involved with USI even though it is a two-AMP operation.
2. Depending on relative selectivity, the Optimizer may use either NUSI, may use both with NUSI Bit Mapping, or may do a FTS.
3. It depends on the selectivity of the index.

Data Conversions

Operands in an SQL statement must be of the same data type to be compared. If operands differ, internal data conversion is performed.

Data conversion is expensive in terms of system overhead and adversely affects performance. The physical designer should make every effort to minimize the need for data conversion. The best way to do this is to implement data types at the **Domain** level which should eliminate comparisons across data type. If data values come from the same Domain, they must be of the same data type and therefore, can be compared without conversion.

Columns used in addition, subtraction, comparison, and join operations should always be from the same domain. Multiplication and division operations involve columns from two or three domains.

In the Teradata Database the Byte data types can only be compared to a column with the Byte data type or a character string of XB'_ _ _ _...'

For example, the system converts a CHARACTER value to a DATE value using the DATE conversion. On the other hand, converting from BYTE to NUMERIC is not possible (indicated by "ERROR").

Data Conversions

- Columns (or values) must be of the same data type to be compared without necessary conversion.
- Character data is compared using the host's collating sequence.
  - Unequal-length character strings are converted by right-padding the shorter one with blanks.
- If column (or values) types differ, internal conversion is performed.
  - Numeric values are converted to the same underlying representation.
  - Character to numeric comparison requires the character value to be converted to a numeric value.
- Data conversion is expensive and generally unnecessary.
- Implement data types at the Domain level.
  - Comparison across data types may indicate that Domain definitions are not clearly understood.

Storing Numeric Data

You should always store numeric data in numeric data types. Teradata will always convert character data to numeric data prior to doing a comparison.

When Teradata is asked to do a comparison, it will always apply the following rules:

To compare 2 columns, they must be of the same data type.

Character data types will always be converted to numeric.

The example on the slide page demonstrates the potential performance hit that could occur, when you store numeric data as a character data type.

In Case 1 (Numeric values stored as Character Data Type):

Statement 1 uses a character literal – Teradata will do a PI access (no data conversion required) to perform the comparison.

Statement 2 uses a numeric value – Teradata will do a **Full Table Scan (FTS)** against the EMP1 table converting Emp_no to a numeric value and then do the comparison.

In Case 2 (Numeric values stored as Numeric Data Type):

Statement 1 uses a numeric value – Teradata will do a PI access (no data conversion required) to perform the comparison.

Statement 2 uses a character literal – Teradata will convert the character literal to a numeric value, then do a PI access to perform the comparison.

Storing Numeric Data

When comparing character data to numeric, Teradata will always convert character to numeric, then do the comparison.

Comparison Rules:

To compare columns, they must be of the same Data types.

Character data types will always be converted to numeric (when comparing character to numeric).

Bottom Line:

Always store numeric data in numeric data types to avoid unnecessary and costly data conversions.

Case 1

```
CREATE TABLE Emp1
   (Emp_no      CHAR(6),
    Emp_name    CHAR(20))
UNIQUE PRIMARY INDEX
                (Emp_no);
```

Statement 1

```
SELECT   *
FROM     Emp1
WHERE    Emp_no = '1234';
```

Statement 2

```
SELECT   *
FROM     Emp1
WHERE    Emp_no = 1234;
```

Results in Full Table Scan

Case 2

```
CREATE TABLE Emp2
   (Emp_no      INTEGER,
    Emp_name    CHAR(20))
UNIQUE PRIMARY INDEX
                (Emp_no);
```

Statement 1

```
SELECT   *
FROM     Emp2
WHERE    Emp_no = 1234;
```

Statement 2

```
SELECT   *
FROM     Emp2
WHERE    Emp_no = '1234';
```

Results in unnecessary conversion

Data Conversion Example

The example on the facing page illustrates how data conversion adversely affects system performance.

You can see the results of the first EXPLAIN. Note that total estimated time to perform this **SELECT** is minimal. The system can process this request quickly because the data type of the literal value matches the column type. A character column value (col1) is being compared to a character literal ('8') which allows TERADATA to use the UPI defined on c1 for access and for maximum efficiency. The query executes as a UPI SELECT.

In the second SELECT statement, the character column value (col1) is compared with a numeric value (8). You should notice that the total "cost" for this **SELECT** is nearly 30 times the estimate for the preceding **SELECT**. The system must do a Full Table Scan and convert the character values in **col1** to numeric to compare them against the numeric literal (8).

If the column was numeric and the literal value was character, the literal would convert to numeric and the result could be hashed, allowing UPI access.

Data Conversion Example

```
CREATE SET TABLE TFACT01.Table1
    (col1 CHAR(12) NOT NULL)
UNIQUE PRIMARY INDEX (col1);
```

```
EXPLAIN SELECT * FROM Table1 WHERE col1 = '8';
```

1) First, we do a single-AMP RETRIEVE step from TFACT01.Table1 by way of the unique primary index "TFACT01.Table1.col1 = '8' " with no residual conditions. The estimated time for this step is 0.00 seconds.
-> The row is sent directly back to the user as the result of statement 1. The total estimated time is 0.00 seconds.

```
EXPLAIN SELECT * FROM Table1 WHERE col1 = 8;
```

1) First, we lock a distinct TFACT01."pseudo table" for read on a RowHash to prevent global deadlock for TFACT01.Table1.
2) Next, we lock TFACT01.Table1 for read.
3) We do an all-AMPs RETRIEVE step from TFACT01.Table1 by way of an all-rows scan with a condition of ("(TFACT01.Table1.col1 (FLOAT, FORMAT '-9.99999999999999E-999')UNICODE)= 8.00000000000000E 000") into Spool 1, which is built locally on the AMPs. The size of Spool 1 is estimated with no confidence to be 1,001 rows. The estimated time for this step is 0.28 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.28 seconds.

Matching Data Types

There are a few data types that the hashing algorithm treats identically.

The best way to make sure that you don't run into this problem is to administer the data type assignments at the Domain level. Designing a system around domains helps ensure that you give matching Primary Indexes across tables the same data type.

Matching Data Types

The following data types are identical to the hashing algorithm:

BYTEINT = SMALLINT = INTEGER = BIGINT = DATE = DECIMAL (x,0)
CHAR = VARCHAR = LONG VARCHAR
BYTE = VARBYTE
GRAPHIC = VARGRAPHIC

Administer data type assignments at the domain level.

Give matching Primary Indexes across tables the same data type.

Counting I/O Operations

Understanding the cost of various Teradata transactions in terms of I/O will help you avoid unnecessary I/O overhead when doing your physical design.

Many factors can influence the number of physical I/Os in a transaction. Some are listed on the facing page.

The main concept of the next few pages is to help you understand the relative cost of doing INSERT, DELETE, and UPDATE operations. This understanding enables you to detect subsequent problems when doing performance analysis on a troublesome application. When counting I/O, it is important to remember that all such calculations give you a relative – not the absolute – cost of the transaction. Any given I/O operation may or may not cause any actual physical I/O.

– Normally, when making a change to a table (INSERT, UPDATE, and DELETE), not only does the actual table have to be updated, but before-images have to be written in the Transient Journal to maintain transaction integrity. Transient Journal space is automatically allocated and is integrated with the WAL (Write-Ahead-Logic) Log, which has its own cylinders and file system.

Additional I/O

A table may also have Join Indexes, Hash indexes, or a Permanent Journal associated with it. Join Indexes can also have secondary indexes.

In additional the number of I/Os for changes to a table, these options will result in additional I/Os.

Whenever Permanent Journaling is used, additional I/O is incurred. The amount of this I/O varies according to whether you are using Before Imaging, After Imaging, or both, and whether the imaging is single or dual. The table on the facing page shows how many I/O operations are involved in writing the Permanent Journal block and the Cylinder Index.

To calculate the Total Permanent Journal I/O for PJ INSERTs, DELETEs and UPDATEs, you apply the appropriate formula shown on the facing page.

Permanent Journal I/O is in addition to any I/O incurred during the operation itself. In order to calculate the TOTAL I/O for an operation, you must sum the I/Os from the operation with the Total PJ I/O corresponding to that operation.

Counting I/O Operations

- **Many factors influence the number of physical I/Os in a transaction:**
  - **Cache hits**
  - **Rows per block**
  - **Cylinder migrates**
  - **Mini-Cylpacks**
  - **Number of spool files and spool file sizes**
- **I/Os may be done serially or in parallel.**
- **Data and index block I/O may or may not require Cylinder Index I/O.**
- **Changes to data rows and USI rows require before-images (undo rows) and after-images (redo rows) to be written to the WAL log.**
- **Logical I/O counts indicate the relative cost of a transaction.**
  - **A given I/O operation may not cause any actual physical I/O.**
- **A table may also have Secondary, Join/Hash indexes, or a Permanent Journal associated with it. Join Indexes can also have secondary indexes.**
  - **In additional to the number of I/Os for changes to a table, these options will result in additional I/O.**

Transient Journal I/O

The **Transient Journal (TJ)** exists to permit the successful rollback of a failed transaction. Transactions are not committed to the database until an End Transaction request has been received by the AMPs, either implicitly or explicitly. Until that time, there is always the possibility that the transaction may fail in which case the participating table(s) must be restored to their pre-transaction state.

The **Transient Journal** maintains a copy of all before-images of all rows affected by the transaction. If the event of transaction failure, the before images are reapplied to the affected tables, the images are deleted from the journal and a rollback operation is completed. When the transaction completes (assume successfully), at the point of transaction commit, the before-images for the transaction are discarded from the journal. Normally, when making a change to a table (INSERT, UPDATE, and DELETE), not only does the actual table have to be updated, but before-images have to be written in the TJ to maintain transaction integrity.

– The preservation of the before-change row images for a transaction is the task of the Write Ahead Logic (WAL) component of the Teradata database management software. The system maintains a separate TJ (undo records) entry in the WAL log for each individual database transaction whether it runs in ANSI or Teradata session mode.

–

The WAL Log includes the following:

–

Before-image or undo records used for transaction rollback.
After-image or redo records for updating disk blocks and insuring file system consistency during restarts, based on operations performed in cache during normal operation.

–

– The WAL Log is conceptually similar to a table, but the log has a simpler structure than a table. Log data is a sequence of WAL records, different from normal row structure and not accessible via SQL.

When are transient journal rows actually written to the WAL log? This occurs BEFORE the modification is made to the base table row.

Some situations where Transient Journal is not used when updating a table include:

INSERT / SELECT into an empty table
DELETE FROM tablename ALL;
Utilities such as FastLoad and MultiLoad

When a DELETE ALL is done, the master index and the cylinder indexes are updated. An entry is actually placed in the Transient Journal indicating that a “DELETE ALL” has been issued. Before-images of the individual deleted rows are not stored in the TJ. In the event a node happens to fail in the middle of a DELETE ALL, the TJ is checked for the deferred action that indicates a DELETE ALL was issued. The system checks to ensure that the DELETE ALL has completed totally as part of the restart process.

Transient Journal I/O

The Transient Journal (TJ) is …

- A journal of transaction before-images (or undo records) maintained in the WAL log.
- Provides for automatic rollback in the event of TXN failure.
- Provides "Transaction Integrity".
- Is automatic and transparent.
- TJ images are maintained in the WAL Log. The WAL Log includes the following:
  - Before-images or undo records used for transaction rollback.
  - After-images or redo records for updating disk blocks and insuring file system consistency during restarts, based on operations performed in cache (FSG) during normal operation.

Therefore, when modifying a table, there are I/O's for the data table and the WAL log (undo and redo records).

Some situations where Transient Journal is not used include:

- INSERT / SELECT into an empty table
- DELETE tablename; (Deletes all the rows in a table)
- Utilities such as FastLoad and MultiLoad
- ALTER TABLE

INSERT and DELETE Operations

To calculate the number of I/Os required to INSERT a new data row or DELETE an existing row, it is necessary to do three subsidiary calculations. They are:

Number of I/Os required to INSERT or DELETE the row itself = **five**.
Number of I/Os required for each Unique Secondary Index (USI) = **five**.
Number of I/Os required for each Non-Unique Secondary Index (NUSI) = **three**.

The overall formula for counting I/Os for INSERT and DELETE operations is shown at the bottom of the facing page. The number of I/Os must be doubled if Fallback is used.

INSERT and DELETE Operations

INSERT INTO tablename . . . ; DELETE FROM tablename . . . ; **(* is an I/O operation)**

DATA ROW
- * READ Data Block
- * WRITE Transient Journal record (UNDO row) to WAL Log
- * INSERT or DELETE the data row, and WRITE REDO row (after-image) to WAL Log
- * WRITE new Data Block
- * WRITE Cylinder Index

For each USI
- * READ USI subtable block
- * WRITE Transient Journal record (UNDO index row) to WAL Log
- * INSERT or DELETE the new USI subtable row, and WRITE REDO row (after-image) to WAL Log for the USI subtable row
- * WRITE new USI subtable block
- * WRITE Cylinder Index

For each NUSI
- * READ NUSI subtable block
  ADD or DELETE the ROWID on the ROWID LIST or
  ADD or DELETE the NUSI subtable row
- * WRITE new NUSI subtable block
- * WRITE Cylinder Index

I/O operations per row = 5 + [5 * (#USIs)] + [3 * (#NUSIs)]

Double for FALLBACK

UPDATE Operations

To calculate the number of I/Os required when updating a data column, it is necessary to perform three subsidiary calculations. They are:

The number of I/Os required to UPDATE the column in the data row itself = **five**.

The number of I/Os required to change any USI subtable containing the particular column which was updated = **ten** (five to remove the old subtable row and five to add the new subtable row).

The number of I/Os required to change the subtable of any NUSI containing the particular column which was updated = **six** (three to remove the old Row ID or subtable row and three to add the new Row ID or subtable row).

The overall formula for counting I/Os for UPDATE operations is shown at the bottom of the facing page.

– REMEMBER –
You are simply estimating the relative cost of a transaction.

UPDATE Operations

UPDATE tablename SET colname = exp . . . (other than PI column) (* = I/O Operations)

DATA ROW

* READ Data Block
* WRITE Transient Journal record (UNDO row) to WAL Log
* UPDATE the data row, and WRITE REDO row (after-image) to WAL Log
* WRITE new Data Block
* WRITE Cylinder Index

If colname = USI column

* READ current USI subtable block
* WRITE TJ record (UNDO row) into WAL Log
* DELETE USI subtable row, and WRITE REDO row (after-image) to WAL Log
* WRITE USI subtable block
* WRITE Cylinder Index
* READ new USI subtable block
* WRITE TJ record (UNDO row) into WAL Log
* INSERT new Index Subtable row, and WRITE REDO row (after-image) to WAL Log
* WRITE new USI subtable block
* WRITE Cylinder Index

If colname = NUSI column

* READ current NUSI subtable block
  REMOVE data row's RowID from RowID list or REMOVE NUSI subtable row if last RowID
* WRITE NUSI subtable block
* WRITE Cylinder Index
* READ new NUSI subtable block
  ADD data row's RowID to RowID list or ADD new NUSI subtable row
* WRITE new NUSI subtable block
* WRITE Cylinder Index

I/O operations per row = 5 + [10 * (#USIs)] + [6 * (#NUSIs)]

Double for FALLBACK

Primary Index Value UPDATE

Updating the Primary Index Value is the most I/O intensive operation of all. This is due to the fact that any change to the PI invalidates all existing secondary index "pointers."

To calculate the number of I/Os required to UPDATE a PI column, it is necessary to perform three subsidiary calculations:

The number of I/Os required to UPDATE the PI column in the data row itself.
The number of I/Os required to change any USI subtable
The number of I/Os required to change any NUSI subtable

Study the steps on the facing page. Notice that updating a PI value is equivalent to first deleting and then inserting a row. All the steps necessary to do a DELETE are performed, and then all the steps necessary to do an INSERT are performed. Changing the PI value involves actually moving the row to the location determined by the new hash value. Thus, the number of steps involved in this process is exactly double the number of steps to perform either an INSERT or a DELETE.

The formula for calculating the number of I/Os involved in a PI value update (shown at the bottom of the facing page) can be derived by doubling the formula for INSERTing or DELETing:

Formula for PI Value Update = 10 + (5 * # USIs) + (6 * # NUSIs)

Remember to double the number if Fallback is used.

Note: If the USI changes, then the number of I/O's for each changed USI is 8 in the preceding formula.

Primary Index Value Update

UPDATE tablename SET PI_column = *new_value* . . . ; **(* = I/O Operations)**

Note: Assume only PI value is changed – all Secondary Index subtable rows are updated.

DATA ROW	**	READ current Data Block, WRITE TJ record (UNDO row) to WAL Log
	*	DELETE the Data Row, and WRITE REDO row (after-image) to WAL Log
	**	WRITE new Data Block, WRITE Cylinder Index
	**	READ new Data Block, WRITE TJ record (UNDO row) to WAL Log
	*	INSERT the DATA ROW, and WRITE REDO row (after-image) to WAL Log
	**	WRITE new Data Block, WRITE Cylinder Index
For each USI	*	READ USI subtable block
	*	WRITE TJ record (UNDO row) into WAL Log
	*	UPDATE the USI subtable row with the new RowID, and WRITE REDO row (after-image) to WAL Log
	*	WRITE new USI subtable block
	*	WRITE Cylinder index
For each NUSI	*	Read NUSI subtable block on AMP for current PI value
	*	Read NUSI subtable block on AMP for new value
		UPDATE the RowID list for both of the subtable blocks
	**	WRITE new NUSI subtable blocks
	**	WRITE Cylinder Indexes

I/O operations per row = 10 + [5 * (#USIs)] + [6 * (#NUSIs)]

Double for FALLBACK`

Table Level Attributes

Because ANSI permits the possibility of duplicate rows in a table, a table level attribute (SET, MULTISET) specifies whether or not to allow duplicates. Maximum data block sizes can now be specified as part of a table creation, thus allowing for smaller or larger blocks depending on the needs of the processing environment. Typically, decision support applications prefer larger block sizes while on-line transaction processing applications generally use smaller block sizes.

Additionally, a parameter may be set to allow for a particular cylinder fill factor during table loading (FREESPACE). This factor may be set high for high subsequent file maintenance activity, or low for relatively static tables.

The Checksum parameter (table level attribute not listed on facing page) feature improves Teradata's ability to detect data corruption in user data at the earliest occurrence. The higher levels of checksums cause more sampling of data and more performance impact. The default system value is normally NONE which has no performance impact. The CHECKSUM is a calculated value (XOR logic) and is stored separate from the data segment. It is stored in the Cylinder Index. This option is not as necessary with latest Disk Array Controller's DAP-3 protection.

When a CHECKSUM value other than NONE is used, the data rows (in blocks) are not updated in place. These "safe" writes prevent the system from not being able to recover from an interrupted write corruption error.

Options for this parameter are:

DEFAULT	Calculate (or not) checksums based on system defaults as specified with the DBS Control utility and the Checksum fields.
NONE	Do not calculate checksums.
LOW	Calculate checksums by sampling a low percentage of the disk block. Default is to sample 2% of the disk block, but this value is determined by the value in the DBS Control Checksum definitions.
MEDIUM	Calculate checksums by sampling a medium percentage of the disk block. Default is to sample 33% of the disk block, but this value is determined by the value in the DBS Control Checksum definitions.
HIGH	Calculate checksums by sampling a high percentage of the disk block. Default is to sample 67% of the disk block, but this value is determined by the value in the DBS Control Checksum definitions.
ALL	Calculate checksums using the entire disk block (sample 100% of the disk block to generate a checksum).

Table Level Attributes

```
CREATE MULTISET TABLE  Table_1,   FALLBACK, DATABLOCKSIZE = 64 KBYTES,
                                  FREESPACE = 15, MERGEBLOCKRATIO = 60
   (column1          INTEGER          NOT NULL,
    column2          CHAR(5)          NOT NULL,
    CONSTRAINT       table_constraint CHECK (column1 > 0)
   )
    PRIMARY INDEX    (column1)
    INDEX            (column2);
```

SET	Don't allow duplicate rows
MULTISET	Allow duplicate rows (ANSI default)

DATABLOCKSIZE = BYTES or KBYTES	Maximum multi-row block size for table in: **BYTES** **Rounded to nearest sector (512)** **KILOBYTES (or KBYTES)** **Increments of 1024**
MINIMUM DATABLOCKSIZE	(7168)
MAXIMUM DATABLOCKSIZE	(130,560)
IMMEDIATE	May be used to immediately re-block the data with ALTER.

FREESPACE = integer [PERCENT]	Percent of freespace to keep on cylinder during load operations (0 - 75%).

DEFAULT MERGEBLOCKRATIO **MERGEBLOCKRATIO = integer [PERCENT]** **NO MERGEBLOCKRATIO**	The merge block ratio to be used for this table when when Teradata combines smaller data blocks into a single larger data block (13.10). Typical system default is 60%.

Example of Column and Table Level Constraints

Constraints can be placed at the column or the table level. Constraints may be named or unnamed.

PRIMARY KEY	May only be defined on NOT NULL columns; guarantees uniqueness.
UNIQUE	May only be defined on NOT NULL columns; guarantees uniqueness.
CHECK	Allows range or value constraints to be placed on the column.
REFERENCES	Requires values to be referenced checked before being allowed.

Note: Columns with a REFERENCES constraint must refer to a column that has been defined either with a PRIMARY KEY or UNIQUE constraint.

With Teradata, attributes and/or constraints can be assigned at the column when the table is created (CREATE TABLE) or altered (ALTER TABLE). Some examples of attributes/constraints that can be implemented include:

No Nulls – e.g., NOT NULL
No duplicates – e.g., UNIQUE
Data type – e.g., INTEGER
Size – e.g., VARCHAR(30)
Check – e.g., CHECK (col2 > 0)
Default – e.g., DEFAULT CURRENT_DATE
References – e.g., REFERENCES parent(col4)

Table Level Constraints

Constraints may also be specified at the table level. This is the only way to implement constraints that involve more than one column. **Table level constraints** follow all column level definitions. As previously, constraints may be either named or unnamed.

Example of Column and Table Level Constraints

There are four types of constraints.

PRIMARY KEY	No Nulls, No Duplicates
UNIQUE	No Nulls, No Duplicates
CHECK	Verify values or range
REFERENCES	Relates to other columns

Constraints can be defined at the column or table level.

Notes for the following example:

- Some constraints are named, some are not.
- Some constraints are at column level, some are defined at the table level.
- The SHOW TABLE command will display this table differently for 13.0 and 13.10.

```
CREATE TABLE Department
    ( dept_number       INTEGER         NOT NULL CONSTRAINT primary_1   PRIMARY KEY
     ,dept_name         CHAR(20)        NOT NULL UNIQUE
     ,dept_mgr_number   INTEGER
     ,budget_amount     DECIMAL (10,2)  COMPRESS 0
     ,CONSTRAINT        refer_1         FOREIGN KEY (dept_mgr_number)
                                            REFERENCES  Employee (employee_number)
     ,CONSTRAINT        dn_gt_1000      CHECK  (dept_number > 1000)
    );
```

Example (13.0) – SHOW Department Table

The **SHOW TABLE** command shows a definition that is slightly altered from the original script.

Note:

The **PRIMARY KEY is** implemented as a unique primary index.
The **UNIQUE** constraint is implemented as a unique secondary index.
The **REFERENCES** constraint is implemented as a FOREIGN KEY at the table level.
The **CHECK** constraint is implemented at the table level.

Additional notes: Since this table was created in Teradata mode, the following also applies:

The table is created as a SET table.
The character field is implemented with a NOTCASESPECIFIC attribute.

It is advisable to keep original scripts for documentation, as the original coding will otherwise be lost.

Example (13.0) – SHOW Department Table

This is an example of SHOW TABLE with Teradata 13.0.

```
SHOW TABLE Department;

CREATE SET TABLE PD.Department, FALLBACK,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
    (
     dept_number INTEGER NOT NULL,
     dept_name CHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
     dept_mgr_number INTEGER,
     budget_amount DECIMAL(10,2)  COMPRESS 0,
     CONSTRAINT dn_gt_1000 CHECK ( dept_number >  1000 ),
CONSTRAINT refer_1 FOREIGN KEY ( dept_mgr_number ) REFERENCES
              PD.EMPLOYEE ( EMPLOYEE_NUMBER ))
UNIQUE PRIMARY INDEX primary_1 ( dept_number )
UNIQUE INDEX ( dept_name );
```

Notes:

- **In Teradata 13.0, the SHOW TABLE command does not show the Primary Key and Unique constraints.**
- **Since Primary Key and Unique constraints are implemented as unique indexes, the Show Table command shows these constraints as indexes.**
- **All constraints are specified at table level with SHOW TABLE.**

Example (13.10) – SHOW Department Table

An example of the same SHOW TABLE with Teradata 13.10 follows:

```
SHOW TABLE Department;

CREATE SET TABLE PD.Department , FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT,
    DEFAULT MERGEBLOCKRATIO
    (
     dept_number INTEGER NOT NULL,
     dept_name CHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
     dept_mgr_number INTEGER,
     budget_amount DECIMAL(10,2) COMPRESS 0,
     CONSTRAINT dn_1000_plus CHECK ( dept_number >  999  ),
CONSTRAINT primary_1 PRIMARY KEY ( dept_number ),
UNIQUE ( dept_name ),
CONSTRAINT refer_1 FOREIGN KEY ( dept_mgr_number )
                              REFERENCES PD.EMPLOYEE ( EMPLOYEE_NUMBER )) ;
```

The **SHOW TABLE** command again shows a definition that is slightly altered from the original script; however the Teradata 13.10 version shows PRIMARY KEY and UNIQUE constraints as originally specified.

Note:

The **PRIMARY KEY** is implemented as a unique primary index.
The **UNIQUE** constraint is implemented as a unique secondary index.
The **REFERENCES** constraint is implemented as a FOREIGN KEY at the table level.
The **CHECK** constraint is implemented at the table level.

Additional notes: Since this table was created in Teradata mode, the following also applies:

The table is created as a SET table.
The character field is implemented with a NOTCASESPECIFIC attribute.

As before, it is advisable to keep the original scripts for documentation, as the original coding will otherwise be lost.

Example (13.10) – SHOW Department Table

This is an example of SHOW TABLE with Teradata 13.10.

```
SHOW TABLE Department;

CREATE SET TABLE PD.Department, FALLBACK,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT,
    DEFAULT MERGEBLOCKRATIO
   (
    dept_number INTEGER NOT NULL,
    dept_name CHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
    dept_mgr_number INTEGER,
    budget_amount DECIMAL(10,2)  COMPRESS 0,
    CONSTRAINT dn_gt_1000 CHECK ( dept_number >  1000 ),
CONSTRAINT primary_1 PRIMARY KEY ( dept_number ),
UNIQUE ( dept_name ),
CONSTRAINT refer_1 FOREIGN KEY ( dept_mgr_number )
                              REFERENCES PD.EMPLOYEE ( EMPLOYEE_NUMBER )) ;
```

Notes:

- In Teradata 13.10, the SHOW TABLE command does show the Primary Key and Unique constraints.
- As always, Primary Key and Unique constraints are implemented as unique indexes.
- All constraints are specified at table level with SHOW TABLE.

Altering Table Constraints

Once a table has been created, constraints may be added, dropped and in some cases, modified. The ALTER TABLE command can also be used to add new columns (up to 2048) to an existing table.

UNIQUE Constraints

Uniqueness constraints may also be added or dropped as needed. They may apply to one or more columns. Columns must be defined as NOT NULL before a uniqueness constraint may be applied to them. Uniqueness constraints are physically implemented by Teradata as unique indexes, either primary or secondary. If the specified columns do not contain data that is unique, the constraint will be rejected and an error will be returned.

Unique constraints may be dropped either by referencing their name, or by dropping the index on the specified columns.

PRIMARY KEY Constraints

Adding a **primary key constraint** to a table via ALTER TABLE will always result in the primary key being implemented as a unique secondary index (USI). This can only be done if there has not already been a primary key defined on the table.

Dropping a **primary key constraint** may be done either by dropping the named constraint or by dropping the associated index. It is not possible to drop a primary key constraint that is implemented as a primary index.

FOREIGN KEY Constraints

Foreign key constraints may be named or unnamed, however the syntax for dealing with them differs accordingly. Foreign keys may be added to a table assuming:

a.) The referenced column is defined as unique and not null
b.) The number and type of columns agree between referenced and referencing columns
c.) There are no data values in the current referencing table that are not also found in the referenced table.

Foreign keys may always be dropped with no concerns about inconsistencies. **Foreign key constraints** may be named and if so may be dropped by name.

REFERENCES constraints are covered in detail in the next module.

CHECK Constraints

CHECK constraints may be added or dropped. A **named CHECK constraint** may also be modified assuming that the existing data conforms to the new constraint, otherwise an error is returned.

Altering Table Constraints

To add constraints to a table:

```
ALTER TABLE tablename
      ADD CONSTRAINT constrname CHECK . . .
      ADD CONSTRAINT constrname UNIQUE . . .
      ADD CONSTRAINT constrname PRIMARY KEY . . .
      ADD CONSTRAINT constrname FOREIGN KEY . . .
```

To modify existing constraints:

```
ALTER TABLE tablename
      MODIFY CONSTRAINT constrname . . . ;
```

Note:
Only constraint that can be modified is a named CHECK constraint.

To drop constraints:

```
ALTER TABLE tablename
      DROP CONSTRAINT constrname ;
```

The ALTER TABLE command can also be used to add new columns (up to 2048) to an existing table.

Identity Column – Overview

This feature, also known as the ***DBS Generated Unique Primary Index*** causes the system to generate a table-level unique number for the column for every inserted row, whether for single or bulk inserts. The feature works for these types of inserts:

Single inserts
Multi-session concurrent single-statement insert requests; for example, BTEQ import
Multi-session concurrent multi-statement insert request; for example, TPump inserts
INSERT SELECT

Business Value

Identity Columns save overhead and maintenance costs because they:

Easily define an identity value that guarantees row uniqueness in a table
Avoid the performance overhead incurred by specifying a uniqueness constraint
Eliminate the need to generate unique IDs for applications outside of Teradata
If used as a Primary Index, they guarantee even data distribution which benefits performance
May be used to generate unique Primary Key values such as employee numbers, order numbers, item numbers, etc.
Comply with the ANSI Standard
Save DBA's and/or Application Developers time by automating a function that previously had to be hand coded and maintained

Business Usage

Use this feature to generate a unique number for each row as rows are added to a table. This feature is most useful if the table has no intrinsically unique column or combination of columns.

There are three reasons that you might want to use this feature:

To guarantee row uniqueness in a table
To guarantee even row distribution for a table
To optimize and simplify first port from other databases that utilize generated keys

Identity Column – Overview

Also known as a *DBS Generated Unique Primary Index:* A table-level unique number system-generated for rows as the rows are inserted in the table.

Identity Columns may be used to ...

- Guarantee row uniqueness in a table
- Guarantee even row distribution for a table
- Optimize and simplify initial port from other databases that use generated keys

Identity columns values can be assigned by the PE or the AMP.

- PE and/or AMPs reserve a pool of values and assign numbers out of their pool.
- PE assigns identity column values for single inserts and TPump.
- AMP assigns identity column values for FastLoad, MultiLoad, and SQL INSERT/SELECT operations.

Identity Columns Save Overhead/Maintenance Costs:

- Reduce need for uniqueness constraints
- Reduce manual coding tasks
- Generate unique PK values
- Comply with the ANSI Standard

Identity Column – Implementation

DBS Control setting IdColBatchSize – indicates the size of the pool of numbers to be reserved for generating numbers for a batch of rows to be bulk-inserted into a table with an identity column. The valid range of values is 1 – 1000000. The default is 100000.

Identity Column data type may be any exact numeric type. For example, an Identity column can be INTEGER, DECIMAL(18,0), BIGINT, etc.

Implicit uniqueness is guaranteed only for GENERATED ALWAYS + NO CYCLE Identity Columns. CYCLE causes the numbering to restart from MINVALUE for positive increments and MAXVALUE for negative increments after the maximum/minimum number is generated.

Performance

Initial bulk-load of an Identity Column table may create an initial performance hit as every VPROC that has rows reserves a range of numbers from DBC.IdCol and sets up its local cache entry. Thereafter, as data skew spaces out the numbers reservation, the contention should diminish. Generating Identity Column values takes a few seconds per couple of thousand rows inserted.

Process for Generating Identity Column Numbers

The system allocates identity column numbers differently depending on whether an operation is 1) a single row or USING clause-based insert or 2) an INSERT … SELECT insert.

For this type of insert operation …	Identity column numbers are cached on the …
Single row or USING clause	PE
INSERT … SELECT	AMP

When the initial batch of rows for a bulk insert arrives on a VPROC (PE or AMP), a range of numbers is first reserved before processing the rows. Each VPROC retrieves the next available value for the identity column from the new DBC.IdCol dictionary table and immediately updates this value with an amount equal to a new DBSControl setting. Once a range of numbers is reserved, the first number in the range is stored in a VPROC local Identity Column cache. Different tasks doing concurrent inserts on the same identity column table allot a number for each row being inserted and increment it in the cache. When the last reserved number is issued, the VPROC again reserves another range of numbers and updates the identity column's entry in DBC.IdCol. For example, each Parsing Engine reserves a cache of numbers. That PE owns that range. It takes the next number from the cache. When it runs out, it goes and gets another group.

Due to Teradata's parallel architecture, numbers generated do not reflect the chronological order of rows inserted. Numbering gaps can occur, exact incrementing is not guaranteed, i.e., 1000 rows inserted into the table may not be numbered from 1 to 1000. Numbering gaps can occur because the process favors scalability and performance over enforced sequential numbering.

Identity Column – Implementation

Characteristics of the IDENTITY Column feature are ...

- Implemented at column level in a CREATE TABLE statement
  - Data type may be any exact numeric type – INTEGER, DECIMAL (x,0)
  - GENERATED ALWAYS always generates a value.
  - GENERATED BY DEFAULT generates a value only when no value is specified.
- GENERATED ALWAYS + NO CYCLE implies uniqueness.
- CYCLE restarts numbering after the maximum/minimum number is generated.
- DBSControl setting indicates the number pool size to reserve for generating numbers.
  - Each Vproc may reserve 1 – 1,000,000 numbers; default is 100000.
- Numbering gaps can occur.
  - Generated numbers do not reflect row insertion sequence.
  - Exact incrementing is not guaranteed.
- Scalability and performance are favored over enforced sequential numbering.

Identity Column – Example 1

The facing page contains an example of using the GENERATED ALWAYS option.

When an Identify column is created, a row is placed into a dictionary table name DBC.IdCol. Columns in this table include:

TableID
DatabaseID
AvailValue
StartValue
MinValue
MaxValue
Increment
Cyc (Cycle)

The values of these columns for this example are:

TableID	00004B090000
DatabaseID	0000AF04
AvailValue	801001
StartValue	100001
MinValue	100001
MaxValue	2147483647
Increment	1
Cyc	N

Note: If MINVALUE is not specified, then the minimum value for an integer column will be -214783647.

If the identify column is defined without a minimum and Decimal (18, 0), then the values in DBC.IdCol are shown below.

```
CREATE TABLE Table_C
        (Cust_Number    DECIMAL (18,0) GENERATED ALWAYS AS IDENTITY,
         LName          VARCHAR(15),
         Zip_code       INTEGER);
```

The values of these columns for Table_C are:

TableID	00004D090000
DatabaseID	0000AF04
AvailValue	1
StartValue	1
MinValue	-1,000,000,000,000,000,000
MaxValue	1,000,000,000,000,000,000
Increment	1
Cyc	N

Identity Column – Example 1

Example 1: GENERATED ALWAYS AS IDENTITY

This command always generates a value. It does not cycle and does not repeat prior used values.

```
CREATE TABLE Table_A
  (Cust_Number    INTEGER GENERATED ALWAYS AS IDENTITY
                          (START WITH 100001  INCREMENT BY 1  MINVALUE 100001 NO CYCLE),
   LName          VARCHAR (15),
   Zip_code       INTEGER);
```

```
INSERT INTO Table_A SELECT c_custid, c_lname, c_zipcode FROM Customer;
```

```
SELECT * FROM Table_A ORDER BY 1;
```

Cust_Number	LName	Zip_Code
100001	Tatem	89714
100002	Kroger	98101
100003	Yang	77481
100004	Miller	45458
:	:	:
200001	Powell	57501
200002	Gordan	89714
200003	Smoothe	80002
:	:	:

- Customer has 5000 rows – new customer numbers generated are not sequentially numbered from 100001 to 105000.
- Numbering gaps can occur – exact incrementing is not guaranteed.
- INTEGER provides a limited range of values – maybe use DECIMAL(18,0)
- Default for next allocation pool is DBSControl parameter value of 100,000.

Identity Column – Example 2

The facing page contains an example of using the GENERATED BY DEFAULT option.

When an Identify column is created, a row is placed into a dictionary table name DBC.IdCol. Columns in this table include:

TableID
DatabaseID
AvailValue
StartValue
MinValue
MaxValue
Increment
Cyc (Cycle)

The values of these columns for this example are:

TableID	00004C090000
DatabaseID	0000AF04
AvailValue	920000000
StartValue	1000000000
MinValue	0
MaxValue	2147483647
Increment	-1
Cyc	N

Note: If INCREMENT BY is positive and CYCLE is specified, renumbering begins from MINVALUE when MAXVALUE is reached.

Identity Column – Example 2

Example 2: GENERATED BY DEFAULT AS IDENTITY

This option generates a value only when no value is specified for the column.

```
CREATE TABLE Table_B
  (Cust_Number   INTEGER   GENERATED BY DEFAULT AS IDENTITY
                           (START WITH 1000000000  INCREMENT BY -1  MINVALUE 0),
   LName         VARCHAR(15),
   Zip_code      INTEGER);
```

```
INSERT INTO Table_B SELECT NULL, c_lname, c_zipcode FROM Customer;
```

```
SELECT * FROM Table_B ORDER BY 1 DESC;
```

Cust_Number	LName	Zip_Code
1000000000	Tatem	89714
999999999	Kroger	98101
999999998	Yang	77481
999999997	Miller	45458
:	:	:
999900000	Powell	57501
999899999	Gordan	89714
999899998	Smoothe	80002
:	:	:

- Customer has 5000 rows – new customer numbers are generated because NULL was part of SELECT.
- If MINVALUE is not used, the minimum value for an INTEGER is -2,147,483,647.
- CYCLE option is not used – default is NO CYCLE.
- GENERATED BY DEFAULT – provides capability of copying the contents of one table with an Identity column into another.

Identity Column – Considerations

Generated Always Identity Columns typically define the Primary Index. **Define an Identity Column as the Primary Index *only if it is the primary path to the table***. For instance, if there is an Identity Column in the Product table but all the retrievals or joins are done on SKU, then SKU should be the Primary Index. If the Identity Column is also used occasionally as an access path, consider it as a Secondary Index.

Generated Default Identity Columns primarily facilitate copying data from one table with an Identity Column into another without losing the system generated values in the source table. Use a numeric type large enough to hold all the values that will ever be required. *Never* use an Identity Column to import a schema/application from another DBMS *as a substitute* for a good logical database design. Many schema choices that rely on generated Identity Columns do not optimally utilize Teradata join and access capabilities. Think carefully before using this feature as a primary schema foundation.

You can drop an identity column from an existing table, but you cannot drop just the identity column attribute and retain the column.

Limited to DECIMAL(18,0)

The maximum numeric data type ranges are DECIMAL(18,0) and NUMERIC(18,0), or approximately 1×10^{18} rows.

This is true even when the DBS Control flag MaxDecimal is set to 38 and you define an identity column with more than 18 digits of precision. For example, you can create a table with an identity column with BIGINT or DECIMAL(38,0) data types. The create table command will not fail and you will not get a warning message. However, the values generated by the identity column feature remain limited to the DECIMAL(18,0) type and size.

Restrictions

Additional restrictions on Identity Columns are listed on the facing page.

Identity Column – Considerations

Generated Always Identity Columns

- Typically define the Primary Index.
- Define as the Primary Index *only if it is the primary path.*
- If it is also used as an access path, consider it as a Secondary Index.

Generated By Default Identity Columns

- Facilitate copying data from one table into another.
- Use a numeric type large enough to hold all the values that will ever be required.
- *Never* use *as a substitute* for a good logical database design.
- May not optimally utilize Teradata join and access capabilities.

Restrictions

- A table can only have 1 Identity column.
- ALTER TABLE statement can not add an Identity Column to an existing table.
- Cannot be part of a composite primary or a composite secondary index.
- Cannot be used with Global Temporary or Volatile tables.
- Cannot be used in a join index, hash index, PPI or value-ordered index.
- Atomic UPSERTs are not supported on a table with an Identity Column as its PI.
- GENERATED ALWAYS Identity Column value updates are not supported.

Module 21: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 21: Review Questions

1. Which one of the following situations requires the use of the Transient Journal?
   a. loading a table with FastLoad
   b. DELETE all the rows in a table
   c. UPDATE all the rows in a table
   d. INSERT/SELECT into an empty table

2. What is a negative impact of updating a UPI value?

   _____

3. What are the 4 types of constraints?

   _____ _____ _____ _____

4. True or False? A primary key constraint is always implemented as a primary index.
5. True or False? A primary key constraint is always implemented as a unique index.
6. True or False? Multi-column constraints must be coded as table level constraints.
7. True or False? Only named check constraints may be modified.
8. True or False? Named primary key constraints may always be dropped if they are no longer needed.
9. True or False? Using the "START WITH 1" and "INCREMENT BY 1" options with an Identity column will provide sequential numbering with no gaps for the column.

Notes

Module 22

TERADATA
Raising Intelligence

Referential Integrity

After completing this module, you will be able to:

- **Specify how is the reference index subtable hash distributed?**
- **Specify how a reference index row can be marked as "invalid".**
- **List the differences between standard referential integrity, batch referential integrity, and soft referential integrity.**

Notes

Table of Contents

Referential Integrity 22-4
Parent-Child Relationships 22-4
Three Types of Referential Constraints 22-6
  Standard Referential Integrity 22-6
  Batch Referential Integrity 22-6
  Referential Constraints (Soft RI) 22-6
Reference Index Subtables 22-8
Reference Index Example – Add REFERENCES 22-10
  Reference Index Example – INSERTs 22-12
  Reference Index Example – UPDATEs 22-14
  Reference Index Example – DELETEs 22-16
Fixing Referential Integrity Problems 22-18
  Unresolved Reference Constraints 22-18
  Inconsistent References 22-18
Batch Referential Integrity 22-20
Soft Referential Integrity 22-22
    Customer Benefits 22-22
    Limitations 22-22
Referential Integrity Example 22-24
Referential Integrity Example (cont.) 22-26
Join Optimization with RI 22-28
  Join Optimization with RI (cont.) 22-30
Summary 22-32
Module 22: Review Questions 22-34

Referential Integrity

Referential Integrity (RI) is the term used to describe a database feature that ensures that when a non-null value is placed in a FK column that the value exists in a PK within the database. Teradata's implementation of RI also allows a FK to reference a UNIQUE, NOT NULL column(s).

Reasons to implement Referential Integrity include:

- Data integrity and consistency
- Increases development productivity – it is not necessary to code SQL statements to enforce referential constraints
- Requires fewer application programs to be written – all update activities are programmed to ensure that referential constraints are not violated.
- Improves performance – the Teradata Database chooses the most efficient method to enforce the referential constraints.
- User applications may rely on Referential Integrity for their functionality.

Referential integrity is the concept of relationships between tables based on the definition of a primary key and a foreign key. It provides for specification of columns within a referencing table that are foreign keys for columns in some other referenced table. Referenced columns must be defined as either:

- PRIMARY KEY or UNIQUE constraints
- UNIQUE indexes

Referential integrity is a reliable mechanism that prevents accidental database corruption when users execute INSERT, UPDATE, and DELETE statements.

Referential integrity states that a row cannot exist in a table with a non-null value for a referencing column if an equal value does not exist in a referenced column.

Parent-Child Relationships

The operative rule of parent-child relationships is that a child must have a parent. Any maintenance done to the table must honor the integrity of this relationship as specified in the constraint definition.

Child tables are called "referencing tables" and parent tables are called "referenced tables".

Referential Integrity

Referential Integrity: **A concept where a Foreign Key value (other than NULL) must exist within the Primary Key.**

- **Referential Integrity is a feature which ensures data integrity and consistency between primary and foreign key columns.**
- **In Teradata, a Foreign Key can reference a "parent key" which effectively is a unique index (UPI or USI).**
- **The table containing the PK is referred to as the "referenced" table.**
- **The table containing the FK is referred to as the "referencing" table.**

Primary Key (Teradata's Implementation)	**Foreign Key**
Must be unique.	**May be unique or non-unique.**
Must be NOT NULL.	**May be NULL.**
Can only be deleted or modified if no Foreign Key values reference it.	**Can be deleted at any time. Can be updated to NULL or only to a valid Primary Key value.**

Three Types of Referential Constraints

Standard Referential Integrity

Standard referential integrity is a form of referential integrity that incurs modest performance overhead because it tests, on a row-by-row basis, the referential integrity of each insert, delete, and update operation on a column with a defined referential integrity constraint. Each such operation is checked individually by AMP software, and if a violation of the specified referential integrity relationship occurs, the operation is rejected and an error message is returned to the requestor.

You cannot use utilities like FastLoad, MultiLoad, and Teradata Parallel Transporter (LOAD and UPDATE operators) on tables defined with standard referential integrity.

Batch Referential Integrity

Batch referential integrity is a form of referential integrity checking that is less expensive to enforce in terms of system resources than standard referential integrity because it is enforced as an all-or-nothing operation (the entire transaction must complete successfully) rather than on a row-by-row basis, as standard referential integrity is checked.

Batch RI also conserves system resources by not using REFERENCE index subtables.

Batch referential integrity relationships are defined by specifying the WITH CHECK OPTION phrase for a REFERENCES constraint. When you specify this phrase, the database enforces the defined RI constraint at the granularity of a single transaction or SQL statement.

You cannot use utilities like FastLoad, MultiLoad, and Teradata Parallel Transporter (LOAD and UPDATE operators) on tables defined with batch referential integrity.

Referential Constraints (Soft RI)

In some circumstances, the Optimizer is able to create significantly better query plans if certain referential relationships have been defined between tables specified in the request.

The Referential Constraint feature permits you to take advantage of these optimizations without incurring the overhead of enforcing the suggested referential constraints.

Referential Constraint (soft RI) relationships are defined by specifying the WITH NO CHECK OPTION phrase for a REFERENCES constraint. When you specify this phrase, the database does not enforce the defined RI constraint.

You can use utilities like FastLoad, MultiLoad, and Teradata Parallel Transporter (LOAD and UPDATE operators) on tables defined with soft referential integrity.

Three Types of Referential Constraints

There are three types of referential constraints that can be implemented with Teradata.

- **Referential Integrity Constraint – tests each row (inserted, deleted, or updated) for referential integrity. Also referred to as Standard RI.**
- **Batch Referential Integrity Constraint (Batch RI) – tests a batch operation (implicit SQL transaction) for referential integrity.**
- **Referential Constraint (Soft RI) – does not test for referential integrity.**

In addition to providing data integrity constraints, use of a PF-FK constraint enables optimization techniques such as Join Elimination.

- **There is no need to join the child table to the parent table if you set up an FK-PK join, unless you need some additional columns from the parent table.**

This module will cover the standard RI constraint first.

Reference Index Subtables

When a reference is created between a FK and a PK, Teradata software creates a Reference Index subtable associated with the Foreign Key. To accommodate referential constraints, a new field that describes the Foreign Key columns is added to the table header. Each referential constraint defined on a table has a reference index descriptor added to the table header that identifies the Reference Index Subtable.

The Reference Index subtable is hash distributed based on the Foreign Key value, which means the subtable row will always reside on the same AMP as the corresponding Parent UPI or USI row. Hence, all RI constraint checks can be done by the same AMP responsible for the Foreign Key value.

For each RI constraint defined on a Child table, Teradata creates a reference index subtable. This subtable is hashed distributed based on the FK value. Each row in this subtable consists of the RowID, the FK value, a Boolean Flag, and a Row Count.

- RowID – included row hash of FK value and uniqueness value
- FK value – included with the subtable row for validation
- Boolean Flag – denotes a valid or invalid reference index row, which indicates that there is no corresponding Foreign Key value in the Parent Table.
- Row Count – tracks the number of Child data rows containing that Foreign Key value.

To preserve data integrity, a RI constraint violation usually rolls back the statement and leaves a consistent database. There are two exceptions to this rule.

1) Adding a RI constraint to an a populated table
2) Revalidating a RI constraint after a table is restored

In these cases, a Reference Index subtable row for a foreign key that doesn't have a matching Parent Key (a.k.a., Primary Key) value is marked as "Invalid". Inserts, updates, and deletes are still legal operations on a table with invalid index reference rows.

When standard or batch RI is established on an already populated table, it is possible to have rows in the child table with an invalid FK value (e.g., invalid department number). The reference index subtable identifies these rows internally as invalid. Once RI is established, it is not possible to insert new rows into the child table with that same invalid value. Teradata software checks the Boolean flag reference index subtable, and if the flag is "invalid", Teradata will check the parent table to see if that value now exists in a PK row (e.g., insert a new department row into the department table). If the value still does not exist, the insert of the row with an invalid FK value will fail.

Reference Index Subtables

- **With standard RI, for each RI constraint defined on a Child table, there exists a Reference Index subtable.**
  - **Each row in this subtable consists of the following:**

    RowID – row hash and assigned uniqueness value of FK value

    FK value – used to verify row

    Boolean Flag – denotes a valid or invalid reference index row. Invalid indicates that there is no corresponding FK value in the PK column.

    Row Count – tracks the number of Child data rows containing that Foreign Key value.

- **The Reference Index subtable is hash distributed based on the FK value.**
  - **The subtable row will always reside on the same AMP as the corresponding Parent UPI or USI row.**

- **How is a reference index row marked as invalid?**

  1) **Adding a RI constraint to an already populated table**
  2) **Revalidating a RI constraint after a table is restored**

  In these cases, a Reference Index subtable row for a foreign key that doesn't have a matching PK value is marked as "Invalid".

An error table (e.g., Employee_0) will also be created that can be accessed via SQL.

Reference Index Example – Add REFERENCES

The facing page illustrates an example of adding a References constraint to a table. The examples assume a 4 AMP system.

When creating a Child table Foreign Key reference, Teradata checks to verify that the Parent key is defined as a UPI (NOT NULL) or a USI (NOT NULL). It also checks that the Foreign Key has the same data type as the corresponding Parent Key, and that the maximum number of RI constraints has not been exceeded.

For each row in the Child table that does not have a matching Parent Key (a.k.a., Primary Key) value, the Reference Index subtable row for that Foreign Key is marked as "Invalid". Additionally, a row is placed into an "error table". In this example, the error table will be named **EMPLOYEE_0**.

> **Note:** If the References constraint is created as part of the CREATE TABLE statement, then the EMPLOYEE_0 table is not built.

If three references constraints are added to the Employee table (via ALTER TABLE), the error tables are named as follows:

- Employee_0
- Employee_4
- Employee_8

Reference Index Example – Add REFERENCES

Employee

Enum	Name	Dept
PK		FK
UPI		
1	BROWN	200
2	SMITH	310
3	JONES	310
4	CLAY	400
5	PETERS	150
6	FOSTER	200
7	GRAY	310
8	BAKER	310
9	TYLER	405
10	CARR	450

Department

Dept	Dept_Name
PK	
UPI	
150	PAYROLL
200	FINANCE
310	MFG.
400	EDUCATION
450	ADMIN

```
ALTER TABLE Employee
    ADD CONSTRAINT fk1 FOREIGN KEY (Dept)
        REFERENCES Department (Dept);
```

Employee rows hash distributed on Employee.Enum (UPI)

6 FOSTER 200 8 BAKER 310	4 CLAY 400 3 JONES 310 9 TYLER 405	1 BROWN 200 7 GRAY 310	5 PETERS 150 2 SMITH 310 10 CARR 450

Department rows hash distributed on Department.Dept (UPI)

150 PAYROLL	310 MFG.	200 FINANCE 450 ADMIN	400 EDUCATION

Reference Index rows hash distributed on Employee.Dept (FK)

150	0	1	310	0	4	200	0	2	400	0	1
405	1	1				450	0	1			
FK	Boolean Flag	Count	FK	Boolean Flag	Count	FK	Boolean Flag	Count	FK	Boolean Flag	Count

Reference Index Example – INSERTs

An Insert into a Parent table (Primary Key) can never violate the referential constraint since it is perfectly legal for a Primary Key value to have no Foreign Key references. There is no need to update the reference index subtable, since it contains Foreign Key values, not Primary Key values.

The example on the facing page illustrates an INSERT of a new row into the Department table. Note there is no impact on the Reference Index subtable.

Note: If a missing Primary Key is inserted into the Parent table, no attempt is made to reset the reference index row flag to valid. An Insert or an Update of the Child table (FK) validates an invalid reference index row.

A Referential constraint check is required when an Insert into a Child table with a Foreign Key reference is done. Teradata software supports two types of Inserts.

1. A simple Insert without a sub-query is used to insert one row into the target table.

   If the FK value already exists in the Reference Index subtable and the reference index row is "valid", then the reference index row count is incremented and the data row is inserted.

   If the FK value doesn't exist in the Reference Index subtable or if it finds an invalid reference index row, Teradata software checks the Parent table's UPI or USI subtable (PK is either a UPI or USI). If the row exists in the Parent table, a new reference index row is created or an invalid reference row is marked as valid.

   A RI constraint violation results if the Parent table's UPI or USI subtable is searched and no matching Foreign Key value is found. A RI constraint violation will roll back the offending statement and return an error to the user.

2. An Insert-Select is used to insert multiple rows from a sub-query. In this case, the input rows generated by the sub-query are stored in a spool file. Instead of checking one row at time, each AMP redistributes its local spool file based on the Foreign Key values to the appropriate AMPs. When the redistribution process is completed, each receiving AMP then sorts the rows based on the Foreign Key values and combines duplicate Foreign Key values into one with a row count indicating the number of duplicates. The rest of the RI constraint check logic is identical to that described for a simple insert.

Reference Index Example – INSERTs

Employee

Enum	Name	Dept
PK		FK
UPI		
1	BROWN	200
2	SMITH	310
3	JONES	310
4	CLAY	400
5	PETERS	150
6	FOSTER	200
7	GRAY	310
8	BAKER	310
9	TYLER	405
10	CARR	450
11	BENCH	310
12	WARNER	500

Department

Dept	Dept_Name
PK	
UPI	
150	PAYROLL
200	FINANCE
310	MFG.
400	EDUCATION
405	MKTG.
450	ADMIN
500	SALES

```
INSERT INTO Department VALUES (500, 'SALES');
INSERT INTO Employee VALUES (11, 'BENCH', 310);
INSERT INTO Employee VALUES (12, 'WARNER', 500);
INSERT INTO Department VALUES (405, 'MKTG.');
```

Employee rows hash distributed on Employee.Enum (UPI)

6	FOSTER	200
8	BAKER	310
11	BENCH	310

4	CLAY	400
3	JONES	310
9	TYLER	405

1	BROWN	200
7	GRAY	310
12	WARNER	500

5	PETERS	150
2	SMITH	310
10	CARR	450

Department rows hash distributed on Department.Dept (UPI)

150	PAYROLL
405	MKTG.

310	MFG.
500	SALES

200	FINANCE
450	ADMIN

400	EDUCATION

Reference Index rows hash distributed on Employee.Dept (FK)

FK	Boolean Flag	Count
150	0	1
405	1	1

FK	Boolean Flag	Count
310	0	4 → 5
500	0	1

FK	Boolean Flag	Count
200	0	2
450	0	1

FK	Boolean Flag	Count
400	0	1

Note: The RI subtable row for 405 is not affected by the INSERT of Department 405.

Reference Index Example – UPDATEs

A Referential constraint check is required when an Update is executed on a Child table column with a Foreign Key reference. The facing page illustrates two examples of updating a Foreign Key value. The first example shows fixing a RI constraint violation. The second example shows changing a valid FK value to a different FK value.

If the column being updated is included in either a Foreign Key or a Parent Key, then RI constraint checks are required. If the update is to a Child table, then updating the Foreign Key value involves two Reference Index subtable operations. First, the row count in the reference index row is updated to reflect one less row containing the old Foreign Key value. Second, Teradata software searches the reference index subtable for the new Foreign Key value. This process is similar to the Insert statement described earlier.

> Note: In order to handle self-references (within the same table), updating a Parent Key column requires that the RI constraint check is deferred until the end of the Update statement.

If the updated FK value already exists in the Reference Index subtable and the reference index row is "valid", then the reference index row count is incremented and the data row is inserted.

If the updated FK value doesn't exist in the Reference Index subtable or if it finds an invalid reference index row, Teradata software checks the Parent table's UPI or USI subtable (PK is either a UPI or USI). If the row exists in the Parent table, a new reference index row is created or the invalid reference row is marked as valid.

A RI constraint violation results if the Parent table's UPI or USI subtable is searched and no matching Foreign Key value is found. A RI constraint violation will roll back the offending statement and return an error to the user.

Reference Index Example – UPDATEs

Employee

Enum	Name	Dept
PK		FK
UPI		
1	BROWN	200
2	SMITH	310
3	JONES	310
4	CLAY	400
5	PETERS	150
6	FOSTER	200
7	GRAY	310
8	BAKER	310
9	TYLER	405 → 400
10	CARR	450
11	BENCH	310 → 500
12	WARNER	500

Department

Dept	Dept_Name
PK	
UPI	
150	PAYROLL
200	FINANCE
310	MFG.
400	EDUCATION
405	MKTG.
450	ADMIN
500	SALES

UPDATE Employee SET Dept = 400 WHERE Enum = 9;
UPDATE Employee SET Dept = 500 WHERE Enum = 11;

Employee rows hash distributed on Employee.Enum (UPI)

6 FOSTER 200	4 CLAY 400	1 BROWN 200	5 PETERS 150
8 BAKER 310	3 JONES 310	7 GRAY 310	2 SMITH 310
11 BENCH 500	9 TYLER 400	12 WARNER 500	10 CARR 450

Department rows hash distributed on Department.Dept (UPI)

150 PAYROLL 405 MKTG.	310 MFG. 500 SALES	200 FINANCE 450 ADMIN	400 EDUCATION

Reference Index rows hash distributed on Employee.Dept (FK)

FK	Boolean Flag	Count		FK	Boolean Flag	Count		FK	Boolean Flag	Count		FK	Boolean Flag	Count	
150	0	1		310	0	5 → 4		200	0	2		400	0	1 → 2	
~~405~~	~~1~~ (deleted)	~~1~~		500	0	1 → 2		450	0	1					

Reference Index Example – DELETEs

If the table being deleted is a Parent table and the Parent Key value(s) are referenced by a Foreign Key, then the delete is not allowed. If the target is a Child table, the RI constraints can never be violated. However, deleting Foreign Key values still requires that the Reference Index subtable be updated.

If the target is a Child table, as each data row is deleted, the corresponding reference index row is also updated. This requires the AMP (that has the data row to delete) to send a message to the AMP with the reference index row so the AMP can decrement the reference index row count. As long as the database state is consistent, the matching Foreign Key value should always be found in the Reference Index subtable. The Boolean flag in the reference index row has no significance in this case. If the row count becomes zero, the reference index row is deleted.

A special case is deleting rows in a table that has self-references in it. For self- references, the RI constraint check cannot be done as each row is being deleted. The self-reference constraint check causes the AMP not to immediately respond to the user with a RI constraint violation. Instead, during the first pass of RI checks, the AMP inserts the offending Parent Key values into a spool file. When all of the AMPs complete deleting all the target data rows, each AMP will then undertake a second pass of checks by traversing through the spool file to make sure that none of the Parent Key values are still be referenced by any Foreign Key values.

Deleting all of the data rows from a table is also a special delete case. If the target of a delete is a Child table, all Teradata software has to do is delete the entire reference index subtable and the rows in the Child table. If the target is a Parent table, the Parser generates a synchronous abort test step. This is an all-AMPs step that is used to test whether or not the subtable is empty. A RI constraint violation results if the reference index subtable is not empty.

Reference Index Example – DELETEs

Employee

Enum	Name	Dept	
PK		FK	
UPI			
1	BROWN	200	
2	SMITH	310	
3	JONES	310	
4	CLAY	400	
~~5~~	~~PETERS~~	~~150~~	(deleted)
~~6~~	~~FOSTER~~	~~200~~	(deleted)
7	GRAY	310	
8	BAKER	310	
9	TYLER	400	
10	CARR	450	
11	BENCH	500	
12	WARNER	500	

Department

Dept	Dept_Name
PK	
UPI	
150	PAYROLL
200	FINANCE
310	MFG.
400	EDUCATION
405	MKTG.
450	ADMIN
500	SALES

DELETE Employee WHERE Enum = 5;
DELETE Employee WHERE Enum = 6;
DELETE Department WHERE Dept = 310; (fails)

Employee rows hash distributed on Employee.Enum (UPI)

~~6 FOSTER 200~~ (deleted)	4 CLAY 400	1 BROWN 200	~~5 PETERS 150~~ (deleted)
8 BAKER 310	3 JONES 310	7 GRAY 310	2 SMITH 310
11 BENCH 500	9 TYLER 400	12 WARNER 500	10 CARR 450

Department rows hash distributed on Department.Dept (UPI)

150 PAYROLL 405 MKTG.	310 MFG. 500 SALES	200 FINANCE 450 ADMIN	400 EDUCATION

Reference Index rows hash distributed on Employee.Dept (FK)

~~150 0 1~~ (deleted)	310 0 4 500 0 2	200 0 2 ➔1 450 0 1	400 0 2
FK Boolean Flag Count	FK Boolean Flag Count	FK Boolean Flag Count	FK Boolean Flag Count

Fixing Referential Integrity Problems

Altering a table to add foreign key constraints will appear to be a successful operation even if unreferenced values are found. The rows containing the unreferenced values will be written to an error table; however the operation will appear to be successful. It is thus important to check for the existence of the error table following the ALTER TABLE command. Unreferenced values are also referred to as "invalid rows".

There are multiple ways to "clean up" these kinds of inconsistencies, as described on the facing page.

Additional states of reference constraints are listed below. These conditions are described in more detail in the database administration portion of this course.

Unresolved Reference Constraints

Unresolved reference constraints occur when the FK exists, but the PK does not. The DBC.Databases2 view provides a count of unresolved reference constraints for any tables within the database. Situations when an unresolved reference constraint occurs are:

- Creating a table with a Foreign Key before creating the table with the Parent Key.
- Restoring a table with a Foreign Key and the Parent Key table does not exist or hasn't been restored.

Inconsistent References

When either the child or parent table is restored, the entire reference constraint for the child table is marked as inconsistent. Both the FK and the PK exist, but the reference constraint is marked as inconsistent. At this point, no inserts, updates, deletes or table changes are allowed. The DBC.All_RI_Children view provides information about reference constraints.

This marking disallows all maintenance activity against the table. If it is desired to perform maintenance on the table, the table may be marked useable by dropping the inconsistent references flag. This is done by using the following command:

ALTER TABLE <child_tablename> DROP INCONSISTENT REFERENCES;

It becomes the user's responsibility to insure the validity of the referencing information until such time as the Foreign Key constraint is reinstated.

The **REFERENCES** privilege is required to perform the **DROP INCONSISTENT REFERENCES** command. It has no effect on existing data in the table.

Fixing Referential Integrity Problems

Assume an employee is assigned to department 405 which doesn't exist in Department.

Submit

```
ALTER TABLE Employee ADD CONSTRAINT fk1
      FOREIGN KEY (Dept_Number) REFERENCES Department (Dept_Number);
```

Results

```
ALTER TABLE is successful.
      Table Employee_0 is created with the single row for 405.
      User is responsible for fixing referencing table data.
```

Fix Option 1 – Assign a different department

```
UPDATE Employee SET Dept_Number = 400  WHERE Employee_Number = 9;
```

Fix Option 2 – Assign a NULL department number

```
UPDATE Employee SET Dept_Number = NULL
      WHERE Employee_Number IN (SELECT Employee_Number FROM Employee_0);
```

Fix Option 3 – Delete the problem Employee(s)

```
DELETE FROM Employee
      WHERE Employee_Number IN (SELECT Employee_Number FROM Employee_0);
```

Fix Option 4 – Create the required department

```
INSERT INTO Department VALUES (405, 'MKTG.');
```

Batch Referential Integrity

Batch referential integrity is a form of referential integrity checking that is less expensive to enforce in terms of system resources than standard referential integrity because it is enforced as an all-or-nothing operation (the entire transaction must complete successfully) rather than on a row-by-row basis, as standard referential integrity is checked.

Batch RI also conserves system resources by not using REFERENCE index subtables.

Batch referential integrity relationships are defined by specifying the WITH CHECK OPTION phrase for a REFERENCES constraint. When you specify this phrase, the database enforces the defined RI constraint at the granularity of a single transaction or SQL statement.

This form of RI is tested by joining the relevant parent and child table rows. If there is an inconsistency in the result, then the system rolls back the entire transaction.

In some situations, there can be a tradeoff for this enhanced performance because when an RI violation is detected, the entire transaction rolls back instead of one integrity-breaching row.

In many instances, this is no different than the case for standard RI because for most transactions, only one row is involved.

The difference for batch RI is situations in which multiple update operations are involved: for example, an INSERT ... SELECT operation involving thousands or even millions of rows. This would be an expensive operation to have to roll back, cleanse, and then rerun.

Similar very large batch RI rollbacks can occur with ALTER TABLE and CREATE TABLE statements whose referential integrity relationships do not verify.

Because of its all-or-none nature, batch RI is best used only for tables whose normal workloads you can be very confident are not going to cause RI violations.

Notes:

- For populated tables, if the FK has NULLs or FK violations, the ALTER TABLE will fail and no indication of which rows caused the error (Error 3513: RI Violation).

- After the Batch RI constraint is validated (FK doesn't have invalid values or NULLs, you can insert rows into the child table with a FK of NULL.

- You cannot use bulk data loading utilities like FastLoad, MultiLoad, or Teradata Parallel Transporter (LOAD and UPDATE operators) on tables defined with batch referential integrity.

Batch Referential Integrity

ALTER TABLE Employee ADD CONSTRAINT fk1 FOREIGN KEY (Dept_Number)
REFERENCES WITH CHECK OPTION Department (Dept_Number);

Characteristics

- Tests entire insert, delete, or update operation (implicit transaction) for RI. If violations are detected, the entire request is aborted and rolled back.
  - This form of RI is tested internally by joining the relevant parent and child table rows.
  - If there is an inconsistency in the result, then the system rolls back the entire transaction. This rollback could potentially be time-consuming.
- User-specified Referential Integrity constraints that are enforced by Teradata.
  - Enables optimization techniques such as Join Elimination.
- Batch RI is best used only for tables where you are confident there are not going to be RI violations with the normal workloads.

Limitations

- For populated tables, if the FK has violations (e.g., invalid values), the ALTER TABLE will fail and there is no indication of which rows caused the error (Error 3513: RI Violation).

Soft Referential Integrity

This feature (starting with Teradata Release V2R5) enables optimization techniques such as Join Elimination. It provides a mechanism to allow user-specified Referential Integrity (RI) constraints that are not enforced by the database.

The benefit of Soft RI is that there is no overhead of integrity-checking. The reason there is no overhead with soft RI is that Teradata never attempts to validate the relationships when data is updated. Soft RI means you trust the ETL to guarantee that the relationships always have integrity. The optimizer simply accepts, trusts, that the integrity is there, but the database never enforces it. The value of using Soft RI is that the optimizer can sometimes make more efficient query plans, because it may, under some conditions, be possible to NOT access one of the tables at all, based on the relationships in the database.

This is only something useful to sites that have an extremely high level of data integrity and can guarantee that relationships between tables are always kept in sync.

Customer Benefits

Performance improvement for SQL from tools and applications that use join views, or specify extra joins in the SQL. The Optimizer can use RI to eliminate unneeded joins.

Soft RI allows the constraint to be available to the optimizer without the cost of maintaining and checking the RI in the database.

FastLoad and MultiLoad can be used to INSERT/UPDATE into tables specified with Soft RI (unlike tables that have regular RI constraints defined on them).

You may not want to put RI on a FK-PK relationship because of the cost to check and maintain; this would be just like maintaining an index. This is a mechanism to declare a RI constraint and that it is true. Soft part is that it will not be checked by the system. The optimizer will use knowledge of RI constraint when optimizing plans without the cost of maintaining the information.

Use the REFERENCES WITH NO CHECK OPTION in the syntax to define Soft RI.

Limitations

If user promises RI is true and inserts data that is not true, there is potential that the query could get the “wrong” result compared to a table with RI that is checked. If correctness is promised to the system by using Soft RI, and you don’t get what you expected, the system is working as designed.

Soft Referential Integrity

```
ALTER TABLE Employee ADD CONSTRAINT fk1 FOREIGN KEY (Dept_Number)
        REFERENCES WITH NO CHECK OPTION Department (Dept_Number);
```

Characteristics

- Allows user-specified Referential Integrity constraints not enforced by Teradata.
  - Enables optimization techniques such as Join Elimination.
  - There is no need to join the child table to the parent table if you set up an FK-PK join, unless you need some additional columns from the parent table.
- Tables with Soft RI defined can be loaded with FastLoad and MultiLoad.

Limitations

- It is possible to insert data that does not comply with the soft RI constraint.
- Queries could get the “wrong” result compared to a table with RI that is checked.

Notes

- No Reference Index subtable is created.
- No error table is created.

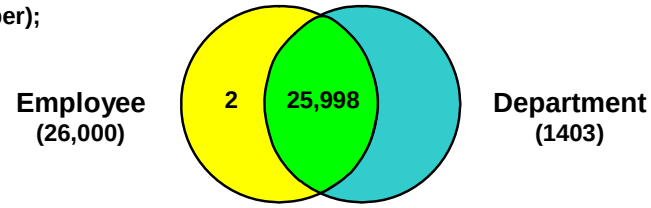
Referential Integrity Example

The facing page contains the CREATE TABLE statements for the Employee and Department tables that will be used in the various RI examples.

Referential Integrity Example

```
CREATE SET TABLE Employee
    ( Employee_Number      INTEGER NOT NULL,
      Location_Number      INTEGER,
      Dept_Number          INTEGER,
      Emp_Mgr_Number       INTEGER,
      Job_Code             INTEGER,
      Last_Name            CHAR(20)
      First_Name           VARCHAR(20)
      Salary_Amount        DECIMAL(10,2) )
  UNIQUE PRIMARY INDEX
    (Employee_Number);
```

```
CREATE SET TABLE Department
    ( Dept_Number          INTEGER NOT NULL,
      Dept_Name            CHAR(20) NOT NULL,
      Dept_Mgr_Number      INTEGER,
      Budget_Amount        DECIMAL(10,2) )
  UNIQUE PRIMARY INDEX
    (Dept_Number);
```



26,000 Employees and 1403 departments; 25,998 Employees with valid departments.

2 employees with invalid department numbers (998 and 999 are invalid).

To establish Batch RI, the Employee rows with department numbers 998 and 999 have the department numbers set to 1001 (a valid department number).

Referential Integrity Example (cont.)

The facing page contains the CREATE VIEW statement for the EmpDept_V that joins the Employee and Department tables. This view will be used in the various RI examples.

The ALTER TABLE commands to add the references constraints to the three Employee tables are:

```
ALTER TABLE Employee
    ADD CONSTRAINT fk1 FOREIGN KEY (Dept_Number)
    REFERENCES Department (Dept_Number);

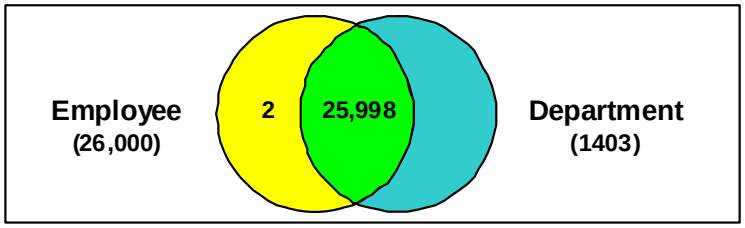
ALTER TABLE Employee
    ADD CONSTRAINT fk1 FOREIGN KEY (Dept_Number)
    REFERENCES WITH CHECK OPTION Department (Dept_Number);

ALTER TABLE Employee
    ADD CONSTRAINT fk1 FOREIGN KEY (Dept_Number)
    REFERENCES WITH NO CHECK OPTION Department (Dept_Number);
```

Referential Integrity Example (cont.)

```
REPLACE VIEW    EmpDept_V
AS SELECT       Employee_Number, Last_Name, First_Name, E.Dept_Number, Dept_Name
FROM            Employee E
INNER JOIN      Department D
ON              E.Dept_Number = D.Dept_Number;
```

EmpDept_V




```
SELECT    Employee_Number, Last_Name, Dept_Number
FROM      EmpDept_V;
```

The tables are NOT joined if a reference constraint exists.

```
SELECT    Employee_Number, Last_Name, Dept_Number, Dept_Name
FROM      EmpDept_V;
```

The tables are joined because columns are requested from both tables.

Join Optimization with RI

The facing page identifies the results of a SELECT from the EmpDept_V view based on four different environments.

- No references constraint between the Employee and Department tables.
- Standard references constraint between the Employee and Department tables.
- References with check option (batch RI) between the Employee and Department tables.
- References with no check option (soft RI) between the Employee and Department tables.

In all cases, Employee rows with a valid department number are returned.
In all cases, Employee rows with a NULL department number are not returned.

Join Optimization with RI

SELECT Employee_Number, Last_Name, Dept_Number FROM EmpDept_V;

Without a references constraint, the two tables are joined together (INNER JOIN).

- **Employees with invalid department numbers are not returned.**

With a Standard RI constraint, the two tables are not joined together.

- **When the RI constraint is created, the Employee_0 table contains rows for invalid departments (998 and 999). It is the user responsibility to "fix" the Employee table.**
  - **If the Employee table still contains Employees with invalid department numbers, then the invalid Employee rows are returned.**
  - **If the Employee table has been "fixed", then there will be no invalid Employee rows and the SELECT will only return valid rows.**

With a Soft RI constraint, the two tables are not joined together.

- **Employee rows with invalid (e.g., 998 and 999) department numbers are returned.**

With a Batch RI constraint, the two tables are not joined together.

- **Since the Employee table will not contain any invalid department numbers (e.g., 999), no invalid Employee rows will be returned.**

Join Optimization with RI (cont.)

Without Referential Integrity, the two tables will be joined together.

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.D.
2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.E.
3) We lock TFACT.D in view EmpDept_V for read, and we lock TFACT.E in view EmpDept_V for read.
4) We do an all-AMPs RETRIEVE step from TFACT.D in view EmpDept_V by way of an all-rows scan with no residual conditions into Spool 3 (all_amps), which is duplicated on all AMPs. The size of Spool 3 is estimated with high confidence to be 19,642 rows (726,754 bytes). The estimated time for this step is 0.02 seconds.
5) We do an all-AMPs JOIN step from Spool 3 (Last Use) by way of an all-rows scan, which is joined to TFACT.E in view EmpDept_V by way of an all-rows scan. Spool 3 and TFACT.E are joined using a single partition hash_ join, with a join condition of ("TFACT.E.Dept_Number = Dept_Number"). The result goes into Spool 2 (group_amps), which is built locally on the AMPs. The size of Spool 2 is estimated with low confidence to be 26,000 rows (1,274,000 bytes). The estimated time for this step is 0.11 seconds.
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 2 are sent back to the user as the result of statement 1. The total estimated time is 0.13 seconds.

When a reference constraint is defined between the two tables (standard RI, batch, or soft RI), the join between the two tables is optimized (effectively eliminated) if all of the columns referenced are in the Employee table.

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.E.
2) Next, we lock TFACT.E in view EmpDept_V for read.
3) We do an all-AMPs RETRIEVE step from TFACT.E in view EmpDept_V by way of an all-rows scan with a condition of ("NOT (TFACT.E in view EmpDept_V.Dept_Number IS NULL)") into Spool 2 (group_amps), which is built locally on the AMPs. The size of Spool 2 is estimated with high confidence to be 26,000 rows (1,274,000 bytes). The estimated time for this step is 0.06 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 2 are sent back to the user as the result of statement 1. The total estimated time is 0.06 seconds.

Join Optimization with RI (cont.)

EXPLAIN SELECT Employee_Number, Last_Name, Dept_Number FROM EmpDept_V;

Without a references constraint, the two tables are joined together.

4) We do an all-AMPs RETRIEVE step from TFACT.D in view EmpDept_V by way of an all-rows scan with no residual conditions into Spool 3 (all_amps), which is duplicated on all AMPs. The size of Spool 3 is estimated with high confidence to be 19,642 rows (726,754 bytes). The estimated time for this step is 0.02 seconds.
5) We do an all-AMPs JOIN step from Spool 3 (Last Use) by way of an all-rows scan, which is joined to TFACT.E in view EmpDept_V by way of an all-rows scan. Spool 3 and TFACT.E are joined using a single partition hash_ join, with a join condition of ("TFACT.E.Dept_Number = Dept_Number"). The result goes into Spool 2 (group_amps), which is built locally on the AMPs. The size of Spool 2 is estimated with low confidence to be 26,000 rows (1,274,000 bytes). The estimated time for this step is 0.11 seconds.

With a references constraint (any of three types), the two tables are not joined together.

3) We do an all-AMPs RETRIEVE step from TFACT04.E in view EmpDept_V by way of an all-rows scan with a condition of ("NOT (TFACT.E in view EmpDept_V.Dept_Number IS NULL)") into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with high confidence to be 26,000 (1,274,000 bytes) rows. The estimated time for this step is 0.06 seconds.

Summary

The facing page summarizes some important concepts regarding the three types of Referential Integrity that can be implemented by Teradata.

Each of the three types of RI has its own application:

- For all types, some queries may be optimized - unnecessary joins eliminated.
- For Standard RI and Batch RI, the parent key and foreign key relationships are validated when inserts, updates, and deletes are executed against the parent or foreign key tables.
- For row-by-row (ad hoc) inserts, updates, and deletes into a child table, the performance for standard RI and Batch RI is similar.

Standard Referential Integrity

- When established on already populated tables, Standard RI establishes RI and generates an error table (e.g., table_0) for rows with invalid foreign key values.
- For updates and deletes against the parent key column(s), the performance for standard RI will be better than Batch RI. Batch RI has to join to the foreign key across all the AMPs.
- Standard RI creates and uses a reference index subtable which requires physical disk space.

Batch Referential Integrity

- When established on already populated tables, if there are invalid FK values, Batch RI fails and does not generate an error table.
- For INSERT/SELECT (e.g., staging table) into a child table, batch RI will be faster than standard RI.

Soft Referential Integrity

- Does not test for RI. The software will assume that the user has somehow enforced Referential Integrity.
- The primary reason for Soft RI is that some queries may be optimized (unnecessary joins eliminated) without the overhead of maintaining referential integrity.

Summary

REFERENTIAL CONSTRAINT TYPE	DDL DEFINITION REFERENCES	ENFORCES RI	LEVEL OF RI ENFORCEMENT	USES RI SUBTABLE	CREATES ERROR TABLE
Referential Integrity Constraint (Standard RI)	REFERENCES	Yes	Row	Yes	Yes
Batch Referential Integrity Constraint	REFERENCES WITH CHECK OPTION	Yes	Implicit	No	No
Referential Constraint (Soft RI)	REFERENCES WITH NO CHECK OPTION	No	None	No	No

Notes:

- **FastLoad and MultiLoad cannot load data into tables defined with standard or batch referential integrity.**
- **For row-by-row (ad hoc) inserts, updates, and deletes into a child table, the performance for standard RI and Batch RI is similar.**
- **For updates and deletes against the parent key column(s), the performance for standard RI will be better than Batch RI. Batch RI has to join to the foreign key across all the AMPs.**
- **For INSERT/SELECT (e.g., staging table) into a child table, batch RI will be faster than standard RI.**

Module 22: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 22: Review Questions

1. Which of the following rules is not one of the relational modeling rules for a Primary Key?
   a. Must not contain NULL values
   b. Must be unique
   c. Must consist of a single column
   d. Must not change

2. Which choice cannot be referenced by a Foreign Key?
   a. Column defined as a USI with NOT NULL
   b. Column defined as UNIQUE with NOT NULL
   c. Column defined as a NUPI with NOT NULL
   d. Column defined as a Primary Key with NOT NULL

3. True or False. A reference index subtable is only created for standard (or full) referential integrity.

4. How is the reference index subtable hash distributed?

5. How can a reference index row be marked as "invalid"?

Notes

Module 23

TERADATA
Raising Intelligence

Sizing

After completing this module, you will be able to:

- **Determine column sizing requirements based on chosen data type.**
- **Determine physical table row size.**
- **Determine table sizing requirements via estimates and empirical evidence.**
- **Determine index sizing requirements via estimates and empirical evidence**

Notes

Table of Contents

General Row Format 23-4
Presence Bits 23-6
NULL and COMPRESS 23-8
Multi-Value Compression 23-10
Implementing Multi-Value Compression 23-12
ALTER TABLE and Compression 23-14
Algorithmic Compression Example 23-18
Detailed Row Format 23-20
Multi-Value Compression vs. VARCHAR 23-22
Teradata Compression Comparison 23-24
Sizing a Data Row Considerations 23-26
Teradata Data Types 23-28
INTEGER Data Types 23-30
DECIMAL and FLOAT Data Types 23-32
NUMBER Data Type (14.0) 23-34
DATE and TIME Data Types 23-36
CHARACTER Data Types 23-38
Character Sets 23-40
BYTE Data Types 23-42
Large Object Data Types 23-44
Variable Column Offsets 23-46
Row Size Calculation Form 23-48
Example: Sizing a Row 23-50
Row Size Exercise 23-54
Sizing Tables and Indexes 23-56
Table Headers 23-58
Sizing a Data Table 23-60
Table Sizing Exercise 23-62
Estimating the Size of a USI Subtable 23-64
Estimating the Size of a NUSI Subtable 23-66
Estimating the Size of a Reference Index Subtable 23-68
Index Sizing Exercise 23-70
Other Sizing Techniques 23-72
Empirical Sizing 23-74
Collect Demographics Command 23-76
Collect Demographics Example 23-78
Spool Space 23-80
Release of Spool 23-82
System Sizing Exercise 23-84
Sizing Summary 23-86
Module 23: Review Questions 23-88
Lab Exercise 23-1 23-90
Lab Exercise 23-2 23-92
Lab Exercise 23-3 (optional) 23-96

General Row Format

The diagram on the facing page represents a typical row. A single row can be up to approximately 64 KB in length. :

Prior to Teradata 13.10, the row structure for 64-bit systems differs from those of 32-bit systems by having additional pad byte fields to ensure row alignment on an 8-byte boundary. To ensure that no unaligned accesses occur with 64-bit systems, the system adds pad bytes within each row to align sections of a row on 8-byte boundaries in both memory and disk. This row format is called **Aligned Row Format**. For 64-bit systems upgraded to Teradata 13.10 (no sysinit), rows are still aligned on 8-byte row boundaries.

Starting with 13.10, new reinitialized systems, the rows are aligned on 2 byte boundaries. This new format is called **Packed64 Format**.

- The illustration on the facing page assumes the Packed64 Format and this course will assume this format unless specifically noted.

- Rows for 32-bit systems are aligned on a 2-byte boundary.

There are three general categories of table columns, which are stored in the row in the order listed:

1 – Fixed Length Columns – this section contains all the fixed length data columns which are not compressible (it is variable in length). Teradata takes the liberty of rearranging the order of data columns within a row. This has no effect on the order in which those columns can be returned to the user. Storage and display are separate issues.

2 – Uncompressed data for Compressible Columns – this section is composed of those data columns that are compressible but are neither compressed nor NULL. It is variable in length.

Includes both value-compressed and algorithmically-compressed data and can be stored in any order.

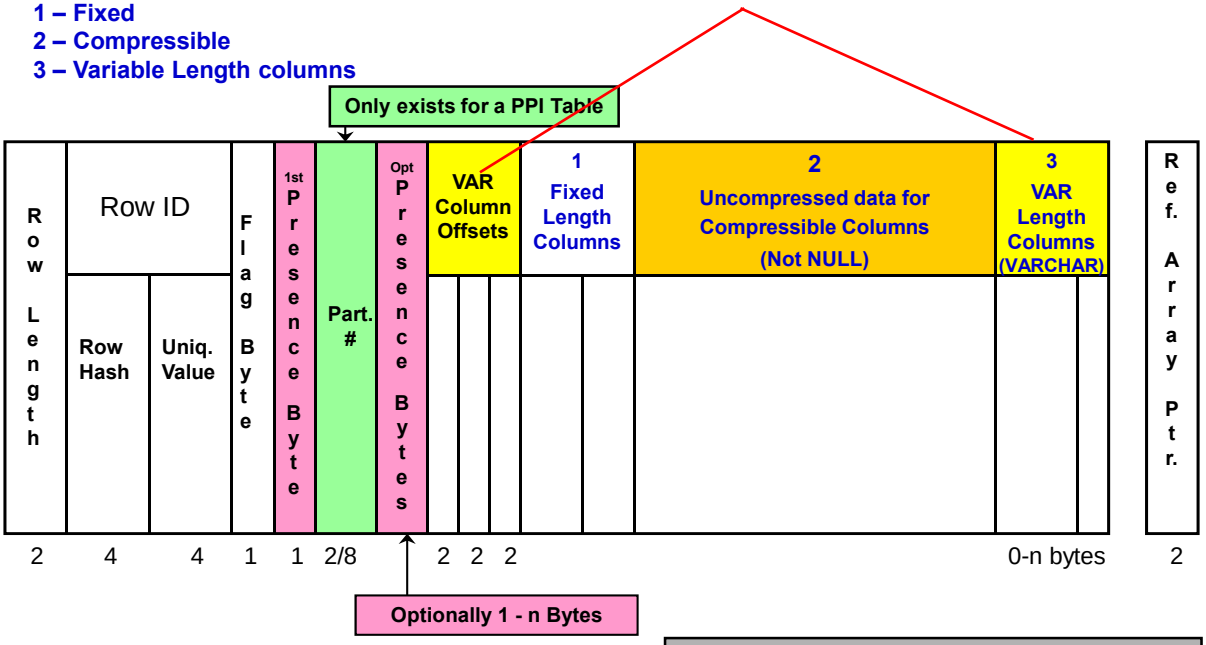
3 – VARCHAR, VARBYTE, VARGRAPHIC, or NUMBER Columns that do not have compression defined – this section contains all the variable length data columns in the row. This section will only be present when variable length columns are declared. The Column Offsets indicate exactly where in this section the variable length columns start.

Storage space is allocated in the row depending on the size of the column.

General Row Format

Data within a row generally falls in one of three categories:

- **1 – Fixed**
- **2 – Compressible**
- **3 – Variable Length columns**



Note: Flag Byte = x'00' NPPI row
x'80' for PPI row

SELECT * returns columns in the order declared in the CREATE TABLE.

Presence Bits

Each row contains at least one Presence Byte. These are used to hold the Presence Bits that indicate the status of nullable and/or compressible rows. Since nullable and compressible columns are defined at the table level, each row in a table will have the same number of Presence Bits.

The first bit of the initial Presence Byte in each row is always set to 1 (see the gray area in the diagram on the facing page) which leaves 7 bits left to hold Presence Bit values (all Presence Bytes other than the first one in a row can hold 8 Presence Bits). This is illustrated on the facing page.

Every data column will have 0 to 10 presence bits.

- A column that is neither nullable nor compressible will have none.
- A column that is either nullable or compressible, but not both, will have 1.
- A column that is nullable and only nulls are compressed will have 1.
- A column that is both nullable and compressible on a value will have 2.
- As more values are compressed (up to 255), there may be as many as 9 presence bits allocated for a column.
- An algorithmically compressed (ALC) column adds an extra bit to indicate whether the column is algorithmically compressed or not, as follows:

When a table is created (CREATE TABLE), the default is that all columns are nullable. Unless otherwise specified, there can be “missing” or “unknown” values in every column. NULL is not the same as Blank, Zero, -1, High Value or other conventions used on host systems to represent missing or unknown data.

There are many cases where this default is acceptable and you will want to allow NULL values. For example, if you have a customer table with a column for an alternate phone number, you need to allow NULLs. Since some customers would not have an alternate phone number, making this an NN column would not be good design. The presence of NULLs does not affect your ability to create an index since any index may contain NULLs.

However, there are occasions when you will want to specify that NULL values are not allowed in certain columns. Do this by issuing the **NOT NULL** clause within the CREATE TABLE command. Three cases where you would want to do this are:

- All Primary Key columns require NOT NULL.
- All UNIQUE columns require NOT NULL.
- All columns where a known data value is required (NN columns).

Nullable Columns

If a column is nullable, there will be one Presence Bit in each row to indicate this fact. This bit will be 0 if the value in the column for that row is NULL and 1 if it is any other value.

Presence Bits

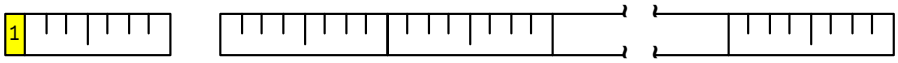
The default for all columns is to allows NULLs (Nullable) and NOT have compression.

- NULLs are not automatically compressed. By default, space is allocated for NULL.

Presence Bits

- Based on the column attributes, Teradata may need 0 to 10 presence bits to represent a NULL and/or compressed data storage (including ALC) for a column in a row.
- Up to 255 different data values (plus NULL) can be compressed for a single column – this type of compression is referred to as Multi-Value Compression.

Presence Bytes to support NULL, Multi-Value Compression, and Algorithmic Compression



1st Byte — One or More Additional Whole Bytes

Column Attribute	# of Bits	Description
NOT NULL	0	No NULLs; no values are compressed
Nullable	1	Allows NULLs; nothing is compressed
Nullable COMPRESS	1	Allows NULLs; only nulls are compressed
NOT NULL COMPRESS ('value')	1	No NULLs; compresses 1 value
Nullable COMPRESS ('value')	2	Allows NULLS; compresses nulls and 1 value
NOT NULL COMPRESS ('value1', 'value2', …)	2 – 8	No NULLs; compresses multiple values
Nullable COMPRESS ('value1', 'value2', …)	3 – 9	Allows NULLs; compresses nulls and values

NULL and COMPRESS

The default for all columns is nullable and not compressible which means that, unless otherwise specified, NULL values are allowed and Teradata will not automatically compress columns no matter what values are in them. You can override the default by using the **COMPRESS** clause at the column level when creating (or altering) a table.

The COMPRESS clause works in two different ways:

- When issued by itself (without a value or values), COMPRESS causes all NULL values for that column to be compressed to zero space.

- When issued with an argument (e.g., COMPRESS "constant"), the COMPRESS clause will compress every occurrence of "constant" in that column to zero space as well as cause every NULL value to be compressed.

The CREATE TABLE example on the facing page causes the system to compress every occurrence of the value "Australia" in the Country column to zero space as well as compress every NULL value in the Country column to zero space. This will save 20 bytes of space for every row where the employee had no country, or where the employee is associated with the country Australia. Compression is case-sensitive. Australia will be compressed in this example, but AUSTRALIA will not be compressed.

If you know in advance which columns will be accessed most frequently, you should declare compressible columns in decreasing order for possibly a small performance improvement.

You **CANNOT** compress the following:

- Components of the primary index
- Identity columns
- Volatile table columns
- Derived table columns
- Referencing and referenced columns with Standard RI cannot be compressed. Batch and Soft RI is allowed with compressed columns.
- Columns defined with a UDT, Period, Geospatial, BLOB, or CLOB data type.

Teradata 13.10 Enhancements

Starting with Teradata 13.10, there have been numerous compression enhancements.

- VARCHAR, VARBYTE, and VARGRAPHIC columns can also be compressed. Before 13.10, only fixed width columns can be compressed.
- Compress up to 510 characters for a value. Before 13.10, limit is 255 characters.

NULL and COMPRESS

- COMPRESS 'value(s)' compresses NULL and 'value(s)' to zero space.
  - Compression is case-sensitive.

How many bits are allocated?

```
CREATE TABLE  Employee
 ( emp_num       INTEGER      NOT NULL,      ---------->  ____
   dept_num      INTEGER      COMPRESS,      ---------->  ____
   country       CHAR(20)     COMPRESS 'Australia',  -->  ____
                 :
```

- COMPRESS all columns where at least 10% of the rows participate.
  - COMPRESS creates smaller rows, therefore more rows/block and fewer blocks in table.
- You cannot COMPRESS Primary Index, referenced and referencing data columns (PK-FK) with Standard RI, Identity columns, derived data columns, or volatile table columns.

Multi-Value Compression

Multi-Value Compression (a.k.a., Value List Compression) allows specification of more than one compressed value on a column. This feature allows multiple values (up to 255 distinct values plus NULL to be compressed on a column (only fixed width columns before 13.10). This compression enhancement further reduces storage cost by storing more logical data per unit of physical capacity. Performance is improved because there is less physical data to retrieve during scan-oriented queries.

Multi-value compression is a technology that reduces the effective price of logical data storage capacity, and improves query performance. Teradata can compress up to 255 values per column.

Performance is improved because there is less physical data to retrieve for scan-oriented queries. Usually compression requires much more CPU time to compress/uncompress, but Teradata Multi-Value Compression does not have that limitation.

Examples of Presence Bits with Multi-Value Compression:

Definition	Value	Presence Bits
City CHAR(10) NOT NULL	'Atlanta'	None
City CHAR(10) NOT NULL COMPRESS ('Atlanta')	'Atlanta'	0
	'Boston'	1
City CHAR(10) COMPRESS	NULL	0
	'Atlanta'	1
City CHAR(10) COMPRESS ('Atlanta')	NULL	00
	'Atlanta'	01
	'Boston'	10
	'Chicago'	10

Note: First Presence Bit is 0 if compressed or 1 if not compressed.

City CHAR(10) COMPRESS ('Atlanta', 'Boston', 'Chicago', 'Denver')

NULL	0000
'Atlanta'	0001
'Boston'	0010
'Chicago'	0011
'Denver'	0100
'Eureka'	1000

Note: First Presence Bit is 0 if compressed or 1 if not compressed.

City CHAR(10) NOT NULL COMPRESS 'Atlanta', 'Boston', 'Chicago', 'Denver')

'Atlanta'	001
'Boston'	010
'Chicago'	011
'Denver'	100
'Eureka'	000

Multi-Value Compression

What is Multi-Value Compression?

- A feature that allows up to 255 distinct values (plus NULL) to be compressed per column – a.k.a., Value List Compression.
  - Reduces storage cost and performance is improved because there is less physical data to retrieve during scan-oriented queries.
- You can not add a compressed value to a column that if doing so, causes the table header row to exceed its maximum length.
  - The Table Header row contains an array of the compressed values.
- Algorithm compression (13.10) adds an additional presence bit to indicate if the column is algorithmically compressed or not.

Number of Presence Bits needed for Multi-Value Compression:

Compressed Values	# of Bits
1	1
2 - 3	2
4 - 7	3
8 - 15	4
16 - 31	5
32 - 63	6
64 - 127	7
128 - 255	8

Note:
If column is "nullable", there will be 1 additional presence bit.

Implementing Multi-Value Compression

This feature is implemented during table creation or modification. Customers must research the data demographics in order to decide which values to compress. The list of compressed values for each column must be defined in the CREATE TABLE statement.

As another example, the syntax for compressing several populous cities:

```
CREATE TABLE Properties
 (Street_Address    VARCHAR(40),
  City              CHAR(20) COMPRESS ( 'Chicago', 'Los Angeles', 'New York'),
  State_Code        CHAR(2) );
```

Rules for Compression

Compression does not require extra computer resources to uncompress the block or row. Performance is enhanced since data remains compressed in memory so that the cache can hold more logical rows. The compressed values are stored in the table header row.

Rules for Compression:

- Up to 255 values (plus NULL) can be compressed per column.
- The maximum size of a compress value is 510 bytes (13.10) or 255 (pre 13.10).
- Before Teradata, 13.10, only fixed width columns can be compressed.

You **CANNOT** compress the following:

- Components of the primary index
- Identity columns
- Volatile table columns
- Derived table columns
- Referencing and referenced columns with Standard RI cannot be compressed.

Miscellaneous notes about compression:

- You cannot add a compress value to a column if doing so would cause the table header row to exceed its maximum length (64 KB – V2R5.1, 128 KB – V2R6.0, 1 MB – 13.0)).
- Compression is case-sensitive.

Since the compressed values are stored in table header row, the Create/Alter Table statement will fail if adding the compress value to a column causes the table header row to exceed the maximum size.

Columns with frequently occurring values may be highly compressed. Examples:

NULLs	Zeros Default Values	Flags
Spaces	Binary Indicators (e.g., T/F)	Age (in years)
Gender	Education Level	# of Children
Credit Card Type	State	County
City	Reason	Automobile Make
Status	Category	Codes

Implementing Multi-Value Compression

CREATE TABLE accepts a list of values for field attribute 'COMPRESS'.

Compress 15 popular last names and the top 15 most populated countries.

```
CREATE TABLE People
  ( ID_number      DECIMAL(18,0)          NOT NULL
   ,Last_Name      CHAR(30)               NOT NULL
                          COMPRESS  ('Smith',      'Johnson',      'Williams',
                                     'Brown',      'Jones',        'Miller',
                                     'Davis',      'Garcia',       'Rodriquez',
                                     'Wilson',     'Martinez',     'Anderson',
                                     'Taylor',     'Thomas',       'Lee')
   ,First_Name     CHAR(20)  COMPRESS
   ,Address        VARCHAR(100)
   ,Country        VARCHAR(40)
                          COMPRESS  ('Australia',  'Bangladesh',   'Brazil',
                                     'China',      'England',      'France',
                                     'Germany',    'India',        'Indonesia',
                                     'Japan',      'Mexico',       'Nigeria',
                                     'Pakistan',   'Russian Federation',
                                     'United States of America' )
   ,Gender         CHAR(1)   NOT NULL
  );
```

How many presence bits are needed for this table?

ALTER TABLE and Compression

You **can** modify the compression specification for an existing column. However, when doing this, include all of the compressed values for the column.

Teradata uses a **lossless** compression method. This means that although the data is compacted, there is no loss of information.

The **granularity** of Teradata compression is the individual field of a row. This is the finest level of granularity possible and is superior for query processing, updates, and concurrency. **Field compression** offers superior performance when compared to row level or block level compression schemes. Furthermore, field compression allows compression to be independently optimized for the data domain of each column.

Teradata performs database operations directly on the compressed fields – there is no need to reconstruct a decompressed row or field for query processing. Of course, external representations and answer sets include the fully uncompressed results.

Up to 255 distinct values in each column can be compressed out of the row body. If the column is nullable, then NULLs are also compressed. The best candidates for compression are the most frequently occurring values in each column. The compressed values are stored in the table header. A bit field in every row header indicates whether the field is compressed and whether or not the value is NULL.

Teradata compression is completely **transparent** to applications, ETL (extraction, transforming, and loading of data), queries, and views. Compression is easily specified when tables are created or columns are added to an existing table.

Expectations / Guidelines

On an existing system, Multi-Value Compression will free up storage space, but is unlikely to free up a significant amount of compute resources since the database will still need to execute the application on the logical data. However, on a system where CPU resource is available, the space savings from compression may allow more applications to be added prior to next system upgrade.

ALTER TABLE and Compression

ALTER TABLE allows you to add a new column with compressed values.

Add an "Education" column.

```
ALTER TABLE People
  ADD Education    CHAR(10)
    UPPERCASE COMPRESS
    ( 'ELEMENTARY',
      'MIDDLE',
      'HIGH',
      'COLLEGE',
      'POST GRAD' ) ;
```

Note:

- **Because UPPERCASE is specified, all values are compressed regardless of case.**

You can use ALTER TABLE to add an additional compressed value to a list of values.

Add the Czech Republic to the list of compressed countries.

```
ALTER TABLE People ADD Country
COMPRESS
   ('Australia',                 'Bangladesh',
    'Brazil',                    'China',
    'England',                   'France',
    'Germany',                   'India',
    'Indonesia',                 'Japan',
    'Mexico',                    'Nigeria',
    'Pakistan',                  'Russian Federation',
    'United States of America',  'Czech Republic'
    );
```

Notes:

- **All of the compressed values must be listed.**
- **A SHOW TABLE will show the compressed values in alphabetical sequence regardless of how they are entered.**

Teradata Compression Enhancements

Multi-Value Compression enhancements of VARCHAR compression and 510 bytes for value starting with Teradata Release 13.10.

Algorithmic Level Compression

Provide the capability that will allow users the option of defining compression/decompression algorithms that would be implemented as UDFs and that would be specified and applied to data at the column level in a row. Initially, Teradata will provide three compression/decompression algorithms that will offer compression for UNICODE and LATIN data columns

The act of compressing/decompressing column data will require CPU cycles to be consumed. Consideration must be given to weighing the CPU cost of compression versus the potential space and performance gains that may be realized from leveraging this feature.

- The algorithms must be defined as regular scalar UDFs. The UDFs need to be specified in the column definition during the execution of a CREATE/ALTER TABLE statement.

- These algorithms will be invoked internally by the Teradata Database to compress/decompress the column data when the data is moved into the tables or when data is retrieved from the tables.

- An algorithmically compressed column adds an extra bit to indicate whether the column is algorithmically compressed or not, as follows:
  - 0; the column is not algorithmically compressed.
  - 1; the column is algorithmically compressed

Block Level Compression

This feature provides the capability to perform compression on whole data blocks at the file system level before the data blocks are actually written to storage.

- BLC will compress/decompress only data blocks but will not be used on any file system structures such Master/Cylinder indexes, the WAL log and table headers.
- Only primary data subtable and fallback copies can be compressed. Both objects are either compressed or neither is compressed.
- Secondary Indexes (USI/NUSI) cannot be compressed but Join Indexes can be compressed since they are effectively user tables.
- Only the compressed form of the data block will be cached, each block access must perform the data block decompression.
- On initial delivery, only one single compression algorithm will be supplied and used.
- Spool data blocks can be compressed via a system-wide tunable, as well as Temporary (Volatile and Global Temporary), WORK (sort Work space) & permanent journal.
- Once BLC is enabled on a table, reversion back to an earlier release for compressed tables is not allowed.

Utilize the FERRET utility to execute the new compression commands:

COMPRESS: "*database_name.table_name"*
UNCOMPRESS: *"database_name.table_name"*

Teradata Compression Enhancements

Enhanced Multi-Value Compression (MVC) or Value-List Compression

- Compress VARCHAR, VARBYTE, or VARGRAPHIC columns
- Number of characters in a compressed value is increased to 510.

Algorithmic Compression (ALC) or Column Level Compression

- Allows users to apply a compression algorithm to data at the column level in a row.
  - One example is to compress two-byte Unicode into one byte when the data is Latin (ASCII).
- Compression/decompression is done by specifying a UDF function.
  - Teradata provides some UDFs to do compression for UNICODE and LATIN data columns.
  - User can create and apply their own compression/decompression algorithms to columns.

Block Level Compression (BLC)

- Compression is performed by Teradata at the file system level on whole data blocks before the data blocks are actually written to/read from storage devices.
- How is BLC set for a specific table?
  - When loading data, SET QUERY_BAND = 'BLOCKCOMPRESSION=YES/NO;' FOR SESSION;
  - Ferret utility has new commands – COMPRESS/UNCOMPRESS *table*
  - DBSControl settings
- There is a CPU cost to compress/decompress on whole data blocks and is generally considered a good trade since CPU cost is decreasing while I/O cost remains high.

Algorithmic Compression Example

In some cases, such as when column values are mostly unique, Algorithmic Compression can provide better compression results than Multi-Value Compression. Algorithmic Compression allows you to define your own compression and decompression algorithms and apply them to data at the column level.

A user can create their own compression algorithms as external C/C++ scalar UDFs, and then specify them in the column definition of a CREATE TABLE/ALTER TABLE statement. Teradata Database invokes these algorithms to compress and decompress the column data when the data is moved into the tables or when data is retrieved from the tables.

ALC allows you to implement the compression scheme that is most suitable for data in particular column. The cost of compression and decompression depends on the algorithm chosen.

You can specify ALC alone, or both MVC and ALC on the same column. If you define both on the same column, ALC is applied only to those non-null values that are not specified in the value compression list of the MVC specification.

You can use ALC to compress columns with the following data types:

- BYTE
- CHARACTER
- GRAPHIC
- VARBYTE
- VARCHAR
- VARGRAPHIC

Teradata provides domain-specific functions for compressing and compressing data. These functions are stored in the TD_SYSFNLIB system database. Two of these algorithms are:

- Compress TransUnicodeToUTF8
  Takes Unicode character input and stores it in UTF8 format.
  This is useful when the input data is predominantly Latin because UTF8 uses one byte to represent Latin characters and Unicode uses two bytes.

- Decompress TransUTF8ToUnicode
  Takes the data previously compressed using the TransUnicodeToUTF8 function and converts it back to Unicode.

Algorithmic Compression Example

Given the following:

- The Description column is defined with a character set of Unicode.
  - Every character uses 2 bytes of space.
  - Use a Teradata-supplied function to save space for common ACSII Latin 7-bit (USA) characters.
- The TransUnicodeToUTF8 function compresses all non-null values in the Description column. Use this function when …
  - Unicode column contains mostly ASCII Latin 7-bit (USA) characters – compress these to 1 byte.
- MVC is used to compress the values 'tools' and 'vacuums' in this Category column.
  - Because UPPERCASE is specified, all values of 'tools' and 'vacuums' are compressed regardless of case.

Example:

```
CREATE TABLE Products
   ( Product_id    INTEGER          NOT NULL,
     Category      CHAR(10)         NOT NULL  UPPERCASE  COMPRESS ('tools', 'vacuums'),
     Description   VARCHAR (500)    CHARACTER SET UNICODE
                                    COMPRESS USING TD_SYSFNLIB.TransUnicodeToUTF8
                                    DECOMPRESS USING TD_SYSFNLIB.TransUTF8ToUnicode);
```

Detailed Row Format

The diagram on the facing page represents a typical row. A single row can be up to approximately 64 KB in length. Each row has at least 14 bytes of overhead. The components of a row are:

Row Length, Ref Array – The first two bytes of the row specify the exact length of the row. There are two bytes at the end of the data block, called the Reference Array pointer, which represents the byte offset (1st byte) of the row in the block.

Row ID – these eight bytes contain the Row ID. The Row ID is determined by the Primary Index Value. The Row ID is, in turn, composed of 4 bytes of Row Hash and 4 bytes of Uniqueness Value. For PPI tables, 2 or 8 bytes are allocated for the partition #.

Flag – this section is one byte in length (x'00' for NPPI tables, x'80' for PPI tables).

Presence Bits (0 - 10 bits) per column – this section is variable in length and contains the Presence Bits for NULLS and/or Values that are compressed for each of the columns.

If a variable or fixed length multi-value compressed column is compressed or is null, then its value is not stored in the row. Instead, Teradata stores one instance of each compressed value within a column in the table header and references it using the presence bits array for the row.

Column Offsets – this section is only present when variable length data columns are declared in the table. This section is variable in length and contains the Column Offsets. There is one 2-byte Column Offset indicator for each variable length data column.

Fixed Length Columns – this section contains all the fixed length data columns which are not compressible. Teradata takes the liberty of rearranging the order of data columns within a row.

Uncompressed data for compressible columns – this variable length section is composed of those data columns that are compressible but are neither compressed nor NULL.

When compression does not apply to a value in a column that specifies multi-value compression, Teradata stores the column data in the row as a length and data value pair.

Three examples of this type data is shown on the facing page.

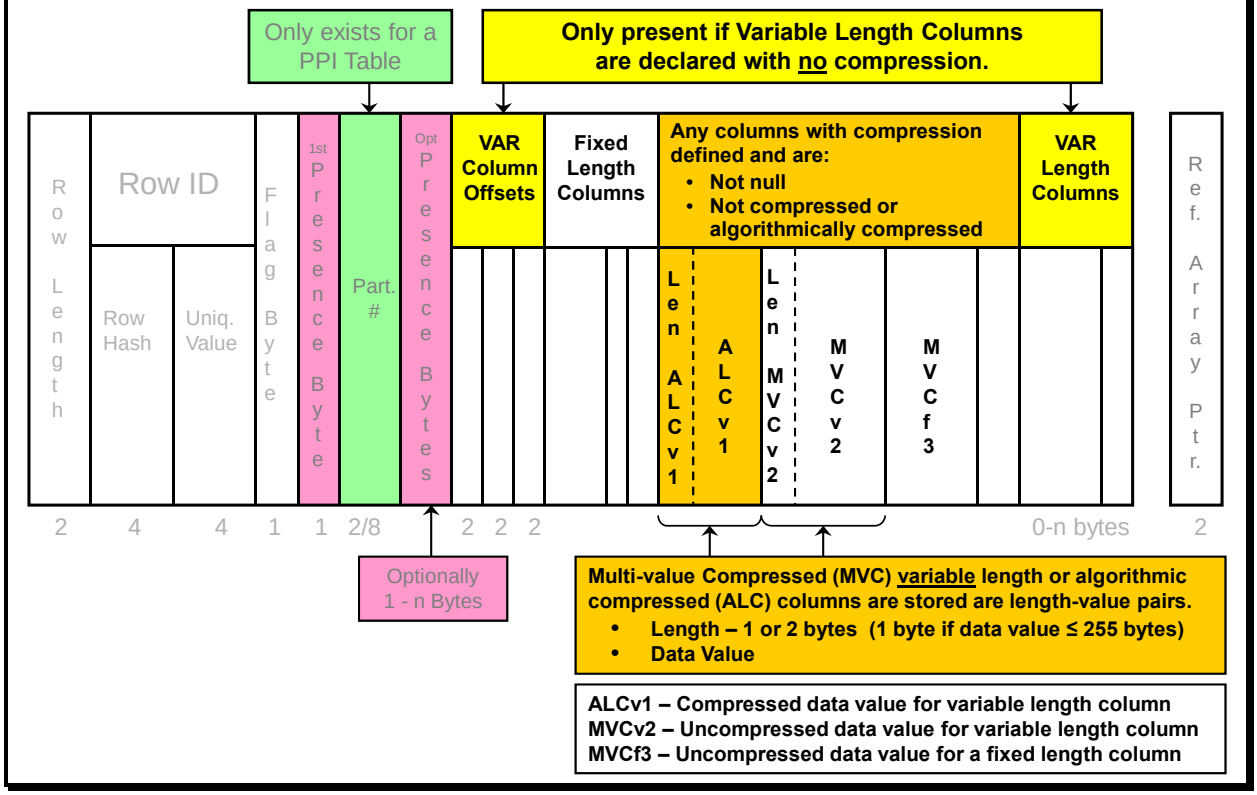
ALCv1 – column that has ALC and is variable in length (kept in "Len ALCv1")
MVCv2 – column that uses MVC and is variable in length (kept in "Len MVCv2")
MVCf3 – column that used MVC and is fixed in length (length bytes not needed)

Since ALCv1 or MVCv2 are variable in length and not compressed, Teradata needs to store this data in a value pair (length:data). If the length of the column data is <= 255 bytes, then Teradata stores the length in 1 byte; otherwise it stores the length in 2 bytes.

Since MVCf3 is a fixed length multi-value compressed column is not compressed, then Teradata stores its values in the row as simple data, but without an accompanying length. No length is needed because Teradata Database pads fixed length data values to the maximum length defined for their containing column.

VARCHAR, VARBYTE, VARGRAPHIC, or NUMBER Columns that do not have compression defined. The Column Offsets indicate exactly where in this section the variable length columns start.

Detailed Row Format



Multi-Value Compression vs. VARCHAR

For character-based data, an alternative to Teradata compression is the VARCHAR (N) data type. The number of bytes used to store each field is the length of the data item plus 2 bytes. Contrast this to a fixed-length CHAR (N) data type that takes N bytes per row, regardless of the actual number of characters in each individual field. Combining Teradata compression with fixed-length character data type can be a very effective space saver.

The data demographics can help determine whether variable-length character data type or fixed length plus compression is more efficient. The most important factors are the maximum field length, the average field length, and the compressibility of the data.

Which is best – Compression or VARCHAR?

Before Teradata 13.10, customers occasionally had to choose between fixed character columns with multi-value compression or variable length columns.

- VARCHAR is more efficient when the difference of maximum and average field length is high and compressibility is low.

- Compression and fixed-length CHAR is more efficient when the difference of maximum and average field length is low and compressibility is high.

When neither is a clear winner, use VARCHAR since it uses slightly less CPU resource.

Compression in Spool Files

Value list compression is extended to intermediate spool files. Thus, when compressed columns are selected, compression is propagated to resulting spool files. Without compression for spool files, the intermediate join results for compressed tables can be very large, causing the need for additional spool space.

Uncompressed VARCHAR data is also carried into spool.

Miscellaneous Considerations

You should not compress columns whose NULL values will be changing. When the value changes, the column will expand and you may get block splits. In practice, there may be exceptions to this rule. For example, it might be a good idea to compress the NULL values in columns related to shipping an order until the order is actually shipped.

Additional sizing considerations:

- Adding a column that is not compressible expands all rows.
- Adding a column that is compressible and there are no spare presence bits expands all rows.
- Dropping a column changes all row sizes where data is present.

Multi-Value Compression vs. VARCHAR

Prior to Teradata 13.10, you cannot compress VARCHAR columns. You may have to choose VARCHAR or compression with a fixed length column.

There is no general rule – evaluate the options.

- **VARCHAR – generally better when data size has a large variance and a low percentage of fields are compressible.**
- **Compression – generally better when data size has a small variance and a high percentage of fields are compressible.**
- **When neither is a clear winner, then use VARCHAR since it uses slightly less CPU resource.**

Starting with Teradata 13.10, you can also compress VARCHAR columns.

Miscellaneous considerations:

- **Multi-value compression and VARCHAR are both carried into intermediate spool files.**
- **You cannot compress BLOB, CLOB, Period, Geospatial, UDT, or Identity columns.**
- **ALTER TABLE considerations:**
  - **Additional compressed values can be added to a column (ALTER). Include the complete list of compressed values with each ALTER command.**
  - **Adding a column (to a table) that is not compressible expands all rows.**
  - **Adding a column (to a table) that is compressible and there are no spare presence bits expands all rows.**

Teradata Compression Comparison

The chart on the facing page summarizes the three types of compression.

Teradata Compression Comparison

	Multi-Value Compression (MVC)	Algorithmic Compression (ALC)	Block Level Compression (BLC)
Ease of Use	Easy to apply to well understood data columns and values.	Easy to apply at column with CREATE TABLE.	Set once and forget.
Analysis Required	Need to analyze data for common values.	Use Teradata algorithms or user-defined compression algorithms to match unique data patterns.	Need to analyze CPU overhead trade-off. You can turn on for all data on system or you can apply on a per table basis.
Flexibility	Works for a wide variety of data and situations.	Automatically invoked for values not replaced by MVC.	Automatically combined with other compression mechanisms.
Performance Impact	No or minimal CPU usage	Depends on compression algorithm used.	Reduced I/O due to compressed data blocks. CPU cycles are used to compress/decompress.
Applicability	Replaces common values	Industry data, UNICODE, Latin data	All Data

A customer can choose any combination or all three on a column/table.

Sizing a Data Row Considerations

There are basically two row formats used by Teradata.

- Aligned Row Format (ARF)
- Packed64

The maximum row length is approximately 64 KB (the actual limit is 64,256 bytes), and this limit is the same for both packed64 and aligned row formats.

Whether a system stores its data in packed64 or aligned row format depends on several factors. Only new systems (or systems that have been sysinit) on Teradata 13.10 can use the packed64 row format. Systems that are simply upgraded to 13.10 will continue to use the aligned row format.

The size of tables on a system that stores data in packed64 format is generally about 7% (range of 3 – 9%) smaller than the size of the same tables on a system that stores data in aligned row format. Storing data in packed64 format reduces the number of I/O operations required to access and write rows in addition to saving disk space.

Rows on Packed64 format systems are always aligned on even-byte boundaries. In other words, rows are never stored with an odd number of bytes. As a result, if a row has an odd byte length, the system adds a filler byte to the end of the row to make its length even.

Rows on Aligned Row Format 64-bit systems (most Teradata 12.0 and 13.0 systems) are always aligned on an 8-byte boundary. Additionally, the following pad bytes were added within the data row. The aligned row format is the only row format for these releases and will also be true if the system is simply upgraded to Teradata 13.10.

Pad Byte Field Name	Purpose
VARCHAR Offsets Array Alignment Pad Bytes	Aligns VARCHAR offsets array at a 2-byte boundary.
Fixed Length Column Alignment Pad Bytes	Aligns fixed length columns within the row on an 8-byte boundary.
Compressible Length Column Alignment Pad Bytes	Aligns value compressible length columns.
VARCHAR Column Alignment Pad Bytes	Aligns variable length columns within the row on an 8-byte boundary
Trailing Pad Bytes	Aligns entire row on an 8-byte boundary.

Also note that for PPI tables, rows in either format also have an additional 2 bytes of overhead between the presence bytes and before the VARCHAR column offsets.

Sizing a Data Row Considerations

Determining a typical row length can be difficult.

- Different column data types occupy different amounts of disk space.
- Row length is determined by 3 categories of table columns.
  - Fixed length uncompressed data – this storage length is the easiest to calculate.
  - Variable length – storage space is allocated in the row depending on the size of the column.
  - Compressible data – includes both value-compressed and algorithmically-compressed data.

Row and data type alignment depends on type (format) of "row architecture".

- Rows are aligned on 8-byte boundaries – applies to 64-bit systems (before release 13.10) or systems upgraded to release 13.10.
  - Specific row sections are also aligned on 2-byte or 8-byte boundaries within the row.
  - Specific data types are also aligned on 2-byte or 4-byte boundaries within the row.
  - This format is referred to as Aligned Row Format (ARF) architecture.
- Rows are aligned on 2-byte boundaries – only applies to new (fresh install – sysinit) Teradata 13.10 systems.
  - This row format does NOT have row section/data type alignment requirements.
  - This row format uses less space (typically 7% smaller) for data tables.
  - This format is referred to as Packed64 architecture.

Note: For sizing exercises, this course will assume the newer 2-byte boundary architecture.

Teradata Data Types

ANSI provides a smaller menu of data types than permitted by Teradata. REAL and DOUBLE PRECISION are implemented as Teradata FLOAT. ANSI NUMERIC is implemented as Teradata DECIMAL. ANSI does not support some of the specific Teradata data type extensions (e.g., BYTEINT).

Examples of the number of bytes allocated to the different data types are:

BYTEINT	1	
SMALLINT	2	
INTEGER	4	
BIGINT	8	
DECIMAL 1-2	1	
DECIMAL 3-4	2	
DECIMAL 5-9	4	
DECIMAL 10-18	8	
DECIMAL 19-38	16	
FLOAT	8	
NUMBER (14.0 Feature)	0 – 18	
DATE	4	
TIME	6/8	6 for packed64; 8 for ARF
TIME WITH TIME ZONE	8	
TIMESTAMP	10/12	10 for packed64; 12 for ARF
TIMESTAMP WITH TIME ZONE	12	
INTERVAL YEAR	2	
INTERVAL YEAR TO MONTH	4	
INTERVAL MONTH	2	
INTERVAL MONTH TO DAY	2	
INTERVAL DAY	2	
INTERVAL DAY TO MINUTE	8	
INTERVAL DAY TO SECOND	10/12	10 for packed64; 12 for ALR
INTERVAL HOUR	2	
INTERVAL HOUR TO MINUTE	4	
INTERVAL HOUR TO SECOND	8	
INTERVAL MINUTE	2	
INTERVAL MINUTE TO SECOND	6/8	6 for packed64; 8 for ARF
INTERVAL SECOND	6/8	6 for packed64; 8 for ARF
Period (Date)	8	
Period (Time)	12/16	12 for packed64; 16 for ARF
Period (Time With Time Zone)	16	
Period (Timestamp)	20	
Period (Timestamp With Time Zone)	24	
Period (Timestamp)	11	When Ending Element Value is UNTIL_CHANGED
Period (Timestamp With Time Zone	13	When Ending Element Value is UNTIL_CHANGED

Packed64 – Also applies to 32-bit systems
ARF – Aligned Row Format for 64-bit systems

Teradata Data Types

Data Type	Size of data type
BYTEINT	1
SMALLINT	2
INTEGER	4
BIGINT	8
NUMBER (14.0)	0 – 18
DATE	4
TIME, TIME WITH TIME ZONE	6 or 8
TIMESTAMP, TIMESTAMP WITH TIME ZONE	10 or 12
INTERVAL YEAR, MONTH, DAY, HOUR, MINUTE, SECOND	2, 4, 6, 8, 10, or 12
PERIOD(DATE)	8
PERIOD(TIME), PERIOD(TIME) WITH TIME ZONE	12 or 16
PERIOD(TIMESTAMP), PERIOD(TIMESTAMP) WITH TIME ZONE	20 or 24
DECIMAL (n,m) (precision to 38 digits) or NUMERIC(n,m)	1 – 16
FLOAT or REAL, DOUBLE PRECISION	8
CHAR(n)	1 – 64,000
BYTE(n)	1 – 64,000
GRAPHIC(n)	1 – 32,000
VARCHAR(n), CHAR VARYING(n), LONG VARCHAR	1 – 64,000
VARBYTE(n)	1 – 64,000
VARGRAPHIC(n), LONG VARGRAPHIC	1 – 32,000
BLOB(n) or BINARY LARGE OBJECT(n)	1 – 2097088000
CLOB(n) or CHARACTER LARGE OBJECT(n)	1 – 2097088000
SYSUDTLIB.*udt_name* (User defined data type)	

INTEGER Data Types

When sizing rows, you need to know how much space will be occupied by the data in the rows. The first of the Teradata data constructs are the three types of Integer data types (BYTEINT, SMALLINT, INTEGER), and date (DATE), which is covered on the next page.

BYTEINT

As the name implies, BYTEINT data requires only a single byte. The first bit of the byte is the SIGN (+ or -) and the remaining seven bits allow the storage of numbers from -128 to +127 (-2^7 to $2^7 - 1$).

SMALLINT

SMALLINT data occupies 2 bytes. The first bit is the SIGN and the remaining 15 bits allow the storage of numbers from -32,768 to +32,767(-2^{15} to $2^{15} - 1$).

INTEGER

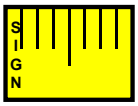
INTEGER data occupies 4 bytes. The first bit is the SIGN and the remaining 31 bits allow the storage of numbers from -2,147,483,648 to +2,147,483,647 (-2^{31} to $2^{31} - 1$).

BIGINT

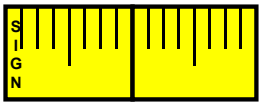
BIGINT data occupies 8 bytes. The first bit is the SIGN and the remaining 63 bits allow the storage of numbers from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 or as (-2^{63} to $2^{63} - 1$).

INTEGER Data Types

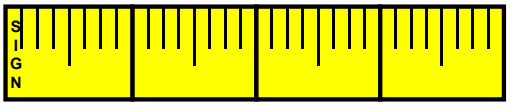
BYTEINT One byte – range -128 to +127 **Non-ANSI**



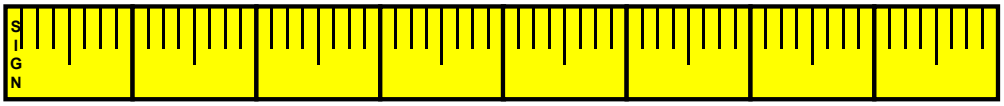
SMALLINT Two bytes – range -32,768 to +32,767



INTEGER Four bytes – range -2,147,483,648 to +2,147,483,647



BIGINT Eight bytes – range -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807



DECIMAL and FLOAT Data Types

Decimal and Numeric

The DECIMAL and NUMERIC data types represent a decimal number of n digits, with m of these digits to the right of the decimal point. The DECIMAL data type is synonymous with NUMERIC and can be abbreviated as DEC.

The specification of a DECIMAL data type is: `DECIMAL [(n[,m])]`

The value of n prior to V2R6.2 has a maximum value of 18. With V2R6.2, value of n can be as large as 38. Values of n between 19 and 38 require 16 bytes of storage.

The Decimal and Numeric data types can take up 1, 2, 4, 8, or 16 bytes of space. The space required depends on the number of digits and is shown in the table on the facing page. Decimal values are stored in scaled binary.

Decimal numbers are scaled by the power of ten equal to the number of fractional digits. The number is stored as a two's complement binary number in 1, 2, 4, 8, or 16 bytes. The number of bytes used for a decimal value depends on the total number of digits in that value.

Float, Real, and Double Precision

The FLOAT, REAL, and DOUBLE PRECISION data types represent a value in sign/magnitude form. These data types represents values that range from 2×10^{-308} to 2×10^{308}.

The mantissa sign is in the most significant bit position; the exponent sign is a part of the exponent field (excess-1024 notation, in which (1024 - exponent) = sign).

Floating point values are stored and manipulated internally in IEEE floating point format. Floating point values are stored with the least significant byte first, with one bit for the mantissa sign, 11 bits for the exponent, and 52 bits for the mantissa. Eight bytes are used to hold a floating point value. Negative numbers differ from positive numbers of the same magnitude only in the sign bit. Float, Real, and Double Precision data require 8 bytes of storage space.

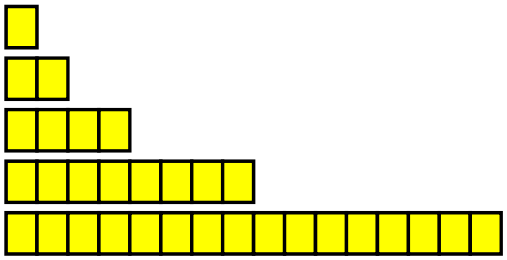
DECIMAL and FLOAT Data Types

DECIMAL and NUMERIC

DECIMAL [(n[,m])]

n = 1 – 38
m = 0 - n
Default = (5,0)
Stored in scaled binary

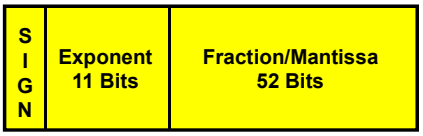
Number of Digits	Number of Bytes
1 to 2	1 byte
3 to 4	2 bytes
5 to 9	4 bytes
10 to 18	8 bytes
19 to 38	16 bytes



FLOAT, REAL, and DOUBLE PRECISION

Notes:

- Range is $2 * 10^{-308}$ to $2 * 10^{+308}$
- 15 significant decimal digit precision.
- Manipulated in IEEE floating point format.
- Corresponds to, but is not identical to, IBM normalized 64 bit floating point.



8 Bytes

NUMBER Data Type (14.0)

NUMBER is a floating point type and, therefore, an approximate data type on the Teradata Database.

If you run a query that includes a NUMBER (or another floating point) value on a database from other vendors and then run the same query on the Teradata Database, the result may be different because the order in which each database evaluates the expression may be different.

NUMBER(n) is equivalent to NUMBER(n,0).

NUMBER Data Type (14.0)

NUMBER or NUMBER(*)
NUMBER(n) or NUMBER(n,m)

Characteristics

- **Teradata 14.0 feature – this is effectively a variable length numeric data type.**
  - Greater efficiency in storing numeric data because the number of bytes to store the data can vary from 0-18 bytes depending on the value stored.
  - NUMBER is effectively stored as a variable length number. 2 bytes are used in the column offset array to represent the starting location of this column in the internal data row.
- **Range is ± 1E-130 to 9.99…9E125**
- **More flexibility, range, and precision for numeric data.**
  - More flexibility in defining numeric columns by providing the ability to change the precision or scale of existing NUMBER columns in tables without modifying the data rows.
  - More flexibility in computation compared with the DECIMAL data type because the result is not limited by the precision or scale of the input.
  - Greater range than the DECIMAL data type.
  - Greater accuracy than the FLOAT data type because NUMBER has greater guaranteed precision. NUMBER can represent common decimals exactly.
- **Increased compatibility with other databases, which include a similar NUMBER data type.**

DATE and TIME Data Types

DATE, TIME, and TIMESTAMP are also SQL functions. CURRENT_DATE, CURRENT_TIME, and CURRENT_TIMESTAMP represent values.

The following data types (not shown on facing page) also require a 2-byte boundary within the row with the 64-bit Aligned Row Format.

- PERIOD (DATE)
- PERIOD (TIME(n))
- PERIOD (TIME(n) WITH TIME ZONE)
- PERIOD (TIMESTAMP(n))
- PERIOD (TIMESTAMP(n) WITH TIME ZONE)

TIME

The format for the TIME data type is: **hh:mi:ss[.ssssss]**. The internal storage format is a string of bytes:

SECOND (DECIMAL(8,6))
MINUTE (BYTEINT)
HOUR (BYTEINT)

Example:

```
CREATE SET TABLE table5
      (col1 DATE,
       col2 TIME(6),
       col3 TIME(6) WITH TIME ZONE,
       col4 TIMESTAMP(6),
       col5 TIMESTAMP(6) WITH TIME ZONE)
PRIMARY INDEX ( col1 );

INSERT INTO table5 VALUES
      (CURRENT_DATE,
       CURRENT_TIME,
       TIME '11:27:00-04:00',
       CURRENT_TIMESTAMP,
       TIMESTAMP '2011-05-18 11:27:00-04:00');

SELECT * FROM table5;

 *** Query completed. One row found. 5 columns returned.
 *** Total elapsed time was 1 second.

col1  11/05/18
col2  13:19:41.000000
col3  11:27:00.000000-05:00
col4  2011-05-18 13:19:41.730000
col5  2011-05-18 11:27:00.000000-05:00
```

DATE and TIME Data Types

DATE (4 Bytes)

((YYYY - 1900)) * 10000 + (MM * 100) + DD

TIME (6 or 8 Bytes)

hh:mi:ss.ssssss
Packed64 format – 6 bytes
Aligned Row format – 8 bytes

TIME WITH TIME ZONE (8 Bytes)

hh:mi:ss.ssssss +/- hh:mi
All formats – 8 bytes

TIMESTAMP (10 or 12 Bytes)

DATE + TIME

Packed64 format – 10 bytes
Aligned Row format – 12 bytes

TIMESTAMP WITH TIME ZONE (12 Bytes)

DATE + TIME + ZONE

All formats – 12 bytes

Note:

- 32-bit systems (e.g., MP-RAS) have the same row format and data type lengths as the Packed64 format.

CHARACTER Data Types

There are three kinds of Character data types: **CHAR, VARCHAR**, and **LONG VARCHAR**. Character data can require from 1 to 64000 bytes of space.

Character data (for the Latin character set) is stored in 8-bit ASCII format. Conversion to and from other host formats is performed by the parsing engine on input and the AMPs upon output. Sorting and comparisons are always done in the collating sequence of the user's host system.

- CHAR is fixed length Character data. CHAR (n) always takes n bytes unless it is compressed.
- The VARCHAR (n) data type represents a variable length character string of length n. The maximum value for n is 64000 characters. Synonyms for VARCHAR(n) are CHAR VARYING [(n)], and CHARACTER VARYING [(n)]
- LONG VARCHAR is equivalent to VARCHAR (64000).

Note: The Unicode and Graphic character sets use 16-bit characters, thus allowing 32K logical characters for 64K physical bytes.

An uncompressed VARCHAR column always has a two-byte offset associated with it, whereas a multi-value compressed column can have its length stored in one or two bytes.

Teradata 13.10 Changes

When a VARCHAR column is compressed (either MVC or ALC), Teradata stores the column data in the row as a length and data value pair.

- If the length of the column data is <= 255 bytes, then Teradata Database stores the length in 1 byte; otherwise it stores the length in 2 bytes.

All variable length compressible columns are treated as an extension of fixed length compressible columns except that uncompressed variable length columns store both a length and the actual column data.

There is an additional presence bit for an algorithmically-compressed column to indicate whether the column data is compressed or not. Teradata Database sets this bit only when data in column is compressed using algorithmic compression. Teradata Database uses the additional presence bit when the compress UDF returns data that is larger than the original column data.

When column data is compressed algorithmically, Teradata Database stores it as length: data pairs interleaved with the other compressible columns in the table. When data is not compressed, or if column data is null, Teradata Database does not store a length, so there is no overhead in this case.

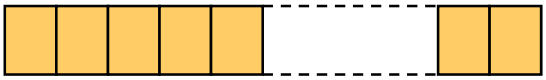
CHARACTER Data Types

CHARACTER

- Typically 1 byte per character – stored in 8 bit ASCII.
- Other Character Sets (e.g., UNICODE) may use multiple bytes per character.
- Compressed VARCHAR (13.10) or any ALC columns have 1 or 2 bytes stored with data in length-value pairs and do not use the 2 byte column offset.
- How do you store a Character data that is greater that 64,000 bytes?

  Define a CLOB (Character Large Object)

CHAR (n) n = 1 - 64000
Fixed length character string



VARCHAR (n) n = 1 - 64000
Variable length character string

2 byte column offset identifies location in row

LONG VARCHAR
Equivalent to VARCHAR (64000)

2 byte column offset identifies location in row

Character Sets

Teradata supports five different character sets for use with the CHARACTER data type.

- LATIN - 8-bit LATIN server character
- UNICODE - 16-bit characters from the Unicode 4.1 standard.
- GRAPHIC - 16-bit characters supported for DB2 compatibility.
- KANJISJIS - intended for Japanese applications that rely on the KANJISJIS character set
- KANJI1 - this character set will be removed in a future release

The Unicode and Graphic character sets use 16-bit characters, thus allowing 32K logical characters for 64K physical bytes. Examples of GRAPHIC character sets are not shown on the facing page, but a brief description follows.

GRAPHIC[(n)] Data Type

The GRAPHIC data type represents a fixed-length, multi-byte character string of length n, where n is the length in logical characters. The maximum value of n is 32000.

The GRAPHIC data type is valid only for Chinese double-byte Hanzi and Japanese double-byte Hiragana character supported sites. If any other site attempts to use this data type, the system generates an error message

VARGRAPHIC(n) Data Type

The VARGRAPHIC data type represents a variable length string of length n, where n is the length in logical characters. The maximum value of n is 32000. There is no default; omitting the length specification results in an error.

LONG VARGRAPHIC Data Type

The LONG VARGRAPHIC data type specifies the longest permissible variable length graphic string. It is equivalent to specifying a data type of VARGRAPHIC(32000).

Graphic Data Validation and Storage

Graphic data must always contain an even number of bytes. Any attempt to insert data that results in an odd number of bytes generates an error. Graphic data is stored without translation; the multi-byte characters remain in the encoding of the session character set. The Teradata RDBMS validates graphic data against the range of hexadecimal values considered valid for the character set of the current session. Hexadecimal constants cannot be validated.

Character Sets

Teradata SQL supports different character sets for use with CHARACTER SET. Examples of character sets are:

LATIN	**- 8-bit LATIN server character**
UNICODE	**- 16-bit characters from the Unicode 4.1 standard.**
GRAPHIC	**- 16-bit characters supported for DB2 compatibility.**

Typically, when 16-bit international character sets are needed, the recommendation is to define these as UNICODE with a CHARACTER data type.

Example:

```
CREATE TABLE People
        ( Last_Name      VARCHAR(30)   CHARACTER SET UNICODE,
          First_Name     CHAR(20)      CHARACTER SET UNICODE,
               :
        ) ;
```

For Last_Name and First_Name, 16-bits (2 bytes) are allocated for each character. For example, 40 bytes are allocated for First_Name.

BYTE Data Types

There are two kinds of Byte data: **BYTE** and **VARBYTE**. They are suitable for various types of data, including graphical data in the form of digitized image information. You cannot compare Character data to Byte data.

Byte data is stored in host format and is never translated by Teradata. It is handled as if it were n-byte, unsigned binary integers.

- BYTE is a fixed length binary string.
- VARBYTE is a variable length binary string. Both BYTE and VARBYTE can have a maximum length (n) of from 1 to 64000 bytes.

BYTE Data Types

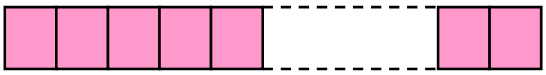
BYTE

- Stored in host format – binary format.
- Never translated by the Teradata Database
- Handled as if they are n-byte, unsigned binary integers
- Suitable for digitized image information (small photo, signature, etc.) that is less than 64,000 bytes.
- Compressed VARBYTE (13.10) or any ALC columns have 1 or 2 bytes stored with data in length-value pairs and do not use the 2 byte column offset.

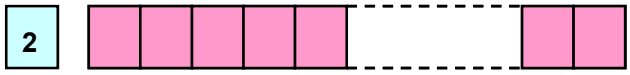
- How do you store a Binary data that is greater that 64,000 bytes?

  Define a BLOB (Binary Large Object)

BYTE (n) n = 1 - 64000
Fixed length binary string



VARBYTE (n) n = 1 - 64000
Variable length binary string



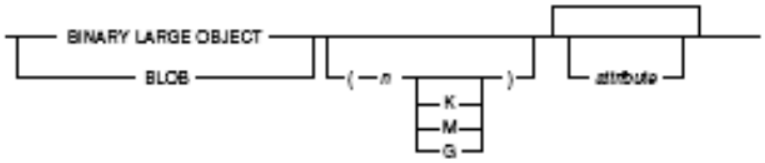
2 byte column offset identifies location in row

Large Object Data Types

For both Binary and Character Large Object (LOB) data types, the base table data row only contains an OID (Object Identifier) to the subtable that actually contains the LOB data.

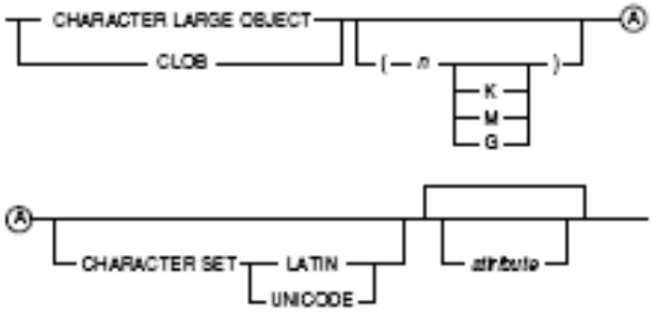
BLOB (n) Data Type

The BLOB data type represents a large binary string of raw bytes. A binary large object (BLOB) column can store binary objects, such as graphics, video clips, files, and documents.



CLOB (n) Data Type

The CLOB data type represents a large character string. A character large object (CLOB) column can store character data, such as simple text, HTML, or XML documents.



For both data types:

n = the number of bytes to allocate for the LOB column. The maximum number of bytes is 2097088000, which is the default if *n* is not specified.

K = that *n* is specified in kilobytes (KB). When K is specified, *n* cannot exceed 2047937.

M = that *n* is specified in megabytes (Mb). When M is specified, *n* cannot exceed 1999.

G = that *n* is specified in gigabytes (GB). When G is specified, *n* must be 1.

Large Object Data Types

Binary or Character Large Object (LOB)

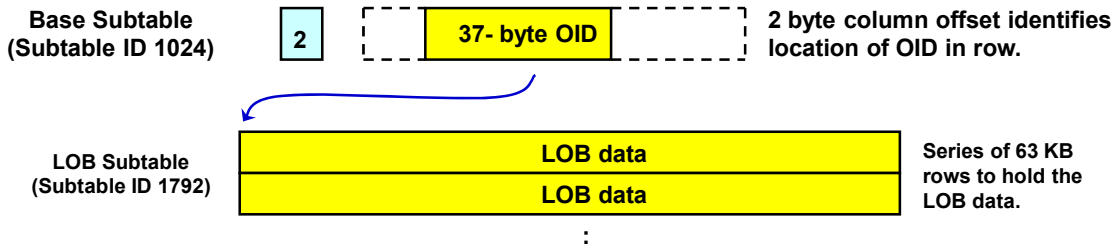
BLOB – a large binary string of raw bytes. A BLOB column can store binary objects, such as graphics, video clips, files, and documents.

CLOB – a large character string. A CLOB column can store character data, such as simple text, HTML, or XML documents.

BLOB (n) n = 1 – 2097088000 bytes (approx. 2 GB)
CLOB (n) n = 1 – 2097088000 bytes (approx. 2 GB)

The data row contains a 2-byte column offset and a 37-byte OID (Object Identifier) for each LOB. LOB data is stored in a separate subtable in a series of 63 KB rows.

- The LOB data is stored on the same AMP as the corresponding data row.
- Maximum of 32 Large Objects in a table. Each LOB is stored in its own subtable.



Variable Column Offsets

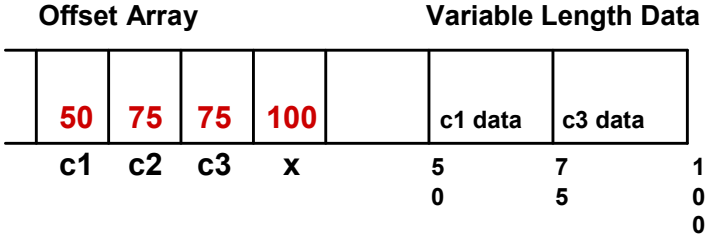
Variable column offset values indicate the starting location of a variable column as is shown on the facing page. They are only present in rows of tables where variable length columns have been defined.

Determine the length of a variable length column by subtracting its starting location from the next column's starting location. In the example on the facing page, column c2 is empty (of length 0) since column c3 starts in the same place (i.e., 75-75=0).

The definition of variable length columns requires an additional 2-byte offset needed by the system to determine the length of the last variable column.

The last offset actually represents what would be the first byte of the next variable length column if it existed.

Variable Column Offsets



- **Only applies to variable length columns without compression defined (MVC or ALC).**
- **Offset values indicate the starting location of a variable column.**
- **Determine the column length by subtracting its starting location from the next column's starting location.**
- **The definition of variable length columns requires one additional 2-byte offset that locates the end of the final variable column.**
  - **Note: the last offset actually represents what would be the first byte of the next variable length column if it existed.**
  - **The OID (Object Identifier) for a large object is also stored as a variable column and uses two bytes in the offset array.**

Row Size Calculation Form

The example on the facing page shows a form that can be used to calculate the physical row size for a typical table.

Row Byte Alignment for 32-bit systems

Rows on 32-bit systems (e.g., UNIX MP-RAS) are always aligned on even-byte boundaries. In other words, rows are never stored with an odd number of bytes. As a result, if a row has an odd byte length, the system adds a filler byte to the end of the row to make its length even. This simply means that with 32-bit systems, a row starts on a 2-byte boundary relative to the start of a data block.

Row Byte Alignment for Aligned Row Format systems

Rows on 64-bit systems prior to Teradata 13.10 (e.g., Linux) are always aligned on 8-byte boundaries. To ensure that no unaligned accesses occur with 64-bit systems, the system adds pad bytes to each row to align them on 8-byte boundaries in both memory and disk. For the sake of uniformity, all row sizes are multiples of 8, starting on an 8-byte boundary relative to the start of a data block. Additionally, padding bytes may be also be added within the row or at the end of a row. For example, the VARCHAR column offsets always start at on even boundary within a data row.

Padding bytes will be added before the following fields to ensure 8-byte boundary alignments.

- VARCHAR offset array
- Fixed length columns
- Compressible columns
- Variable length columns

Padding bytes will be added at the end of a row to ensure a row is a multiple of 8 bytes in length.

Row Byte Alignment for Packed64 Format systems

Rows on Packed64 format systems are always aligned on even-byte boundaries. In other words, rows are never stored with an odd number of bytes. As a result, if a row has an odd byte length, the system adds a filler byte to the end of the row to make its length even.

Only new systems (or systems that have been sysinit) on Teradata 13.10 can use the packed64 row format. Systems that are simply upgraded to 13.10 will continue to use the aligned row format.

Row Size Calculation Form

Table Name _____

Variable Column Data Detail

Column Name	Type	Max	Average
	SUM(a) =		

SUM(a) = SUM of the AVERAGE number of bytes expected for the variable columns.
SUM(n) = SUM of the fixed CHAR and GRAPHIC column bytes.

* Assume rows are on 2-byte boundaries.
* Assume 6 bytes for TIME and 10 bytes for TIMESTAMP.

Data Type	# of Columns	Size	TOTAL
BYTEINT	*	1 =	
SMALLINT	*	2 =	
INTEGER	*	4 =	
BIGINT	*	8 =	
DATE	*	4 =	
TIME	*	6 or 8* =	
TIME with ZONE	*	8 =	
TIMESTAMP	*	10 or 12* =	
TIMESTAMP/ ZONE	*	12 =	
DECIMAL 1-2	*	1 =	
3-4	*	2 =	
5-9	*	4 =	
10-18	*	8 =	
19-38	*	16 =	
FLOAT	*	8 =	
Fixed		SUM(n) =	
Variable		SUM(a) =	
		LOGICAL SIZE =	
		Overhead =	14
Partitioned Primary Index Overhead (2 or 8) =			
Variable Column Offsets (__ * 2) + 2 ; zero if no variable columns =			
_____ Bits for Compressible Columns _____ Nullable Columns _____ / 8 (Quotient only) =			
		PHYSICAL ROW SIZE =	

Example: Sizing a Row

The logical row layout example of the Employee table shown on the facing page will be used in an example of sizing a row. We will examine a row size calculation for the Employee table step-by-step on the following pages.

Example: Sizing a Row

EMPLOYEE

EMP #	SUPV EMP #	DEPT #	JOB CODE	LAST NAME	FIRST NAME	MIDDLE INITIAL	HIRE DATE	BIRTH DATE	SALARY AMOUNT
PK,SA	FK	FK	FK	NN	NN		NN	NN	NN
INTEGER	INTEGER	INTEGER	SMALL INTEGER	VARCHAR 30	VARCHAR 30	CHAR Fixed 1	DATE	DATE	DECIMAL (10,2)

Using this logical row layout, the next page will size a typical row of the Employee table.

Notes:

- **This table is not partitioned.**
- **Data analysis indicates that the typical Last Name is 10 bytes in length and the typical First Name is 8 bytes in length.**
- **Assume this a Teradata 14.0 Linux system and rows are on 2-byte boundaries.**

Example: Completing the Row Size Calculation Form

The example on the facing page shows the use of the "Row Size Calculation Form". We will use this form to determine the typical row size for the Employee table.

Variable Length Columns (Varchar, Varbyte, Vargraphic, etc.)

List all the variable columns on the left side of the form. For each column, determine the average number of bytes expected. Then add these up (SUM(a)). In the Employee table, there are two variable columns, LAST NAME and FIRST NAME. The average number of bytes expected for both columns is 18. Therefore the SUM(a) = 18. Copy this value to the right side of the form for SUM(a).

Fixed Length Columns

Determine how many of each data type there are in the table. In the Employee table, there are 1 SMALLINT, 3 INTEGER, 2 DATE, and 1 DECIMAL (10-18 digits). Enter these numbers to fill in the NUMBER OF COLS portion of the form. Multiply these counts against the sizing factor for that data type. For CHAR(n), BYTE(n), and GRAPHIC(n), add the total number of bytes they will require and list it on the right side of the form for SUM(n). This sum will only be 1 for the MIDDLE INITIAL.

Logical Size

The logical size is the amount of space needed by the data for a typical row. The logical size of an Employee row is 49 bytes.

Physical Size

You must add 14 bytes to allow for Row Length (2), Row Id (8), Spare Byte (1), mandatory Presence Byte (1), and Reference Array Pointer (2).

You must allow for the necessary 2-byte variable column pointers. Since there are 2 variable columns in the Employee table, you must allow for three offsets at 2 bytes each. If there are no variable columns in a table, this entry will be zero.

The system has use of seven bits of the mandatory Presence Byte. Every nullable column and every compressible column requires a Presence bit. The net calculation determines whether additional bytes will be needed for Presence bits. In the Employee table, only three Presence bits are needed. They will be taken from the mandatory Presence byte. No additional bytes are required.

Totaling the results, the size of an average Employee row is 70 or 72 bytes depending if the system has rows in Aligned Row format or Packed64 format.

Example: Completing the Row Size Calculation Form

Table Name EMPLOYEE

Variable Column Data Detail

Column Name	Type	Max	Averag
Last Name	VC	30	10
First Name	VC	30	8
	SUM(a) =		18

SUM(a) = SUM of the AVERAGE number of bytes expected for the variable columns.
SUM(n) = SUM of the fixed CHAR and GRAPHIC column bytes.

* Assume rows are on 2-byte boundaries.
* Assume 6 bytes for TIME and 10 bytes for TIMESTAMP data types.

Data Type	# of Columns		Size		TOTAL
BYTEINT		*	1	=	
SMALLINT	1	*	2	=	2
INTEGER	3	*	4	=	12
BIGINT		*	8	=	
DATE	2	*	4	=	8
TIME		*	6 or 8*	=	
TIME with ZONE		*	8	=	
TIMESTAMP		*	10 or 12*	=	
TIMESTAMP/ ZONE		*	12	=	
DECIMAL 1-2		*	1	=	
3-4		*	2	=	
5-9		*	4	=	
10-18	1	*	8	=	8
19-38		*	16	=	
FLOAT		*	8	=	
Fixed	1		SUM(n)	=	1
Variable			SUM(a)	=	18
			LOGICAL SIZE	=	49
			Overhead	=	14
Partitioned Primary Index Overhead (2 or 8)				=	
Variable Column Offsets (2 * 2) + 2 ; zero if no variable columns				=	6
__0_ Bits for Compressible Columns __4__ Nullable Column __4__ / 8 (Quotient only)				=	0
			PHYSICAL ROW SIZE	=	69 + 1 = 70

Row Size Exercise

Use the information below to fill in the worksheet on the facing page. This will give you the physical size of a Call table row. None of the columns has the COMPRESS option specified.

This table will be partitioned via RANGE_N on Call Date with Monthly intervals for 10 years.

This table is created on a Teradata 14.0 Linux system.

CALL

CALL NUMBER	TAKEN BY EMPOYEE NUMBER	CUSTOMER NUMBER	PLACED BY CONTACT NUMBER	PLACED BY EMPLOYEE NUMBER	ORIGINAL CALL NUMBER	CALL DATE	CALL TIME
PK, SA	FK, NN	FK	FK	FK	FK	NN	NN
UPI		NUSI				NUSI	

CALL STATUS CODE	CALL TYPE CODE	CALL PRIORITY CODE	AREA CODE	PHONE NUMBER	EXTENSION	SYSTEM NUMBER	PART CATEGORY	PART SERIAL NUMBER
FK, NN	FK, NN	FK, NN				FK	FK	
				NUSI				

DOMAIN CHART	DATA TYPE	SIZE
AREA CODE	SMALL INTEGER	2
CALL DATE	DATE	4
CALL NUMBER	INTEGER	4
CALL PRIORITY CODE	SMALL INTEGER	2
CALL STATUS CODE	SMALL INTEGER	2
CALL TIME	TIME WITH ZONE	8
CALL TYPE CODE	CHAR FIXED	2
CONTACT NUMBER	INTEGER	4
CUSTOMER NUMBER	INTEGER	4
EMPLOYEE NUMBER	INTEGER	4
EXTENSION	INTEGER	4
PART CATEGORY	INTEGER	4
PART SERIAL NUMBER	BIG INTEGER	8
PHONE NUMBER	INTEGER	4
SYSTEM NUMBER	INTEGER	4

Row Size Calculation Form

Table Name _____

Variable Column Data Detail

Column Name	Type	Max	Average
	SUM(a) =		

SUM(a) = SUM of the AVERAGE number of bytes expected for the variable columns.
SUM(n) = SUM of the fixed CHAR and GRAPHIC column bytes.

* Assume rows are on 2-byte boundaries.

* Assume 6 bytes for TIME and 10 bytes for TIMESTAMP.

Data Type	# of Columns	Size	TOTAL
BYTEINT	*	1 =	
SMALLINT	*	2 =	
INTEGER	*	4 =	
BIGINT	*	8 =	
DATE	*	4 =	
TIME	*	6 or 8* =	
TIME with ZONE	*	8 =	
TIMESTAMP	*	10 or 12* =	
TIMESTAMP/ ZONE	*	12 =	
DECIMAL 1-2	*	1 =	
3-4	*	2 =	
5-9	*	4 =	
10-18	*	8 =	
19-38	*	16 =	
FLOAT	*	8 =	
Fixed		SUM(n) =	
Variable		SUM(a) =	
		LOGICAL SIZE =	
		Overhead =	14
Partitioned Primary Index Overhead (2 or 8) =			
Variable Column Offsets (__ * 2) + 2 ; zero if no variable columns =			
_____ Bits for Compressible Columns _____ Nullable Columns _____ / 8 (Quotient only) =			
		PHYSICAL ROW SIZE =	

Sizing Tables and Indexes

Teradata supports variable length rows and variable length blocks within a table. These features provide maximum flexibility and save many hours of balancing data types, row sizes and block sizes. Other systems support different fixed length block sizes, but the designer must choose a single block size for each file. Though these other systems may allow variable length records, they can be wasteful of block space.

When sizing tables and indexes, the most important variables are the size of the rows and the number of rows (and whether the table is Fallback or not).

Row size and number determine space requirements. Variable length rows and blocks make it more difficult to accurately estimate the space required for tables and their indexes.

Sizing Tables and Indexes

These features make accurate space estimates for tables and their indexes more difficult.

- **These options on the Teradata Database support variable length rows:**
  - **COMPRESS**
  - **VARCHAR and LONG VARCHAR**
  - **VARBYTE and VARGRAPHIC**
- **Variable length blocks within a table are also supported.**
- **Teradata 13.10 compression features such as Algorithmic Level Compression (ALC) or Block level Compression (BLC).**

However, physical row size and table row count can be used to estimate space requirements.

Table Headers

Teradata creates a **table header** for each table.

- Table headers are a special one or two row block on each AMP having a specific Subtable ID. The header row contains all of the DD information about a table, its columns and its indexes which saves the Parser from having to send this DD information to the AMPs for every request.

- A table header is at least 512 bytes (1 sector). The number of columns and indexes a table has will affect the header size. We will assume that all table headers are at least 1024 bytes and we will assume for sizing exercises that table headers are 4 KB in size.

- Normally, table headers will be at least 1024 bytes in length. If you create a table with 3 or more columns, then the table header will be 1024 bytes in length. If you create a table with 2 columns and a NUSI, then the table header will be 1024 bytes.

An example of a table header is shown on the facing page.

Table Headers

- A copy of the table header exists on every AMP.
  - Table header is a separate subtable (subtable id 0).
- Minimum table header block size is 512 bytes (1 sector) per AMP.
  - The number of columns, secondary indexes, and compressed values impact table header size.
  - For example, tables with 3 or more columns will have a table header that is at least 1 KB in size.
- Typically, a table header will be at least 1024 bytes and may commonly be 4 KB with compression.
- Compressed values are maintained in the table header.
  - Compressed values are maintained in an array in the table header.
  - Example: 255 compressed values @ 20 bytes would increase the table header by 5K.
- Teradata 12.0 – maximum size of 128K
  Starting with Teradata 13.0 – maximum size of 1 MB
- The table header also covers all of its secondary index subtables.

Example of Table Header

STANDARD ROW HEADER LENGTH, ROW ID, PRESENCE/SPARE BYTES	
FIELD OFFSET BYTES (Fields 2-9)	
EXTRA OFFSET	
DATABASE AND TABLE NAMES	FIELD 1
DATABASE ID	
OTHER INTERNAL INFO CREATION DATE, PROTECTION, TYPE OF JOURNALING, JOURNAL ID, STRUCT VERSION, etc.	
INDEX DESCRIPTORS (36 BYTES * # INDEXES) PLUS 20 BYTES PER INDEX COLUMN	FLD. 2
ALWAYS NULL	FLD. 3
FASTLOAD & RESTORE INFORMATION USUALLY NULL	FLD. 4
BASE COLUMN INFO COUNT OF COLUMNS, LOCATION OF FIRST FIXED FIELD, NUMBER OF PRESENCE BITS, etc.	FIELD 5
COLUMN INFORMATION FOR EACH COLUMN 20 BYTES PER COLUMN (+ COMPRESS VALUE), DATA TYPE, OFFSET WITHIN ROW, NULLABLE/NOT NULLABLE, COMPRESS/NO COMPRESS, PRESENCE BIT LOCATION, etc.	
RESTARTABLE SORT INFORMATION USUALLY NULL	FLD. 6
REFERENTIAL INDEX INFORMATION (REF. INDEX DESCRIPTORS)	FLD. 7
LARGE OBJECT INFORMATION (LOB DESCRIPTORS)	FLD. 8
ALWAYS NULL	FLD. 9
ROW LENGTH or **REFERENCE ARRAY POINTER**	

Sizing a Data Table

The formulas shown on the facing page enable you to estimate the size of a data table. The formulas assume that every block is the same size. Though adequate for rough calculations, it is preferable to do the calculations over a range of block sizes. Since rows cannot be split over block boundaries, you must be certain to round down your first answer.

The size ranges for typical blocks are shown on the facing page. The typical block size will vary with the tuning of the maximum block size. The formula for calculating typical block size is also shown.

If the maximum block size is 64 KB, then a typical block size will be approximately 48 KB.

If the maximum block size is 127 KB, then a typical block size will be approximately 96 KB.

Notes:

You will commonly find table headers that are at least 1024 bytes long. This is especially true if you have multiple secondary indexes defined.

Data Block Header is 72 bytes in length. The Data Block Trailer is 2 Bytes long. Therefore, the block overhead is 72 + 2 or 74 bytes.

Sizing a Data Table

Block sizes vary within a table, so compute a range.

- **The typical maximum block size is 127 KB bytes (254 sector blocks), a typical block size is 96 KB, and assume table headers that are 4 KB in size.**

Formula:

(BlockSize - 74) / RowSize = RowsPerBlock	(rounded down)
RowCount / RowsPerBlock = Blocks	(rounded up)
NumAmps * 4096 = Header	(assumes 4 KB table headers)
(Blocks * BlockSize) + Header = NO FALLBACK	(BlockSize = Typical Block Size)
(Blocks * BlockSize) * 2 + Header = FALLBACK	(BlockSize = Typical Block Size)

Parameters:

74 = Block Header + Block Trailer	BlockSize	= Typical block size in bytes
4096 = Typical table header size	NumAmps	= Number of AMPs in the system
	RowCount	= Number of table rows expected
	RowSize	= Physical row size

Note:

For large tables, table headers and block overhead (74 bytes) add a minimal amount of size to the table. Therefore, multiply row size by number of rows and double for Fallback.

Table Sizing Exercise

Given the data on the facing page, estimate the size of this table; assume it is fallback protected. The formulas have been repeated for you.

The formula calculation is as follows:

98,304 – 74 = 98,230 ÷ 98 = 501 rows per block

501,000,000 ÷ 501 = 1,000,000 blocks

50 * 4096 = 204,800 bytes for block headers

No Fallback = (1,000,000 * 98,304) + 204,800 = 98,304,204,800 bytes

Fallback = (1,000,000 * 98,304) * 2 + 204,800 = 196,608,204,800 bytes

As you can see on the facing page, simple multiplying the number of rows by the row size gives a value of 196,392,000,000 which is fairly close to this formula's value.

Table Sizing Exercise

Given this data, estimate the size of a table with Fallback and a typical block size of 96K.

- BlockSize = 98,304 bytes (96 KB) and table headers that are 4 KB (4096)
- NumAmps = 50
- RowCount = 501,000,000
- RowSize = 196 bytes (includes overhead)

Formula:

- (BlockSize - 74) / RowSize = RowsPerBlock (round down)
- RowCount / RowsPerBlock = Blocks (round up)
- NumAmps * 4096 = Header
- (Blocks * BlockSize) + Header = No Fallback
- (Blocks * BlockSize) * 2 + Header = Fallback

Calculation:

(98,304 - 74) / 196 = 501 rows per block
501,000,000 / 501 = 1,000,000 blocks
50 * 4096 = 204,800 for table headers
(1,000,000 * 98,304) + 204,800 = 98,304,204,800 (No Fallback)
(1,000,000 * 98,304) * 2 + 204,800 = 196,608,204,800 (Fallback)

An easier way to estimate this table size:
501,000,000 x 196 bytes x 2 (Fallback) = 196,392,000,000

Estimating the Size of a USI Subtable

The formula on the facing page provides you with a means of calculating the amount of space required for a Unique Secondary Index.

Since there is one USI row for every base table row, the Row Count is the same as the number of rows in the base table. The Index Value Size is the size of the column(s) that comprise the index and varies depending on the USI. There is a minimum of 29 bytes of overhead in each subtable row.

The following formula can be used to estimate size of a USI subtable.

p = Number of data rows in the base table – same as USI subtable

po = Presence bit overhead (ceiling): ((1 + number of nullable USI fields) / 8); if none, then 0.

vo = Variable length field overhead: (number of variable length USI fields + 1) * 2. If none, then 0.

k = Length (in bytes) of a fixed length USI value (or the average length of a variable USI value)

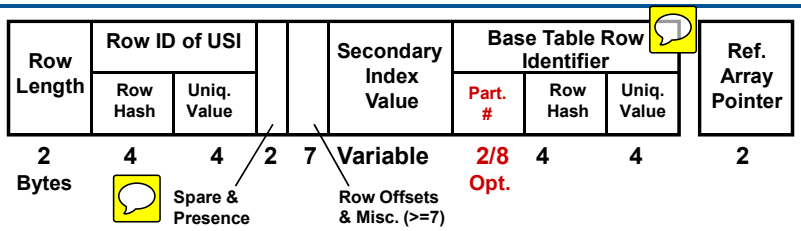
Row length	2	
USI Row ID	8	
Spare0	1	(present for general row compatibility)
Presence	1	(present for general row compatibility)
Offsets	6	(three 2 byte offsets – see note below)
Index Row Continuation	1	(used with NUSI if Row IDs continue to another row; set to x'00' for USI)
Presence/Null Bits (po)	(0+)	(exists only if USI columns are compressible or nullable)
Variable offsets (vo)	(0+)	(exists if SI columns are variable length)
USI value (k)		(variable)
Base Table Row ID	8 – 16	(10 or 16 if base table has a PPI)
Reference Array	2	
	---	
at least	29 – 37	+ USI value length (31/37 for PPI)

USI subtable (no fallback) size = p * (k + po + vo + 29 or 31)
USI subtable (fallback) size = 2p * (k + po + vo + 29 or 31)

Note: The offsets are used as following:

Offset[0] is the offset of the first byte for the USI value.
Offset[1] is the offset where the base table Row ID is found.
Offset[2] is the offset after the base table Row ID.

Estimating the Size of a USI Subtable



There is one Index row for each Base Table row.

USI subtable row size = (Index value size + 29 or 31/37 for partitioned tables)

Where 29 = 4 (Row Header and Row Reference Array pointer)
+ 8 (This row's Row ID)
+ 9 (Spare, presence, and offset bytes – a multi-column index with compressed/null columns may require additional presence bits)
+ 8 (Base table Row ID; or 10/16 for PPI tables; 16 if # partitions is > 65,535)

To estimate the amount of space needed for a USI subtable, you can use the following formulas.

For tables with NPPI, USI Subtable Size = (Row count) * (index value size + 29)

For tables with PPI, USI Subtable Size = (Row count) * (index value size + 31 or 37)

Note: Double this figure for Fallback.

Estimating the Size of a NUSI Subtable

The following information is needed to calculate the size of a NUSI subtable.

- For NPPI tables - (Row Count * 8) is derived from the 8 bytes of Row ID which the subtable stores for each row in the base table. This gives us the total number of bytes devoted to base table Row IDs.

- For PPI tables – (Row Count *10 or 16) is derived from the 8 bytes of Row ID plus 2 or 8 bytes for the partition #. The subtable stores 10/16 bytes for each row in the base table with a PPI. This gives us the total number of bytes devoted to base table Row Identifiers.

- (# distinct values) is an estimate of the number of NUSI subtable rows since a NUSI subtable contains at least one index row per AMP for each distinct index value in the base table on that AMP.

- The 21 bytes of overhead per subtable row are the same as the 29 bytes (minus 8 for Row ID) for a USI. The offsets are used as following:

  Offset[0] is the offset of the first byte for the NUSI value.
  Offset[1] is the offset where the first base table Row ID is found.
  Offset[2] is the offset after the last base table Row ID is found.

  Note: The Index Row Continuation byte is used with NUSIs. If the NUSI subtable row cannot hold all of the Row IDs for data rows on the AMP, this flag is set to indicate that an additional subtable row is needed. The last (or only) subtable row has this flag set to x'00'.

- MIN(NumAMPs, RowsPerValue) is the minimum of the two (see Case 1 and Case 2 below).

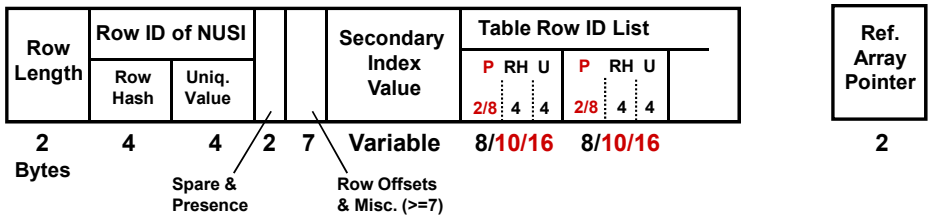
Case 1: NumAMPs < RowsPerValue

If there are fewer AMPs than RowsPerValue, at least one row from each NUSI value will probably be distributed to each AMP. This means the subtable on each AMP will contain a row for every NUSI value.

Case 2: NumAMPs > RowsPerValue

If there are more AMPs than RowsPerValue, at least some AMPs will be missing some NUSI values. This means that some AMPs will not have a subtable row for every NUSI value.

Estimating the Size of a NUSI Subtable



There is at least one index row per AMP for each distinct index value that is in the base table on that AMP.

To estimate the size of a NUSI subtable, …

Size = (Row count) * 8 (or 10/16 for PPI tables; 16 if # partitions is > 65,535)
+ ((#distinct values) * (Index value size + 21) * MIN (#AMPs, Rows per value))

MIN(___ , ___) — use the smaller of the two values.

Double this figure for Fallback.

Example:

More typical rows/value than AMPS:
(50 rows/value, 10 AMPS)

- Every AMP probably has every value.
- Every AMP has a subtable row for every value.
- More weakly selective.
- More rows returned from an equality search.

More AMPs than typical rows/value:
(10 AMPS, 5 rows/value)

- NOT Every AMP has every value.
- NOT Every AMP has a subtable row for every value.
- More strongly selective.
- Fewer rows returned from an equality search.

Estimating the Size of a Reference Index Subtable

The formula on the facing page provides you with a means of calculating the amount of space required for a Reference Index – created to support a References constraint.

A RI row is similar to an USI row except instead of recording a 8 byte data Row ID, a 4 byte foreign key row count and a Valid flag is saved instead.

The following formula can be used to estimate size of a RI subtable.

p = Estimated number of distinct foreign key value excluding foreign key value which contains NULL.

po = Presence bit overhead (ceiling): ((1 + number of nullable foreign key fields) / 8); if none, then 0.

vo = Variable length field overhead: (number of variable length foreign key fields + 1) * 2; if none, then 0.

k = Length (in bytes) of a fixed length foreign key value (or the average length of a variable length foreign key value).

Row length	2	
Refer. Index Row ID	8	
Spare0	1	(present for general row compatibility)
Presence	1	(present for general row compatibility)
Offsets	6	(three 2 byte offsets)
Validity Flag	1	(x'00' is valid; x'01' is invalid)
Presence/Null Bits (po)	(0+)	(exists only if FK columns are nullable)
Variable offsets (vo)	(0+)	(exists if FK columns are variable length)
Foreign key value	(k)	(variable)
Foreign Key Count	4	
Reference Array	2	
	---	

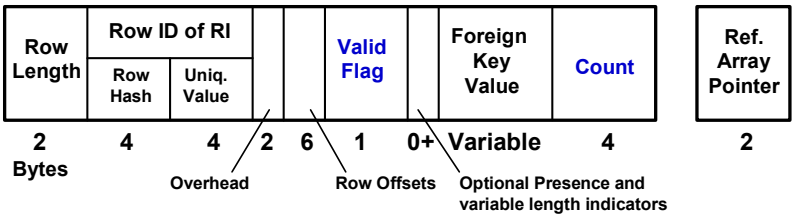
at least 25 + length of Foreign Key

RI subtable (no fallback) size = p * (k + po + vo + 25)
RI subtable (fallback) size = 2p * (k + po + vo + 25)

The offsets are used as following:

Offset[0] is the offset of the validity flag byte.
Offset[1] is the offset where the row count can be found.
Offset[2] is the offset after the row count.

Estimating the Size of a Reference Index Subtable



There is one reference index row for each distinct foreign key value.

RI subtable row size = (Index value size + 25)

Where 25 = 4 (Row Header and Row Ref. Array pointer)
+ 8 (This row's Row ID)
+ 8 (Overhead and row offset bytes)
+ 1 (Validity flag)
+ 4 (Count)

To estimate the size of a Reference Index (RI) subtable, you can use the following formula.

RI Subtable Size = (Distinct count) * (index size + 25)

Double this figure for Fallback.

Index Sizing Exercise

Use the formulas provided to solve the exercise on the facing page.

Index Sizing Exercise

A customer has a Teradata 14.0 system with 168 AMPs.

- **How much space will a USI and NUSI require for 1,000,000 row Fallback table?**
- **The USI is an Integer data type; the NUSI is a CHAR(18) data type and has 20 rows per value with 50,000 distinct values.**
- **The table has multi-level partitioning (< 65,535 partitions). Estimate the space for each secondary index.**

Formulas:

USI Size = Row Count * (IndexValueSize + 31)

NUSI Size = (Row Count * 10) + (#distinct values) * (IndexValueSize + 21) * MIN (#AMPs , rows/value)

Other Sizing Techniques

The DBC.TableSizeV view provides actual table space information, but it is a total value for the data table and all of its subtables (e.g., Fallback, Secondary Indexes, etc.).

Techniques that can be used to help determine the size of data rows and specific index subtables include:

- For a new table, load a portion of the data (e.g., 1%) and use the DBC.TableSizeV view and the "Empirical Sizing" technique to estimate the size of a table and each index as it is created.

  Subtract the size without the index from the size with the index.

- For an existing table, use the COLLECT DEMOGRAPHICS command to capture demographic information into a QCD (Query Capture Database).

  Query the data demographics view to access actual row lengths and row counts.
  Multiply the average subtable row length x number of subtable rows to determine the size of an index subtable.

- For an existing table, use the system utility Ferret with the ShowBlocks command to determine the size of data and index subtables.

  This requires access to system console utilities.
  This technique is not discussed in this course.

Other Sizing Techniques

The DBC.TablesizeV view provides permanent space information about a table and all of its subtables.

- **Sizing information is not provided for individual secondary index subtables.**

Other techniques that can be used to help determine the size of data rows and index subtables include:

- **For a new table, use the DBC.TablesizeV view and the "Empirical Sizing" technique to estimate the size of a table and each index as it is created.**
  - **Subtract the size without the index from the size with the index.**
- **For an existing table, use the COLLECT DEMOGRAPHICS command to capture demographic information into a QCD (Query Capture Database).**
  - **These demographics provide actual row lengths and row counts.**
  - **Multiply the average subtable row length x number of subtable rows to determine the size of an index subtable.**

Empirical Sizing

The most accurate way to size a production table and its indexes is **Empirical Sizing**. This method consists of loading a portion of the table, measuring how much space it takes, and extrapolating the total space required for the table from this information.

The steps involved in Empirical Sizing are shown at the top of the facing page.

The example shows the SQL necessary to query the DD.

- In Step 2, SUM(CurrentPerm) gives you the space, which is required to store the portion of the table itself.

- In Step 4, SUM(CurrentPerm) gives you the amount of space required to store those table rows plus the index which was defined in Step 3. Subtract the results to compute the size of the index.

- Running the SQL statement in Step 2 without the AND clause will give you the amount of space for the entire database, not just a single table.

CurrentPerm is always expressed in whole sectors, rounded up. If only a 10-byte row was in Perm Space, CurrentPerm would be reported as 512 bytes (1 sector).

NOTE: DATABASE is a key word and represents your current database.

Empirical Sizing

An excellent way to size a production table, including indexes is:

1. Load a known percentage of rows onto the system.
2. Query the DD through the view DBC.TableSizeV.
3. Create one index.
4. Query the DD through the view DBC.TableSizeV.
5. Repeat steps 3 and 4 as necessary.
6. Multiply the results to determine the production size.

Example:

Step 1 Load 1% of a table onto a system.

Step 2
```
SELECT    SUM(CurrentPerm) FROM DBC.TablesizeV
WHERE     DatabaseName = DATABASE
AND       TableName = 'Sales';
```

```
Sum(CurrentPerm)
       49,818,624
```

Note: The same query without the SUM keyword returns per/AMP figures which reveal distribution efficiency.

Step 3 CREATE INDEX (sales_date) ON Sales;

Step 4
```
SELECT    SUM(CurrentPerm) FROM DBC.TablesizeV
WHERE     DatabaseName = DATABASE
AND       TableName = 'Sales';
```

```
Sum(CurrentPerm)
       57,300,480
```

Therefore, index size is:

```
  57,300,480
– 49,818,624
   7,481,856
```

If the sample data was 1%, then the index size would be 748 MB.

Collect Demographics Command

The COLLECT DEMOGRAPHICS command collects various table demographics and writes the data to the DataDemographics table of a user-defined QCD database. These demographics are primarily used for subsequent analysis tools such as the Teradata Index Wizard.

You can also query the DataDemographics table in the user-defined QCD via a view named DataDemographicsView.

You must have the following privileges to execute COLLECT DEMOGRAPHICS:

- DELETE on the DataDemographics table in *QCD_name*.
- INSERT on the DataDemographics table in *QCD_name* or INSERT on the *QCD_name database*.
- SELECT on the specified tables or containing databases.

Options: WITH NO INDEX is used to exclude index subtable demographics from the collection and collect only primary data. ALL (data and index demographics is the default).

COLLECT DEMOGRAPHICS determines and writes the following information for each specified table on an AMP-by-AMP basis into the DataDemographics table of the specified QCD (Query Capture Database):

- DatabaseName and TableName
- Subtable type and Subtable ID
- Cardinality and Average row length
- System-related information (Machine Name, Collect Time, etc.)

Demographics captured by COLLECT DEMOGRAPHICS are not deleted when you perform associated DROP actions on the subject table and must be deleted explicitly.

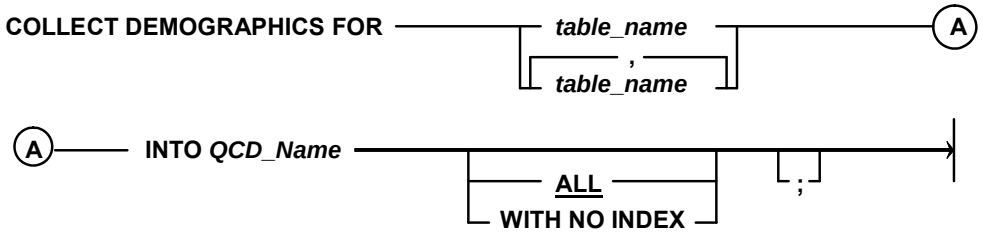
- COLLECT DEMOGRAPHICS does not capture information for the QCD table TableStatistics. The COLLECT STATISTICS (QCD form) is needed to capture statistics for the data analysis tools (e.g., Teradata Index Wizard).

- These are not the same statistics that are used by the optimizer. Statistics collected by this statement are used for index analysis and validation tasks performed by data analysis tools such as the Teradata Index Wizard: these statistics are *not* used by the Optimizer to process queries.

```
COLLECT STATISTICS FOR SAMPLE 100 PERCENT INTO QCD
ON Orders_PPI COLUMN orderid;
```

If you collect data demographics with Visual Explain (INSERT EXPLAIN … WITH DEMOGRAPHICS), when you delete the relevant query plans, then the data demographics are also automatically deleted.

COLLECT DEMOGRAPHICS Command



Notes:

- Collects various table demographics and writes the data to the DataDemographics table of a user-defined QCD database.
- COLLECT DEMOGRAPHICS determines and writes the following information for each specified table per AMP:
  - Subtable ID
  - Cardinality
  - Average row length
  - Subtable type
  - System-related information (Machine name, collected time, etc.)
- If an entry already exists in the DataDemographics table for the specified table, then it is updated with the currently collected data.
- This collection can also be invoked via the Visual Explain or Index Wizard utilities.

Collect Demographics Example

The facing page contains an example of using the COLLECT DEMOGRAPHICS command for a table named Orders_PPI.

The table definition is:

```
CREATE SET TABLE TFACT.Orders_PPI, FALLBACK,
    (orderid INTEGER NOT NULL,
     custid INTEGER NOT NULL,
     orderstatus CHAR(1),
     totalprice DECIMAL(9,2) NOT NULL,
     orderdate DATE FORMAT 'YYYY-MM-DD' NOT NULL,
     orderpriority SMALLINT,
     clerk CHAR(16),
     shippriority SMALLINT,
     ordercomment VARCHAR(79)
PRIMARY INDEX ( orderid )
PARTITION BY RANGE_N(orderdate  BETWEEN DATE '2003-01-01' AND DATE
'2012-12-31' EACH INTERVAL '1' MONTH )
UNIQUE INDEX orderid ( orderid )
INDEX custid ( custid );
```

Note: If you name your index, the index name is captured in the DataDemographics table, but only for the first AMP (AMP 0). The index column names are not captured in this table.

If you haven't named your index, you will have to use the subtable ids to reference the index subtables.

Using the information from the DataDemographicsView, you can calculate the size of the various subtables. Note that this view does not include the fallback subtables.

Primary Data (1024)	= 96000 x 66 x 2 (for fallback) = 12,672,000
USI subtable (1028)	= 96000 x 36 x 2 (for fallback) = 6,912,000
NUSI subtable (1032)	= 90200 x 38 x 2 (for fallback) = 6,855,200

COLLECT DEMOGRAPHICS Example

Example: The Orders_PPI table is partitioned with the following demographics:

- Fallback protected; 96,000 rows; typical row size is 66 bytes (includes overhead)
- Integer NUPI that is partitioned by order date; distributed across a 20-AMP system
- Integer USI
- Integer NUSI – distinct values is 4810, typical rows per value is 24

```
COLLECT DEMOGRAPHICS FOR TFACT.Orders_PPI into QCD;
```

Query the QCD.DataDemographicsView to determine average row length.

```
SELECT      Databasename, Tablename, SubtableType, SubtableID, SUM(RowCount),
            AVG(AvgRowSize)
FROM        QCD.DataDemographicsView
ORDER BY    SubtableID
GROUP BY    Databasename, Tablename, SubtableType, SubtableID;
```

DatabaseName	TableName	SubTableType	SubTableID	Sum(RowCount)	Average(AvgRowSize)
TFACT	ORDERS_PPI	Data	1024	96,000	66
TFACT	ORDERS_PPI	SecondaryIndex	1028	96,000	36
TFACT	ORDERS_PPI	SecondaryIndex	1032	90,200	38

Calculation: The size of the NUSI subtable is 90,200 x 38 x 2 = 6,855,200 bytes.

Notes: The QCD.DataDemographics table does not include fallback subtables.
This table does include the names of named indexes, but not index column names.

Spool Space

When sizing databases, it is important to estimate the amount of Spool space required. Maximum spool-space needs will vary with table size, use (type of application), and frequency of use.

It is a misconception that all unused space can be used for spool - only free cylinders are available for spool. The unused or fragmented space within permanent cylinders cannot be used for spool. This space is used by the file system for updates/insert/deletes requiring larger/new blocks. When blocks are expanded or split, the file system returns blocks to the free block list and this space is available within the cylinder for future updates/inserts/deletes.

Large systems use more spool space to duplicate tables on each AMP to perform Product Joins.

Empty cylinders are used for Spool. Recall that the Spool limit for a User is specified in the CREATE USER statement and may be changed dynamically. Avoid copying or redistributing entire tables to Spool unless absolutely necessary (to keep from exceeding the Spool limit).

A user may run out of Spool space because of an incorrectly coded query, etc. and will receive a message that their SQL statement has been aborted. If the user exceeds their Spool space limit, they will receive the following error message.

2646 – No more spool space in *username*

If a user has an appropriate amount of spool space for their request and receives this message (2646), usually the problem is that the spool file is poorly distributed among the AMPs. This usually occurs in a join operation when the Optimizer redistributes the tables on a column that is not very unique.

If the AMP runs out of Spool space (insufficient available cylinders for Spool), the following message will be displayed. This message indicates that a request has exceeded the physical storage limits imposed by a given configuration without exceeding the logical limits imposed by the PERM or SPOOL space allocation for a given database/user.

2507 - Out of spool space on disk

Spool Space

- Maximum spool space needs vary with table size, use (type of application), and frequency of use.
- Large systems use more spool to duplicate tables on each AMP.
- Cylinders not currently used for data may be used for spool.
  - It is a misconception that all unused space can be used for spool – only free cylinders are available for spool. The unused space within permanent cylinders cannot be used for spool.
- The user's maximum spool space limit can be changed dynamically.
- Avoid unnecessary copying or redistribution of entire tables to spool.

As user concurrency and/or SQL complexity increases, add more SPOOL.

Running out of Spool Space

- If a user exceeds their Spool space limit, they will receive the following error message.

  2646 – No more spool space in *username*

- If the AMP runs out of Spool space (insufficient available cylinders for Spool), the following message will be displayed.

  2507 - Out of spool space on disk

Release of Spool

Intermediate Spool results are held until they are no longer needed. The EXPLAIN facility uses the term “Last Use” to designate the step after which the Spool is released.

Output (final) Spool results are held until one of the events listed on the facing page occurs. System Restarts cause all Spool results to be released.

- Since Master Indexes are kept in memory and never written to disk, they must be rebuilt from the Cylinder Indexes after every Restart.

Previously, this course stated that the Table ID contained information (Unique Value) which defined the type of table or file (data table, Permanent Journal, or Spool file).

- In rebuilding the Master Index after a Restart, the File System examines every Cylinder Index.

- If a Spool block is found on a cylinder, this means that the entire cylinder must contain only Spool and Free Blocks (since you cannot have both data and Spool on the same cylinder).

- All of the in-use space for that cylinder is put onto the Free Block List and the Cylinder Index is rewritten to indicate that the cylinder is empty. That cylinder is then put onto the Free Cylinder List in the Master Index.

Release of Spool

Intermediate Spool

Intermediate Spool results are held until the (LastUse) Explain step.

Output Spool

Output Spool results are held until:

- Last spool Response
- CLOSE cursor
- ERQ, Terminate function
- Session ends
- System is restarted

- – BTEQ
- – Preprocessor
- – CLI
- – Job Abort, timeout, logoff, etc.

System Restart – each AMP rebuilds its Master Index from its Cylinder Indexes.

The AMPs delete all spool files by moving them to the Free Cylinder List.

This costs only one I/O per spool cylinder.

System Sizing Exercise

The exercise on the facing page provides an estimate only. The intent of this exercise is to show that you have to account for more than just user data.

Obviously, customer requirements can vary greatly and the percentages for spool, work space, indexes, permanent journals, etc. will vary as well.

System Sizing Exercise

Some general guidelines for estimating system size:

40% of total space for Spool
10% of total space for DBC, WAL space (Transient Journal), Permanent Journals, etc.
20 - 50% of data size for indexes

- If not Fallback, multiply the amount of raw data by a factor of 3 or 4 (for 40% spool).
- If using Fallback, multiply the amount of raw data by a factor of 5 or 6 (for 40% spool).
- Note: These factors will be smaller if the amount of required spool is only 25% and if aggressive compression is used (block level, etc.)

Example assuming Fallback:

User raw data	100 TB
Estimate of Vdisk space needed	500 to 600 TB

Proof:

Estimate of Vdisk space	500 to 600 TB
– Spool (40%)	– 200 – 240 TB
– DBC, WAL Space, etc. (10%)	– 50 – 60 TB
	250 to 300 TB
User raw data	100 TB
20 - 50% for indexes	20 to 50 TB
Fallback (for data and indexes)	240 to 300 TB

A 28 node 6650 system with 42 AMPs/node and .54 TB of MaxPerm space/AMP would meet this requirement.

28 x 42 x .54 TB = 635 TB MaxPerm space

Assumes 600 GB disks and 2 disks per AMP with RAID 1.

Sizing Summary

The facing page summarizes the primary points to remember when sizing databases.

- To get accurate space estimates, you must start with accurate row counts and row sizes.
- When calculating the space necessary for a database, you must include all of the components from the list.

Sizing Summary

Accurate row counts and sizes are needed to get good space estimates.

Database sizing includes:

- Tables + Fallback +
- Secondary Indexes + Fallback +
- Reference Indexes + Fallback +
- Join Indexes + Fallback +
- Hash Indexes + Fallback +
- Permanent Journal (dual or single) +
- Stored Procedure space +
- Spool space +
- Temporary space +
- WAL Space (Transient Journal, etc.)

Module 23: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 23: Review Questions

1. Which choice can be used with the COMPRESS option?
   a. Identity column
   b. Non-unique Primary Index
   c. USI as PK with Standard RI
   d. Non-unique Secondary Index

2. Which section of a row identifies the starting location of variable length column data and is present only if variable length columns without compression are declared?
   a. Presence Bits
   b. Column Offsets
   c. VARCHAR Columns
   d. Uncompressed Columns

3. How can you override the default that a column with a NULL value will require row space?
   a. Use the NOT NULL option on the column as part of the CREATE TABLE statement.
   b. Set the user's default so columns will default to COMPRESS when creating a table.
   c. Use the COMPRESS option on the column as part of the CREATE TABLE statement.
   d. Use the DEFAULT NULL option on the column as part of the CREATE TABLE statement.

4. What is the minimum space the table headers will take for a 6-column table on a 10 AMP system?
   a. 1024 bytes
   b. 4096 bytes
   c. 5120 bytes
   d. 10240 bytes

5. What DD view is used to get sizing information about tables? _____

Lab Exercise 23-1

The following SQL can be used to determine the size of a table.

SUM of Perm space using the DBC.TableSizeV view.

```
SELECT      TableName (CHAR(15)), SUM(CurrentPerm)
FROM        DBC.TableSizeV
WHERE       DatabaseName = DATABASE
AND         TableName = 'tablename'
GROUP BY    1
ORDER BY    1;
```

Remember that compression is case-sensitive. The city names have to be compressed as shown on the facing page.

Lab Exercise 23-1

Lab Exercise 23-1

Purpose

In this lab, you will compress multiple values for a column in order to reduce Perm space.

What you need

Populated AP.Accounts table and an empty table in your database

Tasks

1. Populate your Accounts table from the AP.Accounts table using the INSERT/SELECT statement:

   INSERT INTO Accounts SELECT * FROM AP.Accounts;

   Using the DBC.TableSizeV view, what is the amount of Perm space used. Accounts =_____

2. Create a new table, named "Accounts_MVC", based on the Accounts table except compress the following city names:

   Culver City, Hermosa Beach, Los Angeles, and Santa Monica

   Populate your Accounts_MVC table from the AP.Accounts table using INSERT/SELECT.

   Using DBC.TableSizeV, what is the amount of Perm space used. Accounts_MVC =_____

Lab Exercise 23-2

Use DBC.TableSizeV to determine the number of bytes that a table is currently using.

SUM of Perm space using the DBC.TableSizeV view.

```
SELECT      TableName (CHAR(15)), SUM(CurrentPerm)
FROM        DBC.TableSizeV
WHERE       DatabaseName = DATABASE
AND         TableName = 'tablename'
GROUP BY    1
ORDER BY    1;
```

Calculating table size

Since the Row Layout of the Trans table is simple, the row length can easily be calculated as:

Trans_Number	INTEGER	4
Trans_Date	DATE	4
Account_Number	INTEGER	4
Trans_ID	CHAR(4)	4
Trans_Amount	DECIMAL(10,2)	8
Row Overhead		14
Row Length		38

To estimate table size, use the following formula which is to simply multiply the number of rows x typical row length and double if the table is fallback protected.

Row Length x 15,000 (# of rows) x 2 (Fallback) = Estimated Table Size

Lab Exercise 23-2

Lab Exercise 23-2

Purpose

In this lab, you will populate tables, determine tables sizes, and create secondary indexes.

What you need

Populated AP.Trans table and an empty table in your database

Tasks

1. Determine the size of your empty Trans table using DBC.TableSizeV (SELECT with and without the SUM aggregate function).

   Size of empty Trans = _____

   What size are the table headers on each AMP? _____

2. Since the typical row length is 38 bytes (see facing page), estimate the size of this table assuming it will have 15,000 rows.

   Estimated size of Trans = _____

3. Populate your Trans table from the AP.Trans table using the following INSERT/SELECT statement:

   INSERT INTO Trans SELECT * FROM AP.Trans;
   Use the SELECT COUNT(*) function to verify the number of rows. _____

Lab Exercise 23-2 (cont.)

Use DBC.TableSizeV to determine the number of bytes that a table is currently using.

SUM of Perm space using the DBC.TableSizeV view.

```
SELECT      TableName (CHAR(15)), SUM(CurrentPerm)
FROM        DBC.TableSizeV
WHERE       DatabaseName = DATABASE
AND         TableName = 'tablename'
GROUP BY    1
ORDER BY    1;
```

Estimating USI Size

USI Subtable Size = ((Row count) * (index value size + 29))

Double this figure for Fallback.

Estimating NUSI Size

```
NUSI Subtable Size  = (Row count) * 8
                    + ( (#distinct values)
                        *  (Index value size + 21)
                        *  MIN( (#AMPs) , (Rows per value) ) )
```

Double this figure for Fallback.

To determine the number of "rows per value", determine the distinct values in a table and divide the total number of rows by the number of distinct values.

```
SELECT COUNT(DISTINCT(col_name)) FROM tablename;
```

Calculation Hint: Remember that all rows (including index subtable rows start on an even address (the subtable rows are an even number of bytes in length).

Result Note: **The NUSI calculation will not as close because the NUSI has one value that has a large number of duplicate values.** When the NUSI is skewed, the estimation will usually be more than the actual subtable size.

Lab Exercise 23-2 (cont.)

4. Using the DBC.TableSizeV view, determine the actual size of the Trans table by using the SUM aggregate function.

   Size of populated Trans = _____

5. Create a USI on the Trans_Number column.

   Estimate the size of the USI = _____
   Actual size of the USI = _____ (use the empirical sizing technique)

6. Create a NUSI on the Trans_ID column.

   Estimate the size of the NUSI = _____ (Hint: use DISTINCT function)
   Actual size of the NUSI= _____ (use the empirical sizing technique)

Lab Exercise 23-3 (optional)

Use DBC.TableSizeV to determine the number of bytes that a table is currently using.

SUM of Perm space using the DBC.TableSizeV view.

```
SELECT      TableName (CHAR(15)), SUM(CurrentPerm)
FROM        DBC.TableSizeV
WHERE       DatabaseName = DATABASE
AND         TableName = 'tablename'
GROUP BY    1
ORDER BY    1;
```

Lab Exercise 23-3 (optional)

Lab Exercise 23-3 (optional)

Purpose

In this lab, you will determine tables sizes and establish referential integrity between two tables.

What you need

Populated PD tables and empty tables in your database

Tasks

1. Populate your Employee and Emp_Phone tables from the PD.Employee and PD.Emp_Phone tables using the following INSERT/SELECT statements.

```
INSERT INTO Employee     SELECT * FROM PD.Employee;
INSERT INTO Emp_Phone  SELECT * FROM PD.Emp_Phone;
```

2. Using the DBC.TableSizeV view, determine the actual size of the Emp_Phone table by using the SUM aggregate function.

   Size of populated Emp_Phone table = _____

Lab Exercise 23-3 (optional – cont.)

To create a References constraint:

```
ALTER TABLE          child_tablename
ADD CONSTRAINT       constraint_name
    FOREIGN KEY      (child_name)
    REFERENCES       parent_tablename (parent_column);
```

To drop a named References constraint:

```
ALTER TABLE          child_tablename
    DROP CONSTRAINT  constraint_name;
```

SUM of Perm space using the DBC.TableSizeV view.

```
SELECT    TableName (CHAR(15)), SUM(CurrentPerm)
FROM      DBC.TableSizeV
WHERE     DatabaseName = DATABASE
AND       TableName = 'tablename'
GROUP BY  1
ORDER BY  1;
```

Estimating Reference Index Size

Reference Index Subtable Size = ((Distinct # of values) * (index value size + 25))

Double this figure for Fallback.

To determine the number if distinct values in a table:

```
SELECT COUNT(DISTINCT(col_name)) FROM tablename;
```

Calculation Hint: Remember that all rows (including index subtable rows) start on an even address (the subtable rows are an even number of bytes in length).

Lab Exercise 23-3 (optional – cont.)

3. The Foreign key is Employee_Number in PD.Emp_Phone and the Primary Key is the Employee_Number in PD.Employee.

   Create a References constraint on Employee_Number using the following SQL statements.

   ```
   ALTER TABLE Emp_Phone ADD CONSTRAINT fk1
   FOREIGN KEY (Employee_Number)
   REFERENCES Employee (Employee_Number);
   ```

   (use the HELP CONSTRAINT Emp_Phone.fk1; to view constraint information.

4. Using the DBC.TableSizeV view, determine the actual size of the Emp_Phone table by using the SUM aggregate function.

   Estimate the size of the Reference Index = _____

   Size of populated Emp_Phone with references index = _____

   Size of references index = _____

5. Drop the Foreign Key constraint by executing the following SQL command.

   ```
   ALTER TABLE Emp_Phone DROP CONSTRAINT fk1;
   ```

Notes

Module 24

SQL Parser

After completing this module, you will be able to:

- **Describe internal, channel, and LAN parcels.**
- **Explain software cache functionality.**
- **List and identify the function of the main components (phases) of the parser.**
- **Describe the functionality of Request-To-Steps cache.**

Notes

Table of Contents

Internal, Channel and LAN Parcels 24-4
Request Parcel 24-6
The Data Parcel 24-8
SQL Parser Overview 24-10
Software Cache 24-12
Request-To-Steps Cache 24-14
  Request-to-Steps Cache Check 24-16
  Request-To-Steps Cache Logic 24-18
Dictionary Cache 24-20
Syntaxer 24-22
Resolver 24-24
Security 24-26
Optimizer 24-28
Generator 24-30
Apply 24-32
Dispatcher 24-32
SQL Parser Review 24-34
Parser Summary 24-36
Module 24: Review Questions 24-38
  Module 24: Review Questions (cont.) 24-40

Internal, Channel and LAN Parcels

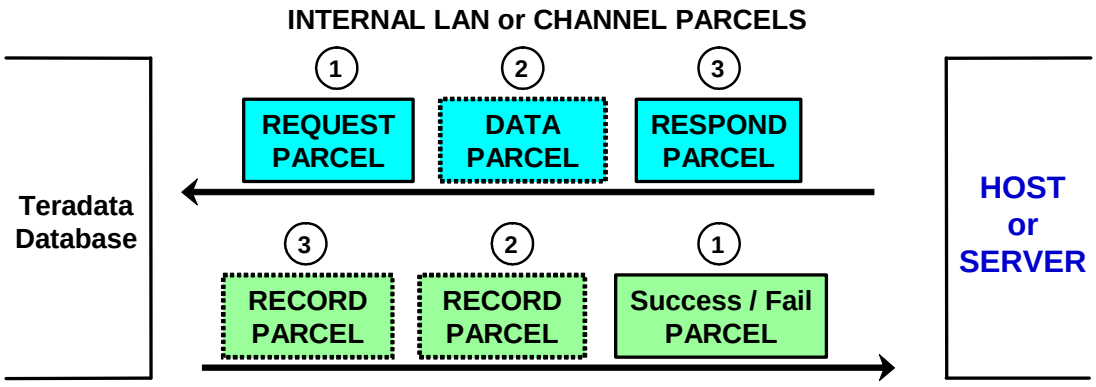
The diagram on the facing page illustrates how an SQL Request is submitted to Teradata by an SQL application (host). It also illustrates how Teradata replies to the SQL Request.

The SQL application sends out a Request parcel followed by a Data parcel (if necessary) and a Respond parcel. In return, Teradata sends back a SUCCESS/FAIL parcel, which may be followed by one or more Record parcels.

SQL requests and responses are "carried" to/from Teradata using the services of the CLI (Call Level Interface). The CLI constructs (and deconstructs) the "parcels" which represent the units of work that Teradata handles. One of the parcels, called a Request Parcel, carries the actual SQL code. It is not "translated" until it gets to the Parser, where it becomes a set of "steps" which will be sent to the AMPs. The AMPs create Response Parcels that contain the information to be sent back to the user session. The CLI (on the client side) receives these parcels, de-blocks them if necessary, and returns the requested information to the user.

Parcels are transparent to the user. The CLI keeps them transparent.

Internal, Channel and LAN Parcels



- A REQUEST parcel is followed by zero or one DATA parcel plus one RESPOND parcel.
- The RESPOND parcel identifies response buffer size.
- A RESPOND parcel may be sent by itself as a continuation request for additional data.
- A SUCCESS parcel may be followed by RECORD parcels.
- Every REQUEST parcel generates a SUCCESS/FAIL parcel.

Request Parcel

A Request parcel contains one or more whole SQL statements. Normally, a Request parcel represents a single transaction. Some transactions may require multiple Request parcels.

If you execute the same SQL statement 100 times, each Request for execution sends the entire SQL statement to Teradata for parsing and execution.

As mentioned earlier, Request parcels are the units upon which the parser acts.

Request Parcel

A Request Parcel:

- Must contain at least one SQL statement.
- May contain two or more SQL statements (multi-statement request).
- May be a transaction by itself (default).
- May be one parcel of a multi-request transaction.

A Request Parcel is the parsing unit.

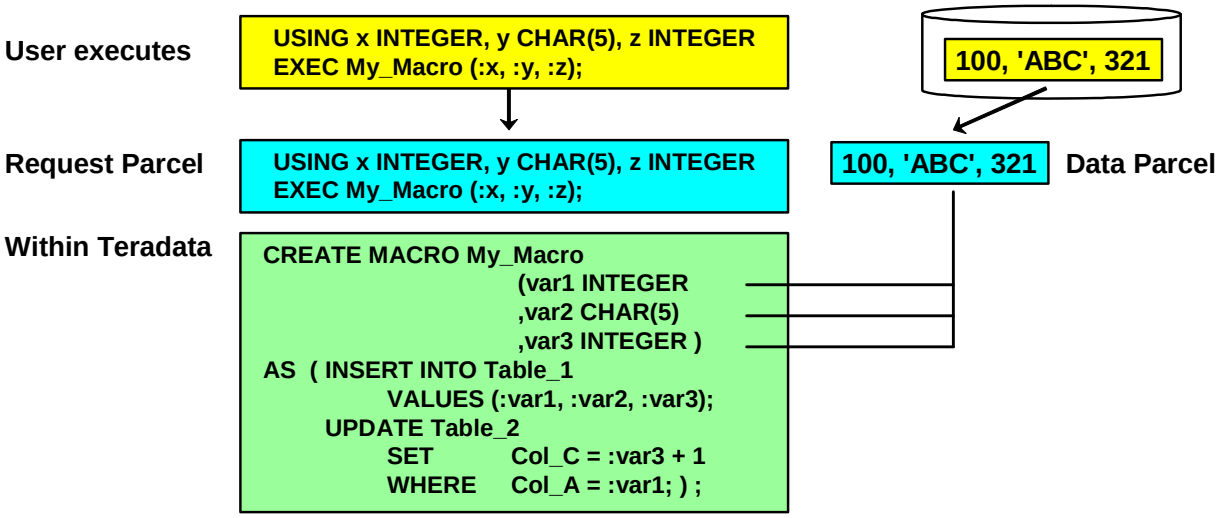
The Data Parcel

If a Request parcel is reusable (contains variable parameters), a Data parcel must be sent with it. The Data parcel contains the values to be inserted into the parameters in the SQL Request.

Hard coded values appear within the Request parcel, while substitutable values appear within a Data parcel. The illustration on the facing page shows how the values contained in the Data parcel (100, ‘ABC’, 321) are inserted into the statements contained in the Request parcel.

The Data Parcel

- Explicitly declare a USING data parcel in BTEQ.
- FastLoad and MultiLoad do not use the SQL protocol, but use this concept through the DEFINE or LAYOUT statements.
- The Preprocessor generates a USING data parcel from macro parameters or SQL statements that reference host program variables.
- CLI programs must create and manage their own DATA parcels.



SQL Parser Overview

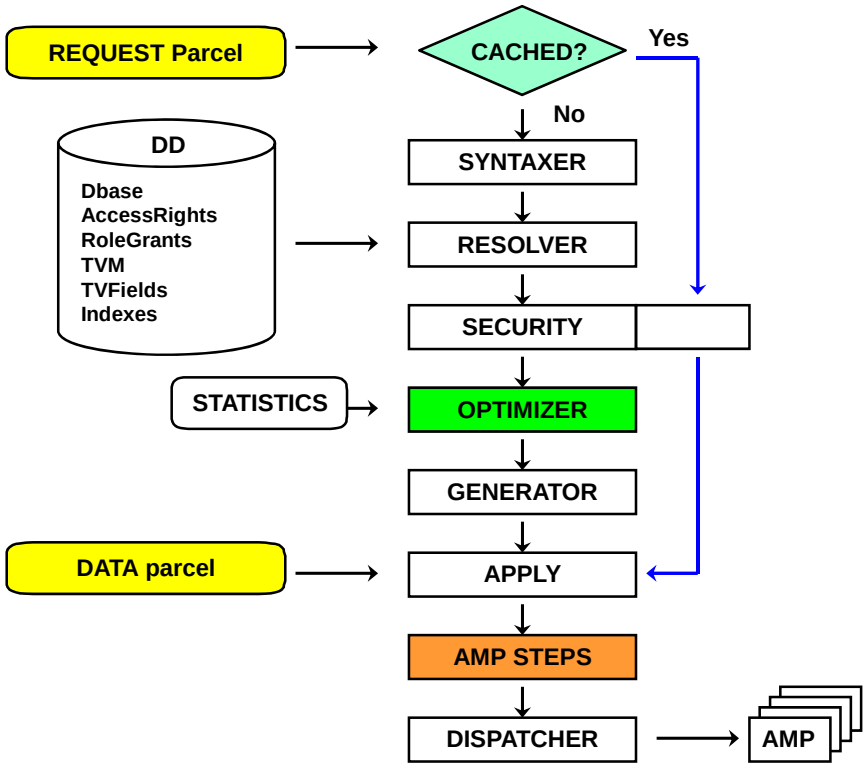
The flowchart on the facing page provides an overview of the SQL parser. As you can see, it is composed of six main sections: **Syntaxer**, **Resolver**, **Security**, **Optimizer**, **Generator** and **Apply**.

When the parser sees a Request parcel it checks to see if it has parsed and cached the execution steps for it. If the answer is NO, then the Request must pass through all the sections of the parser as follows:

- The **Syntaxer** checks the Request for valid syntax.
- The **Resolver** breaks down Views and Macros into their underlying table references through use of DD information.
- **Security** determines whether the Requesting User ID has the necessary permissions.
- The **Optimizer** chooses the execution plan.
- The **Generator** creates the steps for execution.
- **Apply** binds the data values into the steps. (This phase of the Parser is also known as GncApply.)

Note: If the steps in the Request parcel are in cache, the Request passes directly to Apply (after a check by Security). This is illustrated on the facing page by the YES path from the CACHED? decision box.

SQL Parser Overview



Software Cache

All Teradata vprocs use some SMP memory as Software Cache to retain processing steps, data, or both. Cache provides faster performance by eliminating excess disk access to retrieve the needed information.

A PE utilizes software cache to store:

- Processing steps
- The Data Dictionary

When Teradata virtual processors are provided with larger memory, they can allocate more software cache, thus reducing disk accesses.

Software Cache

- ALL Teradata vprocs use some SMP memory as software cache.
- Software cache retains data dictionary information and/or processing steps.
  - **Requests-To-Steps Cache** - holds processing steps in memory
  - **Dictionary Cache** - hold DD/D information in memory
- Cache eliminates regenerating and re-fetching needed information.
- Larger memory gives the processors more software cache and reduces disk accesses.

Request-To-Steps Cache

The **Request-To-Steps Cache** (R-T-S Cache) is where SQL text and AMP steps generated by the Parser are stored. AMP steps are stored without the data values that **Apply** will bind in later. Both Interpretive Steps and Compiled Steps can be held in R-T-S Cache.

Each PE has its own cache memory area. Any steps stored in the R-T-S Cache are available to any User accessing that same PE.

DDL Requests are not cached because they are not considered repeatable. For example, you would not be able to repeat the same CREATE TABLE Request.

Teradata purges cache every four hours. At that time, any "unmarked" entries are removed from the R-T-S Cache. Since demographics change over time, potentially "stale" AMP steps are removed from the R-T-S cache. This cache purge is staggered across all PEs so that no two PEs are purging their cache at the same time.

The system compiles evaluation steps whenever the Optimizer determines that the query will return a large number of rows, resulting in increased performance.

Request-to-Steps Cache

- **Cache stores the SQL text and the AMP steps generated by the Parser without binding in data from the Using DATA parcel.**
  - **Plans do not have to be re-parsed by the user.**
  - **Requests with hard-coded values are not immediately cached.**
- **For cached plans, the plan is retrieved and the SYNTAXER, RESOLVER, OPTIMIZER and GENERATOR steps are bypassed.**
- **Cached steps may be shared by sessions/logons within a Parsing Engine.**
- **Cache is maintained in Least Recently Used sequence so that lower activity plans swap out to disk.**
- **The system purges unmarked entries from cache every four hours.**
  - **All PEs also purge affected cache entries on receipt of a DDL "Spoiling" message.**
- **Demographically independent plans (UPI, USI and some Nested Joins) are marked when placed into cache.**
- **DDL requests are never cached.**

Request-to-Steps Cache Check

In order for an SQL Request to match an entry in R-T-S cache, all of the following must be identical:

- Application (Batch, Interactive)
- Response Mode (Field, Record, Indicator)
- Host, Workstation, or LAN type
- Default database name (if used in resolving the Request)
- National character set
- Request text length
- Request text

Notice that the Request text has to be identical in all respects. A single difference in syntax (spacing, order of columns, etc.) will cause the Request to be treated as new.

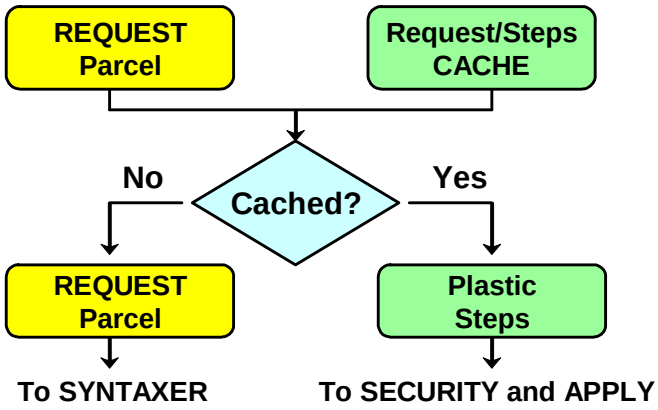
There are two important reasons to use Macros whenever applicable:

- Macros reduce parcel size, thus dramatically improving performance.
- Macros will increase the likelihood of matching the R-T-S cache because users won't have to re-enter their SQL.

The facing page identifies a number of characteristics of R-T-S cache. Additional characteristics to this list include:

- Cache is automatic and transparent.
- Parser output cannot be stored by the user.
- Cache is maintained in Least Recently Used sequence so that lower activity plans swap out to disk.
- Demographically independent plans (UPI, USI and some Nested Joins) are marked when placed into cache.
- Teradata compiles executable evaluation steps if the Optimizer determines that a large number of rows will be returned.

Request-to-Steps Cache Check



If an identical Request exists in Request-To-Steps cache:

- Call SECURITY and APPLY and pass them the memory address of the AMP steps.
- These steps do not have DATA parcel values bound into them; they are called Plastic steps.

Otherwise, the Request Parcel passes the request to the SYNTAXER.

The larger the Request Parcel, the longer these steps take.

Macros reduce parcel size, dramatically improving performance.

Request-To-Steps Cache Logic

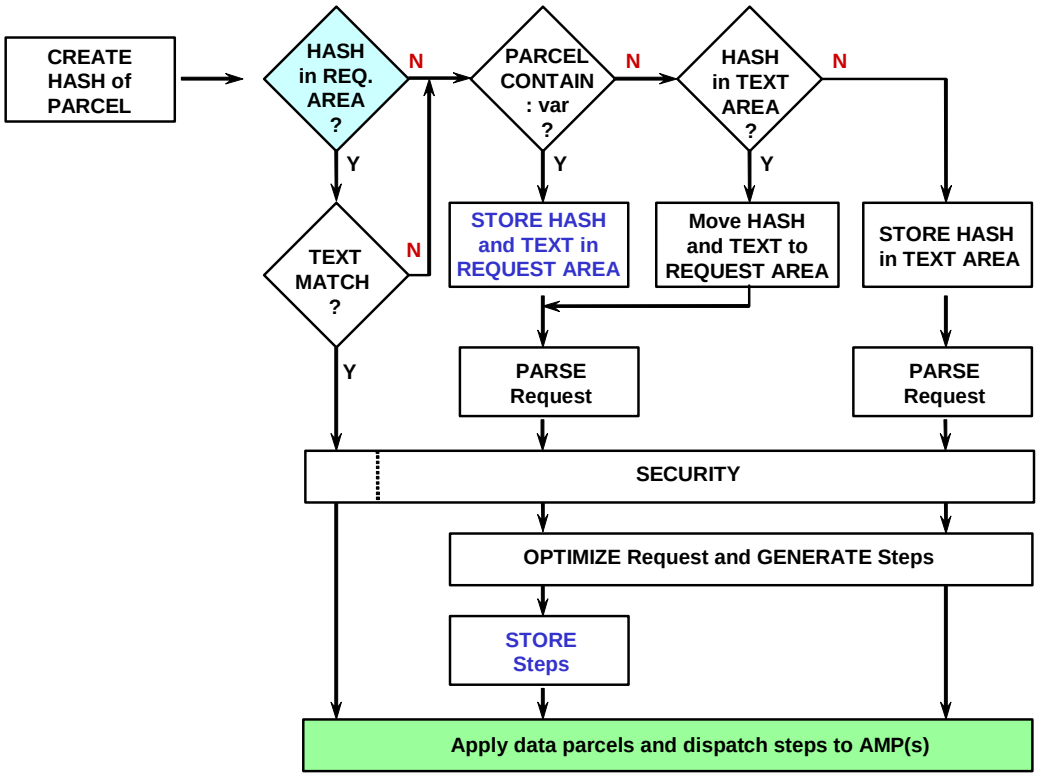
The flowchart on the facing page illustrates how Teradata determines what to do with an SQL request. This flowchart only applies to repeatable DML requests.

INSERT INTO Table_1 VALUES (100, 25.95, 'abcde', 950131) ;

The above DML statement would not be considered repeatable for a Set table since the second attempt would fail with either a duplicate row error or duplicate index error.

The entire Request Parcel is put through the hashing algorithm to produce a 32-bit hash of the parcel. If there is an entry in R-T-S cache with the same hash value, the system must do a byte-by-byte comparison between the incoming text and the stored text to determine if a true a match exists. The larger the size of the Request Parcel, the longer these steps take.

Request-To-Steps Cache Logic



Dictionary Cache

The Data Dictionary (DD) Cache is part of the cache found on every PE. It stores the most recently used DD information including SQL names, their related numeric IDs and Statistics.

Data Dictionary/Directory cache entries include:

- SQL names and their related numeric IDs.
- Demographically dependent Request-To-Steps cache entries that do not have collected statistics.

The DD Cache is purged every four hours by the same process that purges the R-T-S Cache. Only those entries which do not have COLLECTed STATISTICS are purged, since the system considers them demographically dependent. Certain SQL statements can also cause DD contents to be dropped by "spoiling" them. Spoiling occurs when the DD is changed and prevents "stale" DD information from being used.

The DD tables that provide the information necessary to parse DML requests are:

- DBase
- TVM
- AccessRights
- RoleGrants (V2R5)
- TVFields
- Indexes.

The amount of Data Dictionary Cache is determined by the DBSControl Performance parameter named Dictionary/CacheSize. This parameter typically defaults to 1024 KB.

Dictionary Cache

- The Parser needs data from the DD to convert SQL into AMP steps.
- DD cache holds the most recently used items in PE memory:
  - SQL names and their related numeric IDs.
  - Statistical information for the Optimizer.
- SQL statements that change the DD generate “spoiling” messages.
- PEs drop the “spoiled” entries from their cache.
- The system purges cache every four hours:
  - Data demographics can change significantly within four hours.
  - Purging cache forces the Parser to re-optimize all current requests.
- Processors purge their cache on a sequential basis.
- DD tables used by the resolver to parse a request:
  - Dbase
  - TVM
  - AccessRights
  - TVFields
  - Indexes
  - RoleGrants

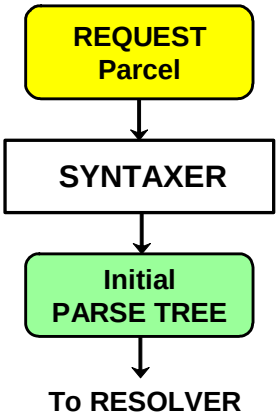
Syntaxer

The Syntaxer checks the syntax of an incoming Request parcel for errors. If the syntax is correct, the Syntaxer produces an initial Parse Tree, which is then sent to the Resolver.

The time required by the Syntaxer depends on the size of the Request parcel. Larger parcels take more time. Using Macros will reduce the size of the parcels and thus reduce the time required by the Syntaxer.

The use of Macros will reduce the number of syntax errors since Users will not have to re-key the SQL code.

Syntaxer



- This module checks the syntax of an incoming Request Parcel.
- If no errors are found, it produces an Initial Parse Tree and calls the RESOLVER.
- The larger the Request Parcel, the longer these steps take.
- Macros reduce parcel size, dramatically improving performance.

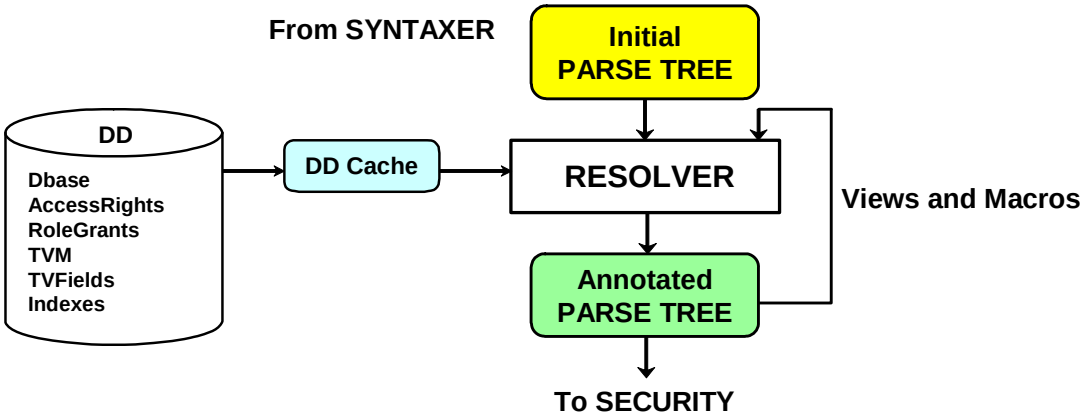
Resolver

The Resolver takes the initial Parse Tree from the Syntaxer and replaces all Views and Macros with their underlying text to produce the Annotated Parse Tree. It uses DD information to "resolve" View and Macro references down to table references. The DD tables shown in the diagram on the facing page (DBase, AccessRights, RoleGrants (V2R5), TVM, TVFields and Indexes) are the tables that the Resolver utilizes for information when resolving DML requests.

Nested Views and Macros can cause the Resolver to take substantially more time to do its job.

The nesting of views (building views of views) can have a very negative impact on performance. At one site, what a user thought was a two-table join was actually a join of two views which were doing joins of other views, which were doing joins of other views, which were doing joins of base tables. When resolved down to the table level, the two "table" join was really doing a 12-table join. The data the user needed resided in a single table.

Resolver



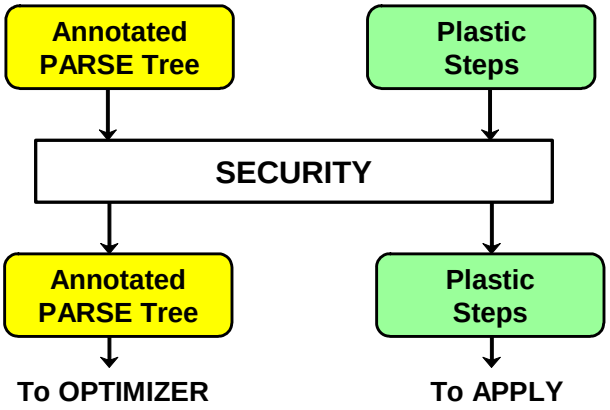
- RESOLVER replaces all View and Macro names with their underlying text.
- Nesting of Views and macros could add substantial time to this process.
- Views and/or Macros can be nested up to 64 levels.
- The RESOLVER uses information from the DD cache when possible.
- It accesses the DD tables if additional information is needed.

Security

Security verifies that the User ID responsible for the SQL Request has the necessary permissions on any objects referenced in the Request. Since permissions can be granted or revoked dynamically, it is very important to have each and every SQL Request (even those cached) pass through Security.

Security is more important the second and subsequent times a Request is processed because the Resolver also checks for permissions during the initial parse. The initial parse and statement execution checks all access rights. Execution from cache checks only user rights.

Security



SECURITY verifies that the requesting User ID has the necessary permissions on the referenced objects.

Permissions can be granted or revoked dynamically.

Each execution of a Request includes a SECURITY check.

- **Initial parse and execute checks ALL access rights.**
- **Execution from cache checks only user rights (i.e., EXEC).**

Optimizer

The Optimizer analyzes the various ways an SQL Request can be executed and determines which is the most efficient. It acts upon the Annotated Parse Tree after Security has verified the permissions and generates an Optimized Parse Tree. (See illustration on the facing page.)

Note that the output of the Optimizer can be passed to the EXPLAIN Facility. EXPLAINing a request does not execute that request.

- **Serial Steps** must be executed in sequence and successfully completed by all affected AMPs before the next step is sent out.

- **Parallel Steps** are multi-AMP processing steps that can be transmitted to the AMPs and complete asynchronously. All Parallel Steps must successfully complete before the next Serial Step is sent out. The Optimizer decides whether steps are serial or parallel.

- **Individual Steps** and **Common Steps** are processing steps generated for a Multi-Statement Request.

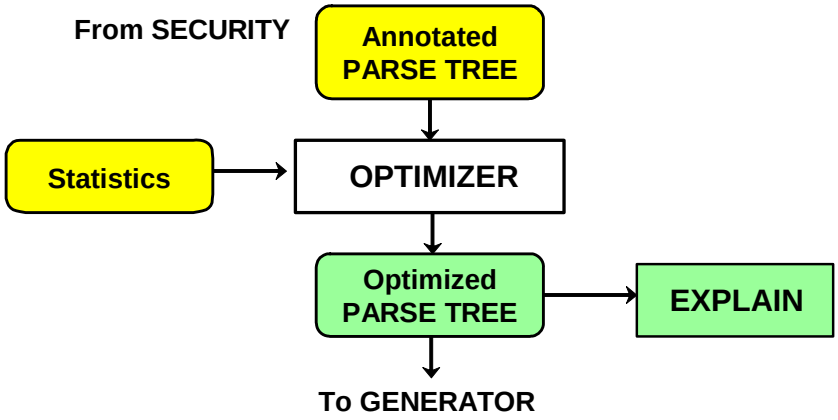
  – Individual Steps are unique to a single SQL statement in the Request.

  – Common Steps are steps that can be used by more than one statement in the Request. A typical example is the creation of a Spool file whose contents can be used more than once. You will learn more about Common Steps in the next module.

Additional processing steps need to be generated in the presence of the following optional features:

- Triggers
- Check Constraints
- References
- Foreign Keys
- Stored Procedures

Optimizer



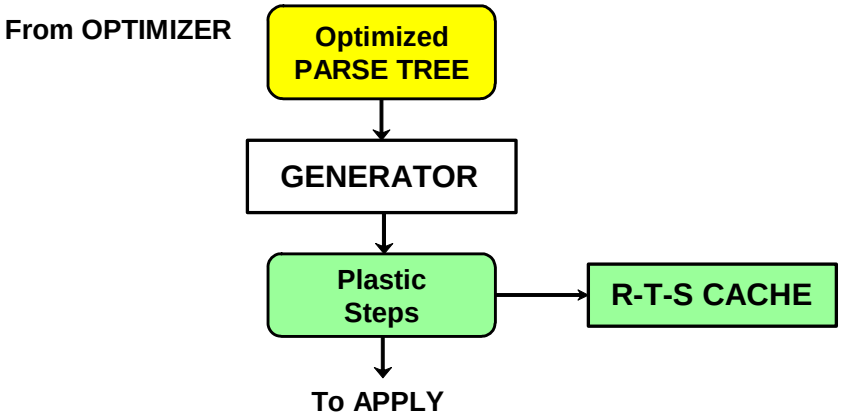
- DD/D operations replace DDL statements in the Parse tree.
- The OPTIMIZER evaluates DML statements for possible access paths:
  - Available indexes referenced in the WHERE clause.
  - Possible join plans from the WHERE clause.
  - Full Table Scan possibility.
- It uses COLLECTed STATISTICS or dynamic samples to make a choice.
- It generates Serial, Parallel, Individual and Common steps.
- OPTIMIZER output is passed to either the Generator or the Explain facility.

Generator

The Generator acts upon the Optimized Parse Tree from the Optimizer and produces the Plastic Steps. Plastic Steps do not have data values from the DATA parcel bound in, but do have hard-coded literal values embedded in them.

Plastic Steps produced by the Generator are stored in the R-T-S Cache unless a request is not cacheable.

Generator



- Plastic Steps are AMP steps without data values from the DATA parcel bound in.
- Hard-coded literal values are embedded in the Plastic Steps.
- Plastic Steps are stored in Request-to-Steps cache.

Apply

Apply acts upon Plastic Steps from the Generator and produces Concrete Steps by binding in data values from the DATA parcel. Apply adds data values one at a time.

The Concrete Steps then go to the Dispatcher, which sends them out to the AMPs.

The Apply phase is also known as GncApply in previous V2 releases. The Apply was referred to as OptApply with Teradata Version 1 software.

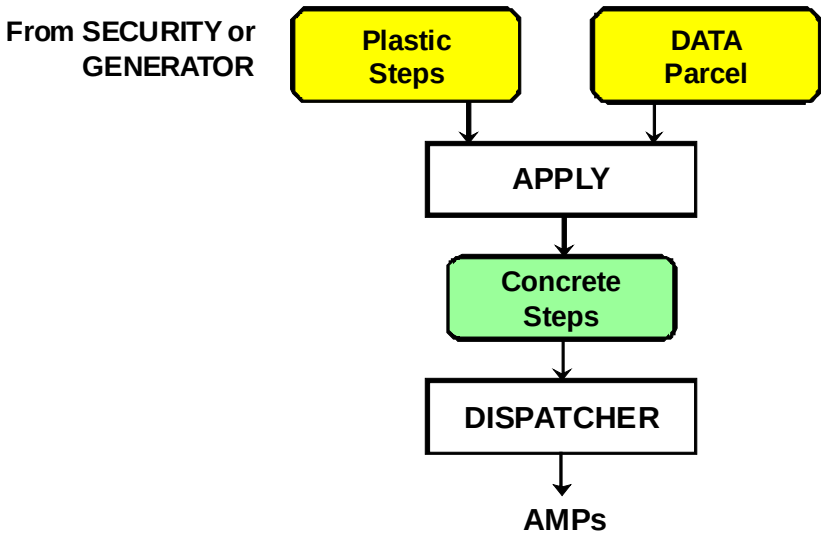
Dispatcher

Dispatcher is responsible for sending the execution steps out to the AMPs.

The AMPs respond back to the Dispatcher which is responsible for sending the host response with one exception.

The exception is because of one of the new V2R6 internal performance features called the Prime-Key Operations Performance feature (actually a UPI/NUPI performance feature). For short queries (PI equality value on a table without fallback or secondary indexes), primary index operations are speeded up by having a shorter CPU length. This is done internally by having an express request generated by the PE (instead of a concrete step request), reduced PDE overhead because of an express request, no transaction group, and the AMP is able to directly respond back to the host/user and bypass the Dispatcher. This feature only works if the table doesn't have fallback or any secondary indexes. If a table has fallback and/or secondary indexes, an internal "transaction group" has to be generated to make sure the fallback and/or secondary indexes are updated as well.

Apply and Dispatcher



APPLY binds the DATA parcel values into the Plastic Steps.

- This produces Concrete Steps.
- Also known as OptApply.

The DISPATCHER sends the Concrete Steps to the AMPs.

- The DISPATCHER is also responsible for sending the response back to the client.

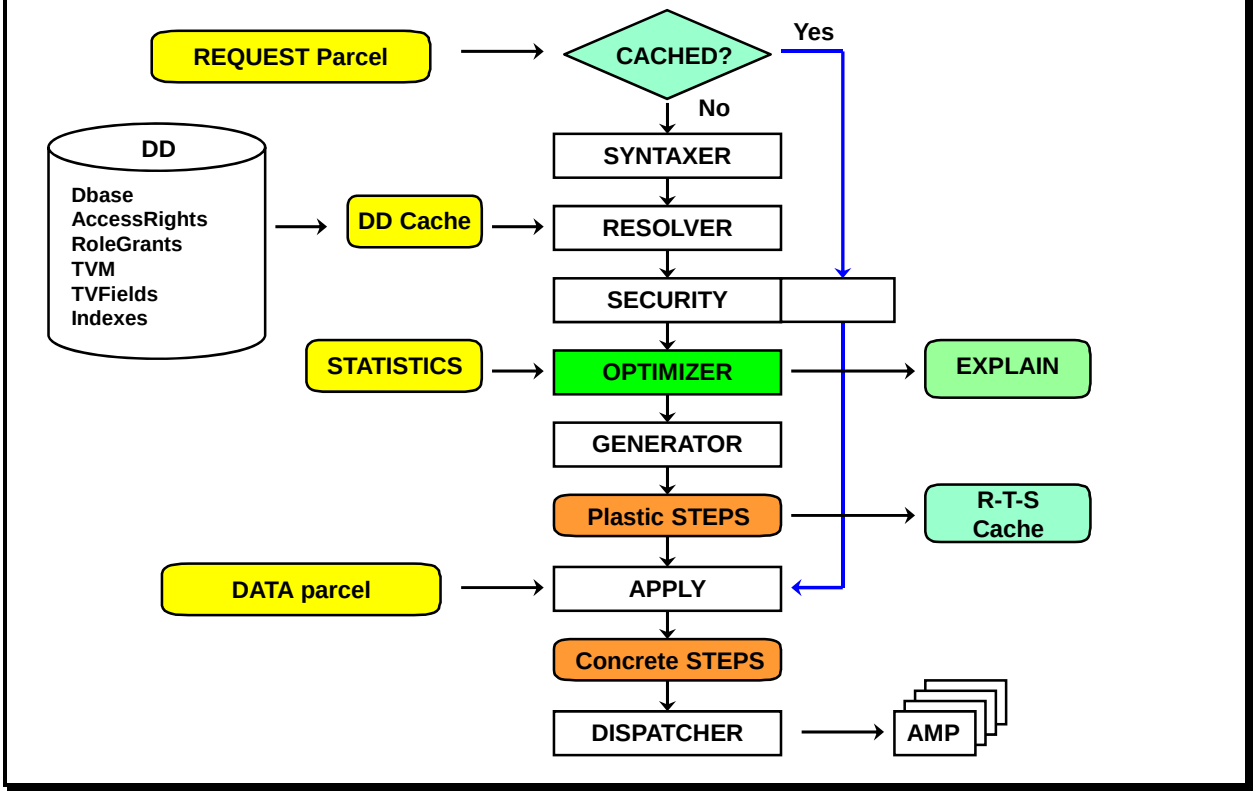
SQL Parser Review

The illustration on the facing page should look familiar since it is almost identical to the diagram you saw in the SQL Parser Overview earlier. It summarizes the information presented in this module.

– IMPORTANT –
Teradata's Late Binding Parser provides maximum flexibility.

This refers to the fact that data values are not bound into the Plastic Steps until the Apply phase of the process.

SQL Parser Review



Parser Summary

Some important points to remember regarding parsing are shown on the facing page.

Parser Summary

- Plastic Steps for Requests with DATA parcels are cached immediately.
- Views and Macros cannot be nested beyond 64 levels.
  - Nested Views and macros take longer for initial parsing.
- Multi-statement requests (including macros) generate more Parallel and Common steps.
- Execution plans remain current in cache for up to four hours.
- DDL “spoiling” messages may purge DD cache entries at any time.
- Requests against purged entries must be re-parsed and re-optimized.

Module 24: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 24: Review Questions

1. What must a REQUEST parcel contain? _____

2. Which two statements about the RESPOND parcel are true? ___
   a. Identifies response buffer size.
   b. Generates a SUCCESS/FAIL parcel.
   c. Always followed by one or more DATA parcels.
   d. May be sent by itself as a continuation request.

3. Match the six SQL Parser phases listed below with its correct description.

__ Syntaxer	a. Determines whether the Requesting User ID has the necessary permissions
__ Resolver	b. Create concrete steps
__ Security	c. Checks the Request for valid syntax.
__ Optimizer	d. Creates the steps for execution.
__ Generator	e. Breaks down Views and Macros into their underlying table references
__ Apply	f. Chooses the execution plan.

4. Which Parser phase benefits the most from the use of macros? ___
   a. Generator
   b. Resolver
   c. Syntaxer
   d. Apply

Module 24: Review Questions (cont.)

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 24: Review Questions (cont.)

5. What is the function of the Request-to-Steps (R-T-S) Cache? ____

   a. Stores the SQL text and AMP steps generated by the Parser.
   b. Resolves View and Macro references down to table references.
   c. Stores the most recently used DD information including SQL names, their related numeric IDs and Statistics.
   d. Analyzes the various ways an SQL Request can be executed and determines which of these is the most efficient.

6. Teradata's Late Binding Parser refers to Apply, which acts upon the _____ from the Generator and produces _____ by binding in the data values from the DATA parcel.

   a. Plastic Steps / Concrete Steps
   b. Interpretive Steps / Compiled Steps
   c. Processing Steps / Execution Steps
   d. AMP steps / Request-to-Steps Cache

Notes

Module 25

TERADATA
Raising Intelligence

Optimizer and Collecting Statistics

After completing this module, you will be able to:

- **Explain how the Optimizer acquires statistics.**
- **Describe random AMP sampling.**
- **State a method for viewing statistics.**
- **Describe how the Teradata Statistics Wizard can be used to collect or re-collect statistics.**

Notes

Table of Contents

Teradata Optimizer....25-4
Optimizer – Cost Based vs. Rule Based....25-6
Optimizer Statistics....25-8
Optimizer's Search for Statistics....25-10
Optimizer – Random AMP Samples....25-12
Random AMP Sampling – How it Works....25-14
Example of an Optimizer Estimate without Collected Statistics....25-16
Statistics....25-18
Statistics Data – What is Collected?....25-20
Statistics Data – What is Collected? (cont.)....25-22
Statistics Data – What is Collected? (cont.)....25-24
Statistics Example....25-26
Statistics Example (cont.)....25-28
COLLECT STATISTICS Command....25-30
Collecting Statistics....25-32
Refresh or Re-Collect Statistics....25-34
COLLECT STATISTICS Command....25-36
COLLECT STATISTICS on a Data Sample....25-38
Collecting Statistics (14.0 Examples)....25-40
Viewing Statistics....25-42
Optimizer's use of Statistics with Uneven NUSI....25-44
Collecting Statistics on PARTITION....25-46
Copying STATISTICS....25-48
Statistics Extrapolation....25-50
Teradata 13.0 Enhancements....25-52
Teradata 14.0 Enhancements....25-54
Teradata Statistics Wizard....25-56
Teradata Statistics Wizard – Main Window....25-58
Teradata Statistics Wizard – Interval Statistics....25-60
Collect, Re-Collect, or Drop Statistics....25-62
Recommendations....25-64
Recommendations (cont.)....25-66
Statistics Summary....25-68
Module 25: Review Questions....25-70

Teradata Optimizer

Teradata's optimizer will actually generate several plans and choose the best plan based on analyzing the low end cost numbers. This type of optimizer is critical to performance in supporting mixed workloads. A cost based optimizer requires statistical information about the data as well as being aware of the machine resources (CPU, disk, memory, processors, etc).

The other type of optimizer is rules based. This is good for transactional workloads where the queries are well known and the data has been physically structured to support this known workload. It is based on a set of rules that have been defined and only performs well in a highly structured transactional environment.

Note for facing page: AMP local – rows from 2 tables are on the same AMP – typically joining two tables on the same PI.

Teradata Optimizer

Teradata uses a "cost-based optimizer".

- **The Optimizer evaluates the "costs" of all reasonable execution plans and the best choice is used.**
  - **Effectively finds the optimal plan**
  - **Optimizes resource utilization - maximizes system throughput**
- **What does the "optimizer" optimize?**
  - **Access Path (Use index, table scan, dynamic bitmap, etc.)**
  - **Join Method (How tables are joined – merge join, product join, hash join, nested join)**
  - **Join Geography (How rows are relocated – redistribute, duplicate, AMP local, etc.)**
  - **Join Order (Sequence of table joins)**
- **Parallelism is automatic**
  - **Aware of table geography and data skew**
- **Parallelism is unconditional**
  - **No operation turns off parallelism**

Optimizer – Cost Based vs. Rule Based

The quality of the optimizer is critical in Data Warehouse environments. Data Warehouse environments require a cost based optimizer to make good decisions regarding join plans for complex DSS queries. Join techniques that are required for DSS are significantly more sophisticated than what is required in an OLTP environment.

The join technique chosen for a query is a dominant factor in performance of the Data Warehouse. Since DSS environments are designed to support ad hoc queries, we must rely on a Cost Based Optimizer to determine the optimal plan.

- Note: Complex queries would be those containing sub-queries. Compound query uses set operators to combine two or more simple or complex queries.

Query optimizers do *not* do either of the following things:

- Guarantee that the access and join plans generated are infallibly the best plans possible. A query optimizer always generates several optimal plans based on the population and environmental demographics it has to work with and the quality of code for the query it receives, then selects the best of the generated plan set to use to respond to the DML statement.

- You should not assume that any query optimizer ever produces absolutely *the* best query plan possible to support a given DML statement.

Optimizers – Cost Based vs. Rule Based

Cost Based Optimizer

- Best for complex DSS queries
- Requires statistics on tables and individual columns within tables
- Plans are independent of table ordering in FROM clause and predicate ordering in the WHERE clause
- Should understand available resources within the architecture (e.g., CPU, Memory, Disk, Processors, etc)
- Generates several plans and chooses best plan based on smallest cost factors

Rules Based Optimizer

- Used in OLTP environments with structured and known access paths
- Rules are defined for access paths and types of SQL statements (e.g., simple, join, complex, compound, etc.)
- Plans are chosen based on access paths available, the ranks of these access paths and type of query
- Not a good choice in DSS environments
- Users can provide hints to help influence a rules based optimizer.

Optimizer Statistics

As we have seen, the Optimizer plans an execution strategy for every SQL query submitted to it. We have also seen that the execution strategy for any query may be subject to change depending on various factors. For the Optimizer to consistently choose the optimum strategy, it must be provided with reliable, complete, and current demographic information regarding all of these factors.

The best way to assure that the Optimizer has all the information it needs to generate optimum execution strategies is to **COLLECT STATISTICS**.

It is interesting to note that the Parser needs the same information to properly plan a query, as you need in properly choosing indexes.

Optimizer and Block-Level Compression

The optimizer is not sensitive to block-level compression. Neither will you see any additional steps added to the explain text when a query accesses a compressed table, as the decompression processes are performed transparently at the file system level. The optimizer does not take into account the extra time or CPU required for decompression when estimated processing times are established. However, the average row size that is calculated during random AMP sampling will be different for compressed tables. When random AMP sampling is performed, one or more cylinder indexes are read. While the cylinder indexes are never compressed, the information they carry is based on physical characteristics of the underlying data.

Even if full statistics have been collected, random AMP sampling is relied upon for determining the average row size as input to query optimization. Because the table's row size is based on the compressed image of the data, estimated processing times, which are influenced by row size, may be slightly less in queries accessing compressed tables. While it is not expected that this discrepancy will be large enough to cause the optimizer to make different decisions in most cases, the row size under-estimation with compression might lead to some query plan changes when block level compression is implemented.

Optimizer Statistics

The optimizer needs information to create the best execution plan for a query.

Environment information:

- Number of nodes
- Number of AMPs
- Number and type of CPUs
- Disk Array information
- Interconnect (BYNET) information
- Amount of memory available

Data Demographics:

- Number of rows in the table
- Row size
- Column demographics
  - Range of values in the table for the column
  - Number of rows per value
  - Number of NULLs for the column
- Index demographics

Are Statistics Collected?

If Yes – use collected statistics

If No,

- For indexed columns – use Random AMP Samples
- For non-indexed columns – use Heuristics (formulas)

Optimizer's Search for Statistics

When there are no statistics available to quantify the demographics of a table or an index, the Optimizer selects (by default) a single AMP to sample for statistics using an algorithm based on the table ID. By inference, these numbers are then assumed to represent the global statistics for the column or index.

Note that the statistics collected by a random AMP sample only apply to indexed columns. If you do not collect statistics on non-indexed columns, then the Optimizer uses various situation-specific heuristics to provide arbitrary estimates of cardinalities.

The process of the optimizer searches for statistics in Teradata 12.0 is as follows:

First, random AMP samples are kept in the table header that resides in memory in dictionary cache. The optimizer first looks for the table header in the dictionary cache in order to extract the random AMP samples. If the table header is not in the cache, it is read from disk, where it can be found as a single row in subtable 0 of the base table. As part of the process of reading the table header from disk, random AMP samples are collected for that table and moved into a field in the table header when the table header is placed in the cache.

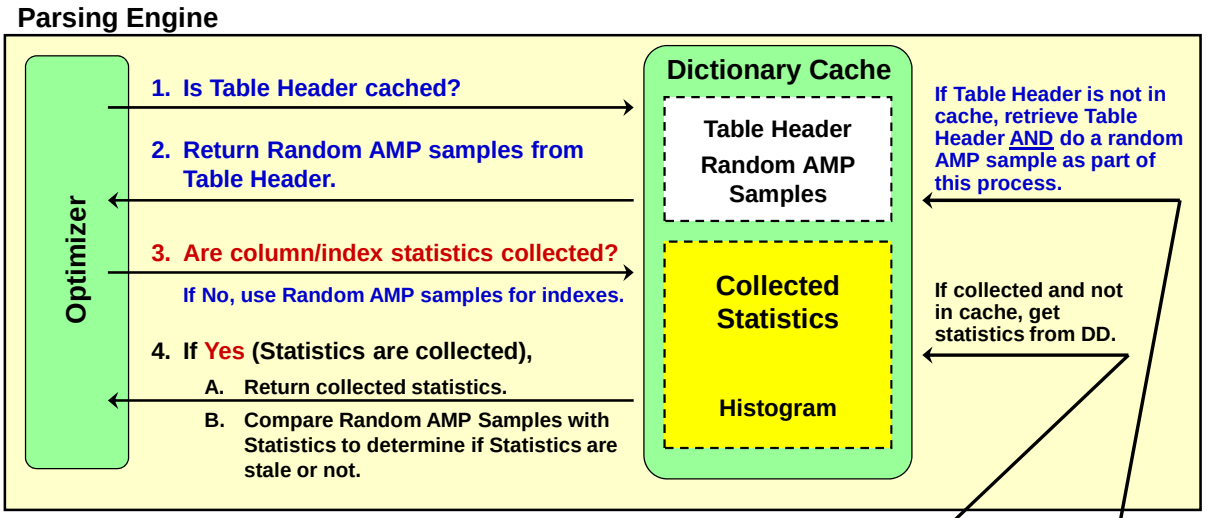
After the table header has been located and the random AMP samples have been accessed, a routine that looks for collected statistics is called. This routine will be called once for each index and/or column on a table for which the optimizer would like statistics. As part of that routine, the dictionary cache is searched for the relevant histogram.

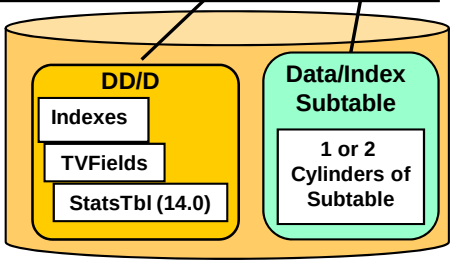
If statistics have been collected for the column/index of interest but the statistics are not cached, an express request is issued that retrieves the collected statistics from the data dictionary tables on disk. If statistics for that column or index are found in the data dictionary, the histogram that contains the statistical information is placed in the dictionary cache for use by other queries and is used by the optimizer for the current query.

If no collected statistics exist and the column is a primary or non-unique secondary index (NUSI), then the random AMP samples that are stored in the table header will be used. Non-indexed columns that have no statistics collected will not use random AMP samples, but will rely on static formulas to determine selectivity estimates.

Statistics are removed from the cache periodically to make sure that what is cached is reasonably current.

Optimizer's Search for Statistics





1 & 2. Get the random AMP statistics from table header in DD cache.

3. If statistics are collected for the index or column, get the statistics from cache or DD.

4. Random AMP sample row counts and collected statistics row counts are compared to determine if collected statistics are stale or if extrapolation is needed.

Optimizer – Random AMP Samples

The Optimizer does Random AMP Sampling with information it gathers from a random AMP. This AMP is selected based on the Table ID. Different tables will get their samples from different AMPs. This assures that no single AMP will be overloaded with Random AMP Sample requests.

Since this method uses only a single AMP for sampling purposes, it is sensitive to the evenness of the data distribution. Badly distributed data gives skewed samples that impact optimization by misleading the Optimizer into making improper choices.

The Optimizer will be more conservative in the choices it makes if it has to rely upon Random AMP Sampling.

DBSControl Options (starting with Teradata 6.0)

When statistics have not been collected for an index, the Optimizer can obtain random samples from more than one AMP when generating row counts for a query plan.

This enhancement improves the row count, row size, and rows per value estimates for a given table. These are passed to the Optimizer, resulting in improved join plans, better query execution times, and better elapsed times.

Tables with heavily skewed data will benefit most from the improvements to random AMP sampling. One AMP sampling of a table with heavily skewed data may result in wrong estimates of the row count and row size information being passed to the Optimizer. With multiple AMP sampling, the Optimizer will receiver better estimates with which to generate a query plan.

An internal parameter (#65) within DBSControl can be set by the Teradata Customer Engineer to specify the type of Random AMP Sampling that will be used in V2R6.

65. RandomAmpSampling – this field determines the number of AMPs to be sampled for getting the row estimates of a table. The valid values are D, L, M, N or A.

    D - The default is one AMP sampling (D is the default unless changed.)
    L - Maximum of two AMPs sampling
    M - Maximum of five AMPs sampling
    N - Node Level sampling - i.e., all the AMPs in a node would be sampled.
    A - System Level sampling - i.e., all the AMPs in a system would be sampled.

Note that a higher number of AMPs sampled will provide better estimates, but can cause short running queries to run slower and long running queries to run faster.

Optimizer – Random AMP Samples

- A Random AMP sample only applies to indexed columns and table row counts.
  - row counts for the table are needed and statistics are not collected on PI.
  - indexed columns are used in the query and statistics do not exist for the indexes.
  - Statistics have been collected, but are considered stale (e.g., 10% change).
- By default, a single AMP is chosen for random AMP data sampling.
  - The AMP chosen is based on Table ID. Different tables will utilize different AMPs.
    - Badly distributed data gives skewed samples that impact optimization.
  - If a table (or index subtable) spans more than 1 cylinder, it will sample the first and the last cylinder. If it fits into 1 cylinder, it will only sample that one cylinder.
- Option – the Optimizer can obtain random samples from more than one AMP when generating row counts for a query plan.
  - Improves the row count, row size, and rows per value estimates for a given table.
  - Random AMP sampling is controlled via a DBS Control parameter.

    D – Dynamic or one AMP sampling (D is the default unless changed)
    L – Low - two AMPs are sampled
    M – Maximum - five AMPs are sampled
    N – Node Level sampling - i.e., all the AMPs in a node are sampled
    A – All or system level sampling - i.e., all the AMPs in a system are sampled

Random AMP Sampling – How it Works

When the Optimizer is not provided with COLLECTed STATISTICS for a table or indexed column, it will perform a Random AMP Sample in order to estimate table demographics.

The Optimizer uses the results of these calculations to provide it with the demographics it needs to make its choices in optimizing the query. If the results are skewed, the Optimizer may be misled and make the wrong choices; performance may not be optimal.

Random AMP samples are stored in the PE Data Dictionary (DD) cache for a maximum of four hours until they are purged by the routine DD four-hour purge job. Each time a table header is read from disk, the system collects a fresh random AMP sample of its statistics and caches the random AMP sample along with the table header.

The Parser is more aggressive with COLLECTed STATISTICS. Features such as NUSI Bit Mapping require COLLECTed STATISTICS.

Random AMP Sampling of USIs

Random AMP sampling assumes that the number of distinct values in a USI equals its cardinality, so it does not read the index subtable for USI equality conditions. The number of distinct values in the USI is assumed to identical to the table cardinality taken from the random AMP sample on the primary index. Because equality conditions on a unique index return only one row by definition, the Optimizer always chooses the direct USI path without costing it or using statistics. However, if a USI will frequently be specified in non-equality predicates, such as range constraints, then you should collect statistics on it.

Random AMP Sampling of NUSIs

Random AMP sampling for NUSIs is very efficient. The system reads the cylinder index that supports the index subtable rows on the sampled AMP and determines the number of rows on that cylinder. Except for situations where the NUSI is very non-unique, there is one subtable row for each distinct value in the NUSI. Using that information, the sampling process assumes that each subtable row it finds translates to one index value.

Sampling Efficiency for Non-indexed Predicate Columns

The heuristic is to estimate the cardinality to be 10% of the table rows. With two selection criteria, the Optimizer assumes that about 7.5% (10% x .75) of the rows will be returned.

For additional equality criteria, each is 75% of the previous level. For example, for 3 columns, the Optimizer assumes 5.2% (7.5% x .75) of the rows will be returned.

For example, if there is one selection criteria column in the WHERE clause for a given table and the column is in an equality condition, and no stats have been collected on it, the optimizer will assume that it is selecting 10% of the table’s rows. If there are two columns each in an equality condition, no stats, the optimizer will assume that 7.5% of the table's rows will be selected by their combination. This can lead to poor plans.

Random AMP Sampling – How it Works

- For a table row count estimate, read 1 or 2 cylinders from 1 (or more) AMPs.
  - Calculate the approximate number of rows in the table:

    Average Number of Rows per Block in the sampled Cylinder(s)
    x Number of Data Blocks in the sampled Cylinder(s)
    x Number of Cylinders with data for this table on this AMP(s)
    x Number of AMPs in this configuration

- For NUSI estimates, read 1 or 2 cylinders from the NUSI subtable.
  - Uses a similar technique by counting the number of NUSI values in the cylinder(s). The table row count is divided by the extrapolated NUSI row count to get a rows/NUSI value.
  - Assumes the number of distinct values on one AMP = total distinct values.

- Any skewed component in the sample skews the demographics.
  - Skewed demographics may mislead the optimizer into choosing a poor plan.

- For non-indexed columns without statistics, the optimizer uses heuristics or fixed formulas to estimate the number of rows. For example,
  - Assumes 10% for one column in an equality condition
  - Assumes 7.5% for two columns, each in an equality condition, and ANDed together

Example of an Optimizer Estimate without Collected Statistics

The facing page shows an example of Explain output for a SELECT from a table using an equality condition on a non-indexed column.

The answer to the question is simply "The Optimizer will assume 10% for one non-indexed equality selection criteria".

If the query added additional non-indexed columns with equality conditions, the Optimizer estimates are as follows. This assumes that no statistics are collected on any of the columns.

With two equality conditions, the approximate estimate is 7.5% (10% x .75).

```
EXPLAIN     SELECT * FROM Sales
            WHERE Store_id = 123
            AND Total_Sold = 1000;
```

The size of Spool 1 is estimated with no confidence to be 66,544 rows.

With three equality conditions, the approximate estimate is 5.2% (10% x .75 x .75).

```
EXPLAIN     SELECT * FROM Sales
            WHERE Store_id = 123
            AND Total_Sold = 1000
            AND Note= 'Ordered';
```

The size of Spool 1 is estimated with no confidence to be 49,032 rows.

With four equality conditions, the approximate estimate is 4.2% (10% x .75 x .75 x .75).

```
EXPLAIN     SELECT * FROM Sales
            WHERE Store_id = 123
            AND Total_Sold = 1000
            AND Note= 'Ordered'
            AND Total_Revenue = 5000;
```

The size of Spool 1 is estimated with no confidence to be 35,831 rows.

If one of the columns has collected statistics and other columns do not have collected statistics, then Teradata will assume the number of values for the column with collected statistics and 75% of that value for each of the other columns without collected statistics.

Example: Assume that statisitcs have been collected on store_id and store_id = 123 exists in 50,000 rows.

With four equality conditions, the approximate estimate is (50,000 x .75 x .75 x .75).

Example of an Optimizer Estimate without Collected Statistics

Table information:

- **Sales table has 931,100 rows and the PI has collected statistics.**
- **Store_id is not indexed and does not have collected statistics.**

Query

```
EXPLAIN  SELECT * FROM Sales WHERE Store_id = 123;
```

Explanation

14.0 EXPLAIN

:

3) We do an all-AMPs RETRIEVE step from TFACT.Sales by way of an all-rows scan with a condition of ("TFACT.Sales.store_id = 123") into Spool 1 (group_amps), which is built locally on the AMPs. The input table will not be cached in memory, but it is eligible for synchronized scanning. The result spool file will not be cached in memory. The size of Spool 1 is estimated with no confidence to be 93,110 rows (11,918,080 bytes). The estimated time for this step is 1.22 seconds.

:

Question?

Why did the optimizer estimate 93,110 rows would be retrieved for a Store_id of 123?

Statistics

The primary purpose for COLLECTing STATISTICS is to tell the Optimizer how many rows/values there are. The Optimizer uses this information to plan the best way to access the data. A few of the ways that STATISTICS can be beneficial:

- Helpful in accessing a column or index with uneven value distribution.
- The Optimizer uses statistics to decide whether it should generate a query plan that use a secondary, hash, or join index instead of performing a full-table scan

.

- Improve the performance of complex queries and joins.
- The Optimizer uses statistics to estimate the cardinalities of intermediate spool files based on the qualifying conditions specified by a query. The estimated cardinality of intermediate results is critical for the determination of both optimal join orders for tables and the kind of join method that should be used to make those joins.
- For example, assume 2 tables or spool files are redistributed and then merge joined, or assume one of the tables or spool files is duplicated and then product joined with the other. Depending on how accurate the statistics are, the generated join plan can vary so greatly that the same query can take only seconds to complete using one join plan, but take hours to complete using another.
- Enable the Optimizer to utilize NUSI Bit Mapping.

Additional Considerations

- DROP TABLE privilege is required to collect or drop statistics.
- COLLECT STATISTICS is a DDL statement and should be scheduled during off-hours. It should not be done during production hours. The operation holds a row-hash Write Lock on DBC.TVFields or DBC.Indexes which means that no new SQL requests involving the affected table can be parsed.
- Remain valid if the system is reconfigured.

Storage of Statistics

- Teradata stores collected statistics only once – collecting statistics on a group of columns that already have statistics collected at the index level (same set of columns) only stores the statistics one time in the DBC.Indexes table.
- Issuing multiple COLLECT STATISTICS statements against the same group of columns, stores only one set of statistics. Column order within the group is irrelevant. For example, collecting on COLUMN (First_Name, Last_Name) and then later on COLUMN (Last_Name, First_Name), results in only one set of statistics.

Statistics

- Statistics basically tell the Optimizer how many rows/value there are.
- The Optimizer uses statistics to plan the best way to access data.
  - Usually improves performance of complex queries and joins.
  - The parser is more aggressive with collected statistics.
  - Stale statistics may mislead the Optimizer into poor decisions.
- Helpful in accessing a column or index with uneven value distribution.
  - NUSI Bit Mapping is much more likely to be considered if there are collected statistics.
- Statistics remain valid across a reconfiguration of the system.
- COLLECT STATISTICS and DROP STATISTICS commands are DDL statements and typically are not executed during production hours.
- COLLECT/DROP STATISTICS places an access lock on the data table and a row-hash write lock on one of the following tables.
  - DBC.TVFields – holds statistics collected for single column or single column index
  - DBC.Indexes – holds statistics collected for multi-column or multi-column index
  - DBC.StatsTbl (14.0) – repository for statistics management data

Statistics Data – What is Collected?

Prior to Teradata 12.0, the demographic information collected by each AMP is sent to one AMP for merging into a frequency distribution of 100 intervals and a summary section.

Starting with Teradata 12.0, the maximum number of intervals for statistics on an index or column was increased from 100 to 200. With a maximum of 200 intervals, each interval can characterize 0.5 percent of the data, as opposed to the former maximum of 100 intervals, which characterized one percent of the data per interval.

Also starting with TD 12.0, there was new value included in statistics for average rows per AMP which is the only value in the statistics that is configuration dependent. This value is not used if the configuration changes so the optimizer may lose a little bit of information.

Starting with Teradata 14.0, collecting statistics has been enhanced to capture more data demographic information so that the Optimizer can generate more accurate plans than it previously could. The maximum number of intervals for statistics on an index or column is increased to 500. The system default is 250 which is also the default maximum. However, you can specify up to 500 intervals with a specific collect statistics statement. With a new default of 250 intervals, each interval can characterize 0.4 percent of the data.

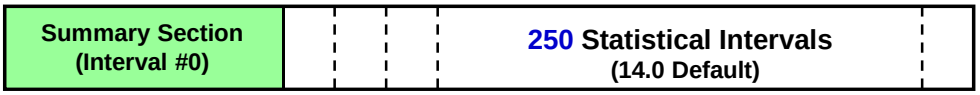
The DBSControl utility has an internal parameter (#127 - MaxStatsInterval) which is set to 250 by default for Teradata 14.0 systems.

The increase in the number of statistics intervals:

- Improves single table cardinality estimates that are crucial for join planning. Having more intervals gives a more granular view of the demographics.
- Increases the accuracy of skew adjustment because of the higher number of modal frequencies that can be stored in a histogram.
- Does not change the procedure for collecting or dropping statistics, although it affects the statistics collected.

The Summary Section has data as shown on the facing page.

Statistics Data – What is Collected?



Statistics for a column or index reside in a frequency distribution of intervals (internal histogram) plus a summary interval.

- **Each interval represents about .4% of the table's rows or high bias values.**
- **Teradata 14.0 – the default number of intervals is 250; 200 was previous default.**
- **Teradata 14.0 – the maxium number of intervals that can specified is 500.**

Summary Section (Interval #0) – Table Level Information

- **Represents domain across entire table**
- **Most frequent value for the column or index – modal value**
- **# of rows with the most frequent value**
- **# of values not equal to the most frequent value – non-modal values**
- **# of rows not equal to the most frequent value**
- **# of NULLs**
- **Minimum value**
- **Average number of rows per AMP – this is the only field that is configuration dependent.**

Statistics Data – What is Collected? (cont.)

The demographic information collected by each AMP is sent to one AMP for merging into a frequency distribution of 250 (assuming Teradata 14.0 default) plus the summary interval for a total of 251 intervals. This module will assume 251 intervals. There may be more intervals if there are high-bias values.

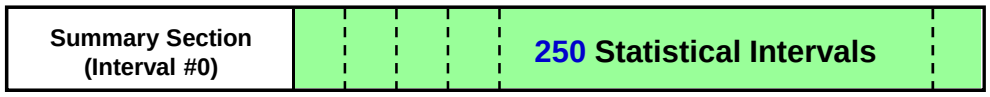
The Teradata Database uses interval histograms to represent the cardinalities and certain other statistical values and demographics of columns and indexes for all-AMPs sampled statistics and for full-table statistics. The greater the number of intervals in a histogram, the more accurately it can describe the distribution of data by characterizing a smaller percentage of its composition. Each interval histogram in the Teradata Database is composed of a maximum of 500 intervals, which in a 500 interval histogram permits each interval to characterize roughly 0.2 percent of the data rows. The default is 250 intervals which means that each interval represents about 0.4 percent of the data rows.

The number of intervals used to store statistics is a function of the number of distinct values in the column set represented. For example, if there are only 10 unique values in a column or index set, the system does not store the statistics for that column across 251 intervals, but across 11 intervals. The summary (interval 0) plus 10 intervals yields a total of 11 intervals.

The number of rows represented by an interval will vary. For example, suppose there are about one million rows in a table. Then there would be approximately 5,000 rows per interval, assuming a 200 interval histogram. Suppose that for a given interval, the system processes the values for 4,900 rows and the next value is present in 300 rows. Those rows would be counted in this interval, and the cardinality for the interval would 5,200. Alternatively, the rows could be counted in the *next* interval, so this interval would still only represent 4,900 rows.

A COLLECT STATISTICS request divides the rows in ranges of values such that each range has approximately the same number of rows, but it never splits rows with the same value across intervals. To achieve constant interval cardinalities, the interval widths, or value ranges, must vary.

Statistics Data – What is Collected? (cont.)



If statistics are collected, the histogram will have 250 intervals by default (Teradata 14.0).

- **With 250 intervals, each interval represents about .4% of the table's rows (minus rows associated with the High Bias values).**

Intervals – Range Level Information – represents ranges within the domain

- **Each range has approximately the same number of rows**
- **Maximum or highest value**
- **Most frequent value – value that occurs mostly frequently in the range – modal value**
- **Number of rows with the most frequent value**
- **Number of other values not equal to the most frequent – non-modal values**
- **Number of rows not equal to the most frequent**

Statistics Data – What is Collected? (cont.)

If any values in a column accounted for more than .20% (1/500) of the rows, these values are saved (with statistics) specifically in intervals known as a **High Bias Interval**. This effectively helps to "even out" the variance of the 200 intervals. The group of High Bias Intervals is referred to as the High Bias Section.

What is a "High Bias" Value?

A high bias (previously called a loner) value is an attribute value whose frequency in the sampled population deviates significantly from a defined criterion; an unusually frequent value indicating significant frequency skew.

Depending on the distribution of values (degree of skew) in a column, any one of three possible histogram types is used to represent its cardinality and statistics.

1. Equal-height interval histogram

   – In an equal-height interval histogram, each interval has the same number of values. To achieve this, the interval widths must vary.

   – When a histogram contains 250 equal-height intervals, each interval effectively represents a single percentile score for the population of attribute values it represents.

   – The statistics for a column are expressed as an equal-height interval histogram if the frequencies of its values are normally distributed (not skewed).

2. High-biased interval histogram

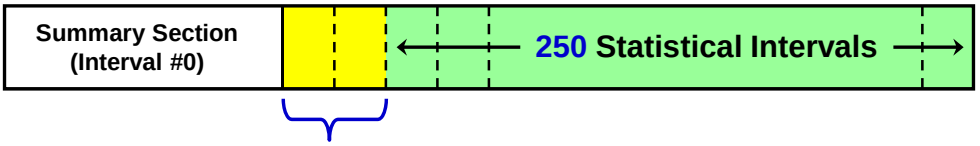
   – In a high-biased interval histogram, each interval has a high-bias value. High-biased intervals are used only when there is significant skew in the frequency distribution of values for the column.

3. Compressed histogram

   – Compressed histograms contain a mix of 250 equal-height and high-biased intervals. Reading from left-to-right, high-biased intervals always precede equal-height intervals in the histogram.

   – The facing page has graphically illustrates the concept of a compressed histogram. An example with values will be provided later.

Statistics Data – What is Collected? (cont.)



High Bias Intervals (each interval can contain 1 or 2 values & frequencies)

- In order to reduce the statistical impact of values that occur frequently, values that occur frequently are treated specially.
  - Assuming 250 intervals, if a specific value in a column occurs in more than .20% (1/500) rows, that value is considered a "High Bias Value".
  - In Teradata 14.0, high bias intervals are separate from the statistics intervals and do not subtract from the count of statistical intervals.
- A "High Bias" value and its frequency is kept in a High Bias interval.
  - Internally, a high bias interval may contain 1 or 2 high bias values and their frequencies.
- This effectively helps to "even out" the variance of the remaining intervals.
- SHOW STATISTICS (14.0) and tools such as Teradata Statistics Wizard can be used to display the statistical detailed information in the intervals.

Statistics Example

If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 = 1200;
The optimizer will assume 400 duplicates exist for this value. 1200 is the high value within the interval.

If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 = 1075;
The optimizer will assume 180 duplicates exist for this value. 1075 is the high value within the interval.

If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 = 1492;
The optimizer will assume 7 duplicates exist for this value. There are 100 other values and 700 other rows. 700 ÷ 100 = 7.

If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 = 1300;
The optimizer will assume 10 duplicates exist for this value. There are 60 other values and 600 other rows. 600 ÷ 60 = 10.

Whenever a range is used for a portion of an interval, the optimizer makes the following calculation:

one-half of the other rows of the other values in the interval
\+ highest value within the range
\+ any values from the high bias section

If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 BETWEEN 1150 AND 1250;
The optimizer will assume 700 duplicates exist for this value.

	300	one-half of the other rows of the other values in Interval #2 (600/2)
+	400	number of duplicates for value 1200

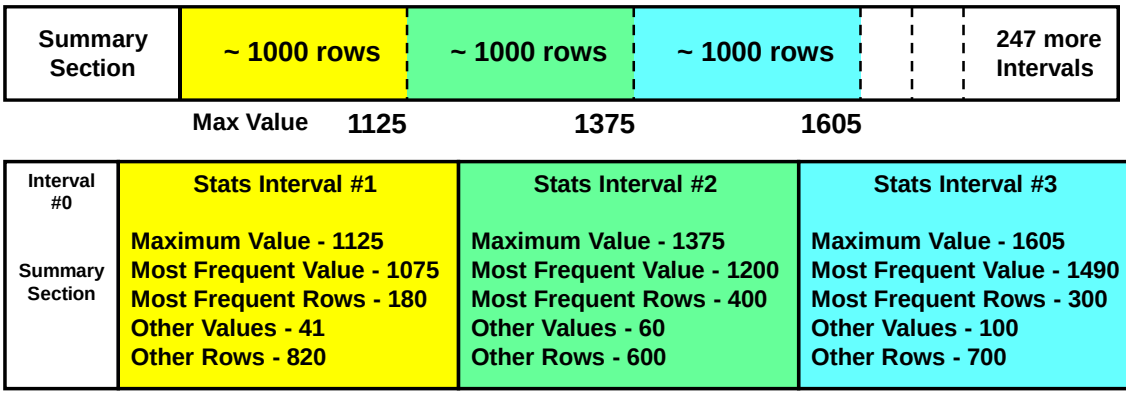
If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 BETWEEN 1150 AND 1550;
The optimizer will assume 1350 duplicates exist for this value.

400 + 300 + 300 + 350 = 1350

Statistics Example

Assume a table has 250,000 rows and statistics are collected on col1. Each interval will represent about 1000 rows.



SQL Statement	Optimizer assumes
SELECT * FROM tabx WHERE col1 = 1200;	400
SELECT * FROM tabx WHERE col1 = 1075;	180
SELECT * FROM tabx WHERE col1 = 1492;	7
SELECT * FROM tabx WHERE col1 = 1300;	_____
SELECT * FROM tabx WHERE col1 BETWEEN 1150 AND 1250;	700
SELECT * FROM tabx WHERE col1 BETWEEN 1150 AND 1550;	_____

Statistics Example (cont.)

If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 = 1275;
The optimizer will assume 2100 duplicates exist for this value. 1275 is a high value within a high bias interval.

If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 = 2480;
The optimizer will assume 900 duplicates exist for this value. 2480 is a high value within a high bias interval.

If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 = 1300;
The optimizer will assume 10 duplicates exist for this value. There are 58 other values and 570 other rows. 570 ÷ 58 = 10.

Whenever a range is used for a portion of an interval, the optimizer makes the following calculation:

one-half of the other rows of the other values in the interval
\+ highest value within the range
\+ any values from the high bias section

If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 BETWEEN 1150 AND 1250;
The optimizer will assume 685 duplicates exist for this value.

	285	one-half of the other rows of the other values in Interval #3 (570/2)
+	400	number of duplicates for value 1200

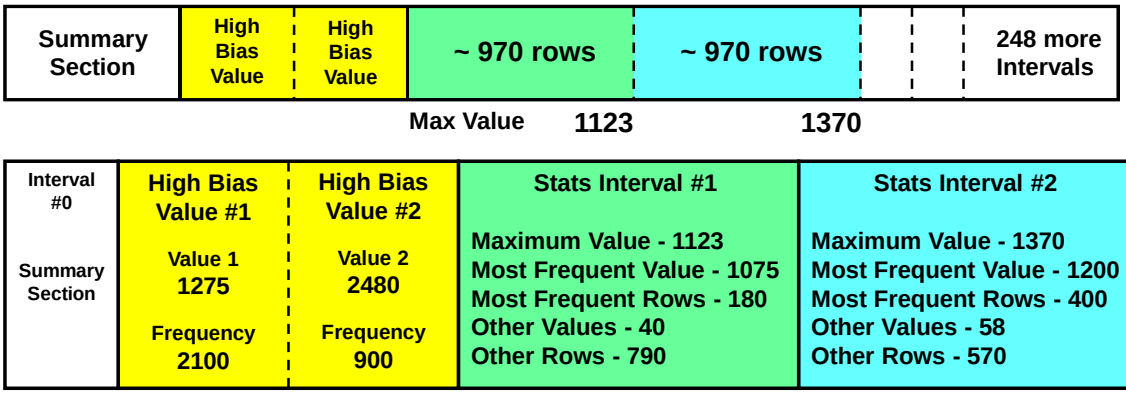
If the user executes the following SQL statement,

SELECT * FROM tabx WHERE col1 BETWEEN 1150 AND 1350;
The optimizer will assume 2785 duplicates exist for this value.

2100 + 285 + 400 = 2785

Statistics Example (cont.)

Assume a table has 250,000 rows and statistics are recollected on col1. Two values occur more than .20% (1/500) on col1. (In this example, more than 500 duplicate values.)



SQL Statement	Optimizer assumes
SELECT * FROM tabx WHERE col1 = 1275;	2100
SELECT * FROM tabx WHERE col1 = 2480;	900
SELECT * FROM tabx WHERE col1 = 1300;	10
SELECT * FROM tabx WHERE col1 BETWEEN 1150 AND 1250;	685
SELECT * FROM tabx WHERE col1 BETWEEN 1150 AND 1350;	2785

COLLECT STATISTICS Command

The Optimizer format for the COLLECT STATISTICS statement is shown on the facing page.

If you are collecting statistics for a table, specify a table name to indicate on which table you want to COLLECT STATISTICS. There are options that allow you to collect for a single column, a list of the columns, or a named index. These options are required to define and collect statistics initially. They may also be used to specify an existing statistic you want refreshed. They can be eliminated if you want the system to refresh all previously defined statistics for the named table.

You can edit your CREATE INDEX statements to create your COLLECT STATISTICS statements. Just replace "CREATE INDEX" with "COLLECT STATISTICS".

Miscellaneous Considerations

The maximum number of columns on which you may collect statistics has is 512 for a table. You can collect or recollect statistics on a combined maximum of 512 implicitly specified columns and indexes. An error results if statistics are requested for more than 512 columns and indexes in a table. This error condition could result during re-collection, when all statistics that have been previously collected are re-generated. For example, if you collected statistics six different times, each for a set of 100 columns, each of those collect statements would work, because they would each be under the 512 limit. However, if you re-collected the statistics the system would attempt to generate all 600 and would fail when it hit the 512 limit.

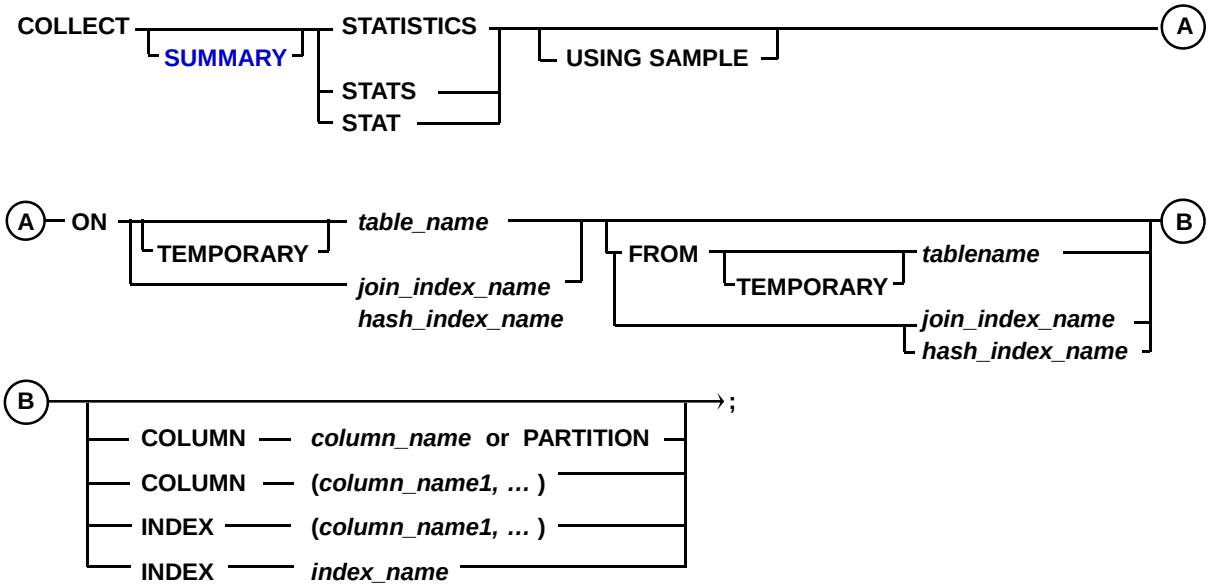
Increasing this limit allows users to issue larger re-collections involving more spool files. While this may increase system resource usage, it also provides better, more accurate information to the optimizer. Improved optimization results in more efficient execution plans, which significantly reduce system resource usage during query processing. Efficient system usage reduces overhead cost and time.

The bottom line on effective use of COLLECT STATISTICS is: Collect ***all*** and ***only*** what is needed to help the Optimizer make the best possible execution plans. Collecting more statistics than are needed or used wastes resources by putting extra processing and storage burdens on the system.

When to Collect Multiple Column Statistics

- If a set of columns are frequently specified together as an equality condition or in a join condition, then collect on the set of multiple columns.
- If a multiple column non-unique index is defined, then collect statistics on the index or the set of columns that comprise the index.

COLLECT STATISTICS Command



SUMMARY (14.0 option) – collects table-level summary statistics – cardinality (number of rows), average # of rows per AMP, etc.

If collecting on a column or index, summary statistics are also collected automatically.

Optionally, you can collect just summary statistics on a table.

Collecting Statistics

Initial Definition and Collection

Each of the statements applies to a separate column of the table. The COLUMN or INDEX parameters must be specified with COLLECT STATISTICS.

Here are some notes about the Explain on the facing page:

- The system acquires an ACCESS lock on the table that will not interfere with existing transactions.
- The Row Hash WRITE lock on DBC.StatsTbl in Step #5 will not prevent new transactions which reference the column **Custid** from parsing.
- Steps 9 and 10 send out a "Spoiling Message" to all PEs. This will spoil any current execution plans that use the **Custid** column for access. It will also remove the column and its existing demographic information from DD cache so that new requests will cause the Parser to access DBC.StatsTbl for the new demographic information.

Example

```
COLLECT STATISTICS ON Orders COLUMN Orderid;
COLLECT STATISTICS ON Orders COLUMN Custid;
COLLECT STATISTICS ON Orders COLUMN Orderdate;
EXPLAIN COLLECT STATISTICS ON Orders;
```

Locks and Concurrency

When you perform a COLLECT STATISTICS statement, the system places an ACCESS lock on the table from which the demographic data is being collected. The system places row-hash-level WRITE locks on the DBC.StatsTbl while collecting statistics at the column and index levels.

In general, COLLECT STATISTICS statements can run concurrently with DML statements, other COLLECT STATISTICS statements, DROP STATISTICS statements, and HELP STATISTICS statements against the same table. COLLECT STATISTICS can also run concurrently with a CREATE INDEX statement against the same table as long as they are not for the same index.

Collecting Statistics

Initial Definition and Collection

```
COLLECT STATISTICS ON Orders COLUMN Orderid;
COLLECT STATISTICS ON Orders COLUMN Custid;
COLLECT STATISTICS ON Orders COLUMN Orderdate;

EXPLAIN COLLECT STATISTICS ON Orders COLUMN Custid;

Explanation                                                                    14.0 EXPLAIN
---------------------------------------------------------------------------------------------------------------------------------
 1) First, we lock DS.Orders for access.
 2) Next, we do an all-AMPs SUM step to aggregate from  DS.Orders by way of a traversal of index # 4
    without accessing the base table with no residual conditions, grouping by field1 (
    DS.Orders.custid). Aggregate Intermediate Results are computed globally, then placed in Spool 8.
    The size of Spool 8 is estimated with low confidence to be 418 rows (12,122 bytes).  The estimated
    time for this step is 0.06 seconds.
 3) Then we save the UPDATED STATISTICS for ('custid') from Spool 8 (Last Use) into Spool 4, which is
    built locally on a single AMP derived from the hash of the table id.
 4) We compute the table-level summary statistics from spool 4 and save them into Spool 6, which is
    built locally on a single AMP derived from the hash of the table id.
 5) We lock DBC.StatsTbl for write on a RowHash.
 6) We do a single-AMP ABORT test from DBC.StatsTbl by way of the primary index with a residual
    condition of ("(DBC.StatsTbl.StatsId = 0) OR ((DBC.StatsTbl.ExpressionList ='custid') OR
    (DBC.StatsTbl.StatsId = 1 ))").
 7) We do a single-AMP MERGE into DBC.StatsTbl from Spool 4 (Last Use).
 8) We do a single-AMP MERGE into DBC.StatsTbl from Spool 6 (Last Use).
 9) We spoil the statistics cache for the table, view or query.
10) We spoil the parser's dictionary cache for the table.
11) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
->  No rows are returned to the user as the result of statement 1.
```

Refresh or Re-Collect Statistics

Statistics should not be "collected" once and then forgotten. They tend to become stale as the tables change, and can cause performance problems by misleading the Optimizer into making poor decisions. Always keep your statistics current and valid by refreshing them on a regular, production-schedule basis. To refresh or recollect statistics for all of the previously collected columns, simply execute the COLLECT STATISTICS command for the table. The following EXPLAIN show the execution plan when multiple columns are recollected.

EXPLAIN COLLECT STATISTICS ON Orders;

1) First, we lock TFACT.orders for access.
2) Next, we do an all-AMPs SUM step to aggregate from TFACT.orders by way of an all-rows scan with no residual conditions, grouping by field1 (TFACT.orders.orderdate). Aggregate Intermediate Results are computed globally, then placed in Spool 18. The size of Spool 18 is estimated with high confidence to be 3,622 rows (105,039 bytes). The estimated time for this step is 0.08 seconds.
3) Then we save the UPDATED STATISTICS for ('orderdate') from Spool 18 (Last Use) into Spool 4, which is built locally on a single AMP derived from the hash of the table id.
4) We do an all-AMPs SUM step to aggregate from TFACT.orders by way of a traversal of index # 8 without accessing the base table with no residual conditions , grouping by field1 (TFACT.orders.custid). Aggregate Intermediate Results are computed globally, then placed in Spool 21. The size of Spool 21 is estimated with high confidence to be 526 rows (15,243 bytes). The estimated time for this step is 0.06 seconds.
5) Then we save the UPDATED STATISTICS for ('custid') from Spool 21 (Last Use) into Spool 9, which is built locally on a single AMP derived from the hash of the table id.

: :

15) We spoil the parser's dictionary cache for the table.
16) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> No rows are returned to the user as the result of statement 1.

Why are Collect Statistics done serially?

Executing individual collect statistics or a single collect on table will single thread the collections – they will be done one at a time. This is good if you want to minimize impact on other work. If you want to get it done in minimum time though, launch multiple sessions and submit individual collects on each. Total elapsed time will be better for several running concurrently than it will be for running them all one at a time. However, this will consume more of the system resources – specifically more CPU cycles.

The Collect Statistics code has to pull the data independently for each stats definition. It basically has to do a large aggregation grouping on the stats columns. Thus it will do a scan and aggregate for each one. It turns out that the scan of data is the least expensive part of the operation, the real cost is the aggregation/computation of the stats.

Normally, collect stats does not utilize sync scan; sync scans would only help if you were running them in separate sessions. However, it does utilize cylinder read which has a big positive impact on stats performance.

Refresh or Re-Collect Statistics

To refresh or recollect statistics for all of the previously collected columns and/or indexes, you can collect at the table level.

```
COLLECT STATISTICS ON Orders;
```

Refresh statistics whenever 5% - 10% of the rows have changed:

- **As part of batch maintenance jobs.**
- **After significant periods of OLTP updating.**
- **After new low and/or high values have extended the range of values.**

Note:

- **When refreshing multiple statistics in a table, this is effectively multiple statistics collections and these are NOT collected in a single pass of the data.**

Why not?

- **The main reason is that a statistics collection is very CPU intensive – aggregation and computation of the statistical data.**
  - **Multiple statistics collection at the same time can use a considerable amount of system CPU.**
- **Optionally, you can run separate statistics collections in different sessions, but you will use a high % of the CPU.**

COLLECT STATISTICS Command

(Index Format with 14.0 Options)

The Index format syntax for the COLLECT STATISTICS statement is shown on the facing page. You can edit your CREATE INDEX statements to create your COLLECT STATISTICS statements. Just replace "CREATE INDEX" with "COLLECT STATISTICS".

16 Byte Limit in Statistics Intervals

Prior to Teradata 14.0, only the first 16-bytes of a column(s) that have collected statistics are stored in the statistic intervals. Lengthy multi-column stats will usually overflow the 16 byte limit of how much meaningful data can be carried within the statistics intervals.

For example, if the first column in a multi-column stats is CHAR(20), then no values from the other columns will get recorded in the histogram intervals. Another example is where a large number of the values in a CHAR column start with a common string, for example "State Department of". As a result, there is no differentiation for these particular values in the histograms, because the differentiation comes after the 16 byte limit, so the optimizer was not able to link the actual value to helpful estimates. Dropping statistics in this case may improve query plans.

The rule of thumb with statistics, when you ask yourself whether to collect them or not on a specific column or set of columns is whether they improve the query plans or not. If they do, collect them, if they don't drop them.

This is no longer an issue starting with Teradata 14.0. You can override the maximum value length (based on column lengths by using the MAXVALUELENGTH *n* option.

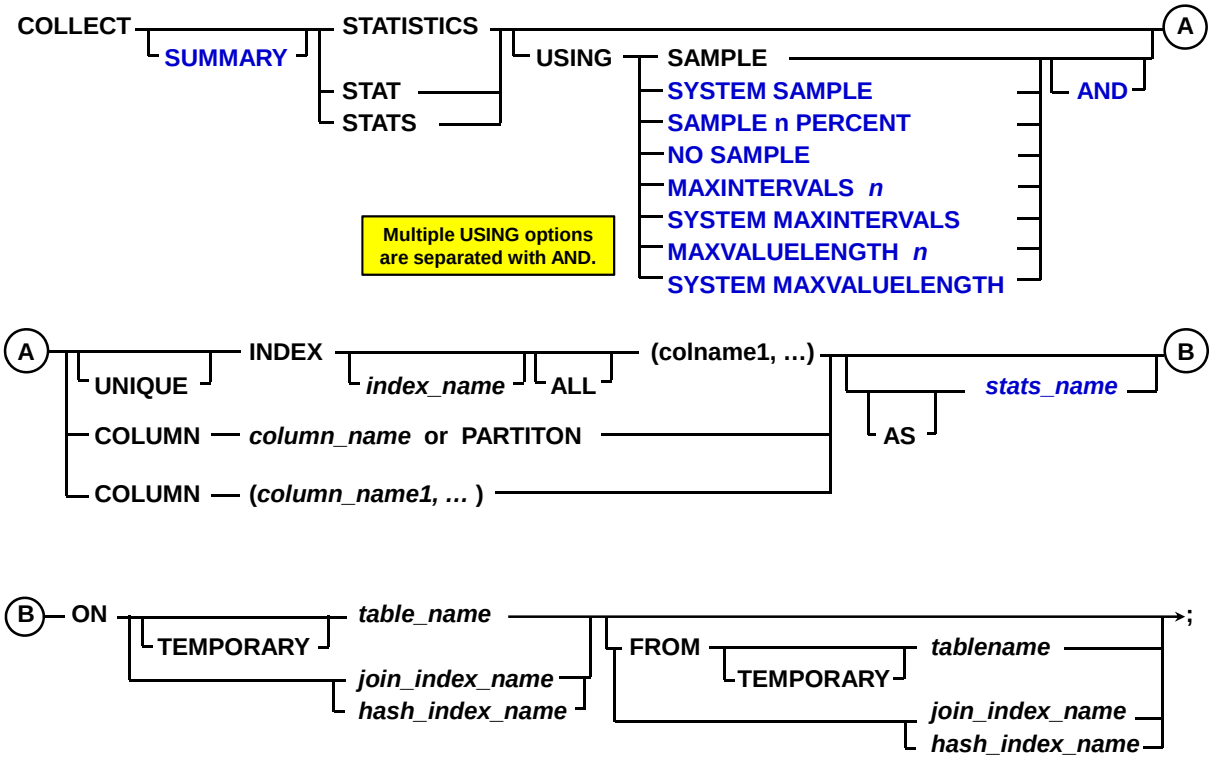
Teradata Database never truncates numeric values for single-column statistics. The system increases the interval size automatically if the specification is not sufficient to accommodate the full value for single-column statistics on numeric columns.

For multicolumn statistics, if the maximum interval size truncates numeric statistical data, Teradata Database automatically increases the maximum interval size to accommodate the numeric column on the maximum size boundary. A larger maximum value size causes Teradata Database to retain the value until the specified maximum is reached, which can enable better single-table and join selectivity estimates for skewed columns. However, you should be selective when increasing the size for the required columns because increasing the maximum value size also increases the size of the histogram, which can increase query optimization time. You can only specify this option if you also specify an explicit column or index.

NO SAMPLE

NO SAMPLE specifies to use a full-table scan to collect the specified statistics. You can only specify this option if you also specify an explicit index or column set.

COLLECT STATISTICS Command
(Index Format with 14.0 Options)



COLLECT STATISTICS on a Data Sample

The new SAMPLING option significantly reduces the resources consumed by COLLECT STATISTICS by collecting on only a percentage of the data. While it should not be used to replace all existing statistics collection, it can be used as an effective query tuning option where statistics are not being collected because of the required overhead. This feature saves time and system resources by collecting statistics on only a percentage sampling of the data.

COLLECT STATISTICS can be very time consuming because it performs a full table scan and its sorts the data to compute the number of occurrences of each distinct value. Most users accept this performance because it can be run infrequently and it benefits query optimization. Without statistics, query performance often suffers. The drawback to sampled statistics is that they may not be as accurate, which in turn may affect the quality of Optimizer plans. In most cases sampled statistics are better than no statistics.

Consider using sampling when:

- Collecting statistics on very large tables.
- Resource consumption from the collection process is a serious performance or cost concern.

Do not use sampling:

- On small tables.
- To replace all existing full scan collections.

Sampling Considerations

The system automatically determines the appropriate sample size to generate accurate statistics for good query plans and performance.

Once initial sampled statistics have been collected, recollections are also sampled. The system does *not* store both sampled and complete statistics for the same index or column set.

Sampled statistics are generally more accurate for data that is uniformly distributed. For example, columns or indexes that are unique or nearly unique are uniformly distributed. Sampling should not be considered for data that is highly skewed because the Optimizer needs to be fully aware of such skew. In addition to uniformly distributed data, sampling is generally more appropriate for indexes than non-indexed column(s). For indexes, the scanning techniques employed during sampling can take advantage of the hashed organization of the data to improve the accuracy of the resulting statistics.

To summarize, sampled statistics are generally most appropriate for:

1. Very large tables
2. Data that is uniformly distributed
3. Indexed column(s)

COLLECT STATISTICS on a Data Sample

SAMPLING reduces resource usage by collecting on a percentage of the data.

- System determines appropriate sample size to generate accurate statistics.
  - If you specify SAMPLE or SYSTEM SAMPLE, the first collection will usually be 100%. Teradata will determine a percentage to use for recollections.
- To request a specific sample of rows on which to collect statistics, use this syntax:

  COLLECT STATISTICS USING SAMPLE n PERCENT
- Re-collection collects statistics using the same mode (full scan or sampled) as specified in the last specified collection.
  - System does *not* store both sampled and complete statistics for the same index/column set.
  - Only one set of statistics is stored for a given index or column.
- Sampled statistics are more accurate for data that is uniformly distributed.
  - Sampling is useful for very large tables with reasonable distribution of data.
  - Sampling should *not* be considered for highly skewed data as the Optimizer needs to be aware of such skew.
  - Sampling is more appropriate for indexes than non-indexed column(s).

Collecting Statistics (14.0 Examples)

USING OPTIONS

SAMPLE or SYSTEM SAMPLE

This option specifies to scan a system-determined percentage of table rows to collect the specified statistics. This option is not valid for single-table views. SAMPLE has the same meaning as SYSTEM SAMPLE and is provided for backward compatibility.

The SYSTEM SAMPLE option specifies to scan a system-determined percentage of table rows to capture the statistical information. The Teradata Database might decide to sample 100% for the first few times before downgrading the sample percent to a lower level.

MAXINTERVALS *n*

Specifies the maximum number of histogram intervals to be used for the collected statistics. Teradata Database might adjust the specified maximum number of intervals depending on the maximum histogram size. Valid range is 0 – 500.

SYSTEM MAXINTERVALS

This option simply indicates to use the system default maximum number of histogram intervals for the collected statistics. The system default is typically 250, but can be adjusted using DBSControl.

MAXVALUELENGTH *n*

This option specifies the maximum size to use for histogram values such as MinValue, ModeValue, MaxValue, etc. This option is valid for both tables and single-table views. The value for n must be an integer number.

For single-character statistics on CHARACTER and VARCHAR columns, *n* specifies the number of characters. For all other options, *n* specifies number of bytes.

For single-character statistics on CHARACTER and VARCHAR columns, *n* specifies the number of characters. For all other options, *n* specifies number of bytes.

For multicolumn statistics, Teradata Database concatenates the values and truncates them if necessary to fit into the specified maximum size.

- For single-column statistics, the valid range of *n* is 1 - maximum size of the column.
- For multicolumn statistics, the valid range of *n* is 1 - combined maximum size of all the columns.

SYSTEM MAXVALUELENGTH

This option simply indicates that the system should use a **system-determined** value for the length of histogram values such as MinValue, ModeValue, MaxValue, etc. This value is dependent on the column data type, number of intervals, etc. This option is valid for both tables and single-table views.

Collecting Statistics (14.0 Examples)

Given: **Orders table** **NUPI on orderid, partitioned on orderdate**
USI on orderid
NUSI on custid

To collect sample statistics using the system default sample:

```
COLLECT STATISTICS USING SYSTEM SAMPLE COLUMN (orderdate) ON Orders;
```

To collect sample statistics by scanning 10 percent of the rows and use 100 intervals:

```
COLLECT STATISTICS USING SAMPLE 10 PERCENT AND MAXINTERVALS 100
                COLUMN (custid) AS custid_stats ON Orders;
```

To change sample statistics to 20 percent (for custid) and use 250 intervals:

```
COLLECT STATISTICS USING SAMPLE 20 PERCENT AND MAXINTERVALS 250
                COLUMN (custid) AS custid_stats ON Orders;
```

To display the COLLECT STATISTICS statements for a table:

```
SHOW STATISTICS ON Orders;
```

To display statistics details – summary section, high bias values, and intervals:

```
SHOW STATISTICS VALUES COLUMN orderdate ON Orders;
```

Viewing Statistics

Help Statistics

HELP STATISTICS returns the following information about each column or index for which statistics have been COLLECTed in a single table:

- The Date the statistics were last COLLECTed or refreshed.
- The Time the statistics were last COLLECTed or refreshed.
- The number of unique values for the column or index.
- The name of the column or column(s) that the statistics were COLLECTed on.

Use Date and Time to help you determine if your statistics need to be refreshed or dropped. The example on the facing page illustrates the HELP STATISTICS output for the Orders table.

Note: Prior to Teradata 14.0, **HELP STATS Orders COLUMN (Custid);** will return the details about the statistics histogram and display the 200 intervals. This command will not work starting with Teradata 14.0.

Starting with Teradata 14.0, to view details about the statistics for an index/column, use the **SHOW STATISTICS VALUES** option.

Help Index

HELP INDEX is an SQL statement which returns information for every index in the specified table. An example of this command and the resulting BTEQ output is shown on the facing page. As you can see, HELP INDEX returns the following information:

- Whether or not the index is unique
- Whether the index is a PI or an SI
- The name(s) of the column(s) which the index is based on
- The Index ID Number
- The approximate number of distinct index values
- Is the index hash-ordered, valued-ordered, or partitioned (H, V, P)

This information is very useful in reading EXPLAIN output. Since the EXPLAIN statement only returns the Index ID number, you can use the HELP INDEX statement to determine the structure of the index with that ID.

Answers to question: The counts are determined by a random AMP sample.

Viewing Statistics

HELP STATISTICS tablename; Displays information about current statistics.

HELP STATISTICS Orders;

Date	Time	Unique Values	Column Names	
12/03/07	00:55:34	43,200	*	(14.0 – The * is Summary Statistics.)
12/03/07	00:53:54	3,622	orderdate	
12/03/07	00:54:01	385	custid_stats	(Note the statistics name is used here.)
12/03/07	00:55:34	1	PARTITION	

Note that the DATE and TIME show when statistics were last collected or refreshed.

HELP INDEX tablename: This statement returns information for every index on a table.

HELP INDEX Orders;

Unique?	Primary or Secondary?	Column Names	Index Id	Approximate Count	Index Name	Ordered or Partitioned?
N	P	orderid	1	32,741	?	P
Y	S	orderid	4	43,654	Orderid	H
N	S	custid	8	449	Custid	H

This command displays the index number and an approximate number of distinct values.

Question: How are the counts determined with HELP INDEX?

Optimizer's use of Statistics with Uneven NUSI

COLLECTing STATISTICS on NUSIs makes it possible for the Optimizer to decide when to use the NUSI for access and when to do a Full Table Scan. The facing page shows how the Optimizer will handle two different uneven NUSIs.

The NUSI on column F has a large distribution spike where F is NULL. The Optimizer will choose to do a Full Table Scan for NULL values and will utilize the NUSI for other values.

The NUSI on column G has a large distribution at a single value. The Optimizer will choose to utilize the NUSI for NULL values, and may or may not utilize the NUSI for other values.

Data Dictionary Views

There are a number of data dictionary views that can be used to determine if statistics have been collected on indexes and/or columns. The following chart illustrates which views will display the various statistics.

These views have been significantly changed in Teradata 14.0 with all new columns.

This chart represents how these views can be used with Teradata 12 through Teradata 13.10.

	Are Statistics provided in the View?		
Stats Collected on …	DBC.IndexStats	DBC.ColumnStats	DBC.MultiColumnStats
Single Column	No	Yes	No
Single Column Index	No	Yes	No
Index with Multi-Columns	Yes	No	No
Multi-Columns *	Yes	No	No
Multi-Column (no index on columns)	No	No	Yes

* An index is on multiple columns, but stats were collected on columns, not the index.

Optimizer's use of Statistics with Uneven NUSI

EXAMPLE

6,000,000 Rows	A		F	G	
PK/FK	PK,SA				
Distinct Values	6M		1.5M	1.5M	
Max Rows/Value	1		9	725K	
Max Rows/NULL	0		725K	5	
Typical Rows/Value	1		3	3	
	UPI				
PI/SI	USI		NUSI?	NUSI?	

Notes:

For column G, assume:

- 725K rows have a value of 0.
- All other values in G return 0 to 30 rows.

Example: **Optimizer will ...**

COLLECT STATISTICS ON Example COLUMN F;

SELECT * FROM Example WHERE F = 'value'; use the NUSI to return 3 - 9 rows
SELECT * FROM Example WHERE F IS NULL; do a FTS to return 725K rows

COLLECT STATISTICS ON Example COLUMN G;

SELECT * FROM Example WHERE G = 'value'; use the NUSI if the value NE 0
SELECT * FROM Example WHERE G IS NULL; use the NUSI to return 5 rows

DROP INDEX (G) ON Example; DROP STATISTICS ON Example COLUMN G;

SELECT * FROM Example WHERE G = 'value'; assumes 10% or 600,000

Collecting Statistics on PARTITION

This feature allows you to collect statistics on the system-derived PARTITION column of a table with a Partitioned Primary Index (PPI).

Statistics on the PARTITION column allow the Optimizer to generate an aggressive plan with respect to PPI tables. Specifically, the Optimizer can use PARTITION statistics to estimate the cost of various operations very accurately, including:

- Static partition elimination
- Dynamic partition elimination

When the Optimizer can use partition elimination more frequently, query efficiency and performance improves. In addition, the collection process itself for single-column PARTITION statistics is highly optimized, which significantly reduces their collection time and overhead.

Prior to Teradata Warehouse V2R6.1, statistics collected on partitioning columns were used only for estimating cardinality (the number of qualifying rows resulting from access to the table). With the new PARTITION statistics feature, the additional statistics can be used to estimate costing (the number of rows, data blocks, and partitions that must be scanned to produce the query result). Statistics can be collected for the system-derived PARTITION column only (single-column PARTITION statistics) or on the PARTITION column and other table columns (multi-column PARTITION statistics). These statistics provide the Optimizer with information on its state when statistics were last collected, such as:

- The number of empty partitions
- How rows are distributed in each partition

The Optimizer can use this information to better estimate the query cost when there are a significant number of empty partitions. If PARTITION statistics are not collected, empty partitions may cause the Optimizer to underestimate the number of rows in a partition.

When the Optimizer uses PARTITION statistics in creating its plan, the EXPLAIN will show the estimate with high confidence.

Notes on Collecting Sample Statistics on the Partitioning column:

You can specify a USING SAMPLE clause to collect single-column PARTITION statistics, but the specification is ignored. Instead, the system automatically resets the internal sampling percentage to 100.

Note that the USING SAMPLE clause is honored for multicolumn PARTITION statistics. The system begins the collection process using a 2 percent sampling percentage and then increases the percentage dynamically if it determines that the values are skewed more than a system-defined threshold value.

Collecting Statistics on PARTITION

You can (and should) collect statistics on the system-derived PARTITION column of all tables, especially PPI tables.

- **Statistics on the PARTITION column provide information about partitions and allow the Optimizer to generate a more aggressive plan with respect to PPI tables.**
- **Specifically, the Optimizer can use PARTITION statistics to estimate the cost of various operations more accurately, including:**
  - **Static partition elimination**
  - **Dynamic partition elimination**
- **The Optimizer can use this information to better estimate the query cost when there are a significant number of empty partitions.**
- **This is a very fast collection as only the Cylinder Indexes need to be analyzed.**
- **It is recommended to collect statistics on PARTITION for every table (including non-partitioned tables).**

Example:

```
COLLECT STATISTICS ON TFACT.Sales_PPI  COLUMN PARTITION;
```

Copying STATISTICS

You can also append an AND STATISTICS option to a WITH [NO] DATA clause to copy any collected source table statistics or empty histograms to a target table. If you specify WITH NO DATA AND STATISTICS, the system sets up the appropriate statistical histograms in the dictionary, but does not populate them with source table statistics. This is referred to as *zeroed statistics*.

General Rules For CREATE TABLE AS … WITH DATA AND STATISTICS. The following list of rules applies only to an AS … WITH DATA AND STATISTICS clause.

- This feature is available starting with Teradata V2R6.2.
- If there are no columns or indexes in the target table for which statistics are eligible to be copied, the system returns a warning message to the requestor.
- If you specify an explicit index definition for the target table, then the system does not copy PARTITION statistics from the source table to the target table.
- This is true for both single-column PARTITION statistics and for composite statistics on a column set that includes the system-derived PARTITION column.
- You cannot copy statistics for a volatile table because you cannot *collect* statistics on a volatile table.
- If you specify WITH DATA AND STATISTICS for a volatile source table, the request aborts and the system returns an error.
- If no statistics have been collected on the specified source table column or index sets, the system ignores the AND STATISTICS option and returns a warning message to the requestor.
- If only a subset of the statistics from the source table are eligible to be copied to the columns and indexes of the target table, the system returns a warning message to the requestor.
- If the number of multicolumn statistics you specify to be copied to the target table exceeds the maximum number of multicolumn statistics allowed, then the system copies multicolumn statistics only up to the limit, does not copy the remainder of the multicolumn statistics to the target table, and reports a warning message to the requestor.
- If all columns in a MULTISET source table are non-unique, and if the target table is a SET table, then the system does not copy statistics to the target table. This is because of the possible violation of the rule of equal cardinalities in the source and target tables: if there are duplicate rows in the source table, the system eliminates them before copying to the target table, resulting in unequal cardinalities between the two tables.
- If the source table has *at least* one non-unique column and the target table is a SET table, then the system copies the statistics from the source table to the target table if all other rules are also obeyed.
- If you specify the NOT CASESPECIFIC attribute for any column in the target table definition, and it does not match the corresponding source table column attribute specification, the system does not copy the statistics for that column or index set because of the possible violation of the rule about not modifying the data in the target table.
- If all the columns in the source table are non-unique and you specify the NOT CASESPECIFIC attribute for any column in the target table definition that does not match the corresponding source table column attribute definition, then the system does not copy the statistics for that column or index because of the possible violation of the rule of equal cardinalities in the source and target tables.

Copying STATISTICS

The COLLECT STATISTICS command includes a FROM option to copy statistics from one table to an identical target table (14.0 option).

```
COLLECT STATISTICS ON Orders_new FROM Orders;
```

The CREATE TABLE AS command includes an option to copy statistics from one table to another by including the "AND STATISTICS" option.

- The CREATE TABLE … AS … WITH DATA AND STATISTICS;
  - In this case, a new table is created via a copy definition (DDL) of an existing table, data is copied from the source table to the target table, and the applicable statistics are replaced with the same statistics (histograms) from the target table.

    ```
    CREATE TABLE Customer_Test2 AS CUSTOMER WITH DATA AND STATISTICS;
    ```

- The CREATE TABLE … AS … WITH NO DATA AND STATISTICS;
  - In this case, a new table is created via a copy definition (DDL) of an existing table, no data is copied from the source table to the target table, and the applicable statistics are replaced with zeroed statistics (histograms).

    ```
    CREATE TABLE Customer_Test1 AS CUSTOMER WITH NO DATA AND STATISTICS;
    ```

Statistics Extrapolation

The optimizer can use extrapolation or derived statistics when it detects stale statistics. Stale statistics is based on 10% or 10,000 row count growth. For very small tables (less than 25 rows per AMP), or for tables with a skewed data distribution, no extrapolation will done. A technique to force extrapolation is to switch to an all AMP random AMP sampling.

If the difference exceeds either of these values, the Optimizer assumes that the histogram statistics are stale and overrides them with the estimates returned by a random AMP sample.

Even if the difference exceeds the value for the absolute deviation (10,000), the Optimizer still uses histogram statistics for its selectivity estimates, and those estimates are extrapolated.

Stale Statistics Detection

Currently, the table row count is estimated from the random AMP sampling or the statistics on the primary index (PI) of the table.

Starting with Teradata 12.0, instead of always trusting Primary Index histogram row count, the row count from Random AMP Sampling and the histogram are compared and a decision is made based on certain normalization heuristics.

- The histogram row count is compared with the table row count and if the deviation is more than the defined threshold, the histogram is determined as stale.
- Stale histograms are specially tagged in the optimizer and value count/row extrapolations are done when they are used for cardinality estimations.
- Stale Statistics Detection also applies to tables that have zero statistics as well and allows for table row count extrapolation.

Extrapolating Statistics Outside Range

This 12.0 enhancement allows the Teradata Optimizer to include a new extrapolation technique specifically designed to more accurately provide for a statistical estimate for date range-based queries that specify a “future” date.

The Optimizer extrapolation technique for date range-based queries that supply “future” dates will result in better query plans due to the fact that cardinality estimation will be much more accurate. Because of the new extrapolation formula it is also possible that statistics for the associated date columns would not have to be re-collected as often.

Some considerations include:

- However, the information displayed within a query Explain plan will change because of the new numbers for estimated rows.
- Specific consideration should be given to collecting statistics less frequently on columns which will now be extrapolated.

Statistics Extrapolation

The optimizer can use extrapolation or derived statistics (random AMP sample) when it detects stale statistics.

Stale Statistics Detection

- **The row count from random AMP sampling and the histogram are compared and a decision is made based on certain normalization heuristics.**
  - **The histogram row count is compared with the table row count and if the deviation is more than a defined threshold, the histogram is determined as stale.**
    - **The stale statistics threshold is based on 10% or 10,000 (small tables) row count growth.**
  - **Stale histograms are specially tagged in the optimizer and value count/row extrapolations are done when they are used for cardinality estimations.**

Extrapolate Statistics Outside of Range

- **This extrapolation technique is designed to more accurately provide for a statistical estimate for date range-based queries that specify a "future" date that is outside the bounds of the collected statistics.**
- **For date range-based queries that supply "future" dates, the optimizer can create better plans because the cardinality estimation will be more accurate.**
- **A benefit is that statistics may not have to be re-collected as often.**

Teradata 13.0 Enhancements

The facing page lists a number of statistics enhancements in Teradata 13.0.

Teradata 13.0 Enhancements

The following enhancements on Collect Statistics statements are available in Teradata 13.0.

- **Multicolumn statistics are available for Hash indexes and Join Indexes.**
- **Statistics on system-derived column PARTITION can be collected on Partitioned Join Indexes.**
- **PARTITION statistics can be collected on Global Temporary tables.**
- **Single, Multicolumn, and PARTITION statistics are available for Volatile tables.**
- **Sampled Statistics can be collected on all the above as well as Partitioning columns.**
- **Collect Statistics can be granted as an access right or privilege.**

Teradata 14.0 Enhancements

The facing page lists a number of statistics enhancements in Teradata 14.0. Benefits of these enhancements include:

- These enhancements allow you to specify whether you or the system should determine a sampling percentage for sampled statistics.
- Enables you to collect or recollect either summary statistics only or both full and summary statistics.
- Removes the restriction on collecting statistics on global temporary tables.
- Enables you to provide a name for statistics collected on a base table or global temporary table.
- These enhancements allow you to explicitly specify the column ordering for multicolumn statistics.
- Improves query optimization time through a new dedicated statistics cache.
- Simplifies privileges:
  - To collect or report statistics on a row-level security (RLS) table, you must have the OVERRIDE SELECT CONSTRAINT privilege.
  - To collect or drop statistics on non-RLS tables, you need only have the STATISTICS privilege.
  - To copy statistics from one table to a duplicate source table, you must have the SELECT privilege on the source table.

Teradata 14.0 Enhancements

The following Teradata 14.0 Collect Statistics enhancements include:

- **SUMMARY option to collect table-level statistics.**
- **SYSTEM SAMPLE option allows the system to determine the sampling percentage for sampled statistics.**
  - **Sampling options have been enhanced (e.g., SAMPLE n PERCENT)**
- **Statistics are stored in a new system table named DBC.StatsTbl.**
  - **This reduces access contention and improves performance.**
  - **New system views (or significantly changed views) include DBC.StatsV, DBC.ColumnStatsV, DBC.MultiColumnStatsV, and IndexStatsV.**
- **SHOW STATISTICS statement reports detailed statistics in either plain text or XML formatting.**
  - **SHOW STATISTICS has separate Optimizer and Query Capture Database (QCD) versions.**
- **Miscellaneous internal enhancements to the Optimizer with respect to histogram structure and use, including:**
  - **Storing statistics data in their native Teradata data types without losing precision**
  - **Enhanced extrapolation methods for stale statistics**
  - **Maintaining statistics history**

Teradata Statistics Wizard

The Teradata Statistics Wizard is a graphical tool that has been designed to automate the collection and re-collection of statistics, resulting in better query plans and helping the DBA to efficiently manage statistics.

The Statistics Wizard enables the DBA to:

- Specify a workload to be analyzed for recommendations specific to improving the performance of the queries in a workload.
- Select an arbitrary database or selection of tables, indexes, or columns for analysis, collection, or re-collection of statistics.
- Make recommendations, based on a specific workload.
- Make recommendations, based on table demographics and general heuristics.
- Defer execution of and schedule the arbitrary collection/re-collections for later.
- Display and modify the interval statistics for a column or index.

Recommendations

Recommendations can be provided by the Teradata Statistics Wizard utility for either a defined workload (set of SQL statements) or for a general set of non-workload settings.

The overall process of collecting statistics for a workload is:

- Define and specify the workload to The Teradata Statistics Wizard.
- The Teradata Statistics Wizard analyzes the workload (SQL statements).
- Once the statistics are chosen by the system, the Teradata Statistics Wizard makes recommendations and provides reports to help the user analyze the recommendations for collection purposes.
- Additionally, the Statistics Wizard permits users to validate the proposed statistics on a production system. This feature enables the user to verify the performance of the proposed statistics before applying the recommendations.

If you choose to specify statistics recommendations rather than capture representative samples of collected statistics within a defined workload, you set user preferences by using the Options choice within the Tools menu.

Teradata Statistics Wizard

Features of the "Teradata Statistics Wizard" include:

- Provides a graphical Windows interface to easily view, collect, recollect, or drop statistics on selected tables.
- Provides non-workload recommendations to collect statistics on tables based on …
  - Table skew
  - General heuristics (based on general rules of thumb)
- Provides non-workload recommendations to re-collect statistics on tables based on …
  - Age of collection
  - Table growth - change in demographics
- Based on a specific workload, provides workload recommendations on columns to re-collect statistics.
  - Integrated with Teradata Analyst tool set – Index Wizard, Statistics Wizard, and Teradata SET (System Emulation Tool).
  - Can utilize a workload defined in QCD (Query Capture Database)
- Display and modify the interval statistics for a column or index.
- Provides numerous reports on statistics recommendations, update cost analysis, table usage, etc.

Teradata Statistics Wizard – Main Window

The Menu bar is located at the top of the Teradata Statistics Wizard window. It contains the pull-down menus displaying the commands that you use to operate the program.

In addition to accessing the commands from the menu bar, the Teradata Statistics Wizard provides shortcuts to many of the commands via the tool buttons along the top and left side of the window.

The Teradata Statistics Wizard window displays Teradata Database statistics in three panes:

- The **Database pane** is a listing of all the databases on the system. Double-click on a database icon to see the children of that database and to retrieve the associated tables.
- The **Table pane** lists the tables contained in the selected database. Double-click a table name to see the statistics for that table.
- **Column/Index Pane** - the Column pane shows statistics for the selected database/table. Double-click a column name to see the interval statistics for that column.

Defining and Using Workloads with Statistics Wizard

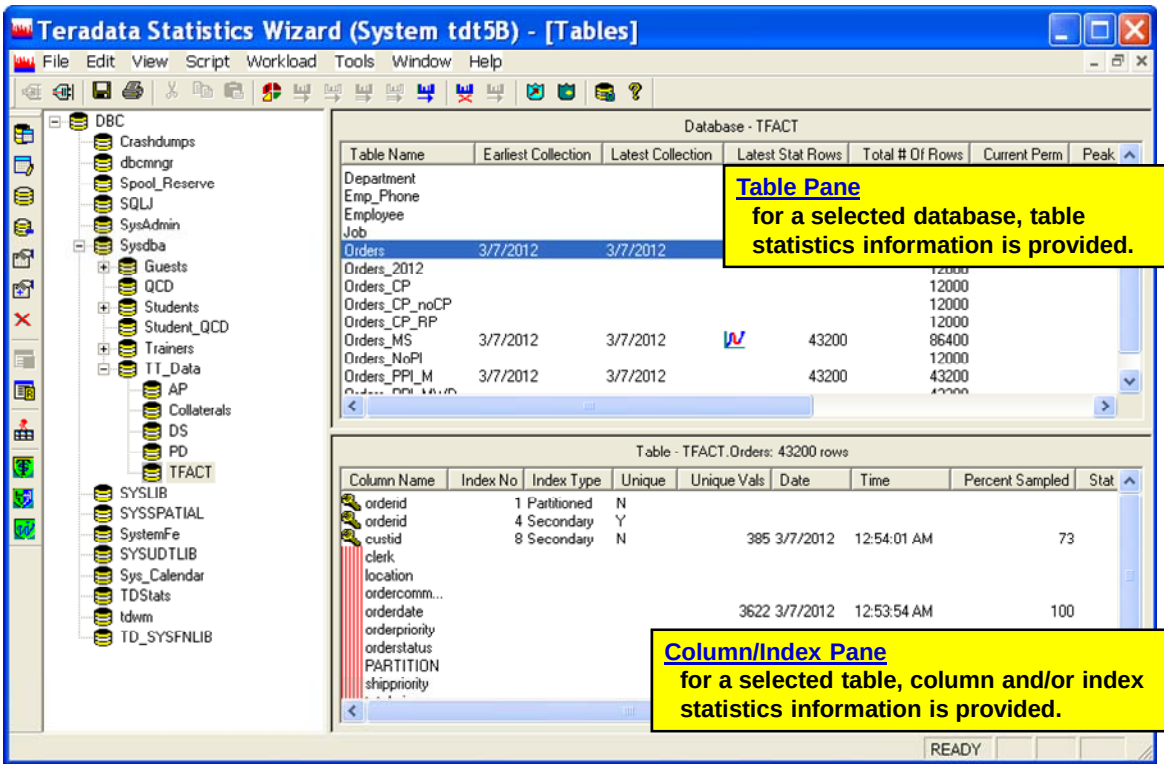
If you choose to use a workload to help provide statistics recommendations, you want representative samples of live SQL statements. A workload is defined as an SQL statement or a set of SQL statements. Typically, a workload is comprised of SQL statements that were executed on a production system.

The Teradata Statistics Wizard analyzes a workload and makes recommendations based on the defined workload. Ways to define a workload include:

- Select queries from the Teradata Database Query Log (DBQL).
- Open a file and directly enter SQL statements into a workload, or select the SQL statements from one or more files.
- Select an existing set of execution plans in a user defined Query Capture Database (QCD) to form a workload.
- Import SQL statements to the test system as a workload from a production system, using the Teradata System Emulation Tool. The production system environment is imported along with the SQL statements.
- Enter the SQL statement manually to add or delete statements, and then optionally save the modified workload under a different name, if needed.
- Once a workload is defined, the next step is to analyze the workload and recommend collection of statistics.

Validation involves collecting sampled statistics on the recommended columns and generating query plans, based on the collected statistics.

Teradata Statistics Wizard – Main Window



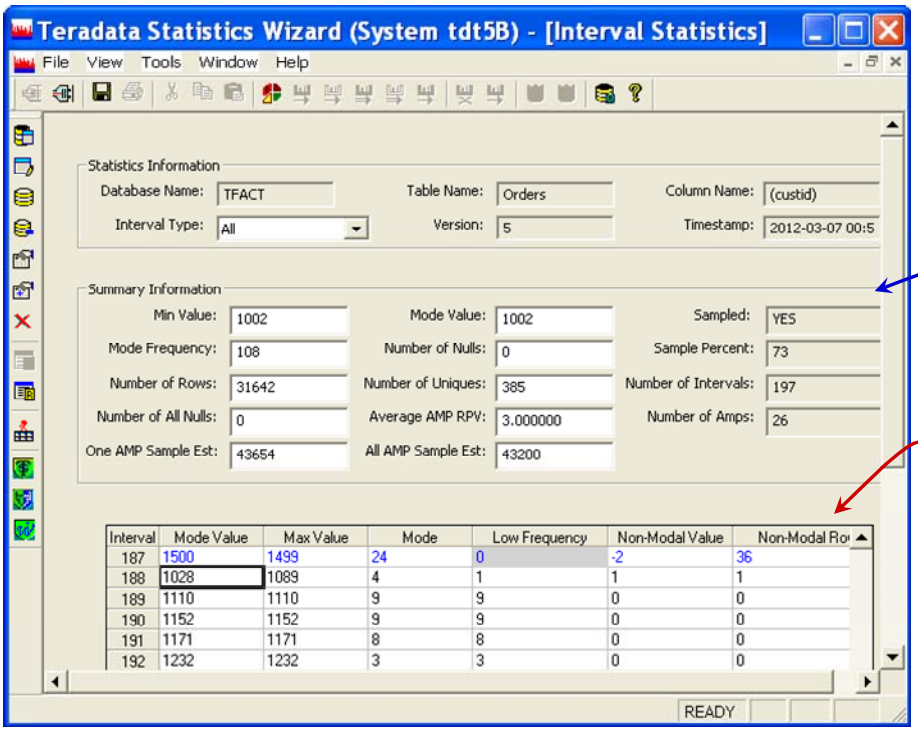
Teradata Statistics Wizard – Interval Statistics

The following chart describes the fields in the Interval Statistics.

Database	Name of database.
Table	Name of table.
Column	Name of column.
Index Number	Number of index, if it is an index.
Timestamp	Time when statistics were collected.
Numeric	Whether the column is numeric or non-numeric.
Number of Intervals	Number of intervals in the frequency distribution for the column/index, up to 200.
Sampled Size	The percentage of statistics that were sampled, if statistics were sampled.
Version	For internal use.
Note: The following information is about the column as a whole. This is also known as interval 0 data or the summary data.	
Number of Rows	Number of rows.
Number of Nulls	Number of empty rows for the entire column/index.
Number of Unique Values	Number of distinct Non-Mode values in the interval.
MinValue	Minimum value for the entire column/index. For example, if entering a date value, it must be in the form of MMDDYYYY.
Mode Value	The most frequently used (popular) value in the interval.
Mode Frequency	The number of rows having the Mode Value.
Note: The following values are displayed in the spreadsheet for all 200 intervals.	
Max Value	The highest value in the interval.
Mode Value	The most frequently used value in the interval.
Mode Frequency	The number of rows having the Mode Value.
Non-Modal Value	The number of distinct non-modal values (values that are not the most frequently used) in the interval. **Note:** If the Non-Modal Value is "-1", it means there is one loner in the interval. If the Non-Modal Value is "-2", it means there are 2 loners in the interval.
Non-Modal Rows	The total number of rows for all the Non-Modal values in the interval.

Note: It is not recommended, but you have the ability to modify interval data and submit it back to the Teradata Database.

Statistics Wizard – Interval Statistics



Double-click on a column or index that has statistics to display the interval details.

Summary Section (Interval #0)

Interval details

Note:
High bias intervals will have a "non-modal value" of -1 or -2 to represent 1 or 2 high bias values in the interval.

What are the two high bias custid's?

Collect, Re-Collect, or Drop Statistics

In addition to making recommendations, the Teradata Statistics Wizard allows you to collect, re-collect, or drop statistics on an arbitrary database, table(s), column(s) and /or index(es).

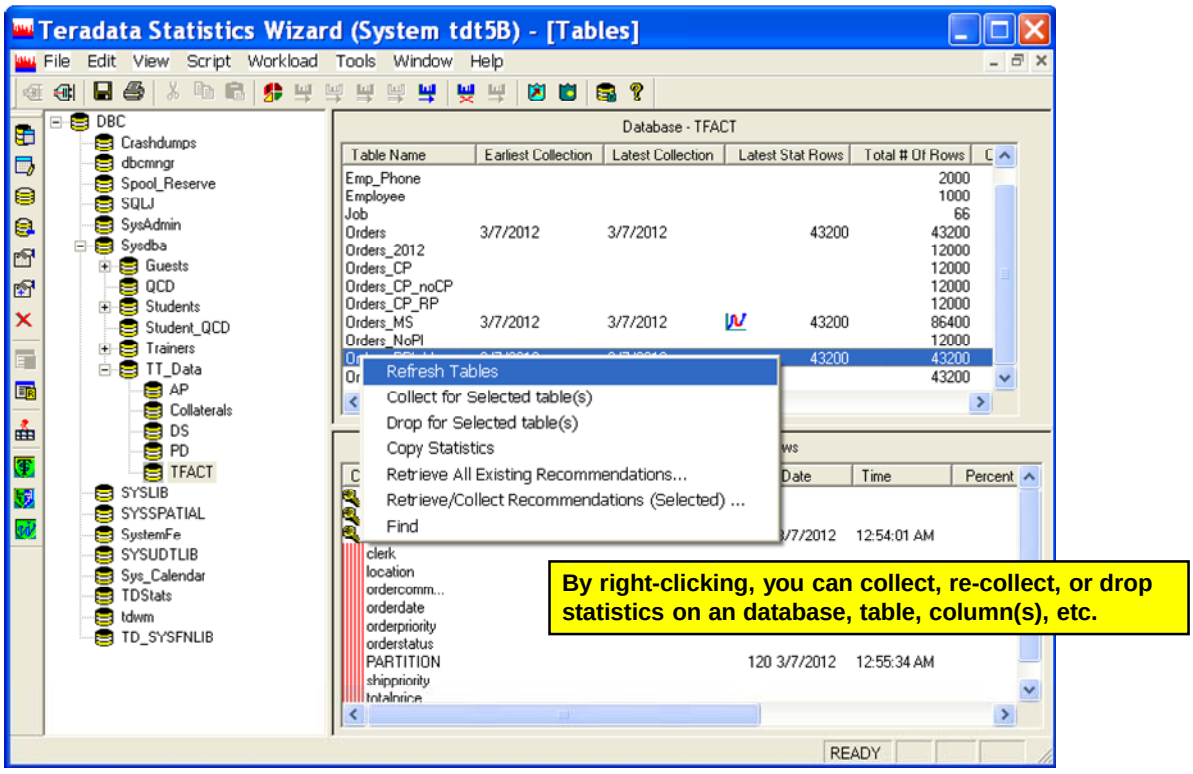
Note: When you right-click in a pane (database, table, or column/index) to select options, you need to click in the appropriate column to get the options dialog box.

For example, when you click in the Table pane, you need to right-click in the Table Name column. When you click in the Column pane, you need to right-click in the Column Name column. This will ensure that the correct Options menu is displayed.

- Right-click in the Database Name column of the Databases pane and select the Collect option to re-collect statistics for a database.
- Right-click in the Database Name column of the Databases pane and select the Drop option to drop statistics for a database.
- Select tables from the Table pane and right-click and select the Collect option to re-collect statistics for one or more tables.
- Right-click in the Column Name column of the Columns pane and select the Select Statistics for Table option to re-collect statistics for one or more tables.
- Select tables from the Table pane and right-click and select the Drop option to drop statistics for one or more tables.
- Right-click in the Column Name column of the Columns pane and select the Drop Statistics for Table option to drop statistics for one or more tables.
- Select one or more columns from the Table pane and right-click and select the Drop Statistics for Selected option to drop statistics for the selected columns in the Table pane.
- Right-click in the Table Name column of the Table pane and select the Drop Statistics for Table option to drop statistics for all columns within the table.

Note: When selecting to collect or drop statistics, statistics are only collected or dropped if the table has statistics defined.

Collect, Re-Collect, or Drop Statistics



Recommendations

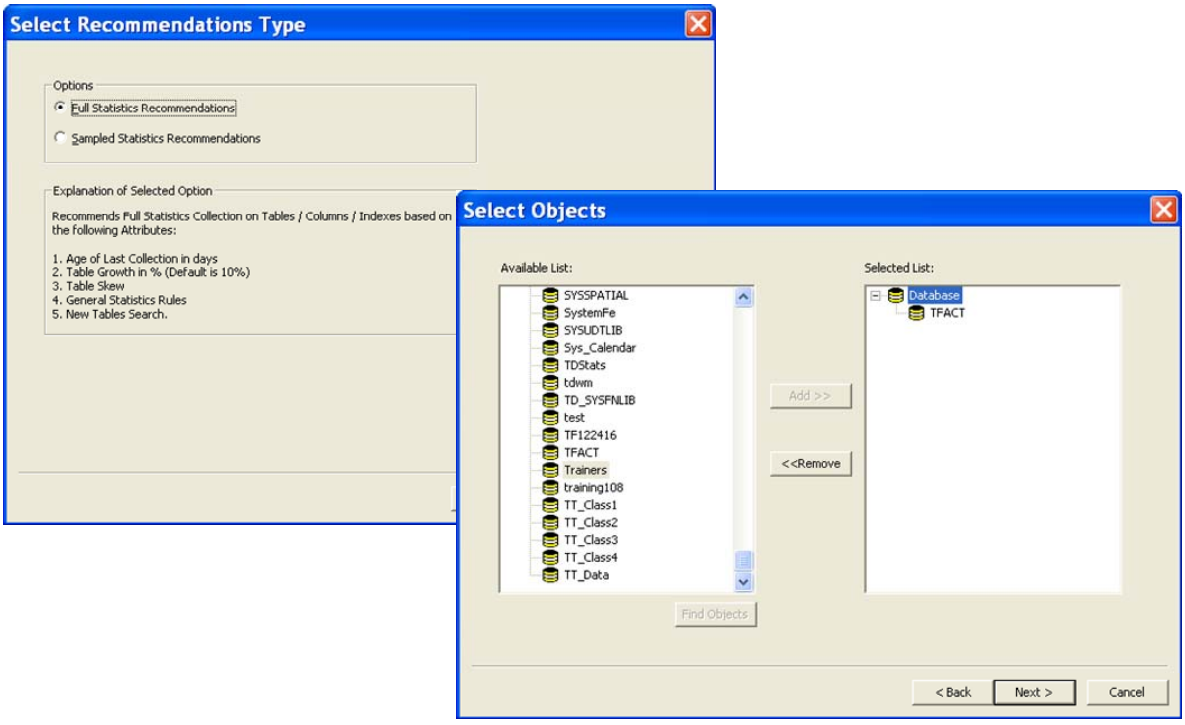
Statistics recommendations can be made when collecting or re-collecting statistics. The Statistics Wizard includes user options to:

- Make recommendations on which column/index would benefit from having statistics collected or re-collected, based on a table's demographics and some general heuristics.
- Retrieve the necessary types of data and apply to a table
- View, schedule, or execute recommendations.

The Teradata Statistics Wizard options are set within the Tools menu, Options. When you set these options, you specify data type and their thresholds, and then you selectively choose which recommendations to retrieve and their specified thresholds. Each type of recommendation is associated with an icon, which is later displayed in the table and column panes.

Recommendations

Tools Menu → Options → Recommendations



Recommendations (cont.)

Re-Collection Recommendations

Statistics re-collection recommendations is the process of making recommendations, based on which tables with existing statistics would benefit from having statistics re-collected. Recommendations are based on the age of collection and table growth.

- **Age of Collection** – recommends re-collection for all columns that have the number of days since statistics were last collected, if they exceed a user-configured threshold.

- **Table Growth** – recommends re-collection for all columns that have the change in row count since statistics were last collected, if they exceed a user-configured threshold.

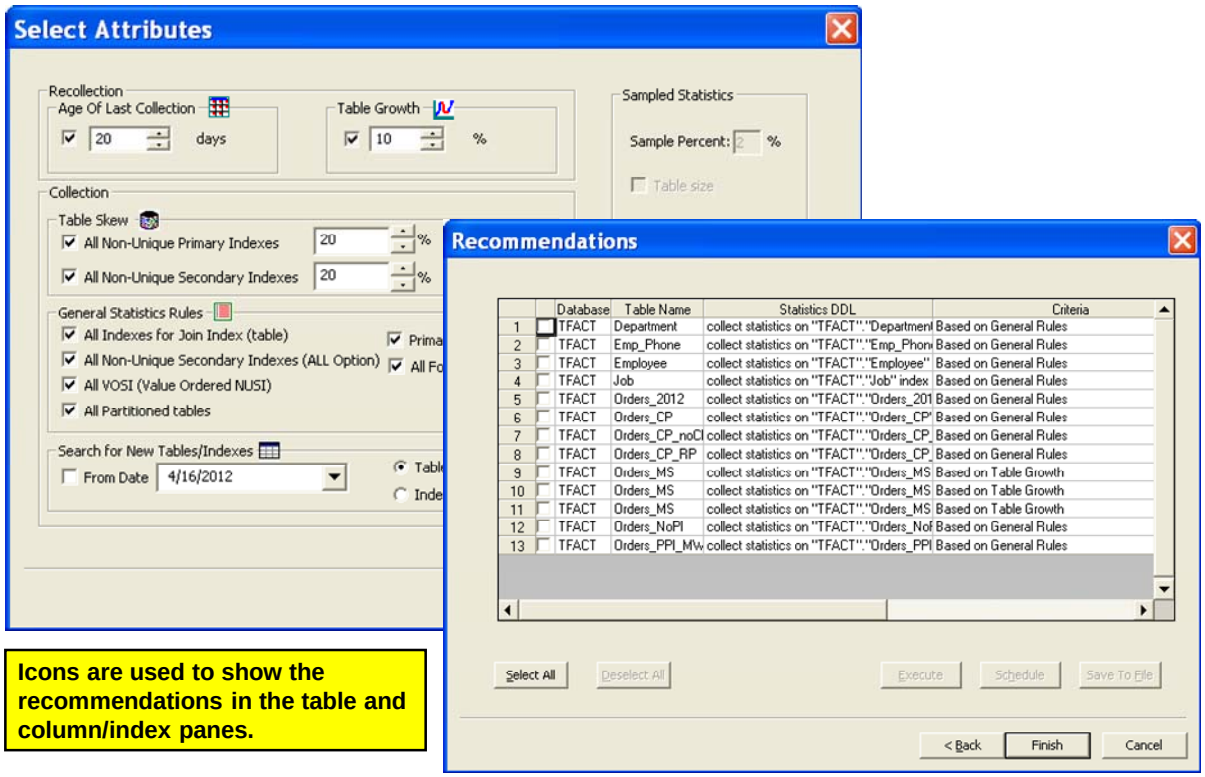
Collection Recommendations

Collection recommendations are based on the following options:

- **Table Skew** – recommends collection on all non-unique primary indexes for tables that have table skews that exceed a user-configured threshold.

- **General Heuristics** – recommends collection based on some rules of thumb, including:
  - All indexes for Join Index table
  - All non-unique secondary indexes with ALL option
  - All VOSI (Value ordered NUSI)
  - All Partitioned Tables

Recommendations (cont.)



Icons are used to show the recommendations in the table and column/index panes.

Statistics Summary

Remember to refresh statistics on a regular basis.

Dropping a secondary index for a single column deletes the index's definition from DBC.Indexes. However, the column definition and single-column statistics stored in DBC.TVFields (13.10 and before) or DBC.StatsTbl (starting with 14.0) will still exist.

Statistics Summary

- **Recommendations for collect statistics include:**
  - **All non-unique indexes.**
  - **The UPI of tables with less than 1,000 rows per AMP – collect full statistics.**
  - **Any non-indexed column used for set selection or join constraints.**
  - **Collect statistics on the key word PARTITION.**
- **Collected statistics are not automatically updated by the system.**
  - **The user is responsible for refreshing collected statistics.**
  - **Refresh statistics when 5% to 10% of the table's rows have changed.**
- **The optimizer is more aggressive when it has COLLECTed STATISTICS. It is more conservative when it must rely on random AMP sampling.**
- **The Teradata Statistics Wizard provides a graphical Windows interface to easily view, collect, recollect, or drop statistics on selected database, tables, columns, or indexes.**
- **Limitations**
  - **Maximum # of columns in a table on which you may implicitly recollect statistics is 512.**
  - **Maximum number of column names within a given COLUMN list is 64.**
  - **Maximum number of multiple column COLLECT STATS statements per table is 32.**

Module 25: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 25: Review Questions

1. When alternatives are available, the Optimizer may require “hints” to ensure that it will make the best choices.
   - T. True
   - F. False

2. If you COLLECT STATISTICS for a NUPI in Teradata 14.0, the statistics are stored in _____.
   - a. DBC.StatsTbl
   - b. DBC.Indexes
   - c. DBC.TVFields
   - d. Data Dictionary (DD) cache

3. Dynamic AMP sample statistics are stored in the _____.
   - a. DBC.TVFields
   - b. DBC.Indexes
   - c. Data Dictionary (DD) cache
   - d. None of the above

4. You can use the _____ to display information (e.g., last collection date) about current column or index statistics.
   - a. EXPLAIN statement
   - b. HELP INDEX statement
   - c. SHOW TABLE statement
   - d. COLLECT STATISTICS statement
   - e. HELP STATISTICS statement

Notes

Module 26

The EXPLAIN Facility

After completing this module, you will be able to:

- **Describe the Explain Facility.**
- **Define Explain terminology.**
- **Describe the EXPLAIN output of a CREATE TABLE statement.**
- **Match EXPLAIN terms to a definition.**
- **Identify the ways to EXPLAIN Macros.**
- **Use EXPLAIN to determine the number of partitions used in various access plans.**

Notes

Table of Contents

The EXPLAIN Facility 26-4
EXPLAIN Facility Output 26-6
Example 1 – EXPLAIN of a Simple SELECT 26-8
Example 2 – EXPLAIN of a SELECT (FTS) 26-10
EXPLAIN Terminology 26-12
" (Last Use) " 26-12
" with no residual conditions " 26-12
" END TRANSACTION " 26-12
" eliminating duplicate rows " 26-12
" by way of the sort key in spool field1 (dbname.tabname.colname)" 26-12
" we do an ABORT test " 26-12
" by way of a traversal of index #n extracting row ids only " 26-12
EXPLAIN Terminology (cont.) 26-14
" we do a SMS (set manipulation step) " 26-14
" we do a BMSMS (bit map set manipulation step) " 26-14
" which is redistributed by hash code to all AMPs " 26-14
" which is duplicated on all AMPs " 26-14
“(one_amp) or (group_amps)” 26-14
"NOT (table_name.column_name IS NULL)" 26-14
Pseudo Table Locks 26-16
An example: 26-16
Understanding Row and Time Estimates (Part 1) 26-18
Understanding Row and Time Estimates (Part 2) 26-20
Parallel Steps 26-22
Example 3 – EXPLAIN with Parallel Steps 26-24
Example 4 – EXPLAIN of a SELECT (BMSMS) 26-26
Example 5 – EXPLAIN of Create Table 26-28
EXPLAINing Macros 26-30
EXPLAIN Terminology for PPI Tables 26-32
Example 6 – Partition Elimination with a PPI Table 26-34
Example 7 – Primary Index Access of PPI Table 26-36
Create a USI or NUSI on the PI for a PPI Table 26-36
Example 8 – Dynamic Partition Elimination 26-38
Example 9 – CURRENT_DATE Improvements 26-40
EXPLAIN Summary 26-42
Module 26: Review Questions 26-44
Module 26: Review Questions (cont.) 26-46

The EXPLAIN Facility

The **EXPLAIN facility** is a Teradata extension that provides you with an "English" translation of the steps chosen by the Optimizer to execute a SQL statement. It may be used on any valid Teradata SQL statement simply by prefacing that statement with “EXPLAIN”.

The following is an example of how you would EXPLAIN a simple SELECT statement:

```
EXPLAIN SELECT last_name, first_name FROM employee;
```

The EXPLAIN facility actually parses the SQL statement but does not execute it. EXPLAIN output provides the physical designer with an “execution strategy”. This execution strategy provides direct feedback on what steps the Optimizer chooses to do, but not why it chooses to do them.

Studying EXPLAIN output can be an excellent way to learn more about the inner workings of the Teradata DBS and how it handles SQL.

The EXPLAIN facility should be used to analyze all Joins and complex queries. Using EXPLAIN regularly can save you lots of time and computing resources by pointing out problems with your SQL statements before you actually run them.

The Explain Facility

May be used on any SQL statement, except EXPLAIN itself.

Translates Optimizer output (the execution plan) into English.

- **Provides direct feedback on what the system intends to do.**

It is a good way to learn about the system and SQL.

Use it consistently to analyze joins, long-running and complex queries.

Time estimates are relative, not absolute.

- **Assumes the query will run stand-alone; doesn't take a loaded system into account.**
- **Time estimates cost formulas based on H/W configuration.**

EXPLAIN Facility Output

In addition to the execution strategy, the EXPLAIN facility provides you with measures of how much work will be performed and a cost estimate expressed as time. Spool size and timing figures are estimates and should be used for comparison purposes only.

For example, the spool size estimates are based on either Dynamic Sampling or COLLECTed STATISTICS. Use “timings” as a cost factor. Even if the timings were calibrated to the CPU and DSU, there is no way that the Parser could determine what other jobs would be running when the request was executed. The developer of the EXPLAIN facility assigned arbitrary units of time to various segments of File System code as a "cost" factor. The sum of these costs is reported to the user.

The primary way to help the Optimizer make the best choices and ensure the most accurate EXPLAIN output is to make sure to provide current STATISTICS.

In many cases, there is more than one way to code an SQL statement to get the desired results. They all should be coded and EXPLAINed to find out which version will perform best for your tables and system configuration.

Always do an EXPLAIN before submitting any Join or complex query.

EXPLAIN Facility Output

The timings and spool sizes shown are ESTIMATES ONLY.

- **Spool sizes are based on dynamic sampling or statistics.**
- **Use them as "figures of merit" for comparison purposes only.**

Know what the Request is supposed to do before EXPLAINing it.

In this module, we will …

1st	**View some text examples of EXPLAINs.**
2nd	**Discuss EXPLAIN terminology that may appear within EXPLAIN output.**
3rd	**Discuss PPI terminology and view some examples.**

The Visual Explain utility will be covered in the next module.

Example 1 – EXPLAIN of a Simple SELECT

The example on the facing page illustrates the output generated by EXPLAINing a simple SELECT operation using a WHERE clause and specifying a value for the primary index.

Final output is generally put into Spool 1.

The view used in the SELECT on the facing page is shown below.

```
REPLACE VIEW TFACT.Daily_Sales_v
     AS SELECT Item_id, Sales_date, Sales
FROM  TFACT.Daily_Sales;
```

The table used in the SELECT on the facing page is shown below.

```
CREATE TABLE  Daily_Sales
    (  item_id          INTEGER NOT NULL
      ,sales_date       DATE FORMAT 'yyyy-mm-dd'
      ,sales            DECIMAL (9,2) )
PRIMARY INDEX (item_id);
```

Example 1 – EXPLAIN of a Simple SELECT

QUERY

EXPLAIN SELECT * FROM Daily_Sales_v WHERE item_id = 5010;

EXPLANATION

12.0 EXPLAIN

---

1) First, we do a single-AMP RETRIEVE step from TFACT.Daily_Sales in view Daily_Sales_v by way of the primary index "TFACT.Daily_Sales in view Daily_Sales_v.Item_id = 5010" with no residual conditions into Spool 2 (one-amp), which is built locally on that AMP. The size of Spool 2 is estimated with high confidence to be 2,191 rows (72,303 bytes). The estimated time for this step is 0.01 seconds.

2) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 2 are sent back to the user as the result of statement 1. The total estimated time is 0.01 seconds.

Notes:

- **Statistics were collected on the Primary Index of this table.**
- **The number of bytes used in the spool file is a feature of Teradata 12.0.**
- **The view name is identified in the Explain plan is also a feature of Teradata 12.0.**

Example 2 – EXPLAIN of a SELECT (FTS)

The example on the facing page illustrates the output generated by EXPLAINing a simple SELECT operation that does not have a primary index value specified.

Final output is generally put into Spool 1.

The table used in the SELECT on the facing page is shown below.

```
CREATE TABLE  Daily_Sales
    ( item_id          INTEGER NOT NULL
      ,sales_date      DATE FORMAT 'yyyy-mm-dd'
      ,sales           DECIMAL (9,2) )
PRIMARY INDEX (item_id);
```

Example 2 – EXPLAIN of a SELECT (FTS)

QUERY

EXPLAIN SELECT * FROM daily_sales ORDER BY 1;

EXPLANATION

12.0 EXPLAIN

---

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.daily_sales.
2) Next, we lock TFACT.daily_sales for read.
3) We do an all-AMPs RETRIEVE step from TFACT.daily_sales by way of an all-rows scan with no residual conditions into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1 (TFACT.daily_sales.Item_id). The input table will not be cached in memory, but it is eligible for synchronized scanning. The result spool file will not be cached in memory. The size of Spool 1 is estimated with high confidence to be 76,685 rows (2,530,605 bytes). The estimated time for this step is 0.09 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.09 seconds.

Notes:

- **Statistics were collected on the Primary Index of this table.**
- **Spool file byte count and column name used as sort key are Teradata 12.0 Explain enhancements.**

EXPLAIN Terminology

In general, EXPLAIN text is clear and easy to understand. However, there are a few phrases and terms that you may need to be familiarized with. Here are some definitions that may prove helpful:

" (Last Use) "

You will find this phrase following a reference to a Spool file. It indicates that the Spool file is being used for the last time and will be DELETEd at the end of the step, thus releasing that Spool space.

" with no residual conditions "

Residual conditions are those conditions in the WHERE clause not used to locate a row(s), but to further qualify the rows. This phrase indicates there are no such conditions present.

With “residual conditions” indicates that there are remaining conditions to be applied. These conditions are maintained in cache memory.

" END TRANSACTION "

When the END TRANSACTION step is sent, transaction locks are released and changes are committed.

" eliminating duplicate rows "

This indicates that a DISTINCT operation is done to ensure that there are no duplicate rows.

" by way of the sort key in spool field1 (dbname.tabname.colname)"

Field1 is created to allow a tag sort. Teradata 12.0 includes the column name used in the sort.

" we do an ABORT test "

ABORT tests are caused by an ABORT or ROLLBACK statement. If the condition is found to be true, then the ABORT or ROLLBACK is performed.

" by way of a traversal of index #n extracting row ids only "

A spool file is built containing the Row IDs found in a secondary index (index #n)

EXPLAIN Terminology

Most EXPLAIN text is easy to understand. The following additional definitions may help:

- ***... (Last Use) ...***

  A spool file is no longer needed and will be released when this step completes.

- ***... with no residual conditions ...***

  All applicable conditions have been applied to the rows.

- ***... END TRANSACTION ...***

  Transaction locks are released, and changes are committed.

- ***... eliminating duplicate rows ...***

  Duplicate rows only exist in spool files, not set tables. Doing a DISTINCT operation.

- ***... by way of the sort key in spool field1 (dbname.tablename.colname) ...***

  Field1 is created to allow a tag sort. Teradata 12.0 includes the column name used for the sort.

- ***... we do an ABORT test ...***

  Caused by an ABORT or ROLLBACK statement.

- ***... by way of a traversal of index #n extracting row ids only ...***

  A spool file is built containing the Row IDs found in a secondary index (index #n).

EXPLAIN Terminology (cont.)

" we do a SMS (set manipulation step) "

The system will combine answer sets using a UNION, EXCEPT (MINUS) or INTERSECT operator.

" we do a BMSMS (bit map set manipulation step) "

NUSI Bit Mapping is being used.

" which is redistributed by hash code to all AMPs "

A step is done because data is being relocated to prepare for a join.

" which is duplicated on all AMPs "

A step is done because data is being duplicated on all AMPs to prepare for a join.

“(one_amp) or (group_amps)”

Indicates one AMP or a subset of AMPs will be used instead of all the AMPs.

"NOT (table_name.column_name IS NULL)"

Feature where optimizer realizes that a nullable column is being referenced in a comparison or join. Such conditions can not evaluate TRUE and are negated to avoid having to participate in the comparison.

EXPLAIN Terminology (cont.)

- *... we do a SMS (set manipulation step) ...*

  Combining rows using a UNION, MINUS, or INTERSECT operator.

- *... we do a BMSMS (bit map set manipulation step) ...*

  Doing a NUSI Bit Map operation.

- *... which is redistributed by hash code to all AMPs (dbname.tablename.colname) ...*

  Redistributing data (in SPOOL) in preparation for a join. Teradata 12.0 includes the column name.

- *... which is duplicated on all AMPs ...*

  Duplicating data (in SPOOL) from the smaller table in preparation for a join.

- *... (one_AMP) or (group_AMPs) ...*

  Indicates one AMP or a subset of AMPs will be used instead of all AMPs.

- *... ("NOT (table_name.column_name IS NULL)") ...*

  Feature where optimizer realizes that a nullable column is being referenced in a comparison or join. Such conditions can not evaluate TRUE and are negated to avoid having to participate in the comparison.

Pseudo Table Locks

Pseudo table locks reduce deadlock situations for all AMP requests.

When you use an all AMP request for a read, write, or exclusive lock, the system goes through pseudo table locking. With pseudo table locking:

- Each table has a table id hash code.
- Table id hash codes are assigned to the AMPs.
- Each AMP becomes a "gate keeper" to the tables for which it is assigned.
- All AMP requests for read, write, or exclusive locks go through the gatekeeper.

An example:

An all-AMP request comes from user1:

1. The PE sends a message to the gatekeeper AMP for the table.
2. The AMP places a pseudo lock on the table hash.
3. There is currently no lock on that table, so user1 gets the lock and may proceed with its all-AMP request.

Meanwhile, another all-AMP request comes from user2 for the same table.

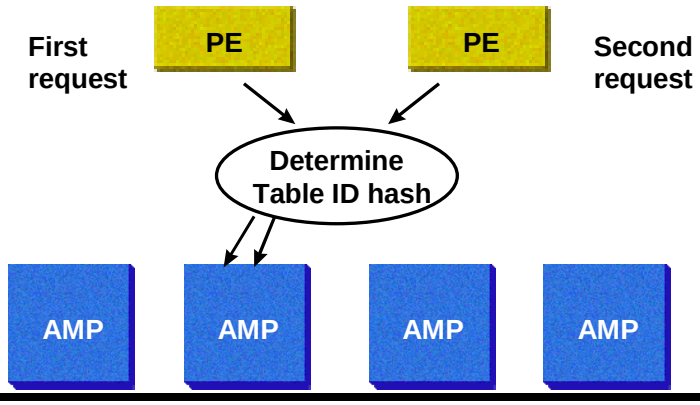
1. The PE sends a message to the gatekeeper AMP for the table.
2. The AMP places a pseudo lock on the table hash.
3. Since user1 already has a lock, user2 has to wait. Because user2 has a pseudo lock on the table, it is next in line.

In essence, the pseudo table lock enables sequential locking. Without pseudo-table locking, if two users send an all-AMP lock request, there could be a deadlock because the requests are sent in parallel and could arrive at the AMPs in a different order.

User1 gets a lock on AMP 3 while User2 gets a lock on AMP 4 first. When User1 tries to get a lock on AMP 4, deadlock would occur. By use of gatekeeper AMP, this cannot occur.

Pseudo Table Locks

- **Internal function to synchronize table-level locks across AMPs.**
  - Prevents two users from getting conflicting locks with all-AMP requests.
  - Effectively results in sequenced locking of a table – first request that requests a table-level lock will get the table-level lock.
- **All-AMP lock requests are handled as follows:**
  - PE determines Table ID hash for an AMP to manage the all-AMP lock request.
  - Place pseudo lock on the table.
  - Acquire lock on all AMPs.



Understanding Row and Time Estimates (Part 1)

The facing page identifies some of the “confidence” phrases and their meanings that you will find in the EXPLAIN output for a data retrieval.

Understanding Row and Time Estimates (Part 1)

The EXPLAIN facility may express "confidence" for a retrieve from a table. Some of the phrases used are:

. . . with high confidence . . .

- Restricting conditions exist on index(es) or column(s) that have collected statistics.

. . . with low confidence . . .

- Restricting conditions exist on index(es) or column(s) having no statistics, but estimates can be based upon a dynamic or random AMP sampling.
- Restricting conditions exist on index(es) or column(s) that have collected statistics but are *"AND-ed"* together with conditions on non-indexed columns.
- Restricting conditions exist on index(es) or column(s) that have collected statistics but are *"OR-ed"* together with other conditions.

. . . with no confidence . . .

- Conditions outside the above.

For a retrieve from a spool, the confidence is the same as the step generating the spool.

Understanding Row and Time Estimates (Part 2)

The facing page identifies some of the “confidence” phrases and their meanings that you will find in the EXPLAIN output for a join.

It is possible to get Index Join Confidence in the Explain output for both NUPI and UPI indexes and whether statistics have been collected or not. If the Explain output indicates that one set is joined to another set that has a Primary Index, then you may see Index Join Confidence.

Explain plans actually specify numeric values for low-end times, rows, and bytes in the EXPLAIN output.

Miscellaneous Notes:

- Estimates too large to display show 3 asterisks (***).
- High-end row and high-end time estimates have been removed starting with V2R5.
- The accuracy of the time estimate depends upon the accuracy of the row estimate.
- Actual performance can be hindered by current workload.

Understanding Row and Time Estimates (Part 2)

The following are "confidence" phrases for a join:

. . . with index join confidence . . .
- A join condition via a primary index.

. . . with high confidence . . .
- One input relation has high confidence and the other has high or index join confidence.

. . . with low confidence . . .
- One input relation has low confidence and the other has low, high, or join index confidence.

. . . with no confidence . . .
- One input relation has no confidence.
- Statistics do not exist for either join field.

Notes:

- Low and no confidence may indicate a need to collect statistics on indexes or columns involved in restricting conditions.
- You may otherwise consider a closer examination of the conditions in the query for possible changes that may improve the confidence.
- Explain plans show "low-end" time, rows, and/or bytes associated with the step.
  - Estimates too large to display show 3 asterisks (***).

Parallel Steps

Parallel Steps are multi-AMP processing steps that can be transmitted to the AMPs. These steps are numbered but execute asynchronously. All Parallel Steps must successfully complete before the next Serial Step is sent out.

In EXPLAIN output, you will see the text, “We execute the following steps in parallel.” Parallel steps will appear after this text.

Teradata will execute up to 20 steps in parallel if they are primary index (single-AMP). The explain may show more steps in parallel but dispatcher will only let 20 run at the same time (when one of those finishes it will start another one). This is 20 worker tasks out of the entire set of worker tasks (80 * #AMPs). If you have 1000 AMPs, which is only 20 out of 80,000 total AWTs.

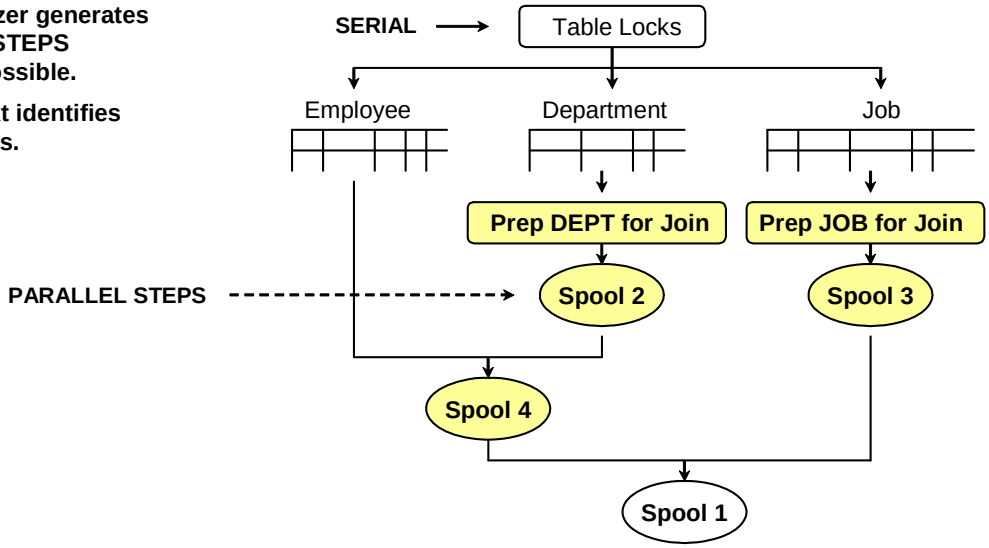
For all-AMP steps, the number of steps in parallel is more limited (the dispatcher doesn't want to have too many all-AMP steps executing at the same time for one query and have this query take over the system). Each all-AMP step takes 1 worker task per AMP (assume 80 AWTs per AMP). For example, with 1000 AMPs, this is $1/80^{th}$ of the total number of AWTs in the system. This is 1000 out of 80,000 and if dispatcher lets 3 steps run in parallel, that is 3000 out of 80000. We have to also assume there is other work going on the system that will need worker tasks. If there are 40 users on the system all trying to do 3 all-AMP steps in parallel, that is 120,000 (more that 80,000 worker tasks on the system). Therefore, some of these queries will wait for AMP worker tasks. In essence, the system is probably very busy.

Parallel Steps

PARALLEL STEPS are AMP steps that can execute concurrently:

- They have no functional overlap and do not contend for resources.
- They improve performance – the Optimizer generates PARALLEL STEPS whenever possible.
- EXPLAIN text identifies Parallel Steps.

```
SELECT      Last_Name, First_Name, Dept_Name, Job_Desc
FROM        Employee E
INNER JOIN  Department D
ON          E.Dept_Number = D.Dept_Number
INNER JOIN  Job J
ON          E.Job_code = J.Job_code
ORDER BY    3, 1, 2;
```



Example 3 – EXPLAIN with Parallel Steps

The example on the facing page illustrates the output generated by an EXPLAIN of a SELECT. This output illustrates parallel steps and relates to the previous example.

This example is based on a Teradata V2R6.1 implementation.

The tables used in the SELECT on the facing page are shown below.

```
CREATE SET TABLE TFACT.Employee
    ( Employee_Number     INTEGER NOT NULL
     ,Dept_Number         INTEGER
     ,Mgr_Emp_Number      INTEGER
     ,Job_Code            INTEGER
     ,Last_Name           CHAR(20)
     ,First_Name          VARCHAR(20)
     ,Salary_Amount       DECIMAL(10,2))
UNIQUE PRIMARY INDEX ( Employee_Number )
INDEX ( Dept_Number )
INDEX (Job_Code);

CREATE SET TABLE TFACT.Department
    (Dept_Number          INTEGER NOT NULL,
     Dept_Name            CHAR(20) NOT NULL,
     Dept_Mgr_Number      INTEGER,
     Budget_Amount        DECIMAL(10,2))
UNIQUE PRIMARY INDEX ( Dept_Number );

CREATE SET TABLE TFACT.Job
     (Job_Code            INTEGER NOT NULL,
      Job_Desc            CHAR(20) NOT NULL)
UNIQUE PRIMARY INDEX ( Job_Code );
```

Example 3 – EXPLAIN with Parallel Steps

```
QUERY    EXPLAIN  SELECT      Last_Name, First_Name, Dept_Name, Job_Desc
                  FROM        Employee E
                  INNER JOIN  Department D     ON  E.Dept_Number = D.Dept_Number
                  INNER JOIN  Job J            ON  E.Job_code = J.Job_code
                  ORDER BY    3, 1, 2;
```

EXPLANATION 12.0 EXPLAIN

: *(Locking steps)*

5) We execute the following steps in parallel.
   1) We do an all-AMPs RETRIEVE step from TFACT.D by way of an all-rows scan with no residual conditions into Spool 2 (all_amps), which is duplicated on all AMPs. The size of Spool 2 is estimated with high confidence to be 19,642 rows (726,754 bytes). The estimated time for this step is 0.02 seconds.
   2) We do an all-AMPs RETRIEVE step from TFACT.J by way of an all-rows scan with no residual conditions into Spool 3 (all_amps), which is duplicated on all AMPs. The size of Spool 3 is estimated with high confidence to be 12,166 rows (450,142 bytes). The estimated time for this step is 0.01 seconds.
6) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.E by way of an all-rows scan with a condition of ("NOT (TFACT.E.Job_Code IS NULL)"). Spool 2 and TFACT.E are joined using a single partition hash_ join, with a join condition of ("TFACT.E.Dept_Number = Dept_Number"). The result goes into Spool 4 (all_amps), which is built locally on the AMPs. The size of Spool 4 is estimated with low confidence to be 26,000 rows (1,690,000 bytes). The estimated time for this step is 0.04 seconds.
7) We do an all-AMPs JOIN step from Spool 3 (Last Use) by way of an all-rows scan, which is joined to Spool 4 (Last Use) by way of an all-rows scan. Spool 3 and Spool 4 are joined using a single partition hash join, with a join condition of ("Job_Code = Job_Code"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort

:

Example 4 – EXPLAIN of a SELECT (BMSMS)

The example on the facing page illustrates the output generated by EXPLAINing a SELECT operation that specifies values for two NUSIs. Statistics have been collected on both NUSIs.

This example is based on a Teradata 12.0 implementation.

The table used in the SELECT on the facing page is shown below.

```
CREATE SET TABLE TFACT.Employee
    ( Employee_Number     INTEGER NOT NULL
     ,Dept_Number         INTEGER
     ,Mgr_Emp_Number      INTEGER
     ,Job_Code            INTEGER
     ,Last_Name           CHAR(20)
     ,First_Name          VARCHAR(20)
     ,Salary_Amount       DECIMAL(10,2))
UNIQUE PRIMARY INDEX ( Employee_Number )
INDEX (Job_Code)
INDEX (Dept_Number);
```

Example 4 – EXPLAIN of a SELECT (BMSMS)

QUERY

```
EXPLAIN   SELECT    *
          FROM      Employee E
          WHERE     Job_Code = 3500
          AND       Dept_Number = 1310;
```

Note:
- Employee table has 26,000 rows.
- 80 Employees are in Dept #1310.
- 248 Employees have Job_Code 3500.
- Only 4 employees in Dept #1310 have the Job_Code of 3500.

EXPLANATION 12.0 EXPLAIN

---

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.E.
2) Next, we lock TFACT.E for read.
3) We do a BMSMS (bit map set manipulation) step that builds a bit map for TFACT.Employee by way of index # 4 "TFACT.E.Job_Code = 3500" which is placed in Spool 2. The estimated time for this step is 0.01 seconds.
4) We do an all-AMPs RETRIEVE step from TFACT.E by way of index # 8 TFACT.E.Dept_Number = 1310" and the bit map in Spool 2 (Last Use) with a residual condition of ("TFACT.E.Job_Code = 3500") into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with low confidence to be 60 rows (4620 bytes). The estimated time for this step is 0.02 seconds.
5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.03 seconds.

Note:

Statistics were collected on the NUSIs Job_Code and Dept_Number.

Example 5 – EXPLAIN of Create Table

The example on the facing page shows how Teradata goes about creating a new table. A new table named **Orders** is created in the TFACT database. The orders table consists of three columns:

- order_id (has a Unique Primary Index)
- order_date
- cust_id

Example 5 – EXPLAIN of a CREATE TABLE

```
QUERY          EXPLAIN   CREATE TABLE Orders
                               (order_id      INTEGER NOT NULL
                               ,order_date    DATE FORMAT 'yyyy-mm-dd'
                               ,cust_id       INTEGER)
EXPLANATION                UNIQUE PRIMARY INDEX (order_id);                12.0 EXPLAIN
```

---

1) First, we lock TFACT.Orders for exclusive use.
2) Next, we lock a distinct DBC."pseudo table" for write on a RowHash for deadlock prevention, we lock a distinct DBC."pseudo table" for write on a RowHash for deadlock prevention, we lock a distinct DBC."pseudo table" for read on a RowHash for deadlock prevention, and we lock a distinct DBC."pseudo table" for write on a RowHash for deadlock prevention.
3) We lock DBC.ArchiveLoggingObjsTbl for read on a RowHash, we lock DBC.TVM for write on a RowHash, we lock DBC.TVFields for write on a RowHash, we lock DBC.Indexes for write on a RowHash, we lock DBC.DBase for read on a RowHash, and we lock DBC.AccessRights for write on a RowHash.
4) We execute the following steps in parallel.
    1) We do a single-AMP ABORT test from DBC.ArchiveLoggingObjsTbl by way of the primary index.
    2) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.
    3) We do a single-AMP ABORT test from DBC.TVM by way of the unique primary index.
    4) We do an INSERT into DBC.TVFields (no lock required).
    : :
    7) We do an INSERT into DBC.Indexes (no lock required).
    8) We do an INSERT into DBC.TVM (no lock required).
    9) We INSERT default rights to DBC.AccessRights for TFACT.Orders.
5) We create the table header.
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> No rows are returned to the user as the result of statement 1.

EXPLAINing Macros

As the page on the right illustrates, there are two distinct ways to EXPLAIN a Macro:

- With hard-coded parameter values
- With “soft” parameter values

The first EXPLAIN statement specifies a parameter value of 100 which will be treated as a hard-coded literal value. In this case, the Optimizer may choose an execution plan, which may not accurately represent production. Do not test production Macros this way.

The second EXPLAIN statement illustrates how you should test production Macros. In this case, both a Request parcel and a DATA parcel will be created due to the “soft” parameter value. The Optimizer will analyze the STATISTICs of the entire table and the resulting execution plan will accurately represent the execution plan you can expect in production.

EXPLAINing Macros

```
CREATE MACRO test (par_1 INTEGER)
    AS
      (SELECT * FROM  table_1
       WHERE cola = :par_1 ;
      );
```

EXPLAIN EXEC test (100) ;

- This creates a Request parcel, but no DATA parcel.
- The parameter value is treated as a hard-coded literal value.
- The execution plan may not accurately represent production execution of a macro.
- Typically, do not EXPLAIN parameterized macros with hard-coded values.

EXPLAIN USING (x INTEGER) EXEC test (:x) ;

- This creates both a Request parcel and a DATA parcel.
- The Optimizer analyzes the entire table's statistics.
- The execution plan accurately represents production execution of a macro.
- Use "soft" parameter values to test parameterized (production) macros.

EXPLAIN Terminology for PPI Tables

When the phrase **"a single partition of" or "*n* partitions of"** is included in the output of an EXPLAIN, it means that partition elimination will occur.

If a query has a range constraint on the partitioning columns in a table with a range-partitioned primary index, an all-AMP row scan ...

- starts at the partition spanning the low end of the range and
- stops at the partition spanning the high end of the range.

Partition elimination can occur for SELECTs, UPDATE, and DELETEs.

- For a DELETE, Optimizer recognizes partitions for which all rows are being deleted and rows in such partitions are deleted without using the transient journal.
- Optimization only performed if DELETE is an implicit transaction or is the last statement in a transaction.
- Similar to DELETE ALL except for a partition.

"SORT to partition Spool m by rowkey"

- Indicates that the optimizer determined that a spool file is to be partitioned based on the same partitioning expression as a table to which the spool file is be joined.
  - That is, the spool is to be sorted by rowkey (partition and hash).

- Partitioning the spool file in this way allows for a faster join with the partitioned table.

"a rowkey-based"

- Indicates an equality join on the rowkey.
  - In this case, there are equality constraints on the partitioning columns and primary index columns.
  - This allows for a faster join since each non-eliminated partition needs to be joined with at most only one other partition.

- If the phrase is not given, the join is hash based.
  - That is, there are equality constraints on the primary index columns from which the hash is derived.
  - For a partitioned table, there is some additional overhead in processing the table in hash order.

- Note that with either method, the join conditions must still be validated.

Teradata Database V2R5.1 introduced the Dynamic Partition Elimination (DPE). DPE can be applied when there are join conditions (instead of single table constraints) on the partitioning column/columns. The partition list that DPE uses depends on the data. The list, called a dynamic partition list, is generated at runtime. This capability has been further enhanced in Teradata Database V2R6.0.

EXPLAIN Terminology for PPI tables

"a single partition of" or "*n* partitions of"

- Indicates that an AMP or AMPs only need to access a single partition or *n* partitions of a table – indicates partition elimination occurred.
- Partition elimination can occur for SELECTs, UPDATE, and DELETEs.
  - For a DELETE, Optimizer recognizes partitions in which all rows are deleted.
  - Rows in such partitions are deleted without using the transient journal.

"SORT to partition Spool m by rowkey"

- The spool is to be sorted by rowkey (partition and hash).
- Partitioning the spool file in this way allows for a faster join with the partitioned table.

"a rowkey-based"

- Indicates an equality join on the rowkey.
- In this case, there are equality constraints on the partitioning columns and primary index columns.

"enhanced by dynamic partition …"

- Indicates a join condition where dynamic partition elimination has been used.

Example 6 – Partition Elimination with a PPI Table

```
EXPLAIN   SELECT    *
          FROM      Claim_PPI
          WHERE     claimdate
          BETWEEN   DATE '2011-01-01' AND DATE '2011-01-31';
```

1) First, we lock a distinct DS."pseudo table" for read on a RowHash to prevent global deadlock for DS.Claim_PPI.
2) Next, we lock DS.Claim_PPI for read.
3) We do an **all-AMPs RETRIEVE step from a single partition** of DS.Claim_PPI with a condition of ("(DS.Claim_PPI.claimdate <= DATE '2011-01-31') AND (DS.Claim_PPI.claimdate >= DATE '2011-01-01')") into Spool 1 (group_amps), which is built locally on the AMPs. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with high confidence to be 21,100 rows (1,856,800 bytes). The estimated time for this step is 0.44 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.44 seconds.

The table named Claim_NPPI is similar to Claim_PPI except it does not have a Partitioned Primary Index, but does have “c_claimid” as a UPI.

```
EXPLAIN   SELECT    *
          FROM      Claim_NPPI
          WHERE     claimdate
          BETWEEN   DATE '2011-01-01' AND DATE '2011-01-31';
```

1) First, we lock a distinct DS."pseudo table" for read on a RowHash to prevent global deadlock for DS.Claim_NPPI.
2) Next, we lock DS.Claim_NPPI for read.
3) We do an **all-AMPs RETRIEVE step** from DS.Claim_NPPI **by way of an all-rows scan** with a condition of ("(DS.Claim_NPPI.claimdate <= DATE '2011-01-31') AND (DS.Claim_NPPI.claimdate >= DATE '2011-01-01')") into Spool 1 (group_amps), which is built locally on the AMPs. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with high confidence to be 21,100 rows (1,856,800 bytes). The estimated time for this step is 49.10 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 49.10 seconds.

Note: Statistics were collected on the claimid, custid, and claimdate of both tables. The Claim table has 1,200,000 rows.

Example 6 – Partition Elimination with a PPI Table

```
QUERY   EXPLAIN   SELECT     *
                  FROM       Claim_PPI
                  WHERE      claimdate
                  BETWEEN  DATE '2011-01-01' AND DATE '2011-01-31';
```

EXPLANATION 13.10 EXPLAIN

:

3) We do an all-AMPs RETRIEVE step from a single partition of DS.Claim_PPI with a condition of ("(DS.Claim_PPI.claimdate <= DATE '2011-01-31') AND (DS.Claim_PPI.claimdate >= DATE '2011-01-01')") into Spool 1 (group_amps), which is built locally on the AMPs. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with high confidence to be 21,100 rows (1,856,800) bytes. The estimated time for this step is 0.44 seconds.

4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

->The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.44 seconds.

Notes:

- The Claim table was partitioned on claimdate with monthly partitions.
- The EXPLAIN text (for this SQL statement) for a non-partitioned Claim table is shown on the facing page in the PDF file. The estimated time is 49.10 seconds.

Example 7 – Primary Index Access of PPI Table

The complete EXPLAIN example of accessing a PPI via the PI is shown on the facing page.

The table named Claim_NPPI is similar to Claim_PPI except it does **not** have a Partitioned Primary Index, but does have “claimid” as a UPI.

```
EXPLAIN   SELECT    *
          FROM      Claim_NPPI
          WHERE     claimid =  260221;
```

1) First, we do a **single-AMP RETRIEVE step** from DS.Claim_NPPI by way of the unique primary index "DS.Claim_NPPI.claimid = 260221" with no residual conditions. The estimated time for this step is 0.00 seconds.
-> The row is sent directly back to the user as the result of statement 1. **The total estimated time is 0.00 seconds.**

Create a USI or NUSI on the PI for a PPI Table

If the partitioning columns are not part of the Primary Index, the Primary Index cannot be unique (e.g., claimdate). To maintain uniqueness on the Primary Index, you can create a USI on the PI (e.g., Claim ID or claimid). Another option is to create a NUSI on the PI column or columns. Access to specific rows via a USI or NUSI will be faster than scanning multiple partitions on a single AMP.

Reasons for a creating a USI instead of a NUSI may include:

- Maintain uniqueness in the column via the USI.
- Establish the USI as a referenced parent in Referential Integrity

```
EXPLAIN   SELECT *
          FROM      Claim_PPI
          WHERE     claimid = 260221
          AND       claimdate = DATE '2008-01-11';
```

1) First, we do a single-AMP RETRIEVE step from DS.Claim_PPI by way of the primary index "DS.Claim_PPI.claimid = 260221, DS.Claim_PPI.claimdate = DATE '2008-01-11'" with a residual condition of ("(DS.Claim_PPI.claimdate = DATE '2008-01-11') AND (DS.Claim_PPI.claimid = 260221)") into Spool 1 (one-amp), which is built locally on that AMP. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with high confidence to be 1 row (88 bytes). The estimated time for this step is 0.00 seconds.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.00 seconds.

Example 7 – Primary Index Access of PPI Table

QUERY EXPLAIN SELECT * FROM Claim_PPI WHERE claimid = 260221;

EXPLANATION 13.10 EXPLAIN

1) First, we do a single-AMP RETRIEVE step from all partitions of DS.Claim_PPI by way of the primary index "DS.Claim_PPI.claimid = 260221" with a residual condition of ("DS.Claim_PPI.claimid = 260221") into Spool 1 (one-amp), which is built locally on that AMP. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with high confidence to be 1 row (88 bytes). The estimated time for this step is 0.09 seconds.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.09 seconds.

Note: If partitioning information isn't provided, all of the partitions have to checked for the Row Hash.

A USI or a NUSI can optionally be placed on the Primary Index of a PPI table. The following example shows the use of a USI.

QUERY EXPLAIN SELECT * FROM Claim_PPI WHERE claimid = 260221;

EXPLANATION (with a USI) 13.10 EXPLAIN

1) First, we do a two-AMP RETRIEVE step from DS.Claim_PPI by way of unique index # 4 "DS.Claim_PPI.claimid = 260221" with no residual conditions. The estimated time for this step is 0.00 seconds.
-> The row is sent directly back to the user as the result of statement 1. The total estimated time is 0.00 seconds.

Example 8 – Dynamic Partition Elimination

Starting with Teradata Database release V2R5.1, the concept of Dynamic Partition Elimination (DPE) was introduced. DPE can be applied when there are join conditions (instead of single table constraints) on the partitioning column/columns. The partition list that DPE uses depends on the data in the columns being joined. The list, called a dynamic partition list, is generated at runtime by the AMPs.

This feature (DPE) was enhanced in TD V2R6.0 and will continue to be enhanced in future releases. Collecting statistics on the column PARTITION (V2R6.1) provides additional information to the optimizer and the optimizer is more likely to consider and use DPE in query plans.

The format of the "Claim_Date" table used in this simple join is:

```
CREATE SET TABLE DS.Claim_Date
    ( claimdate        DATE  FORMAT 'YYYY-MM-DD')
PRIMARY INDEX ( claimdate );
```

The complete EXPLAIN text for the query on the facing page is included:

1) First, we lock a distinct DS."pseudo table" for read on a RowHash to prevent global deadlock for DS.Claim_Date.
2) Next, we lock a distinct DS."pseudo table" for read on a RowHash to prevent global deadlock for DS.Claim_PPI.
3) We lock DS.Claim_Date for read, and we lock DS.Claim_PPI for read.
4) We do an all-AMPs RETRIEVE step from DS.Claim_Date by way of an all-rows scan with a condition of ("NOT (DS.Claim_Date.claimdate IS NULL)") into Spool 2 (all_amps), which is duplicated on all AMPs. Then we do a SORT to partition by rowkey. The size of Spool 2 is estimated with high confidence to be 6 rows (102 bytes). The estimated time for this step is 0.01 seconds.
5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to all partitions of DS.Claim_PPI with no residual conditions. Spool 2 and DS.Claim_PPI are joined using a product join, with a join condition of ("DS.Claim_PPI.claimdate = claimdate") enhanced by dynamic partition elimination. The input table DS.Claim_PPI will not be cached in memory, but it is eligible for synchronized scanning. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with low confidence to be 195 rows (17,940 bytes). The estimated time for this step is 0.04 seconds.
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.05 seconds.

Example 8 – Dynamic Partition Elimination

What is Dynamic Partition Elimination (DPE)?

This feature is applied when there are join conditions on the partitioning column/columns. The partition list that DPE uses depends on the data. The dynamic partition list is generated dynamically at execution time by the AMPs.

```
QUERY  EXPLAIN  SELECT *  FROM Claim_PPI C, Claim_Date D
                WHERE     C.claimdate = D.claimdate;
```

EXPLANATION 13.10 EXPLAIN

:

4) We do an all-AMPs RETRIEVE step from DS.Claim_Date by way of an all-rows scan with a condition of ("NOT (DS.Claim_Date.claimdate IS NULL)") into Spool 2 (all_amps), which is duplicated on all AMPs. Then we do a SORT to partition by rowkey. The size of Spool 2 is estimated with high confidence to be 6 rows (102 bytes). The estimated time for this step is 0.01 seconds.

5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to all partitions of DS.Claim_PPI with no residual conditions. Spool 2 and DS.Claim_PPI are joined using a product join, with a join condition of ("DS.Claim_PPI.claimdate = claimdate") enhanced by dynamic partition elimination. The input table DS.Claim_PPI will not be cached in memory, but it is eligible for synchronized scanning. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with low confidence to be 195 rows (17,940 bytes). The estimated time for this step is 0.04 seconds.

:

Example 9 – CURRENT_DATE Improvements

Starting with Teradata 12.0, the Optimizer peeks at the USING values of a query and may generate a specific plan that is not cached, rather than generating a generic plan that is cached.

Peeking means looking at the USING values during query parsing and using those values when checking all potential optimizations, such as satisfiability, optimum single table access planning, partition elimination, and picking up join index(es). Peeking helps optimize a query based on its specific USING values.

Generic plans are cached, and reusing a cached plan saves parsing time, that is, the time the CPU takes to parse and generate a plan and send it to the Dispatcher. But reusing a cached plan may not provide the most efficient execution plan for all queries.

A *specific* plan is not cached because it cannot be reused for different USING values, although all other details such as the SQL text hash and the host character set, along with the estimated cost, parsing time, and run time are cached.

With respect to queries that use the built-in functions DATE and CURRENT_DATE, the Optimizer generates a specific plan and caches it. But if DATE or CURRENT_DATE changes, the Optimizer disregards the cached plan and generates a new one.

Example 9 – CURRENT_DATE Improvements

Teradata 12.0 Feature

The optimizer can resolve CURRENT_DATE in order to utilize partitioning.

Additionally, for queries that use the built-in functions DATE and CURRENT_DATE, the Optimizer generates a specific plan and caches it. But if DATE or CURRENT_DATE changes, the Optimizer disregards the cached plan and generates a new plan.

```
QUERY EXPLAIN SELECT * FROM Claim_PPI
              WHERE    claimdate = CURRENT_DATE;
```

EXPLANATION 13.10 EXPLAIN

```
  :  (Locking steps)
3) We do an all-AMPs RETRIEVE step from a single partition of DS.Claim_PPI with a condition of
   ("DS.Claim_PPI.claimdate = DATE '2011-01-30'") with a residual condition of
   ("DS.Claim_PPI.claimdate = DATE '2011-01-30'") into Spool 1 (group_amps), which is built locally
   on the AMPs.  The size of Spool 1 is estimated with high confidence to be 16 rows (1,408 bytes).
   The estimated time for this step is 0.02 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1.  The total estimated
   time is 0.02 seconds.
```

EXPLAIN Summary

EXPLAIN output can be of great value to you. Here are some suggestions on how you can make the most of the EXPLAIN facility:

- Make EXPLAIN output an integral part of all design review and formal system documentation.
- Use EXPLAIN results to expose inefficiencies in query structures.
- Rerun EXPLAINs to see if the strategy chosen by the Optimizer has changed.
- This is done because data demographics change overtime.
- Retain EXPLAIN listings to facilitate periodic index re-evaluation.
- Keep formal documentation of the query structure rationale.

EXPLAIN Summary

- Make EXPLAIN output an integral part of all design reviews and formal system documentation.
- EXPLAIN results can expose inefficiencies in query structures.
- Data Demographics change over time.
- Retain EXPLAIN listings to facilitate periodic index re-evaluation.
- Keep formal documentation of the query structure rationale.
- Know what the Request is supposed to do before EXPLAINing it.

Module 26: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 26: Review Questions

Fill in the blanks.

1. _____ steps are multi-AMP processing steps that are numbered but execute at the same time.
2. The primary way to help the Optimizer make the best choices and ensure the most accurate EXPLAIN output is to _____ _____ on appropriate indexes and columns.
3. An EXPLAIN plan will indicate "estimated with _____ confidence" when a value for an index is provided to retrieve the data and the index has collected statistics.
4. Name the two ways to EXPLAIN a Macro:
   a. _____
   b. _____

(Continued on next page.)

Module 26: Review Questions (cont.)

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 26: Review Questions (cont.)

Match each EXPLAIN term to a definition.

___ 1. (Last Use)
___ 2. END TRANSACTION
___ 3. eliminating duplicate rows
___ 4. by way of the sort key in spool field
___ 5. does SMS (set manipulation step)
___ 6. does BMSMS (bit map ...)
___ 7. redistributed by hash code to all AMPs
___ 8. "Pseudo Table"
___ 9. all rows scan
___ 10. "a single partition of" or "n partitions of"
___ 11. "a rowkey-based merge join"
___ 12. group_amps operation
___ 13. "SORT to partition Spool m by rowkey"
___ 14. which is duplicated on all AMPs

a. The spool is to be ordered by partition and hash.
b. Combines answer sets using a UNION, EXCEPT (MINUS) or INTERSECT operator.
c. Indicates transaction locks are released and changes are committed.
d. Internal function to synchronize table-level locks across AMPs.
e. Indicates data is being relocated in preparation for a join.
f. Indicates that NUSI Bit Mapping is being used.
g. Indicates a full table scan.
h. Indicates that a DISTINCT operation is done to ensure that there are no duplicate rows.
i. Indicates that the Spool file will be released at the end of the step.
j. Subset of AMPs will be used instead of all AMPs.
k. Indicates partition elimination will occur.
l. Field1 is created to allow a tag sort.
m. Indicates an equality join based on partition and hash.

Notes

Module 27

Visual Explain

After completing this module, you will be able to:

- **Use the Visual Explain utility.**
- **Specify how to select the base query in a compare.**
- **List the 3 types of QCD users that access rights are associated with.**

Notes

Table of Contents

Teradata Visual Explain 27-4
Visual Explain – Connect to Teradata 27-6
Setting up the Environment 27-8
Placing Plans into QCD 27-10
Creating a Plan using "Execute SQL" Option 27-12
Open Execution Plans 27-14
Open Execution Plans (cont.) 27-16
Open Execution Plans (cont.) 27-18
Visual Explain of a Merge Join 27-20
Visual Explain Options 27-22
Visual Explain – Comparing Multiple Plans 27-24
Visual Explain – Example of Comparing 2 Plans 27-26
Granting Access Rights on a QCD 27-28
Visual Explain Summary 27-30
Module 27: Review Questions 27-32
Lab Exercise 27-1 27-34
Lab Exercise 27-2 27-36

Teradata Visual Explain

The Teradata Visual Explain utility provides a visual depiction of the execution plan chosen by the Teradata Database Optimizer to access data. It does this by turning the output text of the EXPLAIN modifier into a series of easily readable icons.

Teradata Visual Explain makes query plan analysis easier by providing the ability to capture and graphically represent the steps of the plan and performs comparisons of two or more plans. The tool is intended for application developers, database administrators and database support personnel to better understand why the Teradata Optimizer chooses a particular plan for a given SQL query. All of the information required for query plan analysis such as database object definitions, data demographics and cost and cardinality estimates is available through the Teradata Visual Explain interface. The tool is very helpful in identifying the performance implications of data skew and bad or missing statistics.

Teradata Visual Explain can also capture query plans in an emulated database environment. This is helpful for comparing query plans for different configurations or row counts to proactively see the impact of system expansion or table growth for a particular query.

Teradata Visual Explain is especially useful in comparing the execution plans of similar queries. Using the compare feature allows you to easily resolve Optimizer related discrepancies.

Teradata Visual Explain reads the contents of the Query Capture Database (QCD) and turns it into a series of icons. In order to view an execution plan using Teradata Visual Explain, the execution plan information must first be captured into the QCD using the Query Capture Feature (QCF), which includes the “insert explain” and “dump explain” commands.

In summary,

- The Teradata Visual Explain utility provides the Windows GUI.
- Query Capture Database (QCD) is the database that holds the execution plans.
- Query Capture Feature (QCF) is the software that includes the INSERT EXPLAIN and DUMP EXPLAIN commands and places execution plans in the QCD database.

Teradata Visual Explain

What is "Teradata Visual Explain"?

- Windows utility that provides graphical EXPLAIN reports.
  - Turns the output text of the EXPLAIN modifier into a series of readable icons.
- Visually compare two (or more) query plans.
- Visually compare two (or more) steps.
- Utilizes a QCD (Query Capture Database).
  - QCD consists of a set of tables, views, and macros that support user query plans. QCD provides the foundation for Visual Explain.

Additional capabilities of Visual Explain

- Control Center options for manage QCDs, users, and access rights.
- Includes X versions of QCD views and macros for enhanced security.
- Integrated with other Teradata Analyst tools (e.g., Teradata Index Wizard, Teradata Statistics Wizard, and Teradata SET (System Emulation Tool).
- Numerous GUI features including Compressed Views.
- Visual Explain 14.0 provides an improved list of QCDs and Query Plans.

Note: Package name is VEComp – Visual Explain and Compare

Visual Explain – Connect to Teradata

An example of the Visual Explain base screen display is shown on the facing page.

Teradata Visual Explain provides menu options and GUI interfaces to:

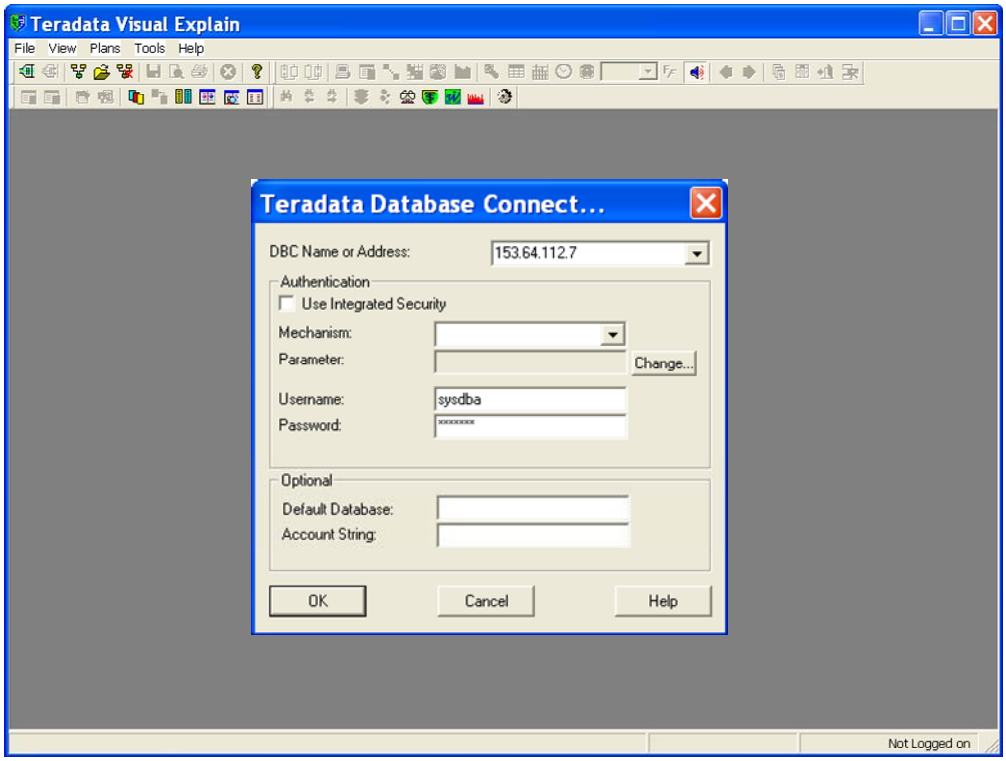
- Define and initialize a QCD database
- Execute an SQL Query and capture the query plan in the specified QCD
- Graphically depict the plan execution steps (including cost estimates and confidence levels)
- Visually compare two or more query plans
- Visually compare two or more steps
- Generate a comparison report for two or more plans

Single or multiple QCDs

The QCD database schema and Visual Explain tool are both designed to allow multiple users share a QCD. The QCD.Query table stores the user that captured a particular execution plan. X views limit access to plans based on this user information. Visual Explain has an interface to setup QCD privileges based on pre-defined user categories. Whether to implement a single or multiple QCDs really depends on how the customer wants to organize it. A customer may choose to create multiple QCDs, each for a different group of users to share. Another customer may choose to create one QCD and store all Visual Explain plans in that QCD for all users.

Visual Explain – Connect to Teradata

Click on the connection icon to connect to a data source and logon to a Teradata Database.



Setting up the Environment

To capture and visualize new query execution plans using Teradata Visual Explain, a QCD must be set up.

Setting up QCD

The easiest technique is to use the Control Center of the Visual Explain facility. This process will automatically create the macros, tables, and views needed in a QCD database.

1 Select Tools > Control Center from the menu bar.
2 Click the Manage QCD tab, then click the Setup QCD button.
3 Select the option Create all QCF database objects (tables and macros).
4 Enter a name for the QCD in the QCD Name field.
5 Enter an owner name in the Owner field. If this field is left blank, the owner defaults to the name of the logged on user.
6 Specify the Perm and Spool Space, selecting the appropriate units (KB, MB or GB) by clicking the desired option. (If no Perm Space is specified, the default is 1MB. If no Spool Space is specified, the default is 0.)
7 If you want the fallback option, check the Fallback check box.
8. If you want to view the schema of the Tables and Macros that will be created in the new QCD, click the View Schema button.

Sizing a QCD

The amount of permanent space required for a QCD depends primarily on the size of the query text and DDL of referenced objects. The Database Design document has a section in chapter 15 titled “Sizing a Query Capture Database” that discusses how to estimate the space requirement. The LIMIT clause of the INSERT EXPLAIN statement can be used to reduce the amount of text captured.

Upgrade the QCD

Teradata Visual Explain upgrades the definition of the QCD and then migrates any existing query plans to the new tables. This process also creates the views and macros required by Teradata Visual Explain to access the QCD data.

A small window is displayed to show the progress of the upgrade.

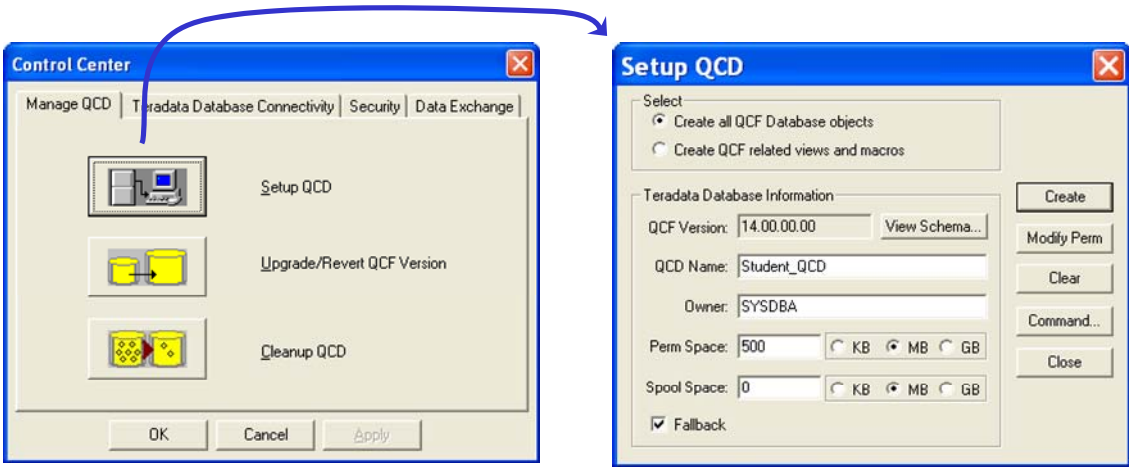
Select the **OK** button to close the Control Center window when the upgrade completes.

Setting up the Environment

The Visual Explain control center allows you to …

- **Manage QCDs – define/create a QCD, upgrade from previous version, or cleanup (delete plans, workloads, or objects in a QCD**
- **Specify the connectivity type (CLIv2 or ODBC) and/or define data sources**
- **Assign access rights to users on specific QCDs and/or create new users**
- **Data Exchange – import/export QCD workloads, plans, or objects to/from a data file**

Use the Tools > Control Center option.



Placing Plans into QCD

Options with the INSERT and DUMP EXPLAIN commands include:

- *QCD_Name* - an optional user-defined query capture database to be used instead of the default QCD database. The database named *QCD_name* need not exist on the target system; however, a database named *QCD_name* must exist on the test system on which the generated script is performed. Use the Control Center feature of the Visual Explain tool to create your QCD databases.

- *Query_Plan_Name* - an optional user-defined name (up to 30 characters) for which the query plan information is to be stored. If no *query_plan_name* is specified, the query plan information is stored with a null name. Because each query plan is stored with a unique non-null Query ID, there is no problem distinguishing among the various query plans within a given database. Note that Query IDs are *not* unique across databases. *query_plan_name* is required to be unique and you can store a name as **"query plan name"** if you enclose the name in quotation marks.

- *XML and NODDLTEXT –XML* captures the output as an XML document. This document is stored in the QCD table named *XMLQCD*. *NODDLTEXT* is used to not capture the DDL text in the XML document.

- *CHECK STATISTICS* - to capture COLLECT STATISTICS recommendations for *SQL_request* into the *StatsRecs* QCD table

What INSERT EXPLAIN Does

INSERT EXPLAIN performs the following actions in the order indicated.

1 Runs an EXPLAIN on the SQL DML statement specified by SQL_query.
2 Captures the Optimizer white tree output of that EXPLAIN.
3 Writes the output to the appropriate tables in a QCD database.

What DUMP EXPLAIN Does

The DUMP EXPLAIN statement is used to export a plan from one system so that it can be loaded into a QCD on another system. DUMP EXPLAIN performs the following actions in the order indicated.

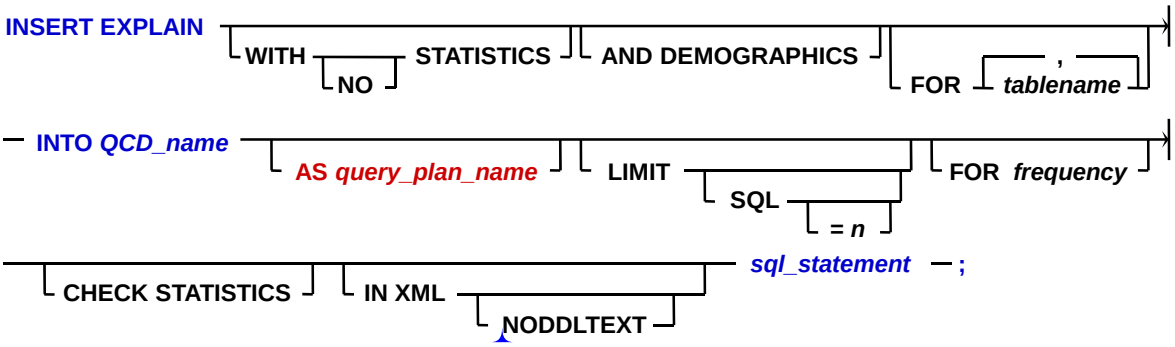
1 Runs an EXPLAIN on the SQL DML statement specified by *SQL_query*.
2 Captures the Optimizer plan output of that EXPLAIN.
3 Returns the output to the requestor as a series of INSERT statements designed to be used to update the appropriate tables in a QCD database.

You might want to use DUMP EXPLAIN rather than INSERT EXPLAIN if you are collecting information from several different machines and you want to ensure that only the selected QCD on the appropriate machine is updated with the results or if you do not want to update the QCD during heavy workload windows. In this case, you could submit the INSERT statements as part of a batch job during a less burdened workload window.

Placing Plans into QCD

There are multiple ways to place query plans into a QCD.

- **INSERT EXPLAIN – places an optimized plan into a QCD database.**
- **DUMP EXPLAIN – typically used to export a plan from one system so it can be loaded into a QCD on another system.**
- **Visual Explain GUI – Launch QCF – perform either function using a Visual Explain.**



Example:

```
INSERT EXPLAIN INTO Student_QCD AS qp1
   SELECT      E.Last_Name , E.First_Name , D.Dept_Name , J.Job_Desc
   FROM        PD.Employee E
   INNER JOIN  PD.Department D  ON D.Dept_Number = E.Dept_Number
   INNER JOIN  PD.Job J         ON E.Job_Code = J.Job_Code
   ORDER BY    3, 1, 2 ;
```

Creating a Plan using "Execute SQL" Option

An example of the Execute SQL screen display is shown on the facing page.

Previous versions of Visual Explain used an option named Launch QCF.

Additional Options that can be included with INSERT EXPLAIN

Some of the key options you can specify with INSERT EXPLAIN are:

- The information captured by the COLLECT STATISTICS (a.k.a., WITH STATISTICS as a clause with INSERT EXPLAIN) is only used by the Index Wizard during index analysis to make secondary index recommendations.

  This option is not needed to analyze plans with Visual Explain. The COLLECT STATISTICS option collects sampled statistics for columns found in a query's where clause. These columns are considered index candidates during the index analysis phase. If the COLLECT STATISTICS option is not used, the INSERT EXPLAIN statement automatically captures Interval 0 (Summary) statistics from the data dictionary. This information can be viewed when the plan is loaded into Visual Explain. If the customer is only going to analyze plans with Visual Explain, then this option is usually not needed.

- The AND DEMOGRAPHICS option captures row count and average row size information per AMP at the subtable level for all of the tables referenced in the plan. This information is not used during index analysis. This information is not automatically captured and is usually not needed. If captured, this information is captured in the QCD.DataDemographics table. The information can be displayed via Visual Explain's View->Show Demographics ... menu option.

- The FREQUENCY value is used to specify the number of times an SQL statement is typically performed within its identified workload. This value is used to weight the respective benefits of each column analyzed for inclusion in the index recommendation computed by Teradata Index Wizard. Any positive integer up to 4 Billion is valid. If this clause is not specified, then the frequency defaults to 1.

- The Limit Text option allows you to place a limit on the size of the query and DDL text captured in the QCD.

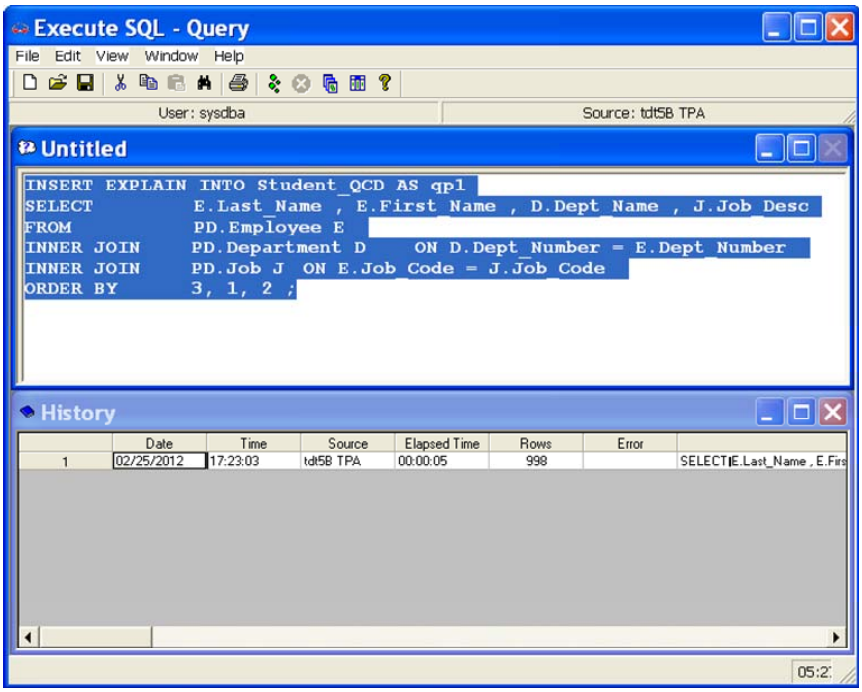
After entering the SQL, you are now ready to capture the execution plan.

Creating a Plan using "Execute SQL" Option

You can use Visual Explain to directly execute SQL.

Use the Tools > Execute SQL option.

Note: This option (Execute SQL) replaces the Launch QCF option found in previous versions of Visual Explain.



The Tools > Options > General tab has an option that can be checked to "use SQL Assistant instead of the Execute SQL window".

Open Execution Plans

The following steps are used to load one or more execution plans.

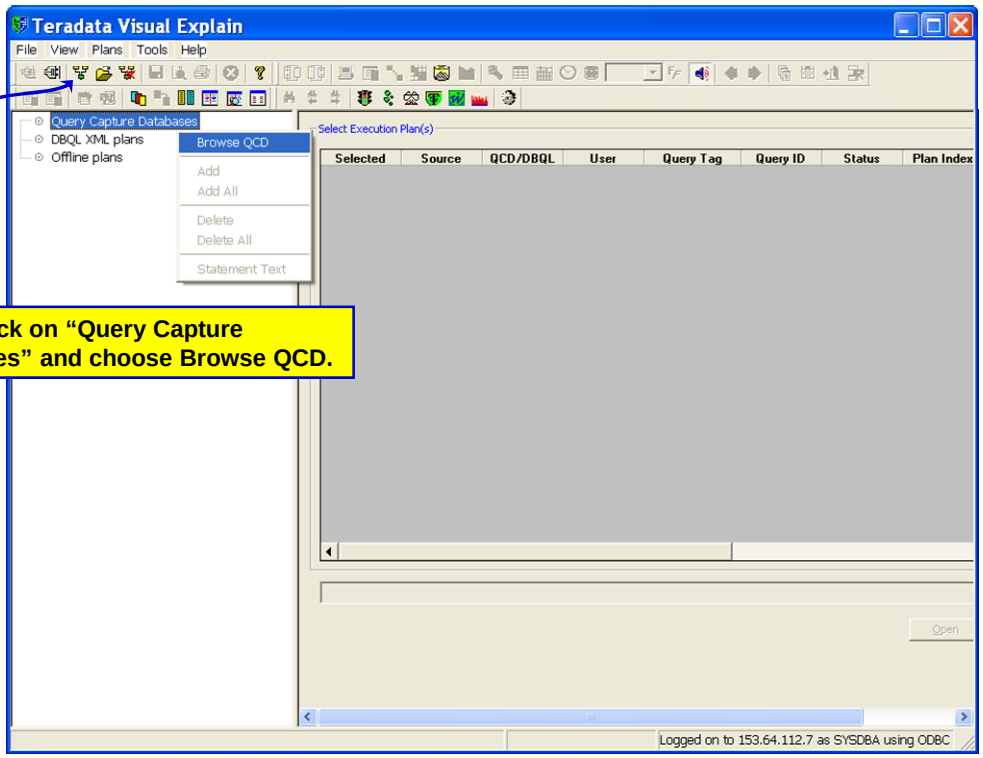
Once you are logged on to the Teradata Database, select File > Open Plan from Database from the menu bar. This displays the Open Plan dialog box.

1. **Right click on “Query Capture Databases” and choose Browse QCD.**

Open Execution Plans

Use the File > Open Plan from Database option.

or use the icon.



1. Right click on "Query Capture Databases" and choose Browse QCD.

Open Execution Plans (cont.)

The following steps are used to load one or more execution plans.

2. **Right-click on a QCD database and choose "Browse Plans".**

   Do one of the following to select the desired queries from the QCD database:

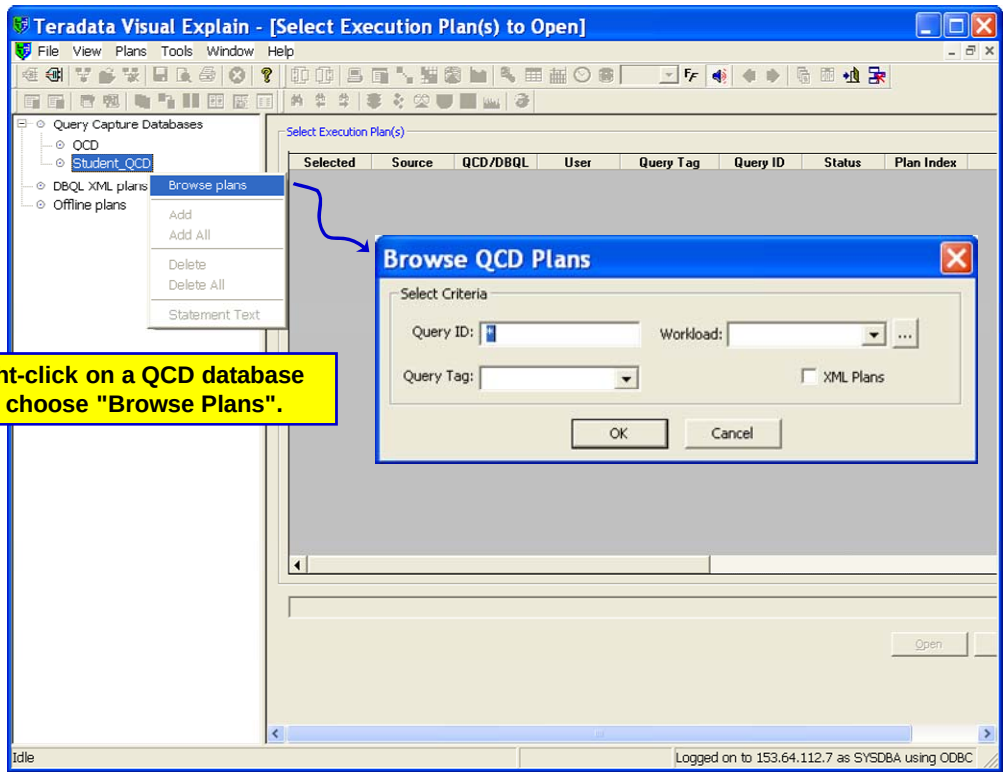
   – To load information for all queries from the specified QCD database, leave Query Tag and Query ID blank, and select Browse QCD.

   – To load queries with a specified plan name, enter the name in the query tag box.

   – To load information for only selected queries or a range of queries, select Query ID Range. Use commas to separate individual queries, and a dash to specify a range of queries. (For example, to load queries 1, 3, 4, 5, 6, and 9, enter 1, 3-6, 9 in the edit box.)

Open Execution Plans

A list of QCD Databases is provided in the left pane.



2. Right-click on a QCD database and choose "Browse Plans".

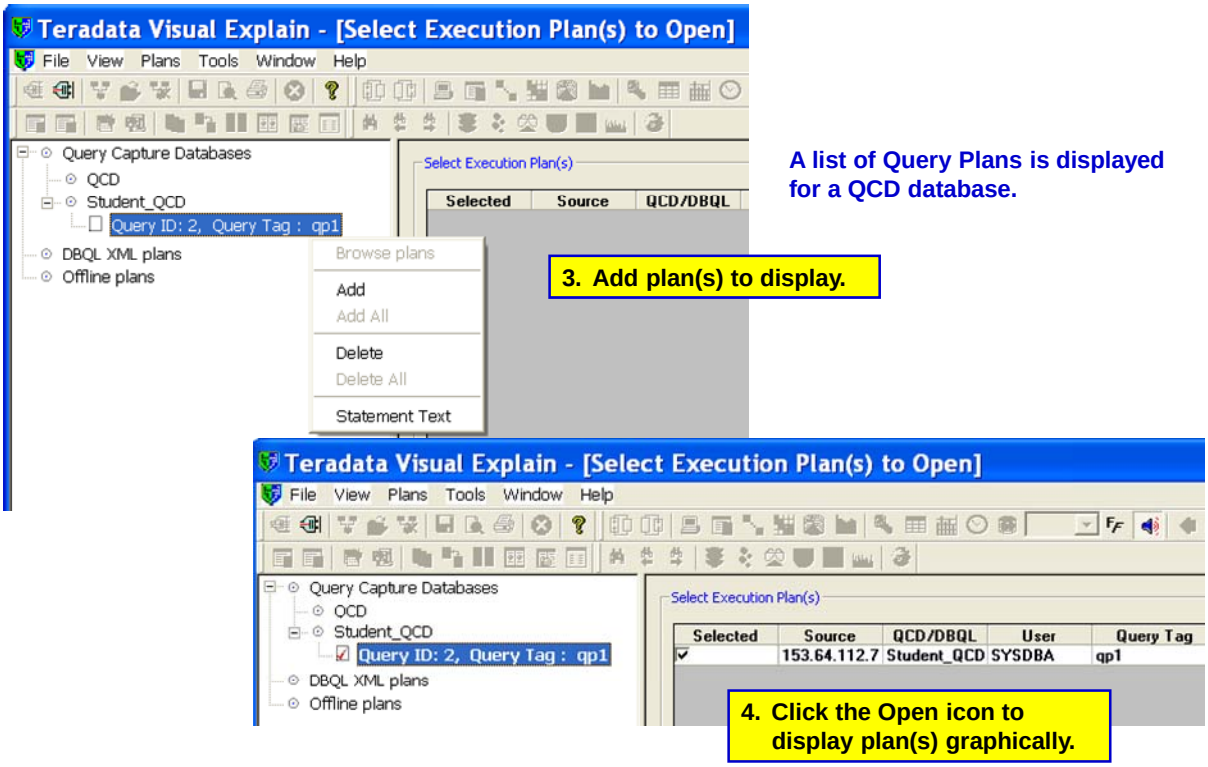
Open Execution Plans (cont.)

The following steps are used to load one or more execution plans.

3. **Add plan(s) to the Selected Execution Plans list.**

4. **Click the Open icon to display the plan(s) graphically.**

Open Execution Plans



Visual Explain of a Merge Join

The Visual EXPLAIN example on the facing page is one where the optimizer joins three tables together via a merge join based on the SQL example shown earlier.

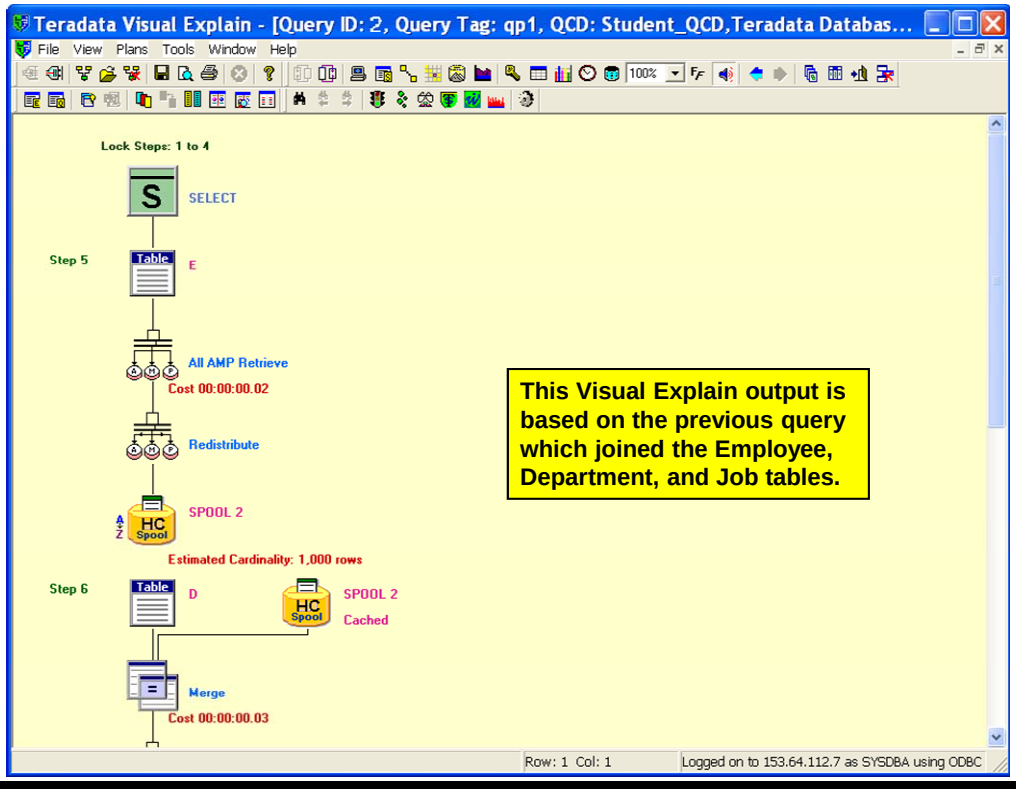
The full EXPLAIN text (Teradata 14.0) is:

1) First, we **lock** a distinct PD."pseudo table" for **read** on a **RowHash** to prevent global deadlock for **PD.J**.
2) Next, we **lock** a distinct PD."pseudo table" for **read** on a **RowHash** to prevent global deadlock for **PD.E**.
3) We **lock** a distinct PD."pseudo table" for **read** on a **RowHash** to prevent global deadlock for **PD.D**.
4) We **lock PD.J** for **read**, we **lock PD.E** for **read**, and we **lock PD.D** for **read**.
5) We do an all-AMPs **RETRIEVE** step from **PD.E** by way of an all-rows scan with a condition of ("NOT (PD.E.Job_Code IS NULL)") into **Spool 2** (all_amps), which is **redistributed** by the hash code of (**PD.E.Dept_Number**) to all AMPs. Then we do a SORT to order **Spool 2** by row hash. The size of **Spool 2** is estimated with high confidence to be 1,000 rows (49,000 bytes). The estimated time for this step is 0.02 seconds.
6) We do an all-AMPs **JOIN** step from **PD.D** by way of a **RowHash match** scan with no residual conditions, which is joined to **Spool 2** (Last Use) by way of a **RowHash match** scan. **PD.D** and **Spool 2** are joined using a **merge join**, with a join condition of ("PD.D.Dept_Number = Dept_Number"). The result goes into **Spool 3** (all_amps), which is **redistributed** by the hash code of (**PD.E.Job_Code**) to all AMPs. Then we do a SORT to order **Spool 3** by row hash. The size of Spool 3 is estimated with low confidence to be 984 rows (63,960 bytes). The estimated time for this step is 0.03 seconds.
7) We do an all-AMPs **JOIN** step from **PD.J** by way of a **RowHash match** scan with no residual conditions, which is joined to **Spool 3** (Last Use) by way of a **RowHash match** scan. **PD.J** and **Spool 3** are joined using a **merge join**, with a join condition of ("Job_Code = PD.J.Job_Code"). The result goes into **Spool 1** (group_amps), which is **built locally** on the AMPs. Then we do a SORT to order **Spool 1** by the sort key in spool field1 (**PD.D.Dept_Name**, **PD.E.Last_Name**, **PD.E.First_Name**). The size of **Spool 1** is estimated with index join confidence to be 984 rows (144,648 bytes). The estimated time for this step is 0.05 seconds.
8) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of **Spool 1** are sent back to the user as the result of statement 1. The total estimated time is 0.10 seconds.

Note: The HC icon on the facing page represents “High Confidence”.

Visual Explain of a Merge Join



Visual Explain Options

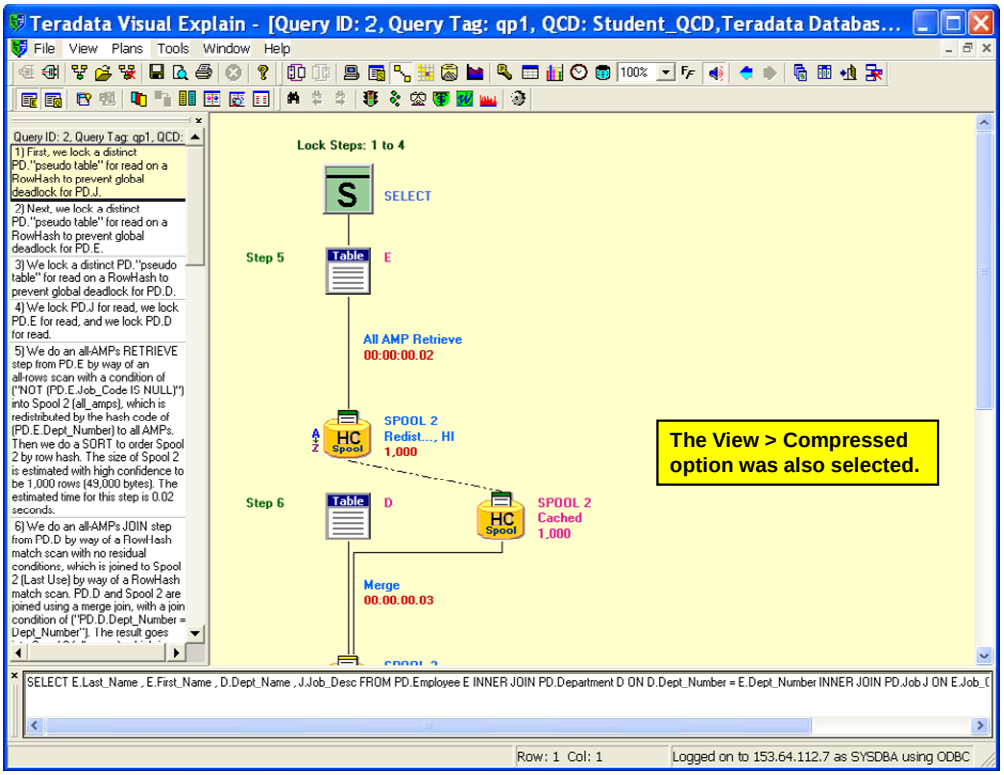
The Visual EXPLAIN example on the facing page illustrates displaying the query statement text and the actual Explain text along with the graphical explain.

Visual Explain Options

These options also display the Explain text and the query.

Plans > Explain Text

Plans > Statement Text



Visual Explain – Comparing Multiple Plans

To visually compare two execution plans:

1. Use the Plans > Compare option to load the plans to be compared.

2. Select at least 2 plans to be compared.

3. Double click on the query to be selected as the base query. The base query will appear in the green base query box.

4. Click on Compare to create the comparison.

5. Differences are displayed by a red arrow next to the appropriate steps. Move the mouse pointer over this red arrow for Tool Tip text explaining the differences.

Display textual difference information

To generate a text report of the differences in multiple execution plans,

1. Create a visual comparison between the execution plans you want to compare

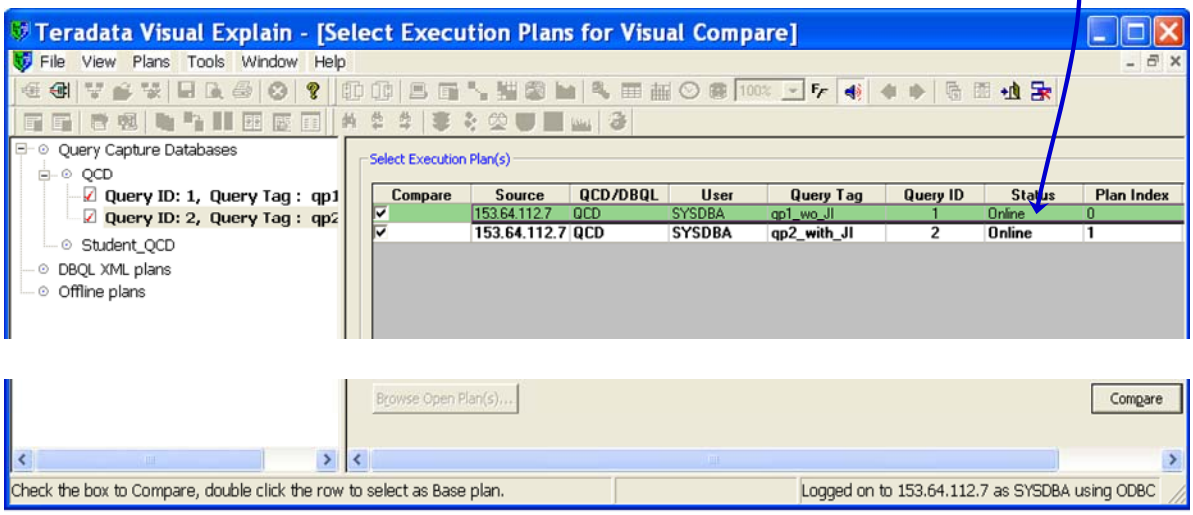
2. Select the type of report information you want to view using the pull-down selector on the Toolbar.

3. Select View > Visual Compare > Textual Output from the menu bar to display the report.

Visual Explain – Comparing Multiple Plans

To visually compare multiple execution plans:

1. **Use the Plans > Compare option.**
2. **Select at least 2 plans to be compared.**
3. **Select one query to be used as the base query by double-clicking on it.**
4. **Click on Compare button (bottom of screen) to create the comparison.**
5. **Differences are displayed by a red arrow next to the appropriate steps.**



Visual Explain – Example of Comparing 2 Plans

The query text is:

```
SELECT          E.Last_Name, E.First_Name,
                D.Dept_Name, J.Job_Desc
FROM            Employee E
INNER JOIN      Department D   ON E.Dept_Number = D.Dept_Number
INNER JOIN      Job J            ON E.job_code = J.job_code
ORDER BY  3, 1, 2;
```

The facing page contains an example of 2 join plans – one without a join index and one with a join index.

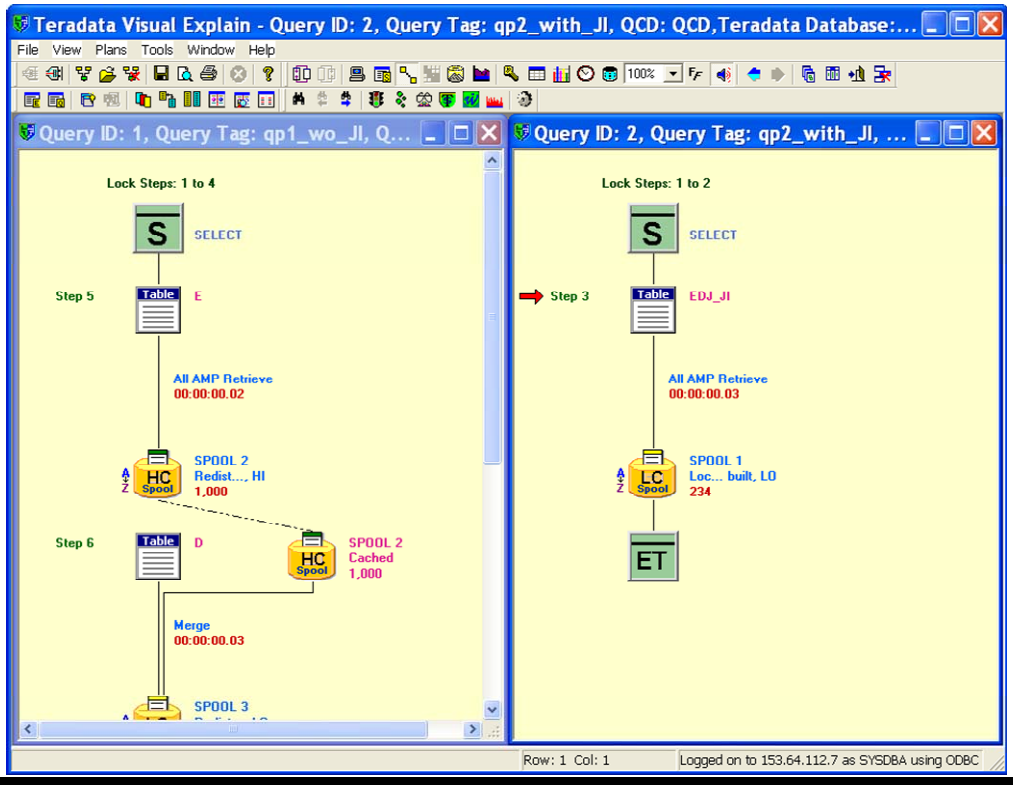
The query is the same except the tables have a Join Index.

Differences are displayed by a red arrow shown next to the appropriate steps. Move the mouse pointer over this red arrow for Tool Tip text explaining the differences.

The SQL to create the join index is:

```
CREATE JOIN INDEX EDJ_Join_Idx, FALLBACK
AS
SELECT          D.Dept_Number, D.Dept_Name,
                E.Employee_Number, E.Last_Name, E.First_Name,
                J.Job_code, J.Job_Desc
FROM            Department D
INNER JOIN      Employee E   ON D.Dept_Number = E.Dept_Number
INNER JOIN      TFACT.Job J ON E.Job_Code = J.Job_Code;
```

Visual Explain – Example of Comparing 2 Plans



Granting Access Rights on a QCD

The Control Center > Security tab can be used to easily grant/revoke access rights to users for a specific QCD.

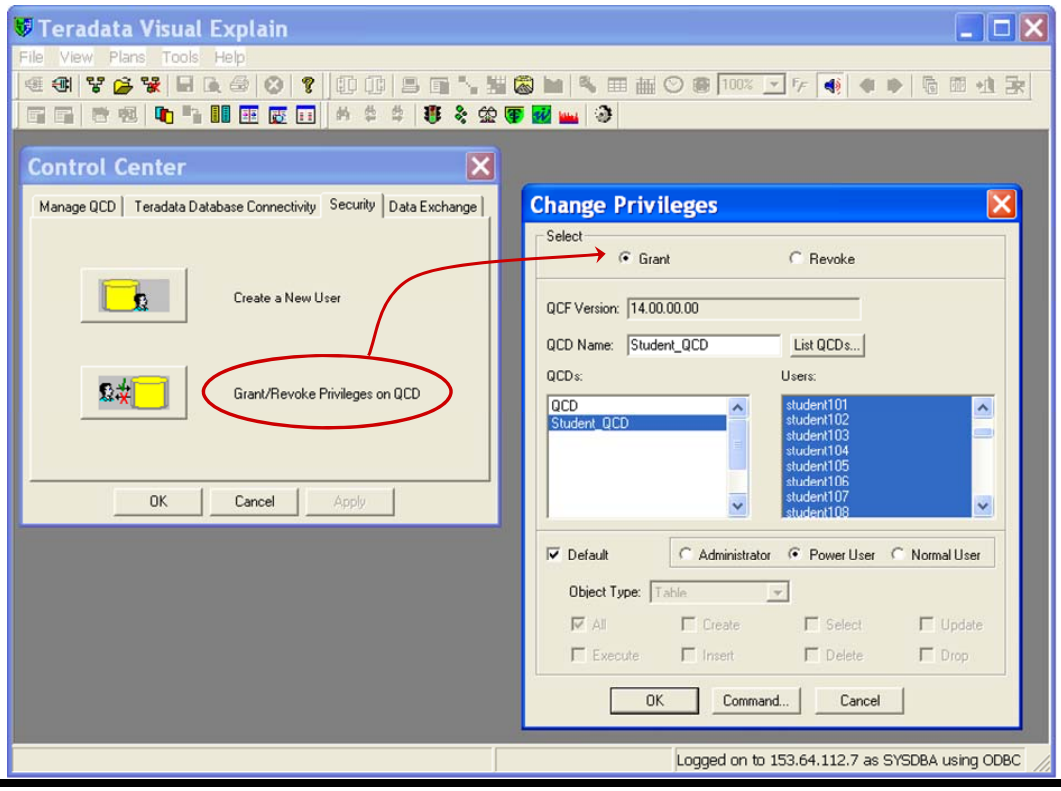
A QCD user can be designated as one of the following:

- Normal User – load, view, or delete user's own plans or workloads only. The Use X-views for QCD information check box must be selected in the Options dialog box for this option to be effective.

- Power User – load and view plans or workloads inserted by any user. Delete user's own plans or workloads only.

- Administrator – load, view, or delete any plan created by any user. The administrator can drop or delete QCD tables. By default, the QCD creator has Administrator privileges.

Granting Access Rights on a QCD



Visual Explain Summary

Teradata Visual Explain has numerous enhancements and features. A partial list is show on the facing page. A description of some of these features follows.

The **Compressed** view essentially displays the plan in a traditional data flow manner without the action icons. The retrieval, redistribution and join information is still displayed in the window, but in an abbreviated format. This feature is ideal for viewing very large plans.

Teradata Visual Explain has been enhanced to display information on data demographics that will be useful for query plan analysis. This information can be displayed in both graphical and report formats.

The Teradata Visual Explain interface also has the capability to display both current and captured index information, object definitions and statistics. This feature is useful for identifying any changes that have occurred since the plan was last captured.

Bulk Compare - this feature allows you to perform multiple plan comparisons with a single operation.

Visual Explain Summary

Summary of features covered in this module:

- Visual Explain allows you to view Explain plans graphically.
- Visual Explain allows you to compare different Explain plans.
- QCD has defined user categories to easily define user access rights.

Additional capabilities:

- Compressed View – essentially displays the plan in a traditional data flow manner without the action icons.
- Menu options are available to display both current and captured index information, object definitions and statistics.
- The QCD is utilized with other Teradata Analyst tools.
  - QCF is enhanced to collect table level data demographics as part of the execution plan capture.
- Bulk Compare – this powerful feature allows you to perform multiple plan comparisons with a single operation.

Module 27: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 27: Review Questions

1. **To place an execution plan into the QCD database, preface a valid parsable Teradata SQL statement with _____ _____ .**

2. **When using Visual Explain to compare multiple plans, one plan must be selected as the _____ query.**

3. **List the 3 types of QCD users that access rights are associated with.**

   _____ _____ _____

Lab Exercise 27-1

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Lab Exercise 27-1

Lab Exercise 27-1

Purpose

In this lab, you will use the COLLECT STATISTICS command.

What you need

Populated AP.Trans table and your empty Trans table.

Tasks

1. Delete all of the rows in the Trans table.

   The Trans table should have a USI on Trans_Number and a NUSI on Trans_ID from a previous lab. Verify with HELP INDEX Trans;

2. Collect statistics on the Trans table primary and secondary indexes. Use the Help Statistics command after collecting statistics on all of the indexes.

3. Populate your Trans table from AP.Trans using the INSERT/SELECT function. Verify (SELECT COUNT) that your Trans table has 15,000 rows.

   Use the Help Statistics command after populating the Trans table. Do the statistics reflect the status of table? _____ How many unique values are there for each column? _____

4. Recollect statistics on the Trans table. Use the Help Statistics command after populating the Trans table. Do the statistics reflect the status of table? _____
   How many unique values are there for Trans_ID? _____

Lab Exercise 27-2

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

SQL hint:

COLLECT STATISTICS ON *table_name* COLUMN *column_name* ;

If your Orders and Orders_PPI tables have a different number of rows, this is probably because Orders only has 9 years of data and Orders_PPI has 10 years of data.

The easiest option to ensure that both tables have the same number of rows is to:

- Delete all the rows in Orders table and populate the Orders table from Orders_PPI.

Lab Exercise 27-2

Lab Exercise 27-2

Purpose

In this lab, you will use Teradata SQL Assistant to explain various SQL access and join functions OR you can use Visual Explain (if installed and available).

What you need

Populated DS tables and your populated tables from previous lab. Make sure that your Orders and Orders_PPI tables have the same number of rows.

Tasks

1. Collect statistics on orderid, orderdate, and PARTITION for Orders and Orders_PPI.
   Execute the following two Explains.

```
EXPLAIN SELECT * FROM Orders
      WHERE orderdate BETWEEN '2012-01-01' AND '2012-01-31';
```

How many AMPs are accessed in this operation? _____
Is this a full table scan? _____ Why or why not? _____

```
EXPLAIN SELECT * FROM Orders_PPI
      WHERE orderdate BETWEEN '2012-01-01' AND '2012-01-31';
```

How many AMPs are accessed in this operation? _____

Is this a full table scan? _____ Why or why not? _____

Is partitioning beneficial for this type of query? _____

Lab Exercise 27-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Lab Exercise 27-2 (cont.)

2. **Execute the following two Explains.**

```
EXPLAIN SELECT * FROM Orders
      WHERE orderid = 418200;
```

How many AMPs are accessed in this operation? _____
How is this row retrieved by Teradata? _____

```
EXPLAIN SELECT * FROM Orders_PPI
      WHERE orderid = 418200;
```

How many AMPs are accessed in this operation? _____
How many partitions are scanned to locate this row? _____
How is this row retrieved by Teradata? _____

Lab Exercise 27-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Lab Exercise 27-2 (cont.)

3. **Execute the following two Explains.**

   EXPLAIN DELETE FROM Orders WHERE orderdate BETWEEN '2009-01-01' AND '2009-12-31';

   How many AMPs are accessed in this operation? _____
   Is this a full table scan? _____ Why or why not? _____

   EXPLAIN DELETE FROM Orders_PPI
   WHERE orderdate BETWEEN '2009-01-01' AND '2009-12-31';

   How many AMPs are accessed in this operation? _____
   Is this a full table scan? _____ Why or why not? _____
   How many partitions are scanned to delete these rows? _____

4. **Collect statistics on orderid for the Orders_CP table and execute the following two Explains.**

   EXPLAIN SELECT Orderid FROM Orders_CP;

   How many column partitions are accessed? _____
   What is the relative cost for this EXPLAIN? _____

   EXPLAIN SELECT Orderid, Custid FROM Orders_CP;

   How many column partitions are accessed? _____
   What is the relative cost for this EXPLAIN? _____

Notes

Module 28

TERADATA
Raising Intelligence

Join Processing Analysis

After completing this module, you will be able to:

- **Identify and describe different kinds of join plans.**
- **Describe join plan strategies.**
  - **Merge Join**
  - **Nested Join**
  - **Hash Join**
  - **Product Join**
  - **Exclusion Merge Join**
- **Describe different types of join strategies that may be used with PPI tables.**

Notes

Table of Contents

SELECT Statement ANSI Join Syntax 28-4
Example of ANSI and Teradata JOIN Syntax 28-6
LEFT Outer Join Example 28-8
RIGHT Outer Join Example 28-10
FULL Outer Join Example 28-12
Join Processing 28-14
Optimizer Minimizes Spool Usage 28-16
Row Selection 28-18
Join Redistribution 28-20
    The Join column(s) is the Primary Indexes of Both Tables 28-20
    The Join column is the Primary Index of one of the tables 28-20
  Join Redistribution (cont.) 28-22
    The Join column is the Primary Index of neither table 28-22
Duplicating a Table in Spool 28-24
General Join Distribution Strategies 28-26
Merge Join 28-28
Merge Join Strategy 28-30
Merge Join – Matching Primary Indexes 28-32
Merge Join – Row Redistribution 28-34
Merge Join – Duplicate the Smaller Table 28-36
Nested Joins 28-38
Product Join 28-40
Cartesian Product 28-42
Product Join – Duplicate the Smaller Table 28-44
Hash Join 28-46
Exclusion Joins 28-48
Exclusion Join Example 28-50
Inclusion Joins 28-52
n-Table Joins 28-54
Join Considerations with PPI 28-56
NPPI to PPI Join – Few Partitions 28-58
NPPI to PPI Join – Many Partitions 28-60
NPPI to PPI Join – Sliding Window 28-62
NPPI to PPI Join – Sliding Window (cont.) 28-64
NPPI to PPI Join – Hash Ordered Spool File Join 28-66
PPI to PPI Join – Rowkey-Based Join 28-68
PPI to PPI Join – Unmatched Partitions 28-70
Additional Join Options with PPI 28-72
Join Processing Summary 28-74
Module 28: Review Questions 28-76
  Module 28: Review Questions (cont.) 28-78

SELECT Statement ANSI Join Syntax

The facing page shows all syntax options for joins using the ANSI standard join conventions.

SELECT Statement ANSI Join Syntax

Teradata supports the ANSI join syntax which provides outer join options.

```
SELECT    cname [, cname , ...]
FROM      tname [aname]
              [INNER] JOIN
              LEFT [OUTER] JOIN
              RIGHT [OUTER] JOIN
              FULL [OUTER] JOIN
              CROSS JOIN
          tname [aname]
              ON  condition ;
```

- A two-table join condition references two different tables.
- A self-join condition references two alias names for one table.

Where:

cname	Column or expression name
tname	Table or view name
aname	Alias for table or view name
condition	Criteria for the join

INNER JOIN	All matching rows.
LEFT OUTER JOIN	Table to the left is used to qualify, table on the right has nulls when rows do not match.
RIGHT OUTER JOIN	Table to the right is used to qualify, table on the left has nulls when rows do not match.
FULL OUTER JOIN	Both tables are used to qualify and extended with nulls.
CROSS JOIN	Product join or Cartesian product join.

Example of ANSI and Teradata JOIN Syntax

An Inner Join (as shown on the facing page) returns an output row for each successful match between the join tables.

The facing page shows an example comparing Teradata and ANSI JOIN syntax.

Notes about this Inner Join include:

- Information about employees and their department names where the employee's department number matches the existing departments.
- No information about employees who have no department number or an invalid department number.
- No information is returned about departments which have no employees assigned to them.

Example of ANSI and Teradata JOIN Syntax

ANSI JOIN Syntax

```
SELECT    D.Department_Number   AS "Dept Number"
         ,D.Department_Name     AS "Dept Name"
         ,E.Last_Name           AS "Last Name"
         ,E.Department_Number   AS "Emp Dept Number"
FROM      Department D
INNER JOIN Employee E
ON        E.Department_Number = D.Department_Number;
```

Teradata JOIN Syntax

```
SELECT    D.Department_Number   AS "Dept Number"
         ,D.Department_Name     AS "Dept Name"
         ,E.Last_Name           AS "Last Name"
         ,E.Department_Number   AS "Emp Dept Number"
FROM      Department D
         ,Employee E
WHERE     E.Department_Number = D.Department_Number;
```

Output is same from either Join.

Dept Number	Dept Name	Last Name	Emp Dept Number
402	software support	Crane	402
100	executive	Trainer	100
501	marketing sales	Runyon	501
301	research and development	Stein	301
301	research and development	Kanieski	301

LEFT Outer Join Example

A **left outer join** produces both rows which match in both tables and rows contained in one of the two tables which do not match. The keyword LEFT indicates that the table syntactically to the left of the join operator will be the “driver” table, that is, all rows of this table will be seen, whether they match or not.

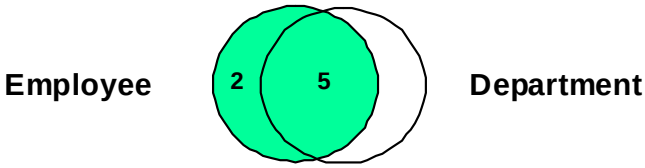
LEFT Outer Join Example

```
SELECT    E.Last_Name              AS "Last Name"
          ,E.Department_Number     AS "Dept Number"
          ,D.Department_Name       AS "Dept Name"
FROM      Employee      E  LEFT OUTER JOIN
          Department    D
ON        E.Department_Number = D.Department_Number
ORDER BY 1;
```

Last Name	Dept Number	Dept Name
Green	?	?
James	111	?
Crane	402	software support
Kanieski	301	research and develop
Runyon	501	marketing and sales
Stein	301	research and develop
Trainer	100	executive

In addition to output from inner join:

- Shows employees with null departments.
- Shows employees with invalid departments.



RIGHT Outer Join Example

A **right outer join** returns both rows that match and those from one table which do not.

The RIGHT keyword indicates that the table located syntactically to the right of the join operator is the “driver” table, i.e., all rows of that table will be returned whether or not they match.

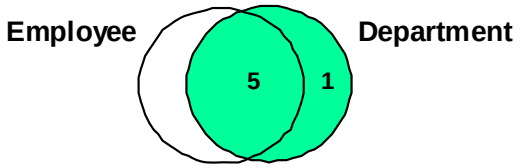
RIGHT Outer Join Example

```
SELECT    D.Department_Number      AS "Dept Number"
          ,D.Department_Name       AS "Dept Name"
          ,E.Last_Name             AS "Last Name"
FROM      Employee    E  RIGHT OUTER JOIN
          Department  D
ON        E.Department_Number = D.Department_Number
ORDER BY 1;
```

Dept Number	Dept Name	Last Name
100	executive	Trainer
301	research and develop	Stein
301	research and develop	Kanieski
402	software support	Crane
501	marketing sales	Runyon
600	new department	?

In addition to output from inner join:

- Shows departments with no employees.



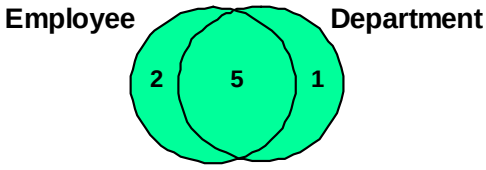
FULL Outer Join Example

Outer join syntax provides the capability of showing not only rows which match, but also those which do not. Using outer joins allows us to see rows with invalid or null entries where there would normally be a match.

FULL Outer Join Example

```
SELECT    D.Department_Number   AS "Dept Number"
          ,D.Department_Name    AS "Dept Name
          ,E.Last_Name          AS "Last Name"
          ,E.Department_Number  AS "Emp Dept Number"
FROM      Employee    E  FULL OUTER JOIN
          Department  D
ON        E.Department_Number = D.Department_Number
ORDER BY 1;
```

Dept Number	Dept Name	Last Name	Emp Dept Number
?	?	Green	?
?	?	James	111
100	executive	Trainer	100
301	research and development	Stein	301
301	research and development	Kanieski	301
402	software support	Crane	402
501	marketing sales	Runyon	501
600	new department	?	?



In addition to output from inner join:

- Shows employees with null departments.
- Shows employees with invalid departments.
- Shows departments with no employees.

Join Processing

For Teradata to process a Join, the rows to be joined must be on the same AMP. Copies of some or all of the rows may have to be moved to a common AMP. (The original rows are never moved.)

Teradata uses several kinds of join plans:

- Product Joins
- Merge Joins
- Nested Joins
- Exclusion (Merge or Product) Joins

No matter what type of Join is being done, the Optimizer will choose the best strategy based upon indexes and demographics (as it does with any SQL request). Use the EXPLAIN facility to be sure what type of Join will occur.

Join Processing

Rows must be on the same AMP to be joined.

- **If necessary, the system creates spool copies of one or both rows and moves them to a common AMP.**
- **Join processing NEVER moves or changes the original table rows.**

The Optimizer chooses the best join strategy based on:

- **Available Indexes**
- **Demographics (COLLECTed STATISTICS or Random AMP Sample)**

EXPLAIN shows what kind of join a query uses.

Typical Relational (or SQL) join types:

- Inner
- Outer
  - Left
  - Right
  - Full
- Product
- Nested
- Cross
  - Cartesian

Typical Teradata Join Plans:

- Merge Join
- Product Join
- Hash Join
- Nested Join
- Exclusion Join
- Inclusion Join
- RowID Join
- Self-Join

Optimizer Minimizes Spool Usage

As you saw on the preceding page, Teradata often has to utilize Spool space to hold the copies of rows redistributed to do a Join. The Optimizer minimizes the amount of Spool required by:

- Projecting (copying) only those columns which the query requires.
- Doing single-table Set Selections first (qualifying rows).
- Putting only the Smaller Table into Spool whenever possible.

Teradata determines the Smaller Table by multiplying the number of qualified rows by the number of bytes in the columns to be projected (qualifying rows * projected column bytes). Which table this turns out to be is not always obvious.

Note: Non-equality Join Operators produce a (partial) Cartesian Product. Join operators should always be equality conditions. Set selection operators may be any condition.

Optimizer Minimizes Spool Usage

1st Example: Select from Dept #1025

```
SELECT      E.emp_number
            E.last_name,
            E.first_name,
            P.check_num
FROM        Employee  E
INNER JOIN  Paycheck  P
ON          E.emp_number = P.emp_number
WHERE       E.dept_number = 1025;
```

- Projection List
- Explicit Join
- Join Condition
- Set Condition

The Optimizer minimizes spool size before the join.

- Applies SET conditions first (WHERE).
- Only the necessary columns are used in Spool.

Assume the following:

Employee (100,000 Rows, each row is 300 bytes)
Paycheck (1,000,000 rows, each row is 250 bytes)

2nd Example: Select from all departments

```
SELECT      E.*, P.check_num
FROM        Employee E
INNER JOIN  Paycheck P
ON          E.emp_number = P.emp_number;
```

In this example, all columns from Employee, but only 2 columns from Paycheck are needed.

Therefore, the following applies for spool space needed.

Employee – 100,000 x 300 bytes = 30 MB
Paycheck – emp# (INTEGER), check# (INTEGER) – 8 x 1,000,000 = 8 MB

Row Selection

Row Selection is a very important part of Join Processing. It is dependent on the conditions specified in the WHERE clause. If no Set Selection conditions exist, then all rows of both tables will participate in the Join.

Reducing the number of rows that participate will improve Join performance. The two examples on the facing page illustrate this. The first SELECT statement has no Set Selection conditions; therefore, there is no Row Selection, and all rows from both tables will participate in the Join.

The second SELECT statement differs from the first one in that a Set Selection condition has been added. By specifying "WHERE Part.Ship# IS NOT NULL," the performance of the Join will be greatly improved. The effect of this new condition is to reduce the number of participating Part Table rows from 30,000,000 to 500,000.

The Optimizer automatically eliminates NULLs for INNER Joins.

Note: The Part Number represents a "serial number" and is unique for a part. Therefore, a particular part number or serial number can only be on 1 shipment.

Row Selection

- Column projection always precedes a join and row selection usually precedes a join.
- Tip: Reduce the rows that participate to improve join performance.

3rd Example:

SHIPMENT

5,000 Rows	SHIP#	. . .	
PK/FK	PK,SA,NN		
Distinct Values	5000		
Max Rows/Value	1		
Max Rows/NULL	0		
Typical Rows/Value	1		
PI/SI	UPI		

PART

30,000,000 Rows	PART#	. . .	SHIP#
PK/FK	PK,SA,NN		FK
Distinct Values	30M		5001
Max Rows/Value	1		200
Max Rows/NULL	0		29.5M
Typical Rows/Value	1		100
PI/SI	UPI		

```
SELECT      . . .
FROM        Shipment  S
INNER JOIN  Part  P
ON          S.ship# = P.ship# ;
```

```
SELECT      . . .
FROM        Shipment  S
INNER JOIN  Part  P
ON          S.ship# = P.ship#
WHERE       P.ship# > 150183 ;
```

Assuming Shipment.Ship# is defined as NOT NULL:

- The Optimizer automatically eliminates all NULLs for INNER joins in the Part table before doing join.
- Even though the Part table has 30,000,000 rows, only 500,000 rows have values (match) and are joined to the Shipment table and output.
- To further eliminate additional rows, add a WHERE condition that reduces the number of rows that participate in the join.

Join Redistribution

The Primary Index is the major consideration used by the Optimizer in determining how to join two tables and deciding which rows to move.

Three general scenarios may occur when two tables are to be Merge Joined:

1. The Join column(s) is the Primary Index of both tables (best case).
2. The Join column is the Primary Index of one of the tables.
3. The Join column is not a Primary Index of either table (worst case).

The three scenarios are described below and pictured on the following pages:

The Join column(s) is the Primary Indexes of Both Tables

This is the best case scenario because rows that can be joined together are already on the same target AMP. Equal primary index values always hash to the same AMP. No movement of data to other AMPs is necessary. The rows are already sorted in Row Hash sequence because of the way in which they are stored by the file system. With no need for sorting or movement of data, the Join can take place immediately.

The Join column is the Primary Index of one of the tables

In this case, one table has its rows on the target AMPs and one does not. The rows of the second table must be redistributed to their target AMPs by the hash code of the Join Column value. If the table is “small,” the Optimizer might decide to simply “duplicate” the entire table on all AMPs instead of hash redistributing. In either case, the rows of one table will be copied to their target AMPs. (If the PI table is the “small” table, the Optimizer might choose to duplicate it on all AMPs rather than redistributing the non-PI table.)

Join Redistribution

Join columns are from the same domain. **No Redistribution needed.**

```
SELECT      . . .
FROM        Table1  T1
INNER JOIN  Table2  T2
ON          T1.A = T2.A;
```

T1

A	B	C
PI		
100	214	433

T2

A	B	C
PI		
100	725	002

Join columns are from the same domain. **Redistribution needed.**

```
SELECT      . . .
FROM        Table3  T3
INNER JOIN  Table4  T4
ON          T3.A = T4.B;
```

T3

A	B	C
PI		
255	345	225

T4

A	B	C
PI		
867	255	566

Redistribute T4 rows in spool on column B.

SPOOL

A	B	C
	PI	
867	255	566

Join Redistribution (cont.)

The Join column is the Primary Index of neither table

If neither column is a Primary Index, then the rows of both tables must be redistributed to their target AMPs. This may be done by hash redistribution of the Join Column value, or by duplicating "small" tables on each AMP. In either case, this approach involves the maximum amount of data movement. The choice of a Primary Index should be heavily influenced by the amount of Join activity anticipated.

Join Redistribution (cont.)

Join is on columns that aren't the Primary Index of either table.

Join columns are from the same domain. **Redistribution needed.**

```
SELECT      . . .
FROM        Table5  T5
INNER JOIN  Table6  T6
ON          T5.B = T6.C;
```

T5

A	B	C
PI		
456	777	876

T6

A	B	C
PI		
993	228	777

Redistribute T5 rows in spool on column B.

Redistribute T6 rows in spool on column C.

SPOOL

A	B	C
	PI	
456	777	876

SPOOL

A	B	C
		PI
993	228	777

If the columns being joined together are not Primary Index columns and are from the same domain, options the Optimizer may choose from include:

- Redistribute both tables in spool (as shown above)
- Duplicate the smaller table in spool across all AMPs

Duplicating a Table in Spool

The facing page highlights the fact that Joins can require considerable Spool space. Take this into consideration when calculating Spool requirements.

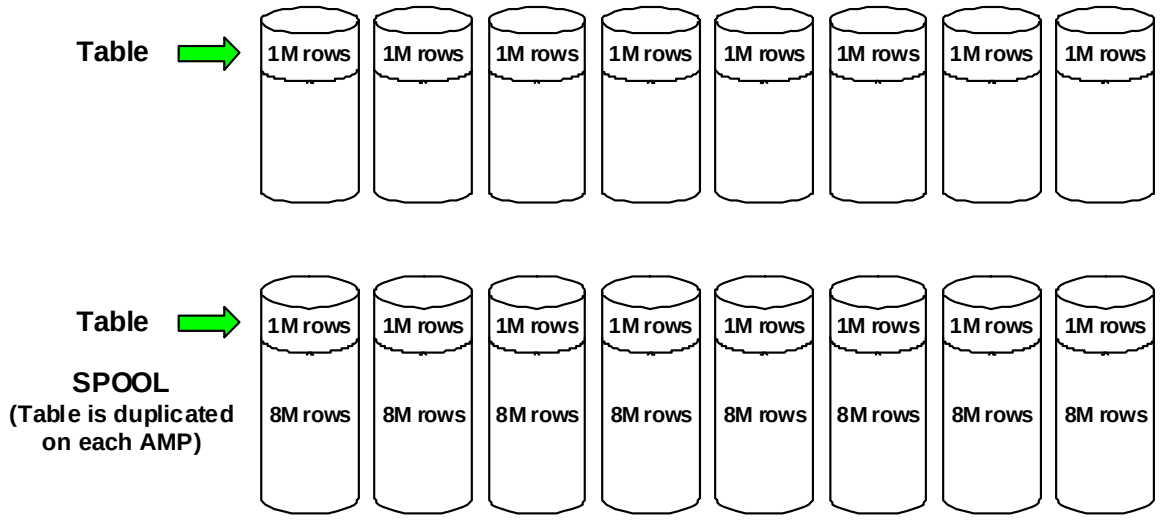
The top diagram shows an 8 million row table distributed evenly across 8 AMPs so that there are 1 million rows on each AMP.

The bottom diagram shows what happens when the table is duplicated in Spool across all the AMPs. You will notice that there are now 9 million rows on each AMP – 1 million for the actual table and 8 million in Spool.

This example should convince you of the importance of using the EXPLAIN facility so that you don't do unnecessary Product Joins.

Duplicating a Table in Spool

- For merge joins, the optimizer may choose to duplicate a small table on each AMP.
- For product joins, the optimizer always duplicates one table across all AMPs.
- In either case, each AMP must have enough spool space for a complete copy.



The Explain plan will indicate that 64M rows are needed for spool.

General Join Distribution Strategies

The Optimizer has many join methods, or modes, to choose from to ensure that any join operation is fully optimized. This module discusses some of the more common join methods, but certainly does not cover all of the join methods available to the Optimizer.

The particular processing described for a given type of join (for example, duplication or redistribution of spooled data) might not apply to all joins of that type.

Some of the common join methods include:

- Merge or Hash
  - When done on matching primary indexes, do not require any data to be redistributed.
  - Hash joins are often better performers and are used whenever possible. They can be used for equijoins *only*.

- Nested
  - Only join expression that generally does not require all AMPs.
  - Preferred join expression for OLTP applications.

- Product
  - Always selected by the Optimizer for WHERE clause inequality conditions.
  - High cost because of the number of comparisons required.

General Join Distribution Strategies

MERGE JOIN

Do nothing if Primary Indexes match and are the join columns.

OR

REDISTRIBUTE one or both sides (depending on the Primary Indexes used in the join).

OR

DUPLICATE the smaller table (or data set) in spool on all AMPs.

After redistributing or duplicating the data, the optimizer may choose to:

- Perform a Hash Join if the redistributed or duplicated table can be held in memory.
- SORT the duplicated table on join column row hash, create a local spool copy of the larger table and sort it on join column hash, and perform a merge join.

NESTED JOIN – Special join case

Equijoin with a constant value for a unique index in one table.
Only join expression that generally does not require all AMPs.

PRODUCT JOIN – Rows do not have to be in any sequence.

DUPLICATE the Smaller Table on all AMPs.

Merge Join

The Merge Join retrieves rows from two tables and then puts them onto a common AMP based on the row hash of the columns involved in the join. The system sorts the rows into join column row hash sequence, then joins those rows that have matching join column row hash values.

Merge Joins are commonly done when the Join condition is based on equality. They are generally more efficient than product joins because the number of rows comparisons is smaller. The general merge join strategy consists of the following steps:

- Identify the smaller table to be joined
- The Optimizer only pursues the following steps if it is necessary to place qualified rows into a spool file.
  - Place the qualifying rows from one or both relations into a spool file.
  - Relocate the qualified spool rows to their target AMPs based on the hash of the join column set
  - Sort the qualified spool rows on their join column row hash values.
- Compare those rows with matching Join Column Row Hash values.

Merge Join

Merge Joins

- Require rows to be on the same AMP to be joined.
- Are usually chosen for an equality join condition.
- Blocks from both tables are read ***only*** once.
- Are generally more efficient than a product join.
- Compare matching join column row hash values for the rows.
- Cause significantly fewer comparisons than a product join.

Merge join process:

- Identify the Smaller Table.
- If necessary:
  - Put qualifying data of one or both tables into spool(s).
  - Move the spool rows to AMPs based on the join column hash.
  - Sort the spool rows into join column hash sequence.
- Compare the rows with matching join column row hash values.

General considerations:

- Join costs rise with the number of rows that are moved and sorted.
- Join plans for the same tables may change as the demographics change.

Merge Join Strategy

Two different general Merge Join algorithms are available:

- Slow Path – the slow path is used when the left table is accessed using a read mode other than an all rows scan. The determination is made in the AMP, not by the Optimizer.

- Fast Path – the fast path is used when the left table is accessed using the all-row scan reading mode.

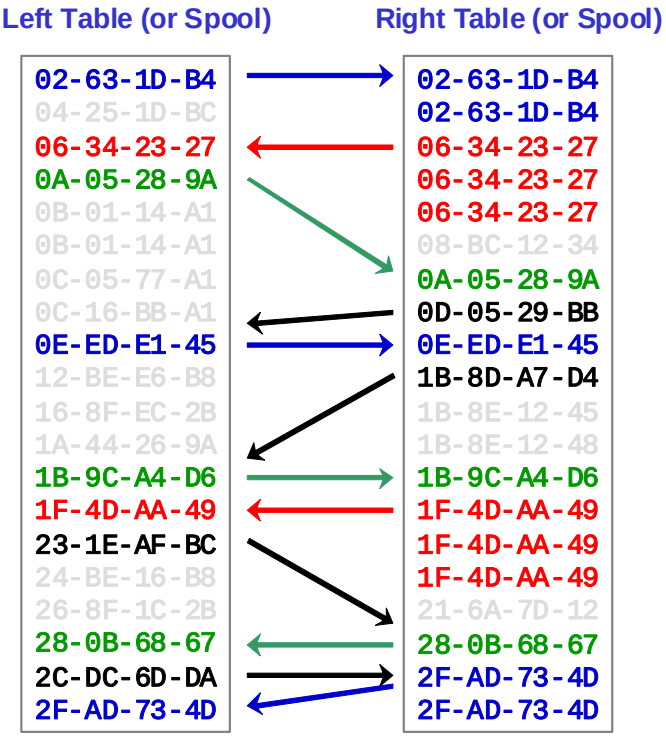
The illustration on the facing page gives you a graphical representation of how a Merge Join compares only those rows with matching Row Hash values.

Merge Join Strategy

There are several versions of merge join. This particular illustration is for the row hash match scan.

With a row hash match scan, both tables (or spool) are sorted on join column row hash sequence.

Note that unnecessary comparisons are ignored, and that each data block is touched only once.



Merge Join – Matching Primary Indexes

The example on the facing page illustrates joining two tables together with matching primary indexes. This strategy does not require any duplication or sorting of rows.

This example is pictured on a 4 AMP System. Teradata merely compares the rows already on the proper AMPs.

This join strategy will occur when the Join Column is the Primary Index of both tables. This is also referred to an AMP Local Join.

Merge Join – Matching Primary Indexes

Employee

Enum	Name	Dept
PK		FK
UPI		
1	BROWN	200
2	SMITH	310
3	JONES	310
4	CLAY	400
5	PETERS	150
6	FOSTER	200
7	GRAY	310
8	BAKER	310
9	TYLER	450
10	CARR	450

Employee_Phone

Enum	Area_Code	Phone
PK		
FK		
NUPI		
1	213	3241576
1	213	4950703
3	408	3628822
4	415	6347180
5	312	7463513
6	203	8337461
8	301	6675885
8	301	2641616

Primary Indexes match: no duplication or sorting needed

Example:

```
SELECT    E.Enum, E.Name, ...
FROM      Employee  E
INNER JOIN Employee_Phone  P
ON        E.Enum = P.Enum ;
```

Employee rows hash distributed on Enum (UPI)

6 FOSTER 200 8 BAKER 310	4 CLAY 400 3 JONES 310 9 TYLER 450	1 BROWN 200 7 GRAY 310	5 PETERS 150 2 SMITH 310 10 CARR 450

Employee_Phone rows hash distributed on Enum (NUPI)

6 203 8337461 8 301 2641616 8 301 6675885	4 415 6347180 3 408 3628822	1 213 3241576 1 213 4950703	5 312 7463513

Merge Join – Row Redistribution

The example on the facing page illustrates an example of redistributing one of the tables. This strategy consists of redistributing the rows of one table and sorting them on the Row Hash of the Join Column.

This example is pictured on a 4 AMP System. Teradata copies the employee rows into Spool and redistributes them on Employee.Dept Row Hash. The Merge Join then occurs with the rows to be joined located on the same AMPs.

This strategy occurs when one of the tables is already distributed on the Join Column Row Hash. The Join Column is the PI of one, not both, of the tables.

Merge Join – Row Redistribution

REDISTRIBUTE one side and SORT on join column row hash.

Employee

Enum	Name	Dept
PK		FK
UPI		
1	BROWN	200
2	SMITH	310
3	JONES	310
4	CLAY	400
5	PETERS	150
6	FOSTER	200
7	GRAY	310
8	BAKER	310
9	TYLER	450
10	CARR	450

Department

Dept	Name
PK	
UPI	
150	PAYROLL
200	FINANCE
310	MFG.
400	EDUCATION
450	ADMIN

Example:

```
SELECT    E.Enum, E.Name, D.Dept, D.Name
FROM      Employee  E
INNER JOIN Department  D
ON        E.Dept = D.Dept ;
```

Employee rows hash distributed on Employee.Enum (UPI)

6 FOSTER 200 8 BAKER 310	4 CLAY 400 3 JONES 310 9 TYLER 450	1 BROWN 200 7 GRAY 310	5 PETERS 150 2 SMITH 310 10 CARR 450

Spool file after redistribution on Employee.Dept row hash

5 PETERS 150	7 GRAY 310 3 JONES 310 8 BAKER 310 2 SMITH 310	1 BROWN 200 6 FOSTER 200 9 TYLER 450 10 CARR 450	4 CLAY 400

Department rows hash distributed on Department.Dept (UPI)

150 PAYROLL	310 MFG.	200 FINANCE 450 ADMIN	400 EDUCATION

Merge Join – Duplicate the Smaller Table

The example on the facing page illustrates a Merge Join by duplicating the smaller table.

This strategy consists of duplicating and sorting the smaller table on all AMPs and locally building a copy of the Larger Table and sorting it.

This example is pictured on a 4 AMP System. Teradata duplicates the department table and sorts it on the department column for all AMPs. The employee table is built locally and sorted on the department Row Hash.

The Merge Join is then performed.

This example is the same as the previous example. If the Parser determines from statistics that it would be less expensive to duplicate and sort the smaller table than to hash redistribute the larger table, it will choose this strategy.

Merge Join – Duplicate the Smaller Table

DUPLICATE and SORT the Smaller Table on all AMPs.
LOCALLY BUILD a copy of the Larger Table and SORT.

Example:

```
SELECT      E.Enum, ...
FROM        Employee  E
INNER JOIN  Department  D
ON          E.Dept = D.Dept ;
```

Employee

Enum	Name	Dept
PK		FK
UPI		
1	BROWN	200
2	SMITH	310
3	JONES	310
4	CLAY	400
5	PETERS	150
6	FOSTER	200
7	GRAY	310
8	BAKER	310
9	TYLER	450
10	CARR	450

Department

Dept	Name
PK	
UPI	
150	PAYROLL
200	FINANCE
310	MFG.
400	EDUCATION
450	ADMIN

Department rows hash distributed on Department.Dept (UPI)

150 PAYROLL	310 MFG.	200 FINANCE 450 ADMIN	400 EDUCATION

Spool file after duplicating and sorting on Department.Dept row hash.

150 PAYROLL 200 FINANCE 310 MFG. 400 EDUCATION 450 ADMIN	150 PAYROLL 200 FINANCE 310 MFG. 400 EDUCATION 450 ADMIN	150 PAYROLL 200 FINANCE 310 MFG. 400 EDUCATION 450 ADMIN	150 PAYROLL 200 FINANCE 310 MFG. 400 EDUCATION 450 ADMIN

Employee rows hash distributed on Employee.Enum (UPI)

6 FOSTER 200 8 BAKER 310	4 CLAY 400 3 JONES 310 9 TYLER 450	1 BROWN 200 7 GRAY 310	5 PETERS 150 2 SMITH 310 10 CARR 450

Spool file after locally building and sorting on Employee.Dept row hash

6 FOSTER 200 8 BAKER 310	3 JONES 310 4 CLAY 400 9 TYLER 450	1 BROWN 200 7 GRAY 310	5 PETERS 150 2 SMITH 310 10 CARR 450

Nested Joins

Nested Joins are the most efficient types of Join. For a Nested Join to be done between Table 1 and Table 2, the Optimizer must be provided with both of the following:

- An equality value for a unique index on Table 1 (this will retrieve a single row).
- A Join on a column of that single row to any index on Table 2

The system will retrieve the single row from Table 1 based on the UPI or USI value, then determine the hash of the value in the Join Column to access matching Table 2 rows.

Nested Joins are the only types of Join that don't always use all of the AMPs. The number of AMPs involved in a Nested Join will vary.

The query on the facing page can also be coded as follows:

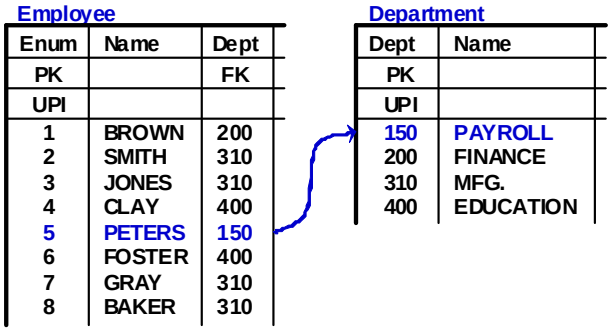
```
SELECT    E.Name
          ,D.Name
FROM      Employee  E, Department  D
WHERE     E.Dept = D.dept
AND       E.Enum = 5;
```

Nested Joins

- This is a special join case.
- This is the only join that doesn't always use all of the AMPs.
- It is the most efficient in terms of system resources.
- It is the best choice for OLTP applications.
- To choose a Nested Join, the Optimizer must have:
  - An equality value for a unique index (UPI or USI) on Table1.
  - A join on a column of that single row to any index on Table2.
- The system retrieves the single row from Table1.
- It hashes the join column value to access matching Table2 row(s).

Example:

```
SELECT     E.Name
          ,D.Name
FROM       Employee  E
INNER JOIN Department  D
ON         E.Dept = D.Dept
WHERE      E.Enum = 5;
```



Employee

Enum	Name	Dept
PK		FK
UPI		
1	BROWN	200
2	SMITH	310
3	JONES	310
4	CLAY	400
5	PETERS	150
6	FOSTER	400
7	GRAY	310
8	BAKER	310

Department

Dept	Name
PK	
UPI	
150	PAYROLL
200	FINANCE
310	MFG.
400	EDUCATION

Product Join

Product Joins are the most general forms of Join. In a product Join, every qualifying row of one table is compared to every qualifying row in the other table. Rows which match on WHERE conditions are saved.

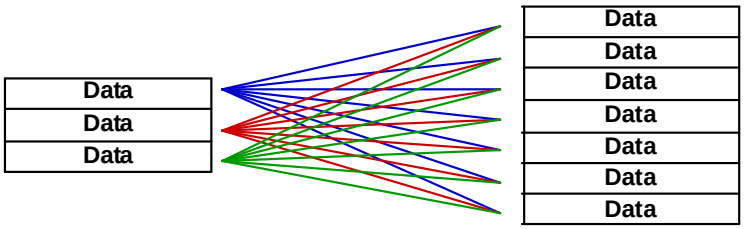
Product Joins are caused by any of the following:

- The WHERE clause is missing.
- A Join condition is not based on equality (NOT =, LESS THAN, GREATER THAN).
- Join conditions are ORed together.
- There are too few Join conditions.
- A referenced table is not named in any Join condition.
- Table aliases are incorrectly used.
- The Optimizer determines that it is less expensive than the other Join types.

Product joins get their name from the fact that the number of comparisons required is the "product" of the number of qualifying rows of both tables. A product Join between a table of 1,000 rows and a table of 50 rows would require 50,000 comparisons and a potential answer set of 50,000 rows.

Because all rows of one side must be compared with all rows of the other, the smaller table is always duplicated on all AMPs. Its rows then are compared with the AMP local rows of the other table. If the entire table cannot fit into memory, blocks will have to be read in more than once. Comparisons that qualify are written to spool. While legitimate cases exist for these joins, they should be avoided whenever possible.

Product Join



Rows must be on the same AMP to be joined.

- Does not sort the rows.
- May re-read blocks from one table if AMP memory size is exceeded.
- It compares every qualifying Table1 row to every qualifying Table2 row.
- Those that match the WHERE condition are saved in spool.
- It is called a Product Join because:

Total Compares = # Qualified Rows Table1 * # Qualified Rows Table2

- The internal compares become very costly when there are more rows than AMP memory can hold at one time.
- They are generally unintentional and often give meaningless output.
- Product Join process:
  - Identify the Smaller Table and duplicate it in spool on all AMPs.
  - Join each spool row for Smaller Table to every row for Larger Table.

Cartesian Product

Cartesian Products are unconstrained Product Joins. Every row of one table is joined to every row of another table.

Since there is rarely any practical business use of Cartesian Product Joins, they generally will occur when an error is made in coding the SQL query. Some reasons why Cartesian Products are caused are shown on the facing page.

Running your queries through EXPLAIN will enable you to avoid unintentional Cartesian Product Joins and thus save a lot of system resources. You should always EXPLAIN a Join before it goes production.

An example of using an alias incorrectly:

```
SELECT TableA.col1
FROM   TableA A;
```

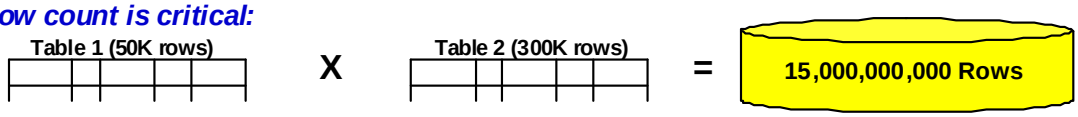
The result will be a product join. Teradata will rename the TAbleA as A and then in the SELECT clause when it sees a reference to TABLEA, it will treat it as a separate instance of TableA.

Teradata assumes that you want to join the table to itself. It will look for a join condition, but as there is none, Teradata will carryout a PRODUCT join.

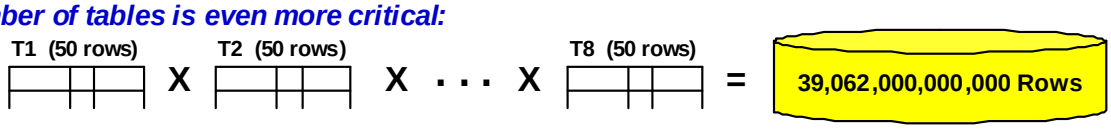
Cartesian Product

- This is an unconstrained Product join.
- Each row of Table1 is joined to every row in Table2.
- Cartesian Product Joins consume significant system resources.

Table row count is critical:



Number of tables is even more critical:



- Cartesian Product Joins are occasionally used in some business cases.
  - Example: Join a single column table (with limited rows) to a second table to populate a third table.
  - The Teradata Database also supports them for ANSI compatibility.
- Cartesian Product Joins may occur when:
  - A join condition is missing or there are too few join conditions.
  - Join conditions are not based on equality.
  - A referenced table is not named in any join condition.
  - Table aliases are incorrectly used.
- The transaction aborts if it exceeds the user's spool limit.

Product Join – Duplicate the Smaller Table

The example on the facing page illustrates the product join strategy. This strategy consists of duplicating the smaller table on every AMP.

This example is pictured on a 4 AMP System. The Product Join plan is caused by a join condition other than equality. As you can see, Teradata determines that the Department Table is the Smaller Table and then distributes copies of those rows to each AMP. The employee rows stay where they were (distributed by Enum). The Product Join then returns those rows which satisfy the Join Condition ("Employee.Dept > Department.Dept") after comparing every row in the employee table with every row in the department table.

Product Join – Duplicate the Smaller Table

DUPLICATE the Smaller Table on every AMP.

Employee

Enum	Name	Dept
PK		FK
UPI		
1	BROWN	200
2	SMITH	310
3	JONES	310
4	CLAY	400
5	PETERS	150
6	FOSTER	200
7	GRAY	310
8	BAKER	310
9	TYLER	450
10	CARR	450

Department

Dept	Name
PK	
UPI	
150	PAYROLL
200	FINANCE
310	MFG.
400	EDUCATION
450	ADMIN

Example:

```
SELECT      E.Enum, E.Name, D.Dept, D.Name
FROM        Employee  E
INNER JOIN  Department  D
ON          E.Dept > D.Dept ;
```

Department rows hash distributed on Department.Dept (UPI)

150 PAYROLL	310 MFG.	200 FINANCE 450 ADMIN	400 EDUCATION

Spool file after duplicating the Department rows.

150 PAYROLL 200 FINANCE 310 MFG. 400 EDUCATION 450 ADMIN	150 PAYROLL 200 FINANCE 310 MFG. 400 EDUCATION 450 ADMIN	150 PAYROLL 200 FINANCE 310 MFG. 400 EDUCATION 450 ADMIN	150 PAYROLL 200 FINANCE 310 MFG. 400 EDUCATION 450 ADMIN

Employee rows hash distributed on Employee.Enum (UPI)

6 FOSTER 200 8 BAKER 310	4 CLAY 400 3 JONES 310 9 TYLER 450	1 BROWN 200 7 GRAY 310	5 PETERS 150 2 SMITH 310 10 CARR 450

Hash Join

The Merge Join requires that both sides of the join have their qualifying row in join column row hash sequence. In addition, if the join column(s) are not the Primary Index, then some redistribution or duplication of rows precedes the sort.

In a Row Hash Join, the smaller table is sorted into join column row hash sequence and then redistributed or duplicated on all AMPs. The larger table is then processed a row at a time and the rows in this table do NOT have to be sorted into join column hash sequence. For those rows that qualify for joining (WHERE or ON), the Row Hash of the join column(s) is used to do a binary search of the smaller table (in memory) for a match. The Optimizer can choose this join plan when the qualifying rows of the small table can be held AMP memory resident. Use COLLECT STATISTICS on both tables to help guide the Optimizer.

You can enable hash joins and control the amount of memory allocated for hash joins with the following DBS Control fields:

- HTMemAlloc
- SkewAllowance

Recommendations that most sites will use for these parameters:

- HTMemAlloc = 1
- Skew Allowance = 75

Consider a different setting if:

- The system is always very lightly loaded. In this case, you may want to increase HTMemAlloc to a value between 2 and 5.
- Data is so badly skewed that the hash join degrades performance. In this case, you should turn the feature off or increase the Skew Allowance to 80 or 90.

<u>Miscellaneous Hash Join Notes:</u>

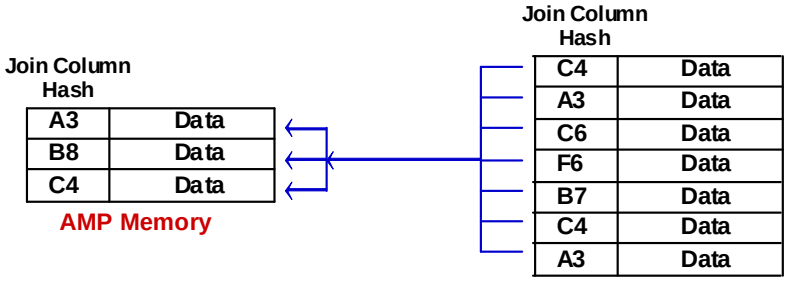
The standard hash join does require the same spooling and the same relocation of rows before the join, whether a table is being duplicated or redistributed. In this regard, it is similar to the merge join.

However, if the small table is small enough to fit into one hash partition and if it is duplicated, the redistribution of the large table can be eliminated by doing a hash join on the fly, a variant of the standard hash join. In such a case, the large table is read directly, without spooling or redistributing, and the hash join is performed between the small table spool and the large table rows.

Both types of hash joins eliminate sorting, and the on the fly version eliminates redistribution as well.

A **Dynamic Hash Join** provides the ability to do an equality join directly between a small table and a large table on non-primary index columns without placing the large table into a spool file. For Dynamic Hash Join to be used, the left table must be small enough to fit in a single partition.

Hash Join



This optimizer technique effectively places the smaller table in AMP memory and joins it to the larger table in unsorted spool. A Hash Join is applicable *only* to equijoins.

Row Hash Join Process:

- Identify the smaller table.
- Redistribute or duplicate the smaller table in memory across the AMPs.
- Usually sort the AMP memory into join column row hash sequence.
- Hold the rows in memory.
- Use the join column row hash of the larger table to search memory for a match.

This join eliminates the sorting, and possible redistribution or copying, of the larger table.

EXPLAIN plans will contain terminology such as “Single Partition Hash Join”.

Exclusion Joins

Exclusion Joins are based on set subtraction and used for finding rows that don't have a matching row in the other table. Queries with the NOT IN and EXCEPT operator lead to Exclusion Joins.

Exclusion Join is a Product or Merge Join where only the rows that do not satisfy (are NOT IN) any condition specified in the request are joined. In other words, Exclusion Join finds rows in the first table that do *not* have a matching row in the second table.

Exclusion Join is an implicit form of the outer join.

The appearance of a null (unknown) join value will return an answer set of NULL. Null join values must be prohibited to get a result other than NULL.

Another way to view exclusion operations, is to look at the 3 rules that are applied from the selected set to the NOT IN set.

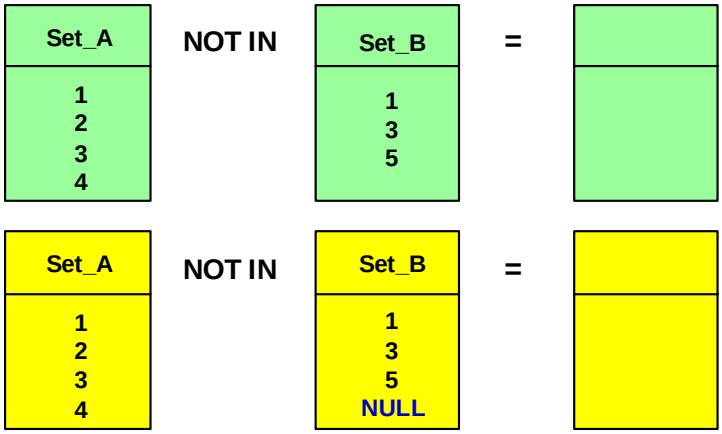
1. Any True – disqualifies the row
2. Any Unknown – disqualifies the row
3. All False – qualifies the row.

Therefore, in the first example, rows 1 and 3 are matched (true) so they are disqualified. Rows 2 and 4 are all false (no matches), so they qualify.

In the second example, row 2 hits the unknown (NULL) and is disqualified, so in essence all of the rows that do not get disqualified by the match (true) will get disqualified by rule number 2 (Any Unknown).

Exclusion Joins

- Finds rows that DON'T have a match.
- May be done as merge or product joins.
- SQL that frequently causes exclusion joins are NOT IN subqueries and EXCEPT or MINUS operations.
- Uses 3-value logic (= , <> , unknown) on nullable columns.
- To avoid NULL result sets:
  - If possible, define columns (used with NOT IN) as NOT NULL on the CREATE TABLE.
  - In queries, include WHERE column_name IS NOT NULL against nullable join columns.



Exclusion Join Example

The example on the facing page illustrates an Exclusion Merge Join. The SQL query is designed to list those salespeople who don't have any customers.

This example is pictured on a 4 AMP System. As you can see, Teradata does the following:

- Performs the subquery (SELECT Sales_Emp_Number FROM Customer).
- Hash redistributes these rows on all AMPs.
- Eliminates duplicate values.
- Returns the name column of those rows in the employee table which do not match the rows in the sub-query.

In this case, the salespersons whose names would be returned would be Peters and Tyler.

Exclusion Join Example

Employee

Enum	L_Name	Job_Code
PK		FK
UPI		
1	BROWN	3100
2	SMITH	2101
3	JONES	3100
4	CLAY	1201
5	PETERS	3100
6	FOSTER	3100
7	GRAY	1302
8	BAKER	3100
9	TYLER	3100
10	CARR	1302

Customer

Cust_Num	Sales_Emp_Number
PK	FK
UPI	
23	6
24	3
25	8
26	1
27	6
28	8
29	1
30	3
31	8

This is an example of an Exclusion Merge Join.

Example:

```
SELECT   Enum, L_Name
FROM     Employee
WHERE    Job_Code = 3100
AND      Enum
         NOT IN  (SELECT  Sales_Emp_Number
                  FROM    Customer);
```

Customer rows hash distributed on Cust_Num (UPI).

30	6	23	6	28	8	25	8
24	3	29	1	27	6	26	1
31	8			30	3		

Customer.Sales_Enum after hashing and duplicate elimination.

6	3	1	
8			

Employee rows hash distributed on Enum (UPI).

6 FOSTER 3100	3 JONES 3100	1 BROWN 3100	5 PETERS 3100
8 BAKER 3100	9 TYLER 3100		

Inclusion Joins

There are two types of Inclusion Join.

- Inclusion Merge Join
  - Read each row from the left table.
  - Join each left table row with the first right table row having the same hash value.
  - End of process.

- Inclusion Product Join
  - For each left table row read all right table rows from the beginning until one is found that can be joined with it.
  - Return the left row if a matching right row is found for it.
  - End of process.

The facing page illustrates an example of an Inclusion Merge Join. Explain plan terminology will indicate either “Inclusion Merge or Inclusion Product” in the explain text.

Inclusion Joins

INCLUSION JOIN – an inclusion join is a Merge or Product Join where the first right table row that matches the left row is joined.

An example of a query that may cause an inclusion merge join:

```
SELECT      Name, EmpNo
FROM        Employee
WHERE       EmpNo IN (SELECT EmpNo FROM Charges)
ORDER BY    Name ;
```

There are two types of Inclusion Join.

- Inclusion Merge Join
  - Read each row from the left table.
  - Join each left table row with the first right table row having the same hash value.
- Inclusion Product Join
  - For each left table row read all right table rows from the beginning until one is found that can be joined with it.
  - Return the left row if a matching right row is found for it.

n-Table Joins

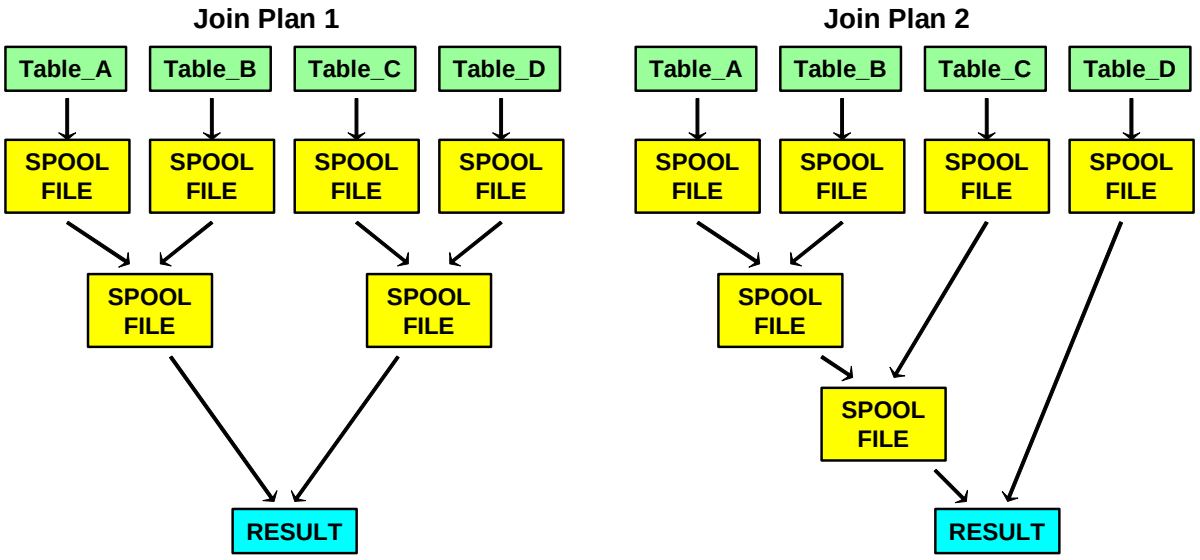
It is not uncommon to have Joins that involve more than two tables. The Optimizer will decide which tables it processes first based on its own decision algorithms. The Optimizer can only work with two tables at a time. The results of that Join operation will then be applied to a third table (or another Join result).

"Smaller" means the size of the table both as a function of selection and projection. "Selection" refers to the number of rows that qualify from that table in the answer set. "Projection" refers to the "row size" as a function of the number of column bytes selected. Each reduces the amount of data carried to subsequent Join steps.

n-Table Joins

- **All n-Table joins are reduced to a series of two-table joins.**
- **The Optimizer attempts to determine the best join order.**
- **Collected Statistics on Join columns help the Optimizer choose wisely.**

SELECT …. FROM Table_A, Table_B, Table_C, Table_D WHERE . . . ;



Join Considerations with PPI

Direct merge joins (in which the table of interest doesn't have to be spooled in preparation for a merge join) are available as an optimizer choice when two non-PPI tables have the same PI, and all PI columns are specified as equality join terms (the traditional merge join).

Direct merge joins of two PPI tables are similarly available as an optimizer choice when the tables have the same PI and identical partitioning expressions, and all PI columns and all partitioning columns are specified as equality join terms (the rowkey-based merge join). In both cases, the rows of the two tables will be ordered in the same way, allowing a merge join without redistribution or sorting of the rows. The performance characteristics of a traditional merge join and a rowkey-based merge join will be approximately the same.

A direct merge join, in the traditional sense, is not available when one table is partitioned and the other is not, or when both tables are partitioned, but not in the same manner, as the rows of the two tables will not be ordered in the same way. However, the traditional merge join algorithms have been extended to provide a PPI-aware merge join, called a "sliding window" join.

The optimizer has three general avenues of approach when joining a PPI table to a non-PPI table, or when joining two PPI tables with different partitioning expressions.

- One option is to spool the PPI table (or both PPI tables) into a non-PPI spool file in preparation for a traditional merge join.

- A second option (not always available) is to spool the non-PPI table (or one of the two PPI tables) into a PPI spool file, with identical partitioning to the remaining table, in preparation for a rowkey-based merge join.

- The third approach is to use the sliding window join of the tables without spooling either one. The optimizer will consider all reasonable join strategies, and pick the one that has the best-estimated performance.

Sliding window joins may be slower than the traditional merge join or the rowkey-based merge join when there are many non-excluded partitions. Sliding window joins can give roughly similar elapsed-time performance when the number of non-excluded partitions is small (but with greater CPU utilization and memory consumption).

Join Considerations with PPI

PPI is based on modular extensions to the existing implementation of Teradata. Therefore, all join algorithms support PPI without requiring a whole new set of join code.

Performance Note

Performance for Row Hash Merge Joins may be worse with PPI (as compared to NPPI tables) if a large number of partitions are not eliminated via query constraints.

Why?

The Row Hash Merge Join algorithm is more complicated and requires more resources with PPI than with NPPI (assuming an equal number of qualified data blocks) since rows are not in hash order, but rather in partition/hash order.

Example

Assume a PPI Transaction table is partitioned on date ranges and an Account table is NPPI.

The PPI Transaction table is not in the same row hash order as the NPPI Account table. A join of this NPPI to PPI will take longer than a typical join of the same tables if they were both NPPI (assuming that both tables have the same PI).

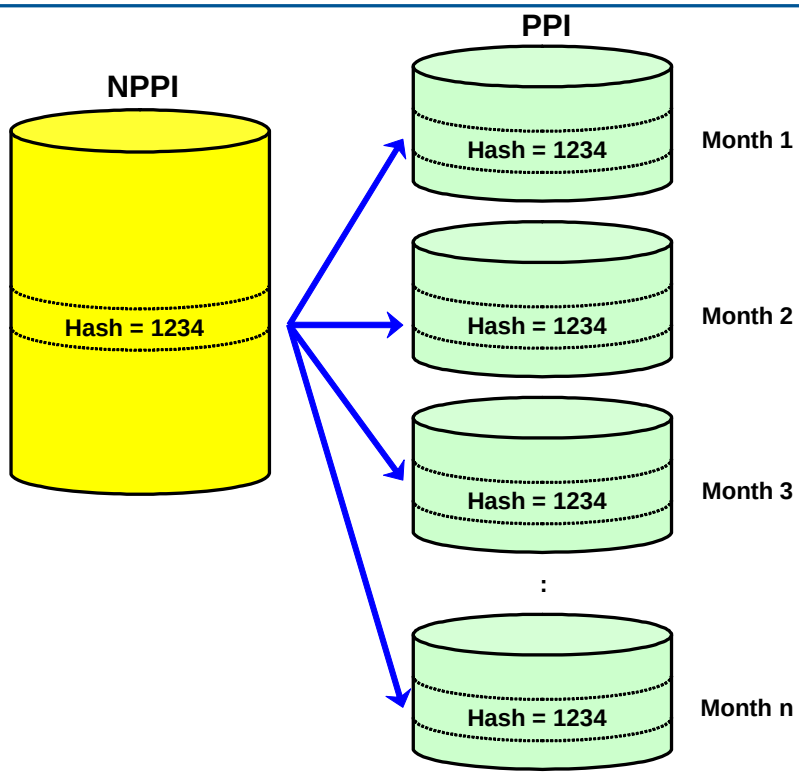
NPPI to PPI Join – Few Partitions

The following discussion assumes that two tables have the same PI and all PI columns are specified as equality join terms.

When joining two tables together (one with an NPPI and the other with a PPI) and after applying constraints, if there are a “small” number of surviving partitions, then the following applies:

- Teradata can keep one block from the NPPI table and one block per partition in memory from the PPI table to facilitate efficient join execution.
- Performance is similar to NPPI to NPPI join even when no partitions are eliminated (assumes that the number of partitions is relatively small).
  - Same number of disk I/Os (except in anomalous cases - large number of rows in a hash)
  - Higher memory requirements
  - Slightly higher CPU utilization
  - You can get significantly better performance when query constraints allow partition elimination.

NPPI to PPI Join – Few Partitions



Teradata can keep one block from the NPPI table and one block per partition in memory from the PPI table to facilitate efficient join execution.

NPPI to PPI Join – Many Partitions

The join algorithms have been enhanced to optimize joins involving PPI tables.

The basic enhancement is the use of a "sliding window" technique, which can be slower than the performance of a direct join. This is true as long as the number of partitions participating in the join is small. Usually, CPU utilization will probably be somewhat higher for a PPI table, and more memory will be used.

When joining two tables together (one with NPPI and the other with PPI) and after applying constraints, if there are a “large” number of surviving partitions, then the following applies:

- Teradata can keep one block from NPPI table and one block for as many partitions as it can fit (k) into memory from the PPI table to facilitate efficient join execution using a “sliding window” technique.

- This type of join will usually have worse performance than NPPI to NPPI join unless partition elimination can reduce total amount of work performed.

  ✓ Higher number of disk I/Os - the data blocks in NPPI table will have to be rescanned multiple times.

  d1 = # of data blocks in NPPI table
  d2 = # of data blocks in PPI table
  p = # of partitions participating in the join
  k = # of partitions that can fit into memory from PPI table

  The number of I/Os for NPPI to PPI join is:

  (p/k * d1) + d2

  The number of I/Os for NPPI to NPPI join is:

  d1 + d2

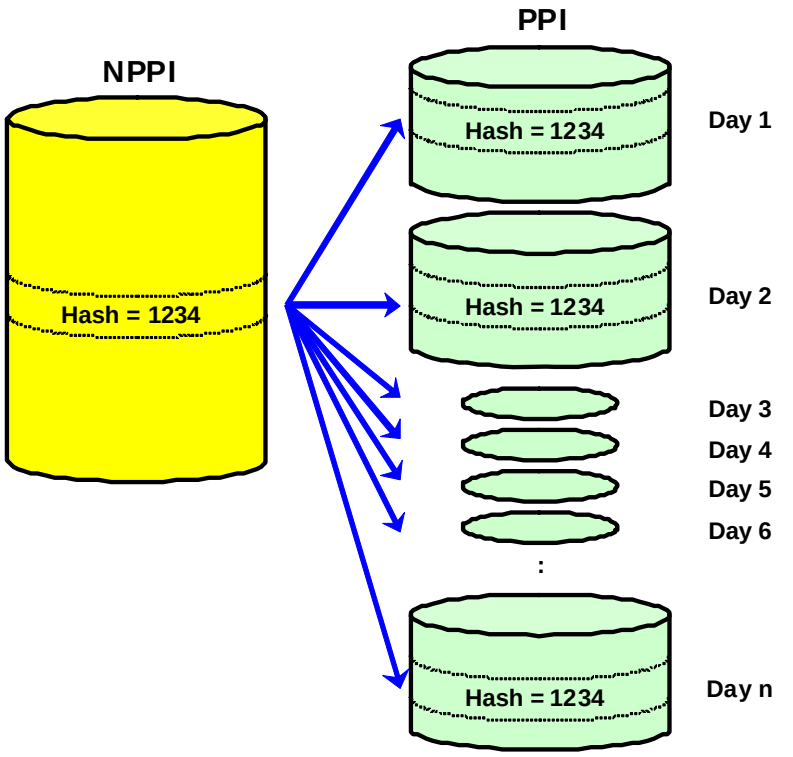
  ✓ Higher memory requirements

  ✓ Higher CPU utilization

- To obtain better performance, eliminate as many partitions as possible with query constraints.

  ✓ The p/k value must be small (less than 3 or 4) for performance to be reasonable.

NPPI to PPI – Many Partitions



One option available to Teradata is to keep one block from the NPPI table and one block for as many PPI partitions as it can fit into memory using a "sliding window" technique.

NPPI to PPI Join – Sliding Window

The simple example on the facing page illustrates that one block from the NPPI table is joined to data blocks from the first 3 partitions in the PPI table.

The most straight-forward way to join an NPPI table to a PPI table would be to make a pass over the NPPI table for each partition of the PPI table, thus doing the join as a series of sub-joins. This would be inefficient, especially for a large non-PPI table, though.

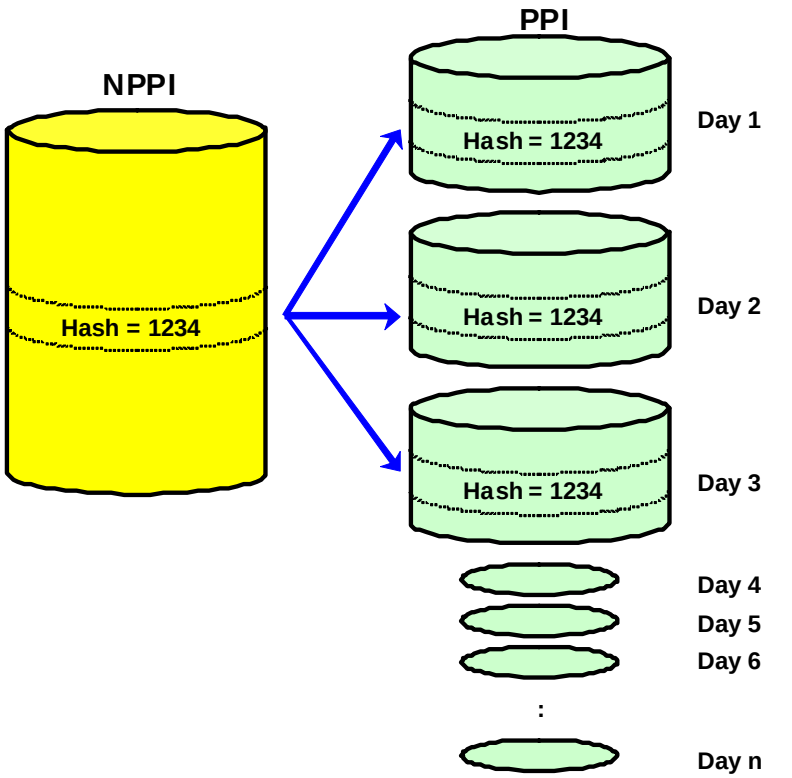
The "sliding window" join uses a similar concept, but minimizes the number of disk reads. The first data block is read for the NPPI table, and the first data block of each (non-excluded) partition of the PPI table is read into memory. The rows from the NPPI data block are compared to the rows of each PPI data block. The join routines present the rows in row hash order, but conceptually you can think of the process as visiting each partition's data block in turn. As the rows of a data block are exhausted, the next data block for that partition is read. This results in each data block of each table being read only once due to partitioning.

There is some additional overhead to manage the pool of data blocks, but join performance is not badly degraded. Overall performance may even be improved, if a non-trivial fraction of the partitions can be eliminated because of query conditions.

With this algorithm, a limiting factor is the number of data blocks that can be held in memory at one time. The file system cache memory (FSG Cache) is used to provide memory for the data blocks. A new user-tunable parameter is provided to control memory usage for this purpose. The DBS Control utility is used to set the parameter ("PPICacheThrP") in the Performance Group.

The value is a number expressed in tenths of a percent that controls the portion of the file system cache available for this purpose. The default value is 10, which represents one percent. A higher value will improve multiple-partition operations, of which joins are the most visible example. A high value may cause more memory contention and disk paging, so higher isn't always better.

NPPI to PPI Join – Sliding Window



In this simple example, the NPPI table is joined to the first 3 days in the PPI table.

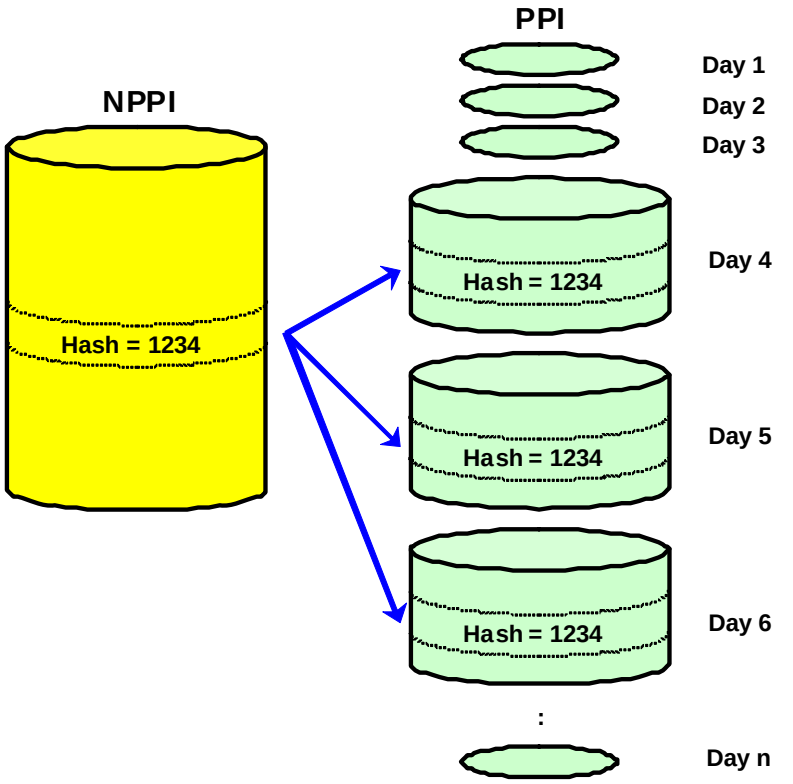
Better join performance is possible by eliminating partitions via query constraints.

NPPI to PPI Join – Sliding Window (cont.)

The example on the facing page illustrates that one block from the NPPI table is joined to the next 3 partitions in the PPI table. The window “slides” from the first 3 partitions to the next 3 partitions. The NPPI table is read again from the beginning each time the window slides in the PPI table.

A more significant degradation of join performance occurs when there are more non-excluded partitions than data block buffers. Assume enough memory for 20 data blocks and a table with 100 partitions. Then the "sliding window" technique lives up to its name. The first 20 partitions are processed as above, then the non-PPI table is read again as the "window" slides down to partitions 21 through 40. A total of five passes through the non-PPI table are required, and the join will take on the order of five times longer than a comparable join where the window covers the entire table. (This assumes that the non-PPI table is roughly the same size as the PPI table.)

NPPI to PPI Join – Sliding Window (cont.)



In continuing this simple example, the NPPI table is joined to the next 3 days in the PPI table.

The NPPI table is read again from the beginning each time the window slides in the PPI table.

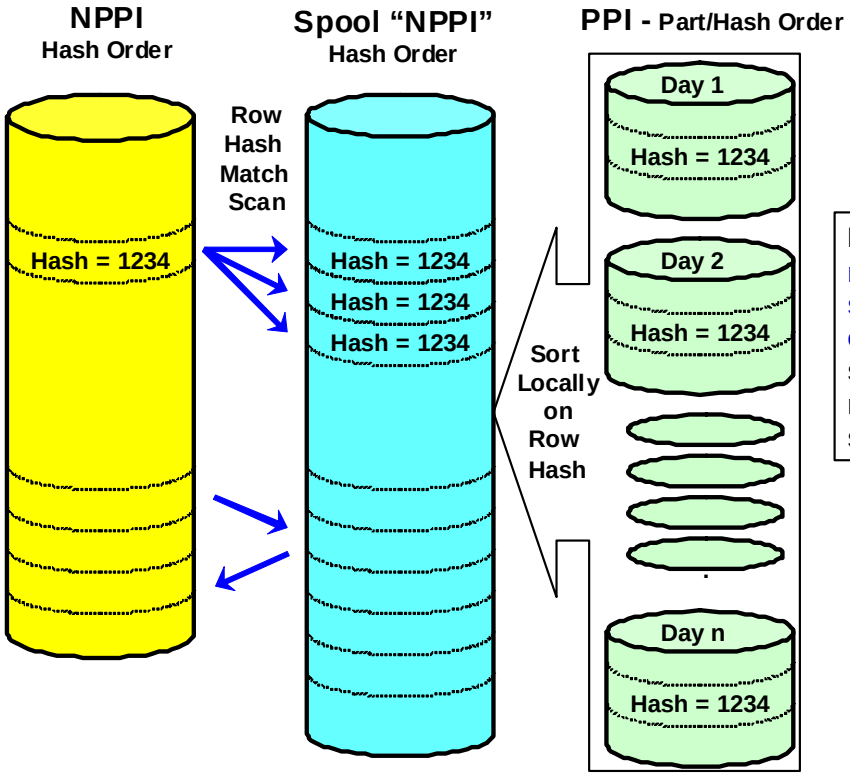
NPPI to PPI Join – Hash Ordered Spool File Join

In some cases, it will be more efficient to create a spool file that is hash ordered from the surviving PPI partitions rather than to use the sliding window technique.

Creation of hash ordered spool file is done relatively efficiently using an n-way merge rather than a full sort.

As always, the cost-based optimizer will figure out the most efficient execution plan using statistics and assessment of various join possibilities.

NPPI to PPI – Hash Ordered Spool File Join



In some cases, it may be more efficient to create a spool file that is hash ordered from the surviving PPI partitions rather than to use the sliding window technique.

PPI to PPI Join – Rowkey-Based Join

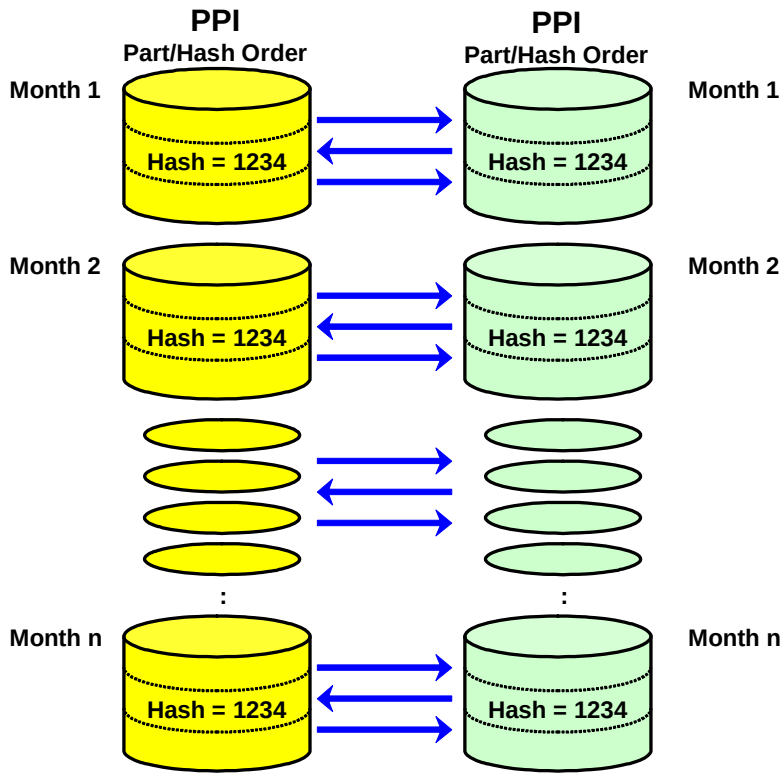
Direct merge joins of two PPI tables are available as an optimizer choice when the tables have the same PI and identical partitioning expressions, and all PI columns and all partitioning columns are specified as equality join terms. This is referred to as a **rowkey-based merge join**. In this case, the rows of the two tables will be ordered in the same way, allowing a merge join without redistribution or sorting of the rows.

If the partitioning column is part of the Primary Index (PI), then it is possible, and usually advantageous, to define all the tables with the same PI with identical partitioning expressions. This allows for a rowkey-based merge join.

If NPPI table is placed into spool (redistributed or duplicated) and this spool is joined with PPI table, then spool can be partitioned the same as the PPI table allowing for this fast join.

The performance characteristics of a traditional merge join (on matching primary indexes) and a rowkey-based merge join will be approximately the same.

PPI to PPI Join – Rowkey Based Join



When joining two PPI tables that have equivalent partition definitions, a **Row Key Match Scan within partitions** can be used to join the tables together.

PPI to PPI Join – Unmatched Partitions

If the partitioning column is not part of the Primary Index (PI), then it may be impossible to similarly partition the other tables having the same PI, as the partitioning column may not exist in the other table(s).

An expensive situation may occur when both tables are partitioned, but have different partitioning expressions. In this case, there is potentially a “sliding window” advancing through both tables.

The following discussion summarizes the number of disk reads for the various types of joins.

Given the following:

d1 = the number of data blocks in table 1
d2 = the number of data blocks in table 2

p1 = the number of non-excluded partitions in table 1
p2 = the number of non-excluded partitions in table 2

k1 = the number of partitions which can be held in memory for table 1
k2 = the number of partitions which can be held in memory for table 2

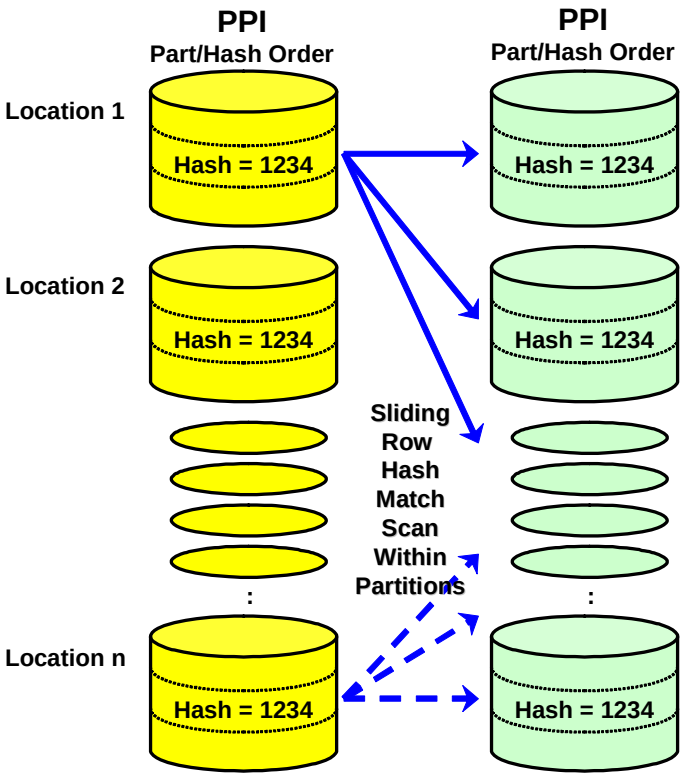
The number of disk reads required for the various types of joins are:

neither table partitioned: d1 + d2
second table partitioned: (p2/k2 * d1) + d2
both tables partitioned: (p2/k2 * d1) + (p1/k1 * d2)

If p/k is less than one, some available memory won't be needed, and a value of one is used for p/k. For one partitioned table, p/k must be small (maybe 4 at the most) for performance to be attractive, unless the query conditions permit a large number of partitions to be excluded. (Without trying to address every join costing consideration, reading the non-PPI table twice, say, and reading 40% of the PPI table due to partition elimination, may be a less expensive operation than reading each table once, as in the non-PPI situation.)

For two partitioned tables, p1/k1 and p2/k2 must both be small (maybe 2 or 3 maximum) for performance to be attractive, unless a large number of partitions can be excluded.

PPI to PPI Join – Unmatched Partitions



When both tables are partitioned, but have different partitioning expressions, there's potentially a "sliding window" advancing through both tables.

Additional Join Options with PPI

The optimizer has other options than the “sliding window” join. As usual, the optimizer estimates the cost of all reasonable alternatives, and chooses the least expensive solution. Given the importance of the term p/k in the previously mentioned formulas, it is important that the optimizer have a realistic estimate of the number of non-excluded partitions.

This is the reason that the earlier examples that partitioned on “store_id” indicated that the RANGE_N example was better than the example that used the column directly. In the RANGE_N example, the optimizer knows that there are a maximum of ten “store_id” partitions with rows, and will use ten or a smaller number as the value of p. When the column was used directly, the maximum number of partitions was 65,535, and the optimizer may use a value much larger than ten when estimating p, especially if statistics haven’t been collected on the “store_id” column.

If one table is partitioned and the other isn't, the optimizer has two viable alternatives in addition to the “sliding window” join approach. One is to spool the partitioned table to a non-partitioned spool file, after which the join is between two non-partitioned relations. The other option, not always available, is to spool the non-PPI table to a partitioned spool file, then directly join two identically partitioned relations. The partitioned spool file option is available only if the query specifies an equality condition on every column that is referenced in the partitioning expression.

If both tables are partitioned, it may be possible to spool one of the tables to a spool file with partitioning identical to the other table. It may also be cost-effective to spool both tables to non-partitioned spool files.

Some join algorithms do not change (e.g., when algorithm is not dependent on the table being in PI hash order), but usually benefit from a reduction in the number of blocks read because of partition elimination.

- Product join
- Classical hash join
- Nested join
- Row ID join

Note: A nested join would not generally be selected without a secondary index on inner table (with or without PPI).

Additional Join Options with PPI

The optimizer has other options than the "sliding window" join.

If one table is partitioned and the other isn't, alternatives to the "sliding window" join approach.

- **One is to spool the partitioned table to a non-partitioned spool file, after which the join is between two non-partitioned relations.**
- **Spool the NPPI table to a partitioned spool file (matching the partitioning of the PPI table), then directly join two identically-partitioned relations.**
  - **This option is available only if the query specifies an equality condition on every column that is referenced in the partitioning expression.**

If both tables are partitioned, possibilities are …

- **Spool one of the tables to a spool file with partitioning identical to the other table.**
- **Spool both tables to non-partitioned spool files.**

As usual, the optimizer estimates the cost of all reasonable alternatives, and chooses the least expensive solution.

Join Processing Summary

The facing page summarizes many of the key points regarding Join Processing. There are three key factors:

1. Physical design choices

2. Availability of COLLECTed STATISTICS for design

3. Quality of SQL coding

Database design is the key to efficient Join Processing because the Optimizer bases its plans on Primary and Secondary Indexes.

COLLECTed STATISTICS are also vital since the Optimizer needs to know table Row Counts as well as Rows per Value. Make sure that the Optimizer always has fresh STATISTICS since data demographics change over time.

MAKE SURE you write efficient SQL code. But remember, even the best SQL code cannot compensate for poor database design choices.

Join Processing Summary

Inefficient joins result from:

- Poor physical design choices
  - Lack of indexes
  - Inappropriate indexes
- Stale or missing Collected Statistics
- Inefficient SQL code
  - Poor SQL code can degrade performance on a good database design.
  - Good SQL code cannot compensate for a poor database design.

The system bases join planning on:

- Primary and Secondary Indexes
- Estimated number of rows in each subtable
- Estimated ratio of table rows per index value

COLLECTed STATISTICS may improve join performance.

The fastest merge joins are based on matching Primary Indexes.

Data demographics change over time.

Join plans for the same tables change as demographics changes.

Revisit ALL index (Primary and Secondary) choices regularly. Make sure they are still serving you well.

Module 28: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 28: Review Questions

1. The best way to be sure what type of Join will occur is to use the EXPLAIN facility.
   a. True
   b. False

2. When two tables are to be Merge Joined, which is the best case of the following scenarios :
   a. The Join column is not a Primary Index of either table.
   b. The Join column is the Primary Index of one of the tables.
   c. The Join column(s) is the Primary Index of both tables.
   d. None of the above

3. Match the four join plans with its correct description.

__ Product Join	a.	Most efficient types of Join; the only type of join that doesn't always use all of the AMPs. The number of AMPs involved will vary.
__ Merge Join	b.	Based on set subtraction; used for finding rows that don't have a matching row in the other table. Queries with the NOT IN and EXCEPT operator lead to this type of join.
__ Nested Join	c.	Every qualifying row of one table is compared to every qualifying row in the other table. Rows that match on their WHERE conditions are then saved.
__ Exclusion Join	d.	Commonly done when the join condition is based on equality. Efficient because every row is not compared with every row in other table.

Module 28: Review Questions (cont.)

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 28: Review Questions (cont.)

Fill in the blanks.

4. When joining two PPI tables that are not partitioned in the same manner, a technique available to the optimizer is referred to as the _____ window.

5. A direct merge join of two PPI tables when the tables have the same PI and identical partitioning expressions is referred to as a _____ - based merge join.

6. The term _____ _____ refers to an automatic optimization in which the optimizer determines, based on query conditions, that some partitions can be skipped.

Notes

Module 29

Explains of Joins and Index Choices

After completing this module, you will be able to:

- **Interpret the EXPLAINS of Nested, Merge, and Product Joins.**
- **Develop Union solutions.**
- **Explain the importance of Join Accesses in Index selection.**
- **Use the Index choice guidelines.**
- **Use the Teradata Index Wizard to analyze secondary index selections.**

Notes

Table of Contents

Join Diagramming 29-4
Nested Join 29-6
Merge Join (Matching Primary Indexes) 29-8
Hash Join 29-10
Visual Explain (Hash Join) 29-12
Merge Join (Joining a Table to Itself) 29-14
Three-Table Join 29-16
  Three-Table Join (cont.) 29-18
Product Join 29-20
  Product Join (cont.) 29-22
A “UNION” Solution 29-24
  A “UNION” Solution (cont.) 29-26
Cartesian Product Join 29-28
Exclusion Join 29-30
Example of PPI to PPI Join 29-32
Join ACCESS 29-34
Exercise 5 – Sample 29-36
Exercise 5 – Making Final Index Choices 29-38
  Exercise 5 – Making Final Index Choices (cont.) 29-40
  Exercise 5 – Making Final Index Choices (cont.) 29-42
  Exercise 5 – Making Final Index Choices (cont.) 29-44
  Exercise 5 – Making Final Index Choices (cont.) 29-46
  Exercise 5 – Making Final Index Choices (cont.) 29-48
Teradata Index Wizard 29-50
Teradata Index Wizard – Main Window 29-52
  Defining and Using Workloads with Index Wizard 29-52
  Additional Workload Functions 29-52
Teradata Index Wizard – Index Analysis 29-54
Teradata Index Wizard – Index Analysis Results 29-56
  Creating a Workload to Analyze using “DUMP EXPLAIN” 29-56
Teradata Index Wizard – Partition Analysis 29-58
Teradata Index Wizard – Partition Analysis Results 29-60
Teradata Index Wizard – Reports 29-62
  Workload Reports 29-62
  Analysis Reports 29-62
  Index Recommendation Report 29-62
Teradata Index Wizard – Validation 29-64
  Index Validation 29-64
Teradata Index Wizard – Validation (View Graph) 29-66
Teradata Index Wizard – Creation 29-68
  Executing Recommendations 29-68
Summary – Index Choice Guidelines 29-70
Module 29: Review Questions 29-72

Join Diagramming

Join Diagramming can save you from costly coding errors by helping you understand what occurs when Teradata performs a Join. The example on the facing page illustrates a two-table Join.

Four distinct pieces of information are provided in Join Diagrams. Three of these are required and one is optional. They are:

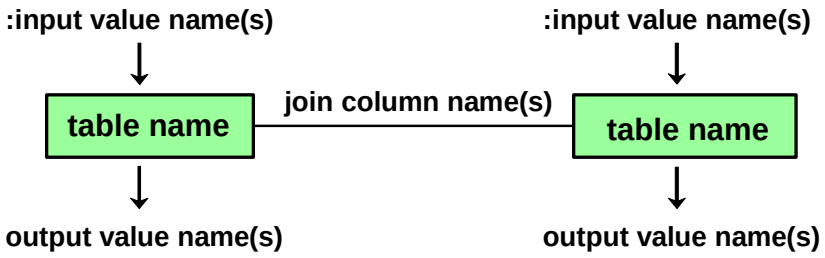
- Table Names appear in the boxes to indicate which tables are being joined. 2 to 64 tables can participate in a Join operation. (Required)

- Join Column Name(s) indicate the column(s) used to do the Join, and are based on the same domain with an equality condition. They appear above the lines between the tables. At least one Join Column Name is between every pair of tables. (Required)

- Input Value Name(s) provide Set Selection values. They appear above the boxes containing the table names. They can be constants, host variables, or macro parameters which appear in the WHERE clause of the SQL statement. (Optional)

- Output Value Name(s) indicate which column(s), aggregate(s), or expression(s) will be output from a particular table. They appear below the boxes containing the table names and there is at least one for the query. (Required)

Join Diagramming



:INPUT VALUE NAME(S) – Optional

Constants, host variables, or macro parameters that supply Set Selection values.

TABLE NAME

From 2 to 64 tables may participate in a Join operation.

JOIN COLUMN NAME(S)

Based on the same domain and an equality condition.

OUTPUT VALUE NAME(S)

Specify at least one column, aggregate or expression.

Nested Join

In this module, you will study various Joins. You will see different types of information relating to Joins. These are presented in the same order that you would use when designing a Teradata query. You would:

- State the query
- Diagram the query
- Code the query
- EXPLAIN the query

You must understand what you are asking the system to do before you attempt to understand how it does it.

In this example, Step 1 is not a locking step as in the other EXPLAINs that you have seen since it does not involve all AMPs. The use of the two UPIs allows the locks to be acquired within the AMP step and immediately released.

The tables used in this Join were created as follows:

```
CREATE SET TABLE TFACT.Employee, FALLBACK
   (Employee_Number        INTEGER NOT NULL,
    Location_Number        INTEGER,
    Dept_Number            INTEGER,
    Emp_Mgr_Number         INTEGER,
    Job_Code               INTEGER,
    Last_Name              CHAR(20),
    First_Name             VARCHAR(20),
    Salary_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX ( Employee_Number )
INDEX ( Job_Code )
INDEX ( Dept_Number );

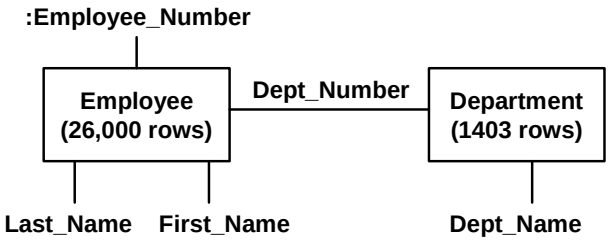
CREATE SET TABLE TFACT.Department, FALLBACK
   ( Dept_Number           INTEGER NOT NULL,
    Dept_Name              CHAR(20 NOT NULL,
    Dept_Mgr_Number        INTEGER,
    Budget_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX ( Dept_Number );
```

The Employee table has 26,000 rows and the Department table has 1403 rows. Statistics were collected on the primary indexes and any join columns.

Nested Join

QUERY

```
EXPLAIN
SELECT      Last_Name,
            First_Name,
            Dept_Name
FROM        Employee E
INNER JOIN  Department D
ON          E.Dept_Number = D.Dept_Number
WHERE       E.Employee_Number = 102001;
```



EXPLANATION

12.0 EXPLAIN

1) First, we do a single-AMP JOIN step from TFACT.E by way of the unique primary index "TFACT.E.Employee_Number = 102001" with a residual condition of ("NOT (TFACT.E.Dept_Number IS NULL)"), which is joined to TFACT.D by way of the unique primary index "TFACT.D.Dept_Number = TFACT.E.Dept_Number". TFACT.E and TFACT.D are joined using a nested join, with a join condition of ("(1=1)"). The input table TFACT.E will not be cached in memory, but it is eligible for synchronized scanning. The result goes into Spool 1 (one-amp), which is built locally on that AMP. The size of Spool 1 is estimated with high confidence to be 1 row (69 bytes). The estimated time for this step is 0.01 seconds.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.01 seconds.

Merge Join (Matching Primary Indexes)

For this example and those that follow, make sure that you understand the Narrative, Join Diagram and Query before looking at the EXPLAIN output.

You must understand what you are asking the system to do before you attempt to understand how it does it.

The EXPLAIN tells you that this is a Merge Join. Note that there is no redistribution of rows or sorting which means that Merge Join Plan is being used. In this example, the Join Columns are the Primary Indexes of both tables. No redistribution or sorting is needed since the rows are already on the proper AMPs and in the proper order for Joining. This is an example of Merge Join Plan M1 from the previous module.

The tables used in this Join were created as follows:

```
CREATE SET TABLE TFACT.Employee, FALLBACK
   (Employee_Number        INTEGER NOT NULL,
    Location_Number        INTEGER,
    Dept_Number            INTEGER,
    Emp_Mgr_Number         INTEGER,
    Job_Code               INTEGER,
    Last_Name              CHAR(20),
    First_Name             VARCHAR(20),
    Salary_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Employee_Number)
INDEX (Job_Code)
INDEX (Dept_Number);

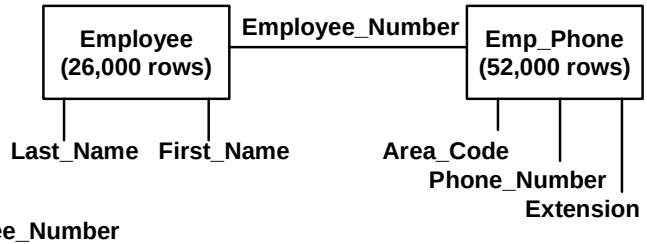
CREATE SET TABLE TFACT.Emp_Phone, FALLBACK
   (Employee_Number        INTEGER,
    Area_Code              SMALLINT,
    Phone_Number           INTEGER,
    Extension              INTEGER)
PRIMARY INDEX (Employee_Number);
```

The Employee table has 26,000 rows and the Employee_Phone table has 52,000 rows. Statistics were collected on the primary indexes and any join columns.

Merge Join (Matching Primary Indexes)

QUERY

```
EXPLAIN
SELECT      Last_Name, First_Name,
            Area_Code, Phone_Number,
            Extension
FROM        Employee E
INNER JOIN  Emp_Phone P
ON          E.Employee_Number = P.Employee_Number
ORDER BY    1, 2;
```



EXPLANATION

12.0 EXPLAIN

: (*Locking steps*)

4) We do an all-AMPs JOIN step from TFACT.E by way of a RowHash match scan with no residual conditions, which is joined to TFACT.P. TFACT.E and TFACT.P are joined using a merge join, with a join condition of ("TFACT.E.Employee_Number = TFACT.P.Employee_Number"). The input table TFACT.E will not be cached in memory, but it is eligible for synchronized scanning. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1 (TFACT.E.Last_Name, TFACT.E.First_Name). The result spool file will not be cached in memory. The size of Spool 1 is estimated with low confidence to be 52,000 rows. The estimated time for this step is 0.12 seconds.

5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.12 seconds.

Hash Join

The EXPLAIN output tells you that this is Hash Join, however, distribution is of the Employee table is required. The rows from the employee table are **"redistributed (hashed) across all AMPs".** The rows from the department table are placed into memory for each AMP and are joined to the employee table via a single partition hash join.

The tables used in this Join were created as follows:

```
CREATE SET TABLE TFACT.Employee, FALLBACK
   (Employee_Number        INTEGER NOT NULL,
    Location_Number        INTEGER,
    Dept_Number            INTEGER,
    Emp_Mgr_Number         INTEGER,
    Job_Code               INTEGER,
    Last_Name              CHAR(20),
    First_Name             VARCHAR(20),
    Salary_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Employee_Number)
INDEX (Job_Code)
INDEX (Dept_Number);

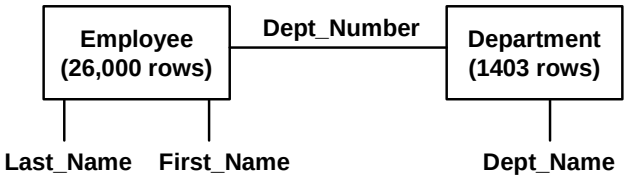
CREATE SET TABLE TFACT.Department, FALLBACK
   ( Dept_Number           INTEGER NOT NULL,
    Dept_Name              CHAR(20) NOT NULL,
    Dept_Mgr_Number        INTEGER,
    Budget_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Dept_Number);
```

The Employee table has 26,000 rows and the Department table has 1403 rows. Statistics were collected on the primary indexes and any join columns.

Hash Join

QUERY

```
EXPLAIN
SELECT      Last_Name,
            First_Name,
            Dept_Name
FROM        Employee E
INNER JOIN  Department D
ON          E.Dept_Number = D.Dept_Number
ORDER BY    1, 2;
```



EXPLANATION

12.0 EXPLAIN

: (*Locking steps)*

4) We do an all-AMPs RETRIEVE step from TFACT.D by way of an all-rows scan with no residual conditions into Spool 2 (all_amps), which is duplicated on all AMPs. The size of Spool 2 is estimated with high confidence to be 19,642 rows (726,754 bytes). The estimated time for this step is 0.02 seconds.

5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.E by way of an all-rows scan. Spool 2 and TFACT.E are joined using a single partition hash_join, with a join condition of ("TFACT.E.Dept_Number = Dept_Number"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1 (TFACT.E.Last_Name, TFACT.E.First_Name). The size of Spool 1 is estimated with low confidence to be 26,000 rows (1,794,000 bytes). The estimated time for this step is 0.08 seconds.

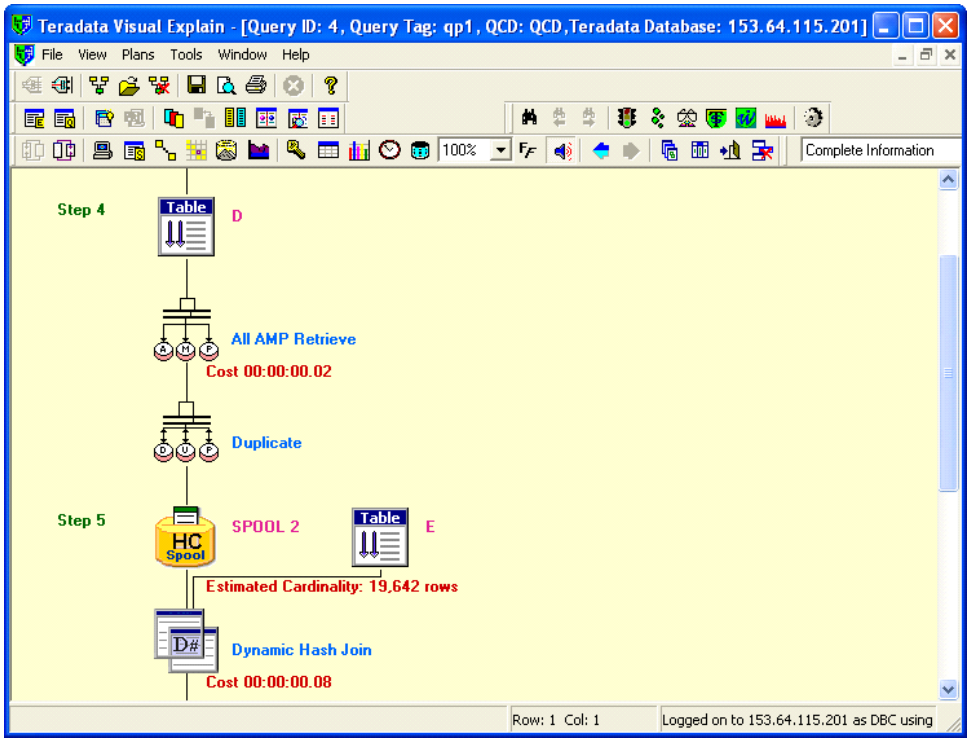
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.10 seconds.

Visual Explain (Hash Join)

The facing page contains output from the Visual Explain facility. This output represents the Join Query on the previous page. This is a way to visually display the Joins, which will help you to understand them more fully. Visual Explain is especially helpful when you are dealing with complex Joins.

Visual Explain (Hash Join)



Merge Join (Joining a Table to Itself)

The facing page shows a Self Join involving the employee table. Two aliases for the employee table (M for manager and E for employee) are defined in the SQL query. This helps the Optimizer treat the single employee table as two tables.

The EXPLAIN output shows that the rows from table alias E are redistributed and sorted before being Joined to the rows from table alias M using a Merge Join.

As before, the table used in this Join was created as follows:

The tables used in this Join were created as follows:

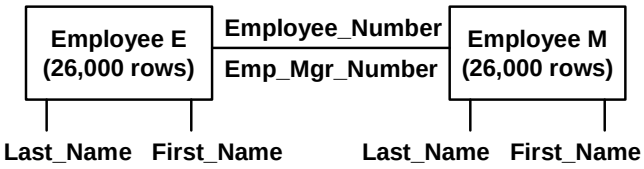
```
CREATE SET TABLE TFACT.Employee, FALLBACK
   (Employee_Number        INTEGER NOT NULL,
    Location_Number        INTEGER,
    Dept_Number            INTEGER,
    Emp_Mgr_Number         INTEGER,
    Job_Code               INTEGER,
    Last_Name              CHAR(20),
    First_Name             VARCHAR(20),
    Salary_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Employee_Number)
INDEX (Job_Code)
INDEX (Dept_Number);
```

The Employee table has 26,000 rows. Statistics were collected on the primary indexes and any join columns.

Merge Join (Joining a Table to Itself)

QUERY

```
EXPLAIN
SELECT    M.Last_Name, M.First_Name,
          E.Last_Name, E.First_Name
FROM      Employee M
INNER JOIN Employee E
ON        M.Employee_Number = E.Emp_Mgr_Number
ORDER BY  1, 3;
```



EXPLANATION 12.0 EXPLAIN

: *(Locking step)*

2) Next, we lock TFACT.E for read.

3) We do an all-AMPs RETRIEVE step from TFACT.E by way of an all-rows scan with a condition of ("NOT (TFACT.E.Emp_Mgr_Number IS NULL)") into Spool 2 (all_amps), which is redistributed by hash code of (TFACT.E.Emp_Mgr_Number) to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with high confidence to be 26,000 rows (1,170,000 bytes). The estimated time for this step is 0.05 seconds.

4) We do an all-AMPs JOIN step from TFACT.M by way of a RowHash match scan with no residual conditions, which is joined to Spool 2 (Last Use) by way of a RowHash match scan. TFACT.M and Spool 2 are joined using a merge join, with a join condition of ("TFACT.M.Employee_Number = Emp_Mgr_Number"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1 (TFACT.M.Last_Name, TFACT.E.Last_Name). The size of Spool 1 is estimated with low confidence to be 26,000 rows (2,938,000 bytes). The estimated time for this step is 0.08 seconds.

:

Three-Table Join

So far, you have only seen the EXPLAINs of Joins involving two tables. In this three-table Join, you will be able to see how the Optimizer chooses a Join Plan that involves a series of two two-table Joins.

The purpose of the query is to display the names of all employees, their department names and their job descriptions. The Join diagram shows that the job, employee and department tables will all be involved in the Join.

As before, the tables used in this Join were created as follows:

The tables used in this Join were created as follows:

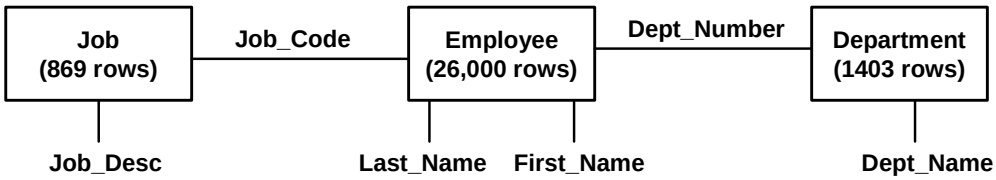
```
CREATE SET TABLE TFACT.Employee, FALLBACK
   (Employee_Number        INTEGER NOT NULL,
    Location_Number        INTEGER,
    Dept_Number            INTEGER,
    Emp_Mgr_Number         INTEGER,
    Job_Code               INTEGER,
    Last_Name              CHAR(20),
    First_Name             VARCHAR(20),
    Salary_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Employee_Number)
INDEX (Job_Code)
INDEX (Dept_Number);

CREATE SET TABLE TFACT.Department, FALLBACK
   ( Dept_Number           INTEGER NOT NULL,
    Dept_Name              CHAR(20) NOT NULL,
    Dept_Mgr_Number        INTEGER,
    Budget_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Dept_Number);

CREATE SET TABLE TFACT.Job, FALLBACK
   (Job_Code               INTEGER NOT NULL,
    Job_Desc               CHAR(20) NOT NULL DEFAULT 'Job Description    ')
UNIQUE PRIMARY INDEX (Job_Code)
INDEX (Job_Desc);
```

The Employee table has 26,000 rows, the Department table has 1403 rows, and the Job table has 869 rows. Statistics were collected on the primary indexes and any join columns.

Three-Table Join



QUERY

```
EXPLAIN
SELECT      E.Last_Name,
            E.First_Name,
            D.Dept_Name,
            J.Job_Desc
FROM        Employee E
INNER JOIN  Department D  ON  E.Dept_Number = D.Dept_Number
INNER JOIN  Job J         ON  E.job_code = J.job_code
ORDER BY    3, 1, 2;
```

EXPLAIN output on following page.

Three-Table Join (cont.)

The EXPLAIN output shows that the duplication of rows is done in parallel. The rows from the job and department tables are duplicated on all AMPs and then sorted. The rows from the employee table are first joined to the Job table (Spool) via a hash join.

The query and table definitions are repeated for your convenience:

```
EXPLAIN
SELECT          Last_Name,
                First_Name,
                Dept_Name,
                Job_Desc
FROM            Employee E
INNER JOIN      Department D      ON E.Dept_Number = D.Dept_Number
INNER JOIN      Job J             ON  E.Job_Code = J.Job_Code
ORDER BY        3, 1, 2;
```

```
CREATE SET TABLE TFACT.Employee, FALLBACK
   (Employee_Number        INTEGER NOT NULL,
    Location_Number        INTEGER,
    Dept_Number            INTEGER,
    Emp_Mgr_Number         INTEGER,
    Job_Code               INTEGER,
    Last_Name              CHAR(20),
    First_Name             VARCHAR(20),
    Salary_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Employee_Number)
INDEX (Job_Code)
INDEX (Dept_Number);

CREATE SET TABLE TFACT.Department, FALLBACK
   ( Dept_Number           INTEGER NOT NULL,
    Dept_Name              CHAR(20) NOT NULL,
    Dept_Mgr_Number        INTEGER,
    Budget_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Dept_Number);

CREATE SET TABLE TFACT.Job, FALLBACK
   (Job_Code               INTEGER NOT NULL,
    Job_Desc               CHAR(20) NOT NULL DEFAULT 'Job Description    ')
UNIQUE PRIMARY INDEX (Job_Code)
INDEX (Job_Desc);
```

Three-Table Join (cont.)

EXPLANATION 12.0 EXPLAIN

: *(Locking steps)*
5) We execute the following steps in parallel.
   1) We do an all-AMPs RETRIEVE step from TFACT.D by way of an all-rows scan with no residual conditions into Spool 2 (all_amps), which is duplicated on all AMPs. The size of Spool 2 is estimated with high confidence to be 19,642 rows (726,754 bytes). The estimated time for this step is 0.02 seconds.
   2) We do an all-AMPs RETRIEVE step from TFACT.J by way of an all-rows scan with no residual conditions into Spool 3 (all_amps), which is duplicated on all AMPs. The size of Spool 3 is estimated with high confidence to be 12,166 rows (450,142 bytes). The estimated time for this step is 0.01 seconds.
6) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.E by way of an all-rows scan with a condition of ("NOT (TFACT.E.Job_Code IS NULL)"). Spool 2 and TFACT.E are joined using a single partition hash_ join, with a join condition of ("TFACT.E.Dept_Number = Dept_Number"). The result goes into Spool 4 (all_amps), which is built locally on the AMPs. The size of Spool 4 is estimated with low confidence to be 26,000 rows (1,690,000 bytes). The estimated time for this step is 0.04 seconds.
7) We do an all-AMPs JOIN step from Spool 3 (Last Use) by way of an all-rows scan, which is joined to Spool 4 (Last Use) by way of an all-rows scan. Spool 3 and Spool 4 are joined using a single partition hash join, with a join condition of ("Job_Code = Job_Code"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1 (TFACT.D.Dept_Name, TFACT.E.Last_Name, TFACT.E.First_Name). The size of Spool 1 is estimated with low confidence to be 26,000 rows (3,822,000 bytes). The estimated time for this step is 0.08 seconds.
8) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.14 seconds.

Product Join

The query in this example results in a Product Join involving the employee and department tables. The query is designed to return the names and department names of all employees who are either workers or managers. Managers may be listed both as an employee of a department and once as a manager of a department (if they are employed by a department other than the one which they manage).

The EXPLAIN tells you that this is a Product Join. The Join makes almost 40,000,000 comparisons and uses considerable system resources.

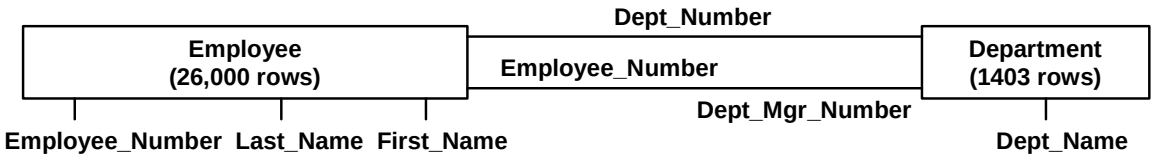
As before, the tables used in this Join were created as follows:

```
CREATE SET TABLE TFACT.Employee, FALLBACK
   (Employee_Number        INTEGER NOT NULL,
    Location_Number        INTEGER,
    Dept_Number            INTEGER,
    Emp_Mgr_Number         INTEGER,
    Job_Code               INTEGER,
    Last_Name              CHAR(20),
    First_Name             VARCHAR(20),
    Salary_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Employee_Number)
INDEX (Job_Code)
INDEX (Dept_Number);

CREATE SET TABLE TFACT.Department, FALLBACK
   ( Dept_Number           INTEGER NOT NULL,
    Dept_Name              CHAR(20) NOT NULL,
    Dept_Mgr_Number        INTEGER,
    Budget_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Dept_Number);
```

The Employee table has 26,000 rows and the Department table has 1403 rows. Statistics were collected on the primary indexes and any join columns.

Product Join




```
QUERY
EXPLAIN
SELECT      D.Dept_Name,
            E.Employee_Number,
            E.Last_Name,
            E.First_Name
FROM        Employee E
INNER JOIN  Department D
ON          E.Dept_Number = D.Dept_Number
OR          E.Employee_Number = D.Dept_Mgr_Number
ORDER BY    1, 2, 3, 4;
```

EXPLAIN output on following page.

Product Join (cont.)

The EXPLAIN tells you that this is a Product Join. This product join uses considerable system resources for even small tables.

The query and table definitions are repeated for your convenience:

```
EXPLAIN
SELECT          D.Dept_Name,
                E.Employee_Number,
                E.Last_Name,
                E.First_Name
FROM            Employee E
INNER JOIN      Department D
ON              E.Dept_Number = D.Dept_Number
OR              E.Employee_Number = D.Dept_Mgr_Number
ORDER BY        1, 2, 3, 4;
```

```
CREATE SET TABLE TFACT.Employee, FALLBACK
   (Employee_Number        INTEGER NOT NULL,
    Location_Number        INTEGER,
    Dept_Number            INTEGER,
    Emp_Mgr_Number         INTEGER,
    Job_Code               INTEGER,
    Last_Name              CHAR(20),
    First_Name             VARCHAR(20),
    Salary_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Employee_Number)
INDEX (Job_Code)
INDEX (Dept_Number);

CREATE SET TABLE TFACT.Department, FALLBACK
   ( Dept_Number           INTEGER NOT NULL,
    Dept_Name              CHAR(20) NOT NULL,
    Dept_Mgr_Number        INTEGER,
    Budget_Amount          DECIMAL(10,2))
UNIQUE PRIMARY INDEX (Dept_Number);
```

The Employee table has 26,000 rows and the Department table has 1403 rows. Statistics were collected on the primary indexes and any join columns.

Product Join (cont.)

EXPLANATION 12.0 EXPLAIN

---

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.D.
2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.E.
3) We lock TFACT.D for read, and we lock TFACT.E for read.
4) We do an all-AMPs RETRIEVE step from TFACT.D by way of an all-rows scan with no residual conditions into Spool 2 (all_amps), which is duplicated on all AMPs. The size of Spool 2 is estimated with high confidence to be 19,642 rows (805,322 bytes). The estimated time for this step is 0.02 seconds.
5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.E by way of an all-rows scan with no residual conditions. Spool 2 and TFACT.E are joined using a product join, with a join condition of ("(TFACT.E.Dept_Number = Dept_Number) OR (TFACT.E.Employee_Number = Dept_Mgr_Number)"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1 (TFACT.D.Dept_Name, TFACT.E.Employee_Number, TFACT.E.Last_Name, TFACT.E.First_Name). The size of Spool 1 is estimated with low confidence to be 27,403 rows (3,809,017 bytes). The estimated time for this step is 1.56 seconds.
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 1.58 seconds.

A “UNION” Solution

This example illustrates that sometimes there are more efficient ways for coding an SQL query. The query on the facing page is designed to yield the same answer set as the previous SQL query.

The query is:

```
EXPLAIN
SELECT   D.Dept_Name, E.Employee_Number, E.Last_Name, E.First_Name
FROM     Employee E INNER JOIN Department D
ON       E.Dept_Number = D.Dept_Number
UNION
SELECT   D.Dept_Name, E.Employee_Number, E.Last_Name, E.First_Name
FROM     Employee E INNER JOIN Department D
ON       E.Employee_Number = D.Dept_Mgr_Number
ORDER    1, 2, 3, 4;
```

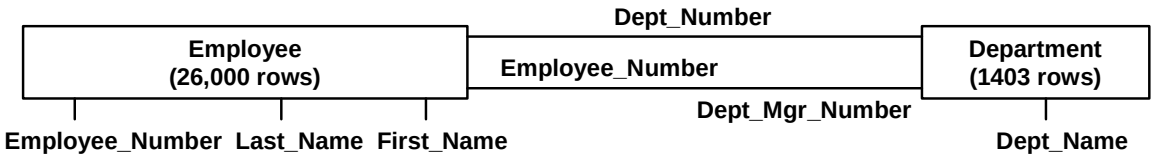
This example illustrates that sometimes there are more efficient ways for coding an SQL query. The query on the facing page is designed to yield the same answer set as the previous SQL query. Both queries return 27,342 rows and exactly the same answer set.

A key to understanding the UNION and UNION ALL is that a UNION will remove any duplicate rows from the output and UNION ALL will keep all duplicate rows.

However, if the queries were changed slightly and the Employee_Number column was not included, the first statement (Product Join via the OR) would still return 27,342 rows. However, the UNION solution would only return 24,035 rows.

This discussion is continued on the following text page and illustrates an example where a PRODUCT JOIN and a UNION might not produce the same result set.

A "UNION" Solution



QUERY

```
EXPLAIN  SELECT     D.Dept_Name, E.Employee_Number, E.Last_Name, E.First_Name
         FROM       Employee E INNER JOIN Department D
         ON         E.Dept_Number = D.Dept_Number
         UNION
         SELECT     D.Dept_Name, E.Employee_Number, E.Last_Name, E.First_Name
         FROM       Employee E INNER JOIN Department D
         ON         E.Employee_Number = D.Dept_Mgr_Number
         ORDER BY   1, 2, 3, 4;
```

EXPLAIN output on following page.

A "UNION" Solution (cont.)

A simple example is provided with the following data in Employee and the Department tables. Assume that the Employee table has two different employees (different Employee Numbers) that have the same name "Paul Winters". The difference in the three examples is that a UNION deletes any duplicate rows and a UNION ALL keeps all duplicate rows.

Employee Rows:

Dept_Number	Employee_Number	Last_Name	First_Name
1104	101056	Winters	Paul
1104	101078	Winters	Paul

Department Row:

Dept_Number	Dept_Name	Dept_Mgr_Number
1104	Education	101078

In essence, the following three queries will each return a different answer set.

```
SELECT      D.Dept_Name, E.Last_Name, E.First_Name
FROM        Employee E
INNER JOIN  Department D
ON          E.Dept_Number = D.Dept_Number
OR          E.Employee_Number = D.Dept_Mgr_Number;
```

Returns:

Dept_Name	Last_Name	First_Name
Education	Winters	Paul
Education	Winters	Paul

```
SELECT    D.Dept_Name, E.Last_Name, E.First_Name
FROM      Employee E INNER JOIN Department D
ON        E.Dept_Number = D.Dept_Number
UNION
SELECT    D.Dept_Name, E.Last_Name, E.First_Name
FROM      Employee E INNER JOIN Department D
ON        E.Employee_Number = D.Dept_Mgr_Number;
```

Returns:

Dept_Name	Last_Name	First_Name
Education	Winters	Paul

```
SELECT    D.Dept_Name, E.Last_Name, E.First_Name
FROM      Employee E INNER JOIN Department D
ON        E.Dept_Number = D.Dept_Number
UNION ALL
SELECT    D.Dept_Name, E.Last_Name, E.First_Name
FROM      Employee E INNER JOIN Department D
ON        E.Employee_Number = D.Dept_Mgr_Number;
```

Returns:

Dept_Name	Last_Name	First_Name
Education	Winters	Paul
Education	Winters	Paul
Education	Winters	Paul

A "UNION" Solution (cont.)

4) We do an all-AMPs RETRIEVE step from TFACT.D by way of an all-rows scan with no residual conditions into Spool 2 (all_amps), which is duplicated on all AMPs. The size of Spool 2 is estimated with high confidence to be 19,642 rows (726,754 bytes). The estimated time for this step is 0.02 seconds.
5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.E by way of an all-rows scan. Spool 2 and TFACT.E are joined using a single partition hash_ join, with a join condition of ("TFACT.E.Dept_Number = Dept_Number"). The result goes into Spool 1 (group_amps), which is redistributed by the hash code of (TFACT.E.First_Name, TFACT.E.Last_Name, TFACT.E.Employee_Number, TFACT.D.Dept_Name) to all AMPs. The size of Spool 1 is estimated with low confidence to be 26,000 rows (3,614,000 bytes). The estimated time for this step is 0.09 seconds.
6) We do an all-AMPs RETRIEVE step from TFACT.D by way of an all-rows scan with a condition of ("NOT (TFACT.D.Dept_Mgr_Number IS NULL)") into Spool 3 (all_amps), which is redistributed by the hash code of (TFACT.D.Dept_Mgr_Number) to all AMPs. Then we do a SORT to order Spool 3 by row hash. The size of Spool 3 is estimated with high confidence to be 1,403 rows (51,911 bytes). The estimated time for this step is 0.01 seconds.
7) We do an all-AMPs JOIN step from Spool 3 (Last Use) by way of a RowHash match scan, which is joined to TFACT.E by way of a RowHash match scan with no residual conditions. Spool 3 and TFACT.E are joined using a merge join, with a join condition of ("TFACT.E.Employee_Number = Dept_Mgr_Number"). The result goes into Spool 1 (group_amps), which is redistributed by the hash code of (TFACT.E.First_Name, TFACT.E.Last_Name, TFACT.E.Employee_Number, TFACT.D.Dept_Name) to all AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1 eliminating duplicate rows. The size of Spool 1 is estimated with low confidence to be 27,403 rows (3,809,017 bytes). The estimated time for this step is 0.06 seconds.
8) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.18 seconds.

Cartesian Product Join

The example on the facing page illustrates a Cartesian Product Join.

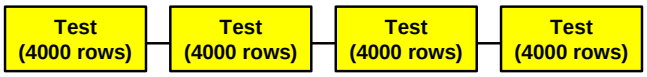
It confirms the admonishment that no SQL should be allowed into production until it has been EXPLAINed and approved.

The TEST table used in this Join was created as follows:

```
CREATE TABLE Test, NO FALLBACK
    ( col1   INTEGER NOT NULL
     ,col2   INTEGER
     ,col3   INTEGER
     ,col4   INTEGER
     ,col5   CHAR(20))
PRIMARY INDEX (col1);
```

The TEST table was populated with 4000 rows and statistics were collected on the primary index.

Cartesian Product Join




```
QUERY
EXPLAIN  SELECT * FROM Test A CROSS JOIN Test B
           CROSS JOIN Test C CROSS JOIN Test D;
```

```
:  (locking steps)                                                    12.0 EXPLAIN
3) We do an all-AMPs RETRIEVE step from TFACT.D by way of an all-rows scan with no residual conditions into Spool 2
   (all_amps), which is duplicated on all AMPs.  The size of Spool 2 is estimated with high confidence to be 56,000 rows
   (2,744,000 bytes).  The estimated time for this step is 0.04 seconds.
4) We do an all-AMPs JOIN step from TFACT.B by way of an all-rows scan with no residual conditions, which is joined
   to Spool 2 by  way of an all-rows scan.  TFACT.B and Spool 2 are joined using a product join, with a join condition of
   ("(1=1)").  The result goes into Spool 4 (all_amps), which is built locally on the AMPs.  The result spool file will not be
   cached in memory.  The size of Spool 4 is estimated with high confidence to be 16,000,000 rows (1,360,000,000
   bytes).  The estimated time for this step is 15.25 seconds.
5) We do an all-AMPs JOIN step from TFACT.A by way of an all-rows scan with no residual conditions, which is joined
   to Spool 2 (Last Use) by way of an all-rows scan.  TFACT.A and Spool 2 are joined using a product join, with a join
   condition of ("(1=1)").  The result goes into Spool 5 (all_amps), which is duplicated on all AMPs.  The result spool file
   will not be cached in memory.  The size of Spool 5 is estimated with high confidence to be 224,000,000 rows
   (19,040,000,000 bytes).  The estimated time for this step is 3 minutes and 3 seconds.
6) We do an all-AMPs JOIN step from Spool 4 (Last Use) by way of an all-rows scan, which is joined to Spool 5 (Last
   Use) by way of an all-rows scan.  Spool 4 and Spool 5 are joined using a product join, with a join condition of
   ("(1=1)").  The result goes into Spool 1 (group_amps), which is built locally on the AMPs.  The result spool file will
   not be cached in memory.  The size of Spool 1 is estimated with high confidence to be 256,000,000,000,000 rows (***
   bytes).  The estimated time for this step is 115,883 hours and 2 minutes.
7) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1.  The total estimated time is 115,883
   hours and 5 minutes.
```

Note: The optimizer recognizes that Spool 2 would be the same for aliases D and C and reuses Spool 2 (for alias C) in step 5.

Exclusion Join

The query on the facing page is designed to show all Sales Representatives who do not have any customers. Here the employee table is merged with the customer table. The Join Diagram shows that the employee table will be joined to the customer table via the Employee_Number (Sales_Emp_Number) column.

The EXPLAIN output tells you that the Optimizer will choose an Exclusion Merge Join. In fact, this is the same query used to illustrate the Exclusion Merge Join in the last module.

NOT IN operation uses three-value logic (equal, not equal, null). If the subquery side of the join contains a null, then the entire answer becomes null. The testing for these null values adds additional processing overhead to NOT IN operation. To reduce the tests down to two-value logic, equal, not equal, the join columns must be declared NOT NULL.

The Employee table was populated with 26,000 rows and the Customer table was populated with 5,000 rows. Statistics were collected on the primary indexes and any “join” columns.

The complete EXPLAIN follows:

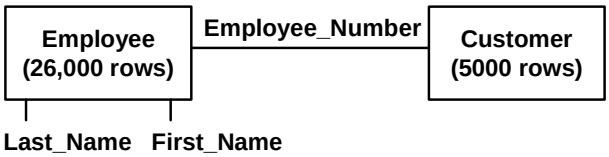
1-3) Locking steps
4) We do an all-AMPs RETRIEVE step from TFACT.C by way of an all-rows scan with a condition of ("NOT (TFACT.C.sales_emp_number IS NULL)") into Spool 3 (all_amps), which is redistributed by the hash code of (TFACT.C.sales_emp_number) to all AMPs. Then we do a SORT to order Spool 3 by row hash and the sort key in spool field1 eliminating duplicate rows. The size of Spool 3 is estimated with high confidence to be 5,000 rows (125,000 bytes). The estimated time for this step is 0.03 seconds.
5) We do an all-AMPs SUM step to aggregate from Spool 3 by way of an all-rows scan. Aggregate Intermediate Results are computed globally, then placed in Spool 4.
6) We do an all-AMPs RETRIEVE step from Spool 4 (Last Use) by way of an all-rows scan into Spool 2 (all_amps), which is duplicated on all AMPs.
7) We do an all-AMPs JOIN step from TFACT.E by way of an all-rows scan with a condition of ("TFACT.E.Job_Code = 3100"), which is joined to Spool 3 by way of an all-rows scan. TFACT.E and Spool 3 are joined using an exclusion merge join, with a join condition of ("TFACT.E.Employee_Number = sales_emp_number"), and null value information in Spool 2. Skip this join step if null exists. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with index join confidence to be 98 rows (4,802 bytes). The estimated time for this step is 0.12 seconds.
8) We do an all-AMPs RETRIEVE step from Spool 3 (Last Use) by way of an all-rows scan into Spool 6 (all_amps), which is redistributed by the hash code of (TFACT.C.sales_emp_number) to all AMPs. Then we do a SORT to order Spool 6 by row hash, and null value information in Spool 2. Skip this retrieve step if there is no null. The size of Spool 6 is estimated with high confidence to be 5,000 rows (125,000 bytes). The estimated time for this step is 0.03 seconds.
9) We do an all-AMPs JOIN step from TFACT.E by way of an all-rows scan with a condition of ("TFACT.E.Job_Code = 3100"), which is joined to Spool 6 (Last Use) by way of an all-rows scan. TFACT.E and Spool 6 are joined using an exclusion merge join, with a join condition of ("TFACT.E.Employee_Number = sales_emp_number"), and null value information in Spool 2 (Last Use). Skip this join step if there is no null. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with index join confidence to be 98 rows (4,802 bytes). The estimated time for this step is 0.12 seconds.
10) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1.

Exclusion Join

QUERY

```
EXPLAIN
SELECT    Last_Name, First_Name
FROM      Employee E
WHERE     Job_Code = 3100
AND       Employee_Number
NOT IN    (SELECT  Sales_Emp_Number FROM Customer C WHERE Sales_Emp_Number IS NOT NULL);
```

Job_Code (3100)



EXPLANATION 12.0 EXPLAIN

4) We do an all-AMPs RETRIEVE step from TFACT.C by way of an all-rows scan with a condition of ("NOT (TFACT.C.sales_emp_number IS NULL)") into Spool 3 (all_amps), which is redistributed by the hash code of (TFACT.C.sales_emp_number) to all AMPs. Then we do a SORT to order Spool 3 by row hash and the sort key in spool field1 eliminating duplicate rows. The size of Spool 3 is estimated with high confidence to be 5,000 rows (125,000 bytes). The estimated time for this step is 0.03 seconds.
5) We do an all-AMPs SUM step to aggregate from Spool 3 by way of an all-rows scan. Aggregate Intermediate Results are computed globally, then placed in Spool 4.
6) We do an all-AMPs RETRIEVE step from Spool 4 (Last Use) by way of an all-rows scan into Spool 2 (all_amps), which is duplicated on all AMPs.
7) We do an all-AMPs JOIN step from TFACT.E by way of an all-rows scan with a condition of ("TFACT.E.Job_Code = 3100"), which is joined to Spool 3 by way of an all-rows scan. TFACT.E and Spool 3 are joined using an exclusion merge join, with a join condition of ("TFACT.E.Employee_Number = sales_emp_number"), and null value information in Spool 2. Skip this join step if null exists. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with index join confidence to be 98 rows (4,802 bytes). The estimated time for this step is 0.12 seconds.

Example of PPI to PPI Join

The complete EXPLAIN example is shown below.

```
EXPLAIN
SELECT          *
FROM            Orders_PPI A
INNER JOIN      Orders_PPI_n B
ON              A.orderid = B.orderid
AND             A.orderdate = B.orderdate;
```

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.B.
2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.A.
3) We lock TFACT.B for read, and we lock TFACT.A for read.
4) We do an all-AMPs JOIN step from TFACT.B by way of a RowHash match scan with no residual conditions, which is joined to TFACT.A by way of a RowHash match scan with no residual conditions. TFACT.B and TFACT.A are joined using a rowkey-based merge join, with a join condition of ("(TFACT.A.orderid = TFACT.B.orderid) AND (TFACT.A.orderdate = TFACT.B.orderdate)"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with low confidence to be 3,600 rows (558,000 bytes). The estimated time for this step is 0.11 seconds.
5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.11 seconds.

Note: Statistics were collected on both tables on orderid and orderdate.

Example of PPI to PPI Join

QUERY

```
EXPLAIN
   SELECT     *
   FROM       Orders_PPI A
   INNER JOIN Orders_PPI_n B
   ON         A.orderid = B.orderid
   AND        A.orderdate = B.orderdate;
```

The Primary Index and the partitioning expression for Orders_PPI and Orders_PPI_n are the same.

EXPLANATION 12.0 EXPLAIN

```
  :  (Locking steps)
  4) We do an all-AMPs JOIN step from TFACT.B by way of a RowHash match scan with no residual
     conditions, which is joined to TFACT.A by way of a RowHash match scan with no residual
     conditions.  TFACT.B and TFACT.A are joined using a rowkey-based merge join, with a join
     condition of ("(TFACT.A.orderid = TFACT.B.orderid) AND (TFACT.A.orderdate =
     TFACT.B.orderdate)").  The result goes into Spool 1 (group_amps), which is built locally on the
     AMPs.  The size of Spool 1 is estimated with low confidence to be 3,600 rows (558,000 bytes).  The
     estimated time for this step is 0.11 seconds.
  5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
  -> The contents of Spool 1 are sent back to the user as the result of statement 1.  The total estimated
     time is 0.11 seconds.
```

Join ACCESS

Join ACCESS occurs whenever a column appears in an Equality Join constraint. The example at the top of the facing page features an Equality Join constraint (Table1.Column = Table2.Column). The Join ACCESS occurs on the ColName column.

Join Access demographics come from Application and Transaction Modeling, and are expressed in two ways:

- **Join Frequency,** which is a measure of how often annually, all known transactions access rows from the table through a Join on this column.

- **Join Rows,** which is a measure of how many rows are accessed annually through joins on this column across all transactions.

Index choices can greatly affect the Join Type, or other access method used:

- UPIs or USIs may make Nested Joins possible.

- NUSIs can only be used for Set Selection in most Joins or to access Table 2 in a Nested Join.

- USIs and NUSIs do not participate in Join Conditions (except in Nested Joins). Only PIs participate in non-Nested Join Conditions.

Join Access

- **Join Access is how often a column appears in an equality constraint.**

  e.g., ON Table1.colname = Table2.colname

- **Two metrics that represent join access are:**

  Join Frequency:

  How often annually all known transaction access rows from the table through a join on this column.

  Join Rows:

  How many rows are accessed annually through joins on this column across all transactions.

- **The demographics result from activity modeling.**

- **Index choices determine the join type or other access method.**

Exercise 5 – Sample

In this exercise, you will make final index choices (based on Join Access demographics) for the same tables you used in Exercises 2, 3, and 4. You will identify the columns that statistics should be collected.

The Final Index Choice Guidelines are:

- Choose one Primary Index (UPI or NUPI) utilizing Join Access demographics
- Do not carry identical Primary and Secondary Indexes

The example on the facing page provides you with an example of how to apply these guidelines.

Exercise 5 – Sample

On the following pages, there are sample tables with change row and value access demographics.

- Make your final index choices, as shown below.
- Identify columns to collect statistics on.
- Join Access demographics have been added.

Final Index Choice Guidelines:

- Choose ONE Primary Index (UPI or NUPI) utilizing Join Access demographics.
- Do not carry identical Primary and Secondary Indexes.

Example 60,000,000 Rows	A	B	C	D	E	F	G	H
PK/FK	PK,SA		FK,NN	NN,ND				
Value Access	5K	2.6K	0	500K	0	0	0	52
Range Access	12	0	0	0	0	0	0	4K
Join Access	1M	0	1K	0	0	0	0	0
Join Rows	50M	0	5K	0	0	0	0	0
Distinct Values	60M	7M	1.5M	60M	8	15M	15M	700
Max Rows/Value	1	12	500	1	8M	9	725K	90K
Max Rows/NULL	0	5	0	0	0	725K	5	10K
Typical Rows/Value	1	7	35	1	7M	3	3	80K
Change Rating	0	1	5	3	0	4	4	9
PI/SI	UPI USI	NUPI NUSI		USI				
Collect Statistics (Y/N)	Y	Y	Y					Y

Exercise 5 – Making Final Index Choices

In this exercise, you will make final index choices (based on Join Access demographics) for the same tables you used in Exercises 2, 3, and 4. You will identify the columns that statistics should be collected.

The Final Index Choice Guidelines are:

- Choose one Primary Index (UPI or NUPI) utilizing Join Access demographics
- Do not carry identical Primary and Secondary Indexes

The example on the facing page provides you with an example of how to apply these guidelines.

Exercise 5 – Making Final Index Choices

ENTITY 1

100,000,000 Rows	A	B	C	D	E	F
PK/FK	PK,UA					
Value Access	50K	0	0	0	0	0
Range Access	0	0	0	0	0	0
Join Access	10M	0	0	0	0	0
Join Rows	10M	0	0	0	0	0
Distinct Values	100M	95M	300K	250K	40M	1M
Max Rows/Value	1	2	400	350	3	110
Max Rows/NULL	0	0	0	0	1.5M	0
Typical Rows/Value	1	1	325	300	2	90
Change Rating	0	3	1	1	1	1
PI/SI	UPI ~~USI~~		~~NUPI~~	~~NUPI~~		~~NUPI~~
Collect Statistics (Y/N)						

Exercise 5 – Making Final Index Choices (cont.)

In this exercise, you will make final index choices (based on Join Access demographics) for the same tables you used in Exercises 2, 3, and 4. You will identify the columns that statistics should be collected.

The Final Index Choice Guidelines are:

- Choose one Primary Index (UPI or NUPI) utilizing Join Access demographics
- Do not carry identical Primary and Secondary Indexes

The example on the facing page provides you with an example of how to apply these guidelines.

Exercise 5 – Making Final Index Choices (cont.)

ENTITY 2

10,000,000 Rows	G	H	I	J	K	L
PK/FK	PK,SA					
Value Access	5K	365	12	12	0	0
Range Access	12	0	0	0	0	260
Join Access	100M	0	0	0	0	0
Join Rows	100M	0	0	0	0	0
Distinct Values	10M	100K	9M	12	50	180K
Max Rows/Value	1	200	2	1M	240K	60
Max Rows/NULL	0	0	100K	0	0	0
Typical Rows/Value	1	100	1	800K	190K	50
Change Rating	0	0	9	1	2	0
	UPI	NUPI				NUPI
PI/SI	USI	NUSI				VONUSI
Collect Statistics (Y/N)						

Exercise 5 – Making Final Index Choices (cont.)

In this exercise, you will make final index choices (based on Join Access demographics) for the same tables you used in Exercises 2, 3, and 4. You will identify the columns that statistics should be collected.

The Final Index Choice Guidelines are:

- Choose one Primary Index (UPI or NUPI) utilizing Join Access demographics
- Do not carry identical Primary and Secondary Indexes

The example on the facing page provides you with an example of how to apply these guidelines.

Exercise 5 – Making Final Index Choices (cont.)

DEPENDENT

5,000,000 Rows	A	M	N	O	P	Q
PK/FK	PK				NN,ND	
	FK	SA				
Value Access	0	0	0	0	0	0
Range Access	0	0	0	0	0	0
Join Access	700K	0	0	0	0	0
Join Rows	1M	0	0	0	0	0
Distinct Values	2M	50	90K	3M	5M	2M
Max Rows/Value	4	200K	75	2	1	5
Max Rows/NULL	0	0	0	390K	0	1M
Typical Rows/Value	1	60K	50	1	1	1
Change Rating	0	0	3	1	0	1
PI/SI	UPI				UPI	
	NUPI					
	USI				USI	
Collect Statistics (Y/N)						

Exercise 5 – Making Final Index Choices (cont.)

In this exercise, you will make final index choices (based on Join Access demographics) for the same tables you used in Exercises 2, 3, and 4. You will identify the columns that statistics should be collected.

The Final Index Choice Guidelines are:

- Choose one Primary Index (UPI or NUPI) utilizing Join Access demographics
- Do not carry identical Primary and Secondary Indexes

The example on the facing page provides you with an example of how to apply these guidelines.

Exercise 5 – Making Final Index Choices (cont.)

ASSOCIATIVE 1

300,000,000 Rows	A	G	R	S	
PK/FK	PK				
	FK	FK,SA			
Value Access	260	0	0	0	
Range Access	0	0	0	0	
Join Access	0	8M	0	0	
Join Rows	0	300M	0	0	
Distinct Values	100M	10M	15K	800K	
Max Rows/Value	5	50	21K	400	
Max Rows/NULL	0	0	0	0	
Typical Rows/Value	3	30	19K	350	
Change Rating	0	0	0	0	
	UPI				
PI/SI	NUPI	NUPI	NUPI?	NUPI	
	USI				
	NUSI				
Collect Statistics (Y/N)					

Exercise 5 – Making Final Index Choices (cont.)

In this exercise, you will make final index choices (based on Join Access demographics) for the same tables you used in Exercises 2, 3, and 4. You will identify the columns that statistics should be collected.

The Final Index Choice Guidelines are:

- Choose one Primary Index (UPI or NUPI) utilizing Join Access demographics
- Do not carry identical Primary and Secondary Indexes

The example on the facing page provides you with an example of how to apply these guidelines.

Exercise 5 – Making Final Index Choices (cont.)

ASSOCIATIVE 2

100,000,000 Rows	A	M	G	T	U	
PK/FK	PK					
	FK		FK			
Value Access	0		0	0	0	
Range Access	0		0	0	0	
Join Access	7M		250K	0	0	
Join Rows	800M		20M	0	0	
Distinct Values	50M		10M	560K	750	
Max Rows/Value	3		150	180	135K	
Max Rows/NULL	0		0	0	0	
Typical Rows/Value	1		8	170	100K	
Change Rating	0		0	0	0	
PI/SI	UPI					
	NUPI		NUPI	NUPI		
	USI					
Collect Statistics (Y/N)						

What additional index choices would be available for the column G?

Exercise 5 – Making Final Index Choices (cont.)

In this exercise, you will make final index choices (based on Join Access demographics) for the same tables you used in Exercises 2, 3, and 4. You will identify the columns that statistics should be collected.

The Final Index Choice Guidelines are:

- Choose one Primary Index (UPI or NUPI) utilizing Join Access demographics
- Do not carry identical Primary and Secondary Indexes

The example on the facing page provides you with an example of how to apply these guidelines.

Exercise 5 – Making Final Index Choices (cont.)

HISTORY

730,000,000 Rows	A	DATE	D	E	F	
PK/FK	PK					
	FK	SA				
Value Access	10M	5K	0	0	0	
Range Access	0	20K	0	0	0	
Join Access	800M	0	0	0	0	
Join Rows	2.4B	0	0	0	0	
Distinct Values	100M	730	N/A	N/A	N/A	
Max Rows/Value	18	1100K	N/A	N/A	N/A	
Max Rows/NULL	0	0	N/A	N/A	N/A	
Typical Rows/Value	3	900K	N/A	N/A	N/A	
Change Rating	0	0	N/A	N/A	N/A	
PI/SI	UPI					
	NUPI					
	USI					
	NUSI	VONUSI				
Collect Statistics (Y/N)						

What additional index choices would be available for the DATE column?

Teradata Index Wizard

Indexing is one of the most powerful tuning options available to a Database Designer or Database Administrator. Traditionally, index selection has been a manual process, often requiring the DBA or designer to have detailed knowledge of the application workloads and data demography of the Warehouse, and also have experience and understanding of query plan optimization. To determine the right set of indexes for the Active Data Warehouse, designers mostly rely on their application experience and intuition to make index design decisions.

The Teradata Index Wizard utility automates this process of manual index design by recommending secondary indexes for a particular workload. Teradata Index Wizard provides a simple, easy-to-use graphical user interface (GUI) that guides the user on how to go about analyzing a database workload and provides recommendations for improving performance through the use of indexes.

The following describes Teradata Index Wizard:

- Teradata Index Wizard is a client tool that provides an interface to the Teradata Database in order to tune physical database design for performance improvement. It automates the selection process for secondary indexes and single table join indexes on the tables accessed by the statements in the workload.
- Each step involved in the index analysis is provided as an easy to use, menu-driven interface.
  - Workload Definition
  - Workload Analysis
  - Index Analysis and/or Partition Analysis (12.0 option)
  - Analysis Reporting
  - Recommendation Validation (optional, but very useful)
  - Recommendation Implementation (optional)
- It works with the Teradata Database Query Log facility for defining a workload from a collection of SQL statements captured in the query log.
- It provides a What-If-Analysis mode allowing users to specify their own set of recommendations.
- It allows users to validate the recommendations provided. This feature enables the users to check the recommendations on the production system before actually creating the new set of indexes. It interfaces with Teradata Visual Explain to provide a visual compare for the plans with and without the recommendations.
- It uses the Teradata Database Optimizer to recommend the indexes.
- The interface is internationalized to allow localization.
- On-line help is provided via the context-sensitive help feature.
- It interfaces with other client tools such as Teradata System Emulation Tool in order to provide flexibility and perform offline analysis of a production system on a test system.
- It can import the workload on a test system for analysis. This way, the production system resources need not be used during analysis.

Teradata Index Wizard

- The Teradata Index Wizard (Windows utility) analyzes a set of queries in a defined workload and recommends a set of secondary indexes, single table join indexes, or partitioning to consider.
  - Example of workloads that be can defined and used include …
    - Queries selected from the Teradata Database Query Log (DBQL) statements
    - Queries selected from Teradata DBQL XML statements
    - A user-specified set or file of SQL statements.
    - Set of execution plans in a user-defined Query Capture Database (QCD)
- The steps involved in index analysis are:
  - Workload Definition (and optional workload analysis)
  - Index and/or Partition Analysis
  - Analysis Reporting
  - Recommendation Validation
  - Index and/or PPI Creation
- Other features and characteristics are:
  - What-If Analysis mode – make your own recommendations
  - The Index Analysis Engine is inside the Teradata Parser and uses the Teradata Optimizer to recommend the indexes.
  - Integrated with Teradata Analyst tool set – Teradata Index Wizard, Teradata Statistics Wizard, and Teradata SET (System Emulation Tool).

Teradata Index Wizard – Main Window

The Menu bar is located at the top of the Teradata Index Wizard window. It contains the pull-down menus displaying the commands that you use to operate the program.

In addition to accessing the commands from the menu bar, the Teradata Index Wizard provides shortcuts to many of the commands via the tool buttons along the top and left side of the window.

Defining and Using Workloads with Index Wizard

A workload is a set of SQL statements created or defined using the Workload Definition dialog. Index Wizard creates several workload reports after a workload is defined. Workloads can be defined in the following ways:

Using Database Query Log (DBQL) – The Database Query Log (DBQL) provides the capability to store, in system tables, the performance-related data for a request

Using Statement Text – SQL statements can be directly keyed into a workload and analyzed. The SQL statements can also be selected from one or more files.

From QCD Statements – An existing set of execution plans in a QCD can be selected to form a workload. The workload is created in the QCD in which the execution plans exist.

Importing Workload – Users can import workloads from other sources including other Teradata client tools. For example, you can import SQL statements to the test system as a workload from a production system, using the Teradata System Emulation Tool. The production system environment is imported along with the SQL statements.

From an Existing Workload – A new workload can be created from an existing workload.

Additional Workload Functions

There are additional functions under the workload menu that allow you to manage workloads better. These functions include the following.

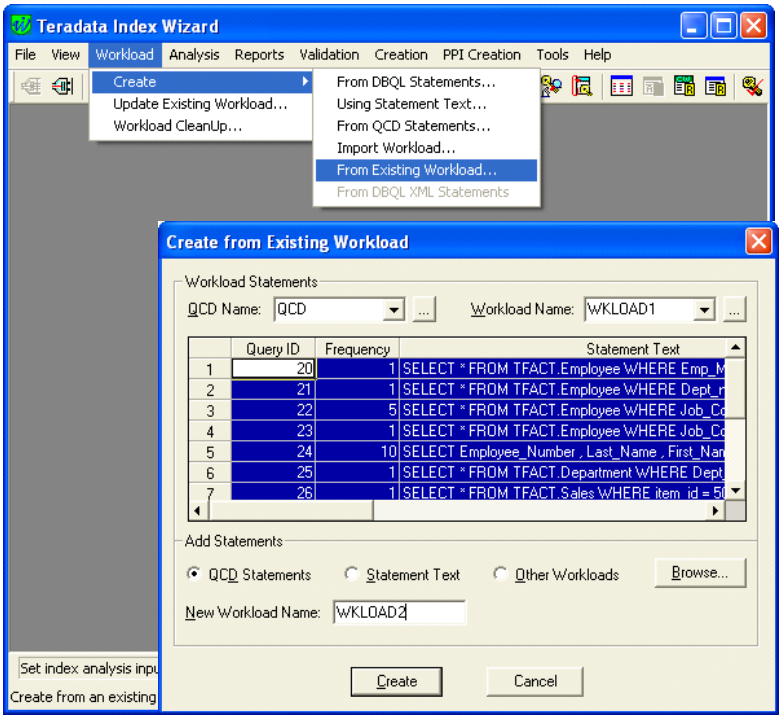
Workload Cleanup – allows you to delete workloads that you no longer need.

View Workload Details – provides the statements and operations on a particular workload.

Workload Summary – shows the workload name and ID as well as the number of statements and the begin and end time.

Teradata Index Wizard – Main Window

1st – Define a Workload of SQL statements to analyze.



The sequence of the menu items represents the order of the steps to follow with the Teradata Index Wizard.

- **Workload**
- **Analysis**
- **Reports**
- **Validation**
- **Creation**

This example defines a workload of 12 SQL statements.

Workloads can be defined …

- **From DBQL statements**
- **Using Statement Text**
- **From QCD Statements**
- **Import Workload**
- **From Existing Workload**
- **FROM DBQL XML Statements**

Note: You can Imports workloads onto a test system for analysis, saving production system resources.

Teradata Index Wizard – Index Analysis

Teradata Index Wizard consists of a database server component and a front-end client application. The Index Analysis Engine is actually inside the Teradata Parser and works closely with the parallel optimizer to enumerate, simulate, and evaluate index selection candidates. The client front end is a graphical Windows interface, providing step by step instructions for workload definition and index analysis, and reports for both workload and index analysis.

After the workload is defined, it is analyzed and a new set of indexes is recommended. The Teradata Index Wizard may also recommend that indexes be added or dropped to enhance system performance.

There are four types of analysis including:

Index Analysis – a new analysis is performed on a workload.

Partition Analysis – a partition analysis is performed on a workload.

Restarting an Analysis – an analysis can be restarted if, for some reason, it was interrupted.

What-If Analysis – allows you to check your own recommendations on the workload to determine the performance benefits.

What-if analysis allows you to suggest indexes for the selected workload and monitor the performance improvement. Statistics in the simulation mode are collected into the QCD and are simulated to the Optimizer during the plan generation. Only the CREATE INDEX and COLLECT STATS are valid in this mode. If DROP INDEX is specified in the simulation mode, the indexes/stats are not provided to the Optimizer, and the generated plans do not consider these indexes/statistics.

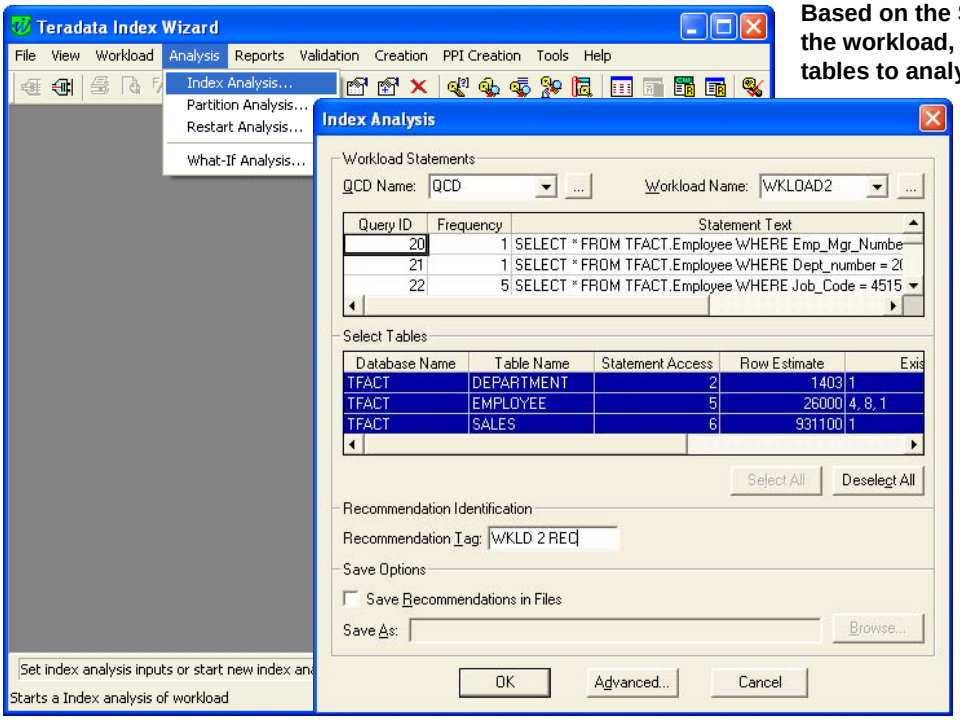
You can build the required CREATE INDEX and COLLECT STATS DDL statements using Index Wizard. The DDL statements specified are submitted in a simulation mode to the Teradata Database and then the workload statements are submitted to generate the new execution plans.

Note: Various parameters that determine index recommendations are set by selecting the **Advanced** button in the **Index Analysis** window. Examples of advanced options include:

Keep Existing Indexes – with NO, recommendations to drop indexes are given.
Maximum Columns per Recommendation – default is 4; max of 16
Maximum Recommendations per table – default is 16; max is 64
Maximum Candidates – default is 256; range is between 128 and 512
Change Rate Threshold – default is 5; 0 is non-volatile and 9 is highly volatile

Teradata Index Wizard – Index Analysis

2nd – Perform an index analysis.



Based on the SQL statements in the workload, you can select which tables to analyze.

Specify a "tag" or name for the index recommendations.

Teradata Index Wizard – Index Analysis Results

When the index analysis is complete, Index Wizard displays the results in a summary window. Click OK to close the window. All the analysis reports are now available.

In addition to performing an index analysis, you can use the Index Wizard to perform a partition analysis on a workload. This analysis will possibly recommend partitioning for a primary index on existing tables.

The example on the facing page depicts the screens involved in performing a partition analysis.

Creating a Workload to Analyze using "DUMP EXPLAIN"

The following steps can be following to create and analyze a workload created from a "DUMP EXPLAIN".

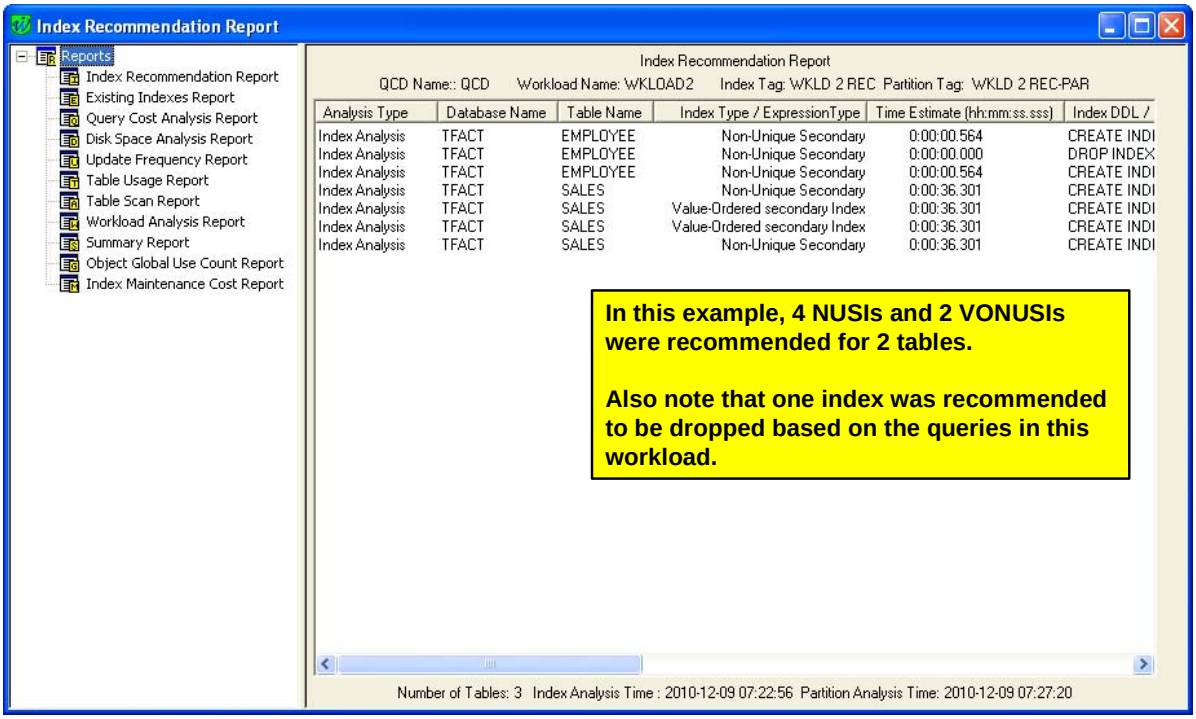
1. Using BTEQ,
   ```
   .EXPORT FILE=dumpexplain.txt
   DUMP EXPLAIN INTO QCD SELECT...
   .EXPORT RESET
   .RUN FILE =dumpexplain.txt
   ```

2. Using Index Wizard,
   Select "Workload->Create->From QCD Statements"
   Enter QCD name and select Browse QCD to list the captured plans.
   Select the plan, enter a new workload name and select OK.

3. Using Index Wizard, perform the index analysis as usual.

Teradata Index Wizard – Index Analysis Results

The Index Analysis phase will provide a results report at its completion.



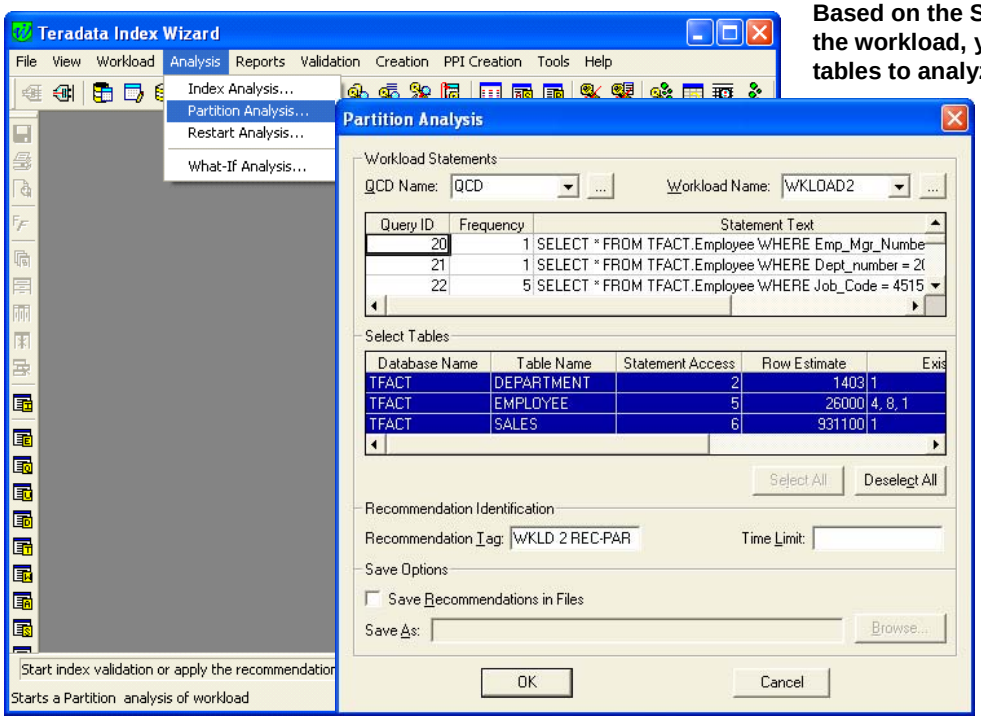
Teradata Index Wizard – Partition Analysis

In addition to performing an index analysis, you can use the Index Wizard to perform a partition analysis on a workload. This analysis will possibly recommend partitioning for a primary index on existing tables.

The example on the facing page depicts the screens involved in performing a partition analysis.

Teradata Index Wizard – Partition Analysis

2nd (cont.) – Perform a partition analysis.



Based on the SQL statements in the workload, you can select which tables to analyze.

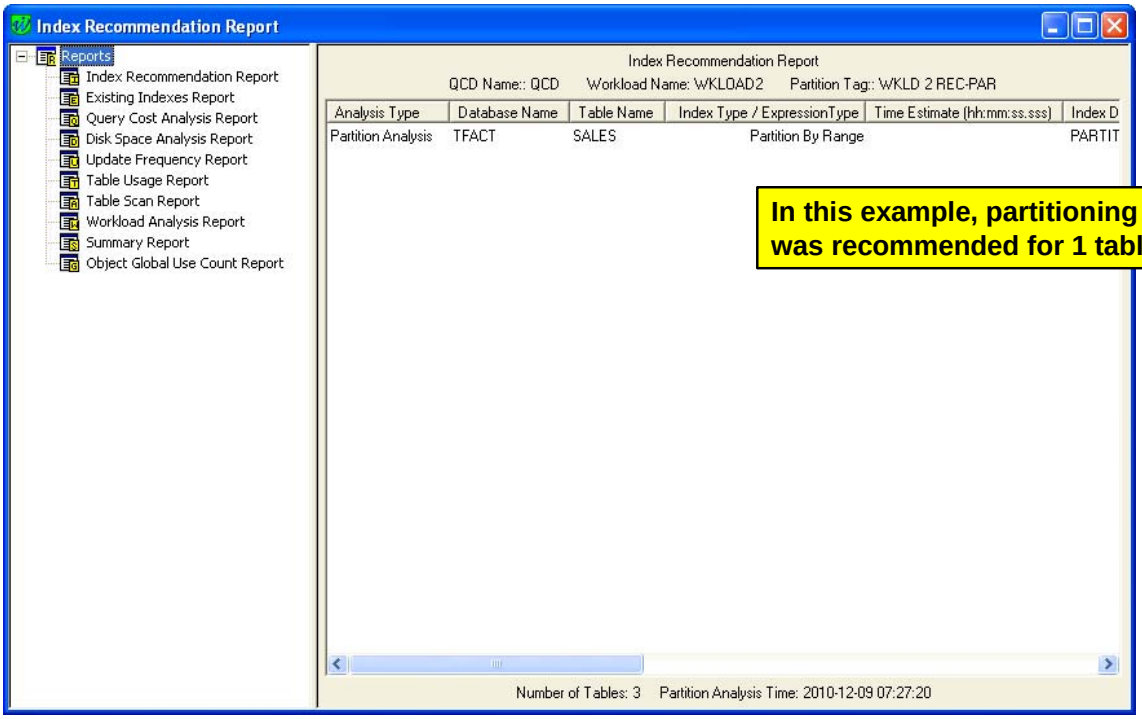
Specify a "tag" or name for the partitioning recommendations.

Teradata Index Wizard – Partition Analysis Results

When the partition analysis is complete, Index Wizard displays the results in a summary window. Click OK to close the window. All the analysis reports are now available.

Teradata Index Wizard – Partition Analysis Results

The Partition Analysis phase will provide a results summary at its completion.



Teradata Index Wizard – Reports

The Teradata Index Wizard creates a set of reports when a workload is created and a set after a workload is analyzed. Workload reports are designed to help you select tables within a workload to be analyzed. Analysis reports provide the data you need to make a decision about the index recommendation

Workload Reports

Workload Reports provide information about the tables in a defined workload. The reports assist you in deciding which tables make the best candidates for an index analysis. These reports also provide a snapshot of table information which can be compared with report information after an analysis. The following list shows the reports available when a workload is created:

- Existing Indexes Report
- Update Cost Analysis Report
- Table Usage Analysis Report
- Table Scan Report
- Workload Analysis Report
- Object Global Use Count Report

Analysis Reports

Analysis reports provide information about the results of the index analysis. They are designed to help you decide if the index recommendation is appropriate or not. The following reports are available after a workload is analyzed:

- Index Recommendation Report
- Existing Indexes Report
- Query Cost Analysis Report
- Update Cost Analysis Report
- Disk Space Analysis Report
- Table Usage Analysis Report
- Table Scan Report
- Workload Analysis Report
- Summary Report
- Object Global Use Count Report

Index Recommendation Report

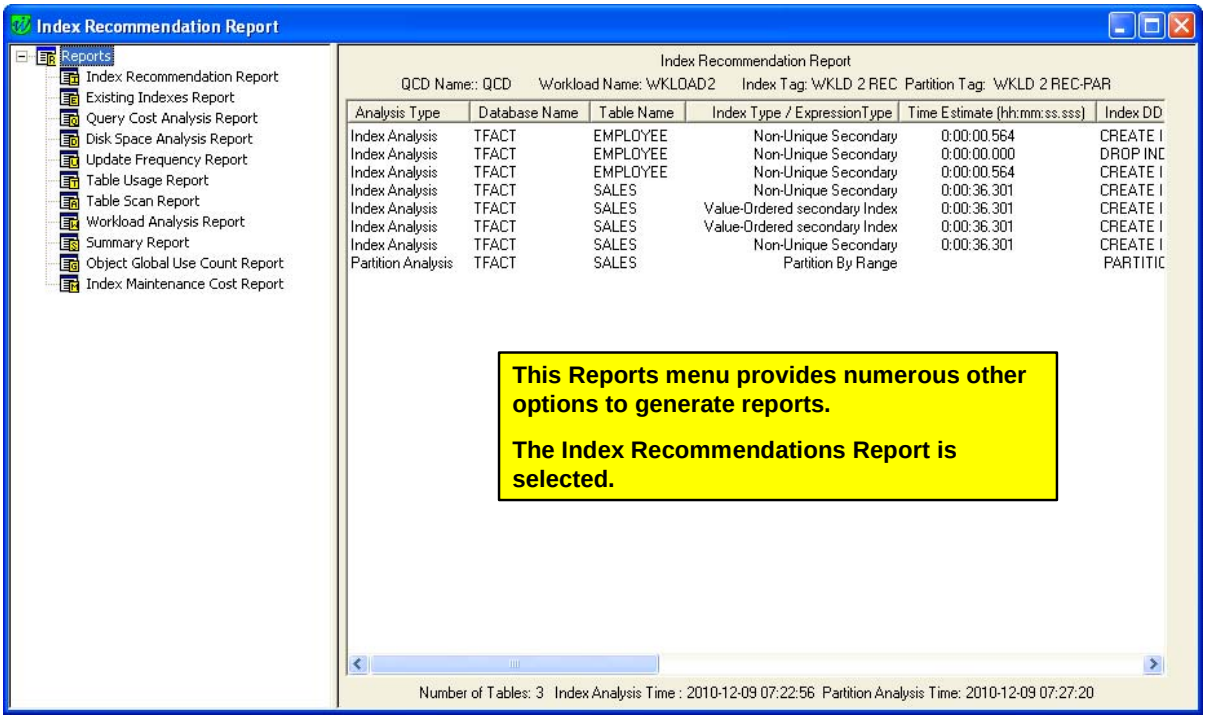
The Index Recommendation report (facing page) shows the recommended secondary index for each table if one is recommended. Use the information in this report table to help you make a decision about whether the index recommended is appropriate or not.

The report also suggests indexes that can be dropped.

The statistics used for the analysis are saved in the Index Recommendations table.

Teradata Index Wizard – Reports

3rd – Use the Reports menu to view the recommendations.



Teradata Index Wizard – Validation

Index Validation

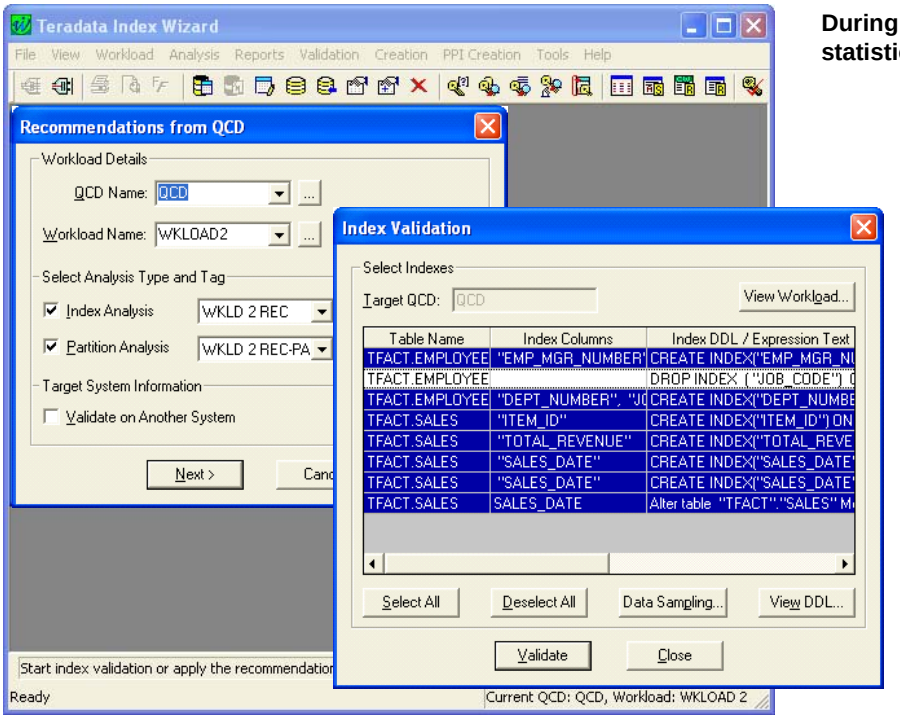
Index validation is the process of checking to see if the recommended indexes actually improve system performance. Validation does not involve the actual execution of the recommendations, but permits the Teradata Optimizer to use the recommended indexes in the plan generation. If the recommendations are created with a COLLECT STATISTICS option, Index Wizard collects statistics on a sample size and saves them in the QCD. The sampled statistics are used during index validation.

The validation process occurs in a simulated session mode. The index and the required statistics (as indicated by the recommendations) are simulated for the plan generation. Index recommendations are validated on the set of statements that were analyzed. The statements are submitted to the Teradata Database in a “no execute” mode. During validation, the query plans are saved into the specified QCD.

Recommendations for the workload are loaded either from QCD or from a file.

Teradata Index Wizard – Validation

4th – Use the Validation menu to check if the recommendations improve performance.



During this phase, sample statistics may be collected.

Two EXPLAINs for each statement will be executed – one without the recommended indexes and one with the recommended indexes.

This phase may take some time to complete.

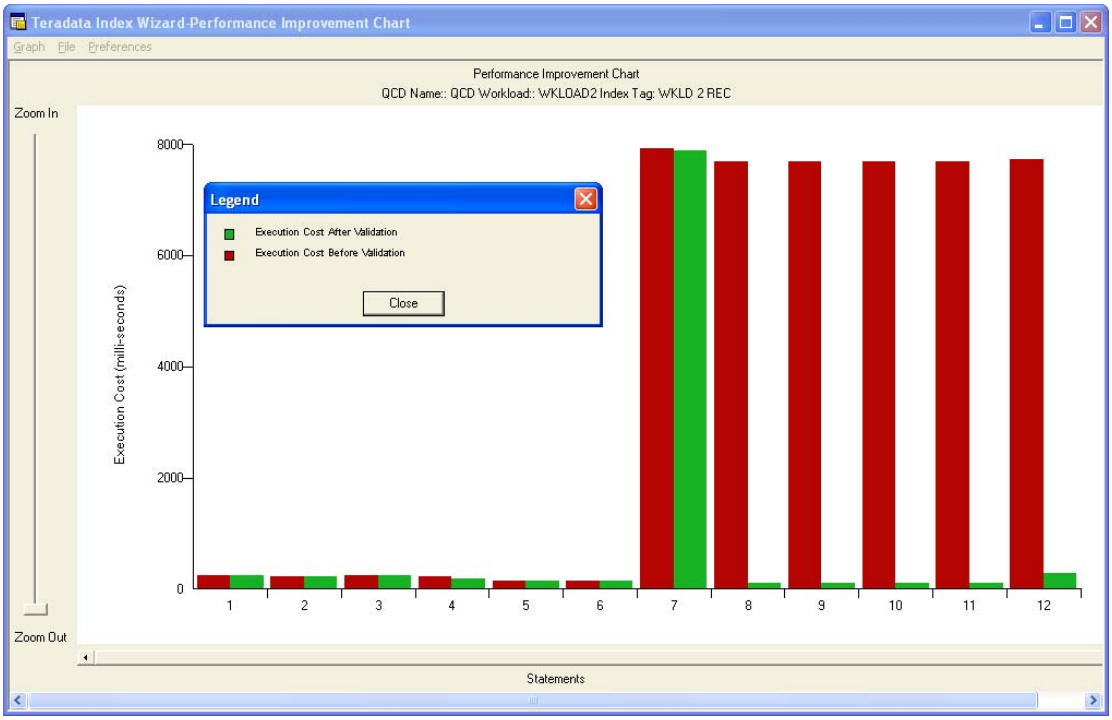
Teradata Index Wizard – Validation (View Graph)

After validating the index recommendations, you can use the Compare option to compare (via Visual Explain) each of the recommendations. You can use the View Graph option to view the estimated performance gains with a summary graph.

The facing page illustrates an example of the View Graph option.

Teradata Index Wizard – Validation (View Graph)

Use the View Graph function to view estimated performance gains for each of the queries.



Teradata Index Wizard – Creation

Executing Recommendations

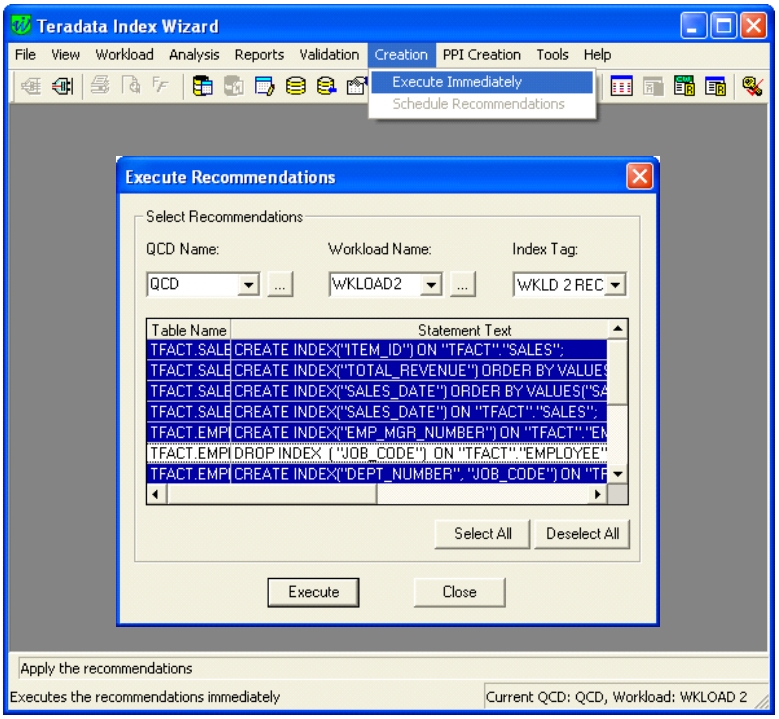
The Teradata Index Wizard has an execute recommendation feature that allows you to execute index recommendations.

Executing a recommendation applies the index recommendations to the database. It is highly recommended that index recommendations be validated before executing.

Partition Analysis indicates tables where partitioning may be beneficial and identifies a recommended partitioning expression. The Index Wizard does not implement partitioning.

Teradata Index Wizard – Creation

5th – If desired, use the Creation menu to execute the recommendations.



SQL is created that you can select to create secondary indexes and collect statistics.

The Execute button submits the SQL.

Notes:

Partition Analysis indicates tables where partitioning may be beneficial and identifies a recommended partitioning expression.

The Index Wizard does not implement partitioning.

Summary – Index Choice Guidelines

The facing page shows some guidelines for choosing Primary Indexes. The most important guidelines are:

- Primary Indexes should have Change Ratings of 0 - 2.
- Try to avoid NUPI indexes that have excessive number of duplicate values. A general rule is that the number of number of duplicate values should be less than the number of rows that can reside in 3 or fewer blocks.
- Collect statistics for all NUPIs and NUSIs.
- Collect statistics for UPIs of small tables (e.g., less than 1000 rows per AMP or 4 blocks per AMP).
- Consider column(s) as primary index candidates if …

  the number of max rows per value and the typical rows per value is relatively close to each other

  AND

  the number of distinct values is greater than the number of AMPs (by at least at factor of 10), then the column may be a candidate for a NUPI.

The facing page shows some guidelines for choosing Secondary Indexes. The most important guidelines are:

- Secondary Indexes should have Change Ratings of 0 - 5.
- Secondary Indexes should have Value Access > 0.
- Collect Statistics for all NUSIs.
- Collect Statistics for USIs that are used with range queries

Summary – Index Choice Guidelines

Primary Index	Unique	Non-Unique
Value Access	1 AMP – 1 Row	1 AMP – 1+ Rows
Join Access	Nested, Merge, Exclusion Joins	Nested, Merge, Exclusion Joins
Distinct Values	Equals row count	Less than row count Much greater than # of AMPs (10X or more)
Max Rows/Value	One	Close to typical value
Max Rows NULL	One	Less than typical value
Typical Rows/Value	One	Close to max value
Change Rating	0, 1, 2	0, 1, 2
Statistics	Collect statistics for small tables	Collect statistics
Secondary Index	**Unique**	**Non-Unique**
Value Access	2 AMP – 1 Row	All AMPs – 1+ Rows
Join Access	Nested Joins	Nested and Merge Joins
Distinct Values	Equals row count	Less than row count
Max Rows/Value	One	Less than # of blocks
Max Rows NULL	One	Less than # of blocks
Typical Rows/Value	One	Less than # of blocks
Change Rating	0 to 5	0 to 5
Statistics	Collect statistics for range queries	Collect statistics

Module 29: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 29: Review Questions

1. When do Cartesian Product Joins generally occur?

    a. When the Join Column is the Primary Index of both tables.
    b. When a column appears in an Equality Join constraint.
    c. When an error is made in coding the SQL query.
    d. None of the above.

2. A Join Condition of (1=1) in an EXPLAIN output is usually indicative of _____.

    a. Cartesian product join
    b. Exclusion merge join
    c. Merge join
    d. Hash join

3. One of the ways in which Join Access demographics are expressed is _____ , which is a measure of how often all known transactions access rows from the table through a Join on this column.

    a. Join Access Rows
    b. Join Access Frequency
    c. High Access Rows
    d. Value and Join Access

Notes

TERADATA
Raising Intelligence

Additional Index Choices

After completing this module, you will be able to:

- **List the three types of Join Indexes.**
- **Describe the situations where a Join Index can improve performance.**
- **Describe how to create a "sparse index".**
- **Describe the difference between a single-table Join Index and a Hash Index.**

Notes

Table of Contents

Additional Index Choices 30-4
Join Indexes 30-6
Options for Join Indexes 30-8
Join Index Considerations 30-10
Join Index Example – Customer and Order Tables 30-12
Compressed Multi-Table Join Index 30-14
Non-Compressed Multi-Table Join Index 30-16
Compressed and Non-Compressed Join Indexes 30-18
Example 1 – Does a Join Index Help? 30-20
Example 2 – Does a Join Index Help? 30-22
Example 3 – Partitioning a Join Index 30-24
Join Index – Single Table 30-26
Join Index – Single Table (cont.) 30-28
Creating a Join Index – Single Table 30-30
Example 4 – Does the Join Index Help? 30-32
Why use Aggregate Join Indexes? 30-34
Aggregate Join Index Properties 30-36
Aggregation without an Aggregate Index 30-38
Creating an Aggregate Join Index 30-40
Aggregation with an Aggregate Index 30-42
Sparse Join Indexes 30-44
Creating a Sparse Join Index 30-46
Creating a Sparse Join Index on a Partitioned Table 30-48
ALTERing a Join Index to CURRENT 30-50
Partitioning a Sparse Join Index 30-52
Global (Join) Indexes 30-54
Global Index – Multiple Tables 30-56
Global Index as a “Hashed NUSI” 30-58
Creating a Global Index (“Hashed NUSI”) 30-60
Example: Using a Global Index as a Hashed NUSI 30-62
Repeating Row Ids in Global Index 30-64
Hash Indexes 30-66
Hash Index – Example 30-68
Hash Index – Example (cont.) 30-70
Summary 30-72
Module 30: Review Questions 30-74
Lab Exercise 30-1 30-76

Additional Index Choices

As part of implementing a physical design, Teradata provides additional index choices that can improve performance for known queries and workloads. These will be described in more detail in this module.

Join indexes are defined to reduce the number of rows processed in generating result sets from certain types of queries, especially joins. Like secondary indexes, users may not directly access join indexes. They are an option available to the optimizer in query planning. The following are properties of join indexes:

- Are used to replicate and "pre-join" information from several tables into a single structure.
- Are designed to cover queries, reducing or eliminating the need for access to the base table rows.
- Usually do not contain pointers to base table rows (unless user defined to do so).
- Are distributed based on the user choice of a Primary Index on the Join Index.
- Permit Secondary Indexes to be defined on the Join Index (except for Single Table Join Indexes), with either "hash" or "value" ordering.

Unlike traditional indexes, join indexes do not store "pointers" to their associated base table rows. Instead, they are a fast path *final* access point that eliminates the need to access and join the base tables they represent. They substitute *for* rather than point *to* base table rows.

The Optimizer can choose to resolve a query using the index, rather than performing a join of two or more tables. Depending on the complexity of the join, this improves query performance considerably. To improve the performance of Join Index creation and Join Index maintenance during updates, consider collecting statistics on the base tables of a Join Index.

Additional Index Choices

As part of the physical design process, the designer may choose to implement join and/or hash indexes for performance reasons.

- **Join Indexes**
  - Can be created to pre-join multiple tables.
  - Can be used on a single table to redistribute the table on a different column – effectively creating an alternative primary index on a foreign key column.
  - Can be used as a summary table to aggregate one or more columns from a table or tables.
- **Hash Indexes**
  - Contains properties of both secondary indexes and single table join indexes.
- **Both Join Indexes and Hash Indexes**
  - Provides the optimizer with additional options and the optimizer may use the join index if it "covers" the query.
  - For known queries, this typically will result in much better performance.

Join Indexes

There are multiple ways in which a join index can be used. Three common ways include:

Single table Join Index — Distribute the rows of a single table on the hash value of a foreign key value

Multi-Table Join Index — Pre-join multiple tables; stores and maintains the result from joining two or more tables.

Aggregate Join Index — Aggregate one or more columns of a single table or multiple tables into a summary table

A join index is a system-maintained index table that stores and maintains the joined rows of two or more tables (**multiple table join index**) and, optionally, aggregates selected columns, referred to as an **aggregate join index**.

Join indexes are defined in a way that allows join queries to be resolved without accessing or joining their underlying base tables. A join index is useful for queries where the index structure contains all the columns referenced by one or more joins, thereby allowing the index to cover all or part of the query. For obvious reasons, such an index is often referred to as a covering index. Join indexes are also useful for queries that aggregate columns from tables with large cardinalities. These indexes play the role of pre-join and summary tables without denormalizing the logical design of the database and without incurring the update anomalies presented by denormalized tables.

Another form of join index, referred to as a **single table join index**, is very useful for resolving joins on large tables without having to redistribute the joined rows across the AMPs.

Join Indexes

- **A Join Index is an optional index which may be created by the user. The basic types of Join Indexes will be described first.**

- **Multi-table Join Index**
  - **Pre-join multiple tables; stores and maintains the result from joining two or more tables.**
  - **Facilitates join operations by possibly eliminating join processing or by reducing/eliminating join data redistribution.**

- **Single-table Join Index**
  - **Distribute the rows of a single table on the hash value of a foreign key value.**
  - **Facilitates the ability to join the foreign key table with the primary key table without redistributing the data.**

- **Aggregate Join Index (AJI)**
  - **Aggregate (SUM or COUNT) one or more columns of a single table or multiple tables into a summary table.**
  - **Facilitates aggregation queries by eliminating aggregation processing. The pre-aggregated values are contained in the AJI instead of relying on base table calculations.**

Options for Join Indexes

The facing page highlights two options that are available with join indexes.

A Sparse Join Index is simply a term that is used when a join index is created with a WHERE condition. You can use a WHERE clause in the CREATE JOIN INDEX statement to limit the rows that are created in the join index. This effectively reduces the size (PERM space) of the join index. The rows included in the join index are a subset of the rows in the base table or tables based on an SQL query result.

A Global Index is simply a term that is used to define a join index that contains the Row IDs of the base table rows. This means that the user includes the ROWID as a user-defined column within the join index. Row IDs that are included within a join index are always 10 bytes in length, regardless if the base table is partitioned or not.

Options for Join Indexes

- **Sparse Join Indexes**
  - When creating any of the join indexes, you can include a WHERE clause to limit the rows created in the join index to a subset of the rows in the base table or tables.
- **Global Join Indexes**
  - You can include the Row ID of the table(s) within the join index to allow an AMP to join back to the data row for columns not referenced (covered) in the join index.
- **Miscellaneous notes:**
  - Materialized Views are implemented as Join Indexes. Join Indexes improve query performance at the expense of slower updates and increased storage.
  - When you create a Join Index, there is no duplicate row check – you don't have to worry about a high number of NUPI duplicates in the join index.

Join Index Considerations

Join Index considerations include:

You cannot specify a column with a data type of either BLOB or CLOB in the definition of a join index (nor for any other kind of index).

Be aware that when you define a join index using an outer join, you must reference all the columns of the outer table in the select list of the join index definition. If any of the outer table columns are not referenced in the select list for the join index definition, the system returns an error to the requestor.

Perm Space — a Join Index is created as a table-like structure in Teradata and uses Perm space.

Fallback Protection — Join Index subtables can optionally be Fallback-protected.

Because join indexes generated from inner joins do not preserve unmatched rows, you should always consider using outer joins to define simple join indexes, noting the following restrictions.

Inequality conditions are not supported under any circumstances for ON clauses in join index definitions.

Outer joins are not supported under any circumstances for aggregate join indexes.

Load Utilities — MultiLoad and FastLoad utilities cannot be used to load or unload data into base tables that have an associated join index defined on them because join indexes are not maintained during the execution of these utilities. The TPump utility, which perform standard SQL row inserts and updates, can be used because join indexes are properly maintained during the execution of such utilities.

Archive and Restore — archiving is permitted on a base table or database that has a Join Index. Prior to Teradata 13.0, during a restore of such a base table or database, the system does not automatically rebuild the Join Index. Instead, the Join Index is marked as invalid.

Permanent Journal Recovery — using a permanent journal to recover a base table (i.e., ROLLBACK or ROLLFORWARD) with an associated Join Index defined is permitted. The join index is not automatically rebuilt during the recovery process. Instead, the join index is marked as invalid and the join index must be dropped and recreated before it can be used again in the execution of queries.

Collecting Statistics — statistics should be collected on the primary index and secondary indexes of the Join Index to provide the Optimizer with baseline statistics, including the total number of rows in the Join Index.

Join Index Considerations

Join Index considerations include:

- Join indexes are updated automatically as base tables are updated (additional I/O).
- Take up PERM space and may (or may not) be Fallback protected.
- You can specify no more than 64 columns per referenced base table per join index.
- BLOB and CLOB data types cannot be defined within a Join Index.
- Load utilities such as MultiLoad and FastLoad can't be used (use TPump).
- With a multi-table join index, you can specify INNER, LEFT OUTER, and RIGHT OUTER joins, but not a FULL OUTER join.

In many respects, a Join Index is similar to a base table.

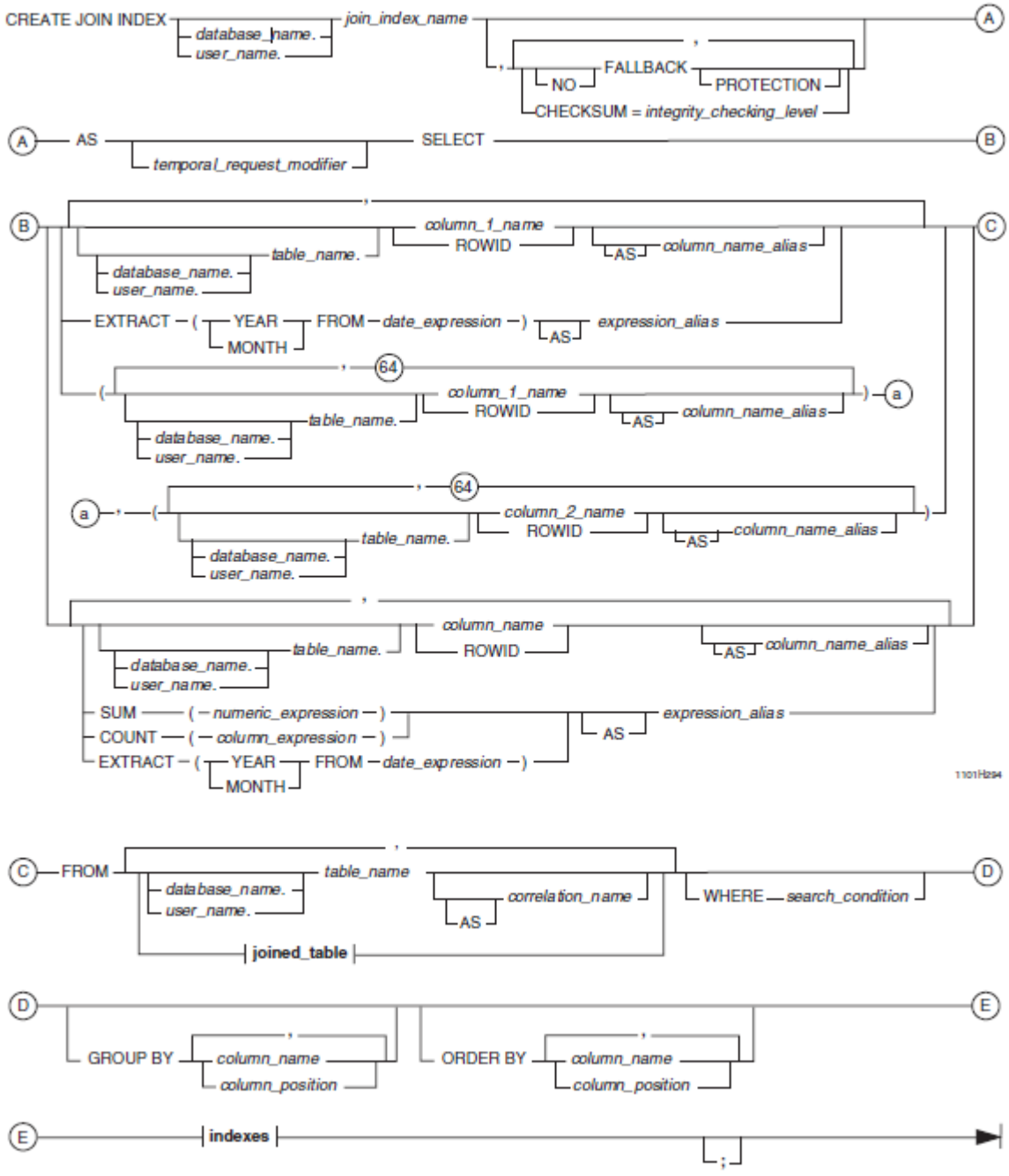
- You may create non-unique secondary indexes on its columns.
- Perform COLLECT/DROP STATISTICS, DROP, HELP, and SHOW statements.

Unlike base tables, you *cannot* do the following:

- Directly query or update join index rows.
- Create unique indexes on its columns.
- Store and maintain arbitrary query results such as expressions.

Join Index Example – Customer and Order Tables

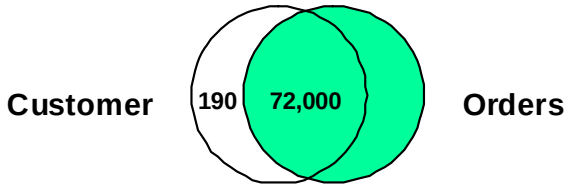
The CREATE JOIN INDEX syntax is shown below. The reference manual provides detailed for all of the CREATE JOIN INDEX options.



Join Index Example
Customer and Order Tables

```
CREATE SET TABLE Customer
    ( custid        INTEGER NOT NULL,
      lname         VARCHAR(15),
      fname         VARCHAR(10),
      address       VARCHAR(50),
      city          VARCHAR(20),
      state         CHAR(2),
      zipcode       INTEGER)
UNIQUE PRIMARY INDEX (custid);
```

```
CREATE SET TABLE Orders
    ( orderid       INTEGER NOT NULL,
      custid        INTEGER NOT NULL,
      orderstatus   CHAR(1),
      totalprice    DECIMAL(9,2) NOT NULL,
      orderdate     DATE
                    FORMAT 'YYYY-MM-DD' NOT NULL,
      orderpriority SMALLINT,
      clerkid       CHAR(16),
      shippriority  SMALLINT,
      ordercomment  VARCHAR(79))
UNIQUE PRIMARY INDEX (orderid);
```



5000 Customers and 72,000 Orders.
72,000 orders are associated with valid customers.
190 customers have no orders.

Compressed Multi-Table Join Index

The facing page includes an example of creating a Multiple Table Join Index using the repeating group option.

The storage organization for join indexes supports a compressed format to reduce storage space.

If you know that a join index contains groups of rows with repeating information, then its definition DDL can specify repeating groups, indicating the repeating columns in parentheses. The column list is specified as two groups of columns, with each group stipulated within parentheses. The first group contains the fixed columns and the second group contains the repeating columns.

As another option, you can elect to store join indexes in value order, ordered by the values of a 4-byte column. Value-ordered storage provides better performance for queries that specify selection constraints on the value ordering column. For example, suppose a common task is to look up sales information by order date. You can create a join index on the Orders table and order it by order date. The benefit is that queries that request orders by order date only need to access those data blocks that contain the value or range of values that the queries specify.

Physical Join Index Row and Compression

A physical join index row has two parts:

A required fixed part that is stored only once.
An optional repeated part that is stored as often as needed.

For example, if a logical join result has *n* rows with the same fixed part value, then there is one corresponding physical join index row that includes *n* repeated parts in the physical join index. A physical join index row represents one or more logical join index rows. The number of logical join index rows represented in the physical row depends on how many repeated values are stored.

Limitations

For a compressed multi-table join index, the maximum number of columns defined in the fixed portion is 64 and the maximum number of columns defined in the repeating portion is also 64. The total maximum number of columns in this type of join index is 128.

With a LEFT or RIGHT OUTER join, at least 1 column from each inner table must be NOT NULL.

FULL OUTER joins are not allowed with a compressed multi-table join index.

Compressed Multi-Table Join Index

```
CREATE JOIN INDEX    Cust_Ord_JI AS
      SELECT         (custid, lname),
                     (orderid, orderstatus, orderdate)
      FROM           Customer C
      INNER JOIN     Orders O
      ON             C.custid = O.custid
      PRIMARY INDEX  (custid);
```

Fixed portion of Join Index (max # columns is 64)

Repeating portion of Join Index (max # of columns is 64)

Maximum # of columns for compressed join index is 128.

Fixed Portion		Variable Portion		
custid	lname	orderid	orderstatus	orderdate
1443	Woods	102292	C	2008-11-30
		135893	C	2009-11-30
		157093	O	2011-10-16
		135993	C	2009-12-14
4000	Mickelson	142000	C	2009-12-20
		149600	C	2009-12-29
		154798	C	2010-04-17
5809	Furyk	149698	C	2009-12-31
		156599	C	2010-10-05
		101199	C	2008-09-11
		158999	O	2011-10-23
		152399	C	2010-06-30

Within the join index, this is one row of data representing the orders for a customer.

One row in Join Index.

One row in Join Index.

Non-Compressed Multi-Table Join Index

The facing page includes an example of creating a Multiple Table Join Index without using the repeating group option.

The storage space in this example for the non-compressed multi-table join index will be higher.

For a non-compressed multi-table join index, the maximum number of columns defined per referenced table is 64. The total maximum number of columns in this type of join index is 2048.

PERM Space Required

The amount of PERM space used by the compressed multi-table join index (previous example) and the non-compressed join index is listed below. Remember that these tables are quite small and note that the join index with repeating data requires less storage space.

TableName	SUM(CurrentPerm)
Cust_Ord_JI	1,044,480
Cust_Ord_JI2	2,706,432

Non-Compressed Multi-Table Join Index

```
CREATE JOIN INDEX    Cust_Ord_JI2 AS
      SELECT         custid, lname,
                     orderid, orderstatus, orderdate
      FROM           Customer C
      INNER JOIN     Orders O
      ON             C.custid = O.custid
      PRIMARY INDEX  (custid);
```

Max # columns per referenced table is 64

Maximum # of columns for non-compressed join index is 2048.

Fixed Portion		Variable Portion		
custid	lname	orderid	orderstatus	orderdate
1443	Woods	102292	C	2008-11-30
1443	Woods	135893	C	2009-11-30
1443	Woods	157093	O	2011-10-16
1443	Woods	135993	C	2009-12-14
4000	Mickelson	142000	C	2009-12-20
4000	Michelson	149600	C	2009-12-29
4000	Michelson	154798	C	2010-04-17
5809	Furyk	149698	C	2009-12-31
5809	Furyk	156599	C	2010-10-05
5809	Furyk	101199	C	2008-09-11
5809	Furyk	158999	O	2011-10-23
5809	Furyk	152399	C	2010-06-30

Within the join index, each order is represented by a separate join index row.

Compressed and Non-Compressed Join Indexes

The facing page illustrates the difference between a compressed and a non-compressed join index.

PERM Space Required

The amount of PERM space used by the compressed multi-table join index (previous example) and the non-compressed join index is listed below. Remember that these tables are quite small and note that the join index with repeating data requires less storage space.

TableName	SUM(CurrentPerm)
Cust_Ord_JI	1,044,480
Cust_Ord_JI2	2,706,432

Compressed and Non-Compressed Join Indexes

Example of storage of compressed and non-compressed join indexes.

```
CREATE JOIN INDEX Cust_Ord_JI AS
SELECT      (custid, lname),
            (orderid, orderstatus, orderdate)
FROM        Customer C
INNER JOIN  Orders O
ON          C.custid = O.custid
PRIMARY INDEX  (custid);
```

```
CREATE JOIN INDEX Cust_Ord_JI2 AS
SELECT      custid, lname,
            orderid, orderstatus, orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          C.custid = O.custid
PRIMARY INDEX  (custid);
```

Compressed Join Index

Join Index Row ID		Fixed Columns	Repeating Columns (Orders)		
RH		5001 ...	Order1	Order2	Order3
RH		5002 ...	Order4	Order5	

Non-Compressed Join Index

Join Index Row ID		Join Index Column Values	
RH		5001 ...	Order1
RH		5001 ...	Order2
RH		5001 ...	Order3
RH		5002 ...	Order4
RH		5002 ...	Order5

```
SELECT      TableName, SUM(CurrentPerm)
FROM        DBC.TableSize
WHERE       DatabaseName = USER
GROUP BY    1 ORDER BY 1;
```

TableName	SUM(CurrentPerm)
Cust_Ord_JI	1,044,480
Cust_Ord_JI2	2,706,432

Example 1 – Does a Join Index Help?

This EXPLAIN is **without** a Join Index.

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.O.
2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
3) We lock TFACT.O for read, and we lock TFACT.C for read.
4) We do an all-AMPs RETRIEVE step from TFACT.O by way of an all-rows scan with a condition of ("TFACT.O.orderstatus = 'O'") into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with high confidence to be 3,970 rows. The estimated time for this step is 0.41 seconds.
5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of a RowHash match scan, which is joined to TFACT.C by way of a RowHash match scan with no residual conditions. Spool 2 and TFACT.C are joined using a merge join, with a join condition of ("TFACT.C.custid = custid"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 3,970 rows. The estimated time for this step is 0.07 seconds.
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.49 seconds.

This EXPLAIN is **with** a Join Index.

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.CUST_ORD_JI.
2) Next, we lock TFACT.CUST_ORD_JI for read.
3) We do an all-AMPs RETRIEVE step from TFACT.CUST_ORD_JI by way of an all-rows scan with a condition of ("TFACT.CUST_ORD_JI.orderstatus = 'O'") into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with no confidence to be 3970 rows. The estimated time for this step is 0.18 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.18 seconds.

Example 1 – Does a Join Index Help?

List the valid customers who have open orders?

```
SELECT      C.custid, C.lname, O.orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          C.custid = O.custid
WHERE       orderstatus = 'O'
ORDER BY    1;
```

custid	lname	orderdate
1391	Poppy	2011-10-06
1906	Putman	2011-10-22
1969	Mitchell	2011-09-14
2916	Rotter	2011-10-24
2954	Agnew	2011-10-15
4336	Carson	2011-09-20
5396	Murphy	2011-10-21
:	:	:

SQL Query	Time
Without Join Index	.49 seconds
With Join Index	.18 seconds

All referenced columns are part of the Join Index.

Optimizer picks Join Index rather than doing a join.

Join Index covers query and helps this query.

The Compressed Join Index was used for this example.

Example 2 – Does a Join Index Help?

This EXPLAIN is **without** a Join Index.

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.O.
2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
3) We lock TFACT.O for read, and we lock TFACT.C for read.
4) We do an all-AMPs RETRIEVE step from TFACT.O by way of an all-rows scan with a condition of ("TFACT.O.orderstatus = 'O'") into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with high confidence to be 3,970 rows. The estimated time for this step is 0.41 seconds.
5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of a RowHash match scan, which is joined to TFACT.C by way of a RowHash match scan with no residual conditions. Spool 2 and TFACT.C are joined using a merge join, with a join condition of ("TFACT.C.custid = custid"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 3,970 rows. The estimated time for this step is 0.07 seconds.
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.49 seconds.

This EXPLAIN is **with** a Join Index.

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.CUST_ORD_JI.
2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
3) We lock TFACT.CUST_ORD_JI for read, and we lock TFACT.C for read.
4) We do an all-AMPs JOIN step from TFACT.C by way of a RowHash match scan with no residual conditions, which is joined to TFACT.CUST_ORD_JI by way of a RowHash match scan with a condition of ("TFACT.CUST_ORD_JI.orderstatus = 'O'"). TFACT.C and TFACT.CUST_ORD_JI are joined using a merge join. The input table TFACT.CUST_ORD_JI will not be cached in memory. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 3,970 rows. The estimated time for this step is 0.25 seconds.
5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.25 seconds.

Example 2 – Does a Join Index Help?

List the valid customers and their addresses who have open orders?

```
SELECT      C.custid, C.lname,
            C.address, C.city, C.state,
            O.orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          C.custid = O.custid
WHERE       O.orderstatus = 'O'
ORDER BY    1;
```

SQL Query	Time
Without Join Index	.49 seconds
With Join Index	.25 seconds

- Some of the referenced columns are NOT part of the Join Index. The Join Index does not cover the query, but is used in this example.
- A Join Index is used in this query and is merge joined with the Customer table.

Results:

custid	lname	address	city	state	orderdate
1391	Poppy	2300 Madrona Ave	Carson City	NV	2011-10-06
1906	Putman	903 La Pierre Ave	Jackson	MS	2011-10-22
1969	Mitchell	36 Main Street	New York	NY	2011-09-14
2916	Rotter	4564 Long Beach Blvd	Trenton	NJ	2011-10-24
2954	Agnew	1083 Beryl Ave	Los Angeles	CA	2011-10-15
4336	Carson	4021 Eleana Way	St. Paul	MN	2011-09-20
5396	Murphy	5603 Main Street	Pierre	SD	2011-10-21
:	:	:	:	:	:

Example 3 – Partitioning a Join Index

The facing page includes an example of partitioning a non-compressed join index.

Example 3 – Partitioning a Join Index

```
CREATE JOIN INDEX     Cust_Ord_JI3 AS
      SELECT          C.custid, C.lname,
                      O.orderid, O.orderstatus, O.orderdate
      FROM            Customer C
      INNER JOIN      Orders O
      ON              C.custid = O.custid
      PRIMARY INDEX   (custid)
          PARTITION BY RANGE_N
              (orderdate BETWEEN DATE '2003-01-01' AND DATE '2012-12-31'
                                                  EACH INTERVAL '1' MONTH) ;
```

List the valid customers who have open Orders for August, 2012?

```
SELECT      C.custid, C.lname, O.orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          C.custid = O.custid
WHERE       O.orderstatus = 'O'
AND         O.orderdate BETWEEN DATE '2012-08-01' AND DATE '2012-08-31'
ORDER BY    1;
```

The join index is used in this query and partition elimination will occur on the join index.

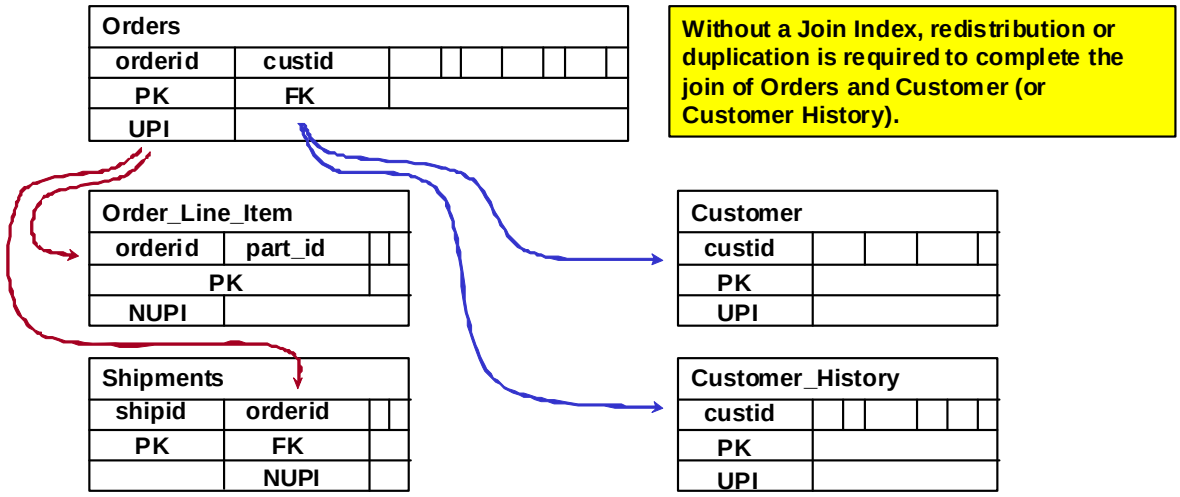
Join Index – Single Table

A denormalization technique is to replicate a column in a table to avoid joins. If an SQL query would benefit from replicating some or all of its columns in another table that is hashed on the join field (usually the primary index of the table to which it is to be joined) rather than the primary index of the original base table, then you should consider creating one or more single table join indexes on that table.

For example, you might want to create a single table join index to avoid redistributing a large base table or to avoid the possibly prohibitive storage requirements of a multi-table join index. For example, a single table join index might be useful for commonly made joins having low predicate selectivity but high join selectivity.

Join Index – Single Table

- The **Single Table Join Index** is useful for resolving joins on large tables without having to redistribute the joined rows across the AMPs.
- In some cases, this may perform better than building a multi-table join index on the same columns.



Join Index – Single Table (cont.)

You can also define a simple join index on a single table. This permits you to hash some or all of the columns of a large replicated base table on a foreign key that hashes rows to the same AMP as another large table. In some situations, this may perform better than building a multi-table join index on the same columns. The advantage comes from less under-the-covers update maintenance on the single table form of the index. Only testing can determine which is the better design for a given set of tables, applications, and hardware configuration.

The example on the facing page shows a technique where the join index is effectively substituted for the underlying base table. The join index has a primary index that ensures that rows are hashed to the same AMPs as rows in tables being joined. This eliminates the need for row redistribution when the join is made.

Even though each single table join index you create partly or entirely replicates its base table, you cannot query or update them directly just as you cannot directly query or update any other join index.

In this example, the compressed format for a single table join index can be used.

```
CREATE JOIN INDEX     Orders_JI  AS
    SELECT            (custid),
                      (orderid,
                       orderstatus,
                       totalprice,
                       orderdate)
    FROM              Orders
    PRIMARY INDEX     (custid);
```

PERM Space Required

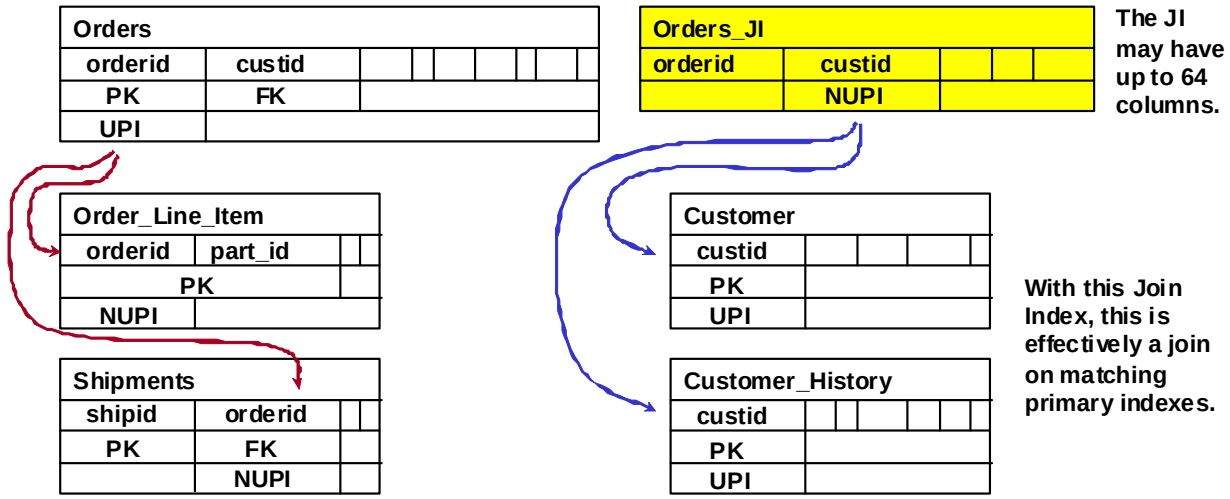
The amount of PERM space used by the compressed multi-table join index (previous example) and the single table join index is listed below. Remember that these tables are quite small and note that the join index with repeating data requires less storage space.

TableName	SUM(CurrentPerm)
Cust_Ord_JI	1,044,480
Orders_JI	1,238,584
Orders_JI2	2,308,608

Note that in this example that the single table join index uses more permanent space. However, the single table join index has some columns (from the Orders table) that not part of the compressed multi-table join index.

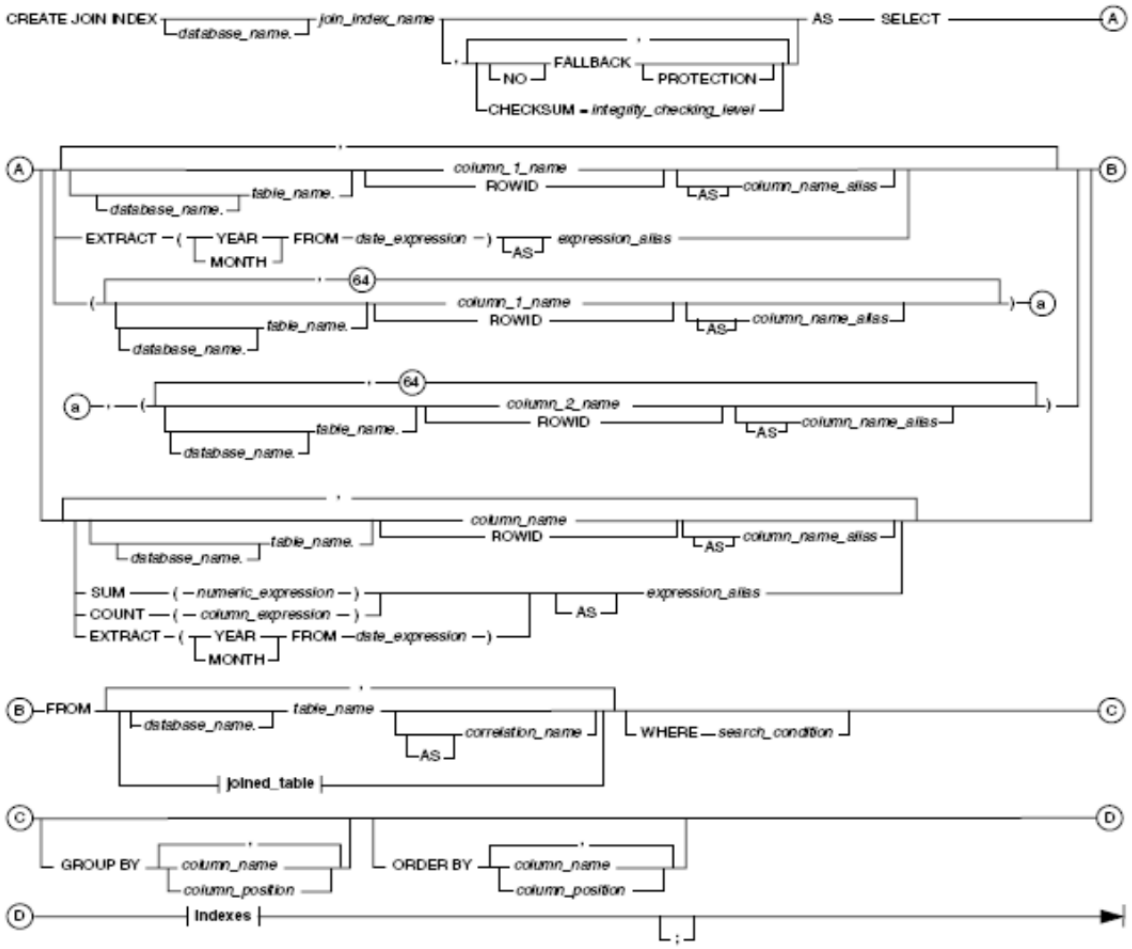
Join Index – Single Table (cont.)

- **Possible advantages include:**
  - **Less update maintenance on the single table Join Index than a multi-table Join Index.**
  - **Maybe less storage space for a single table Join Index than for a multi-table Join Index.**

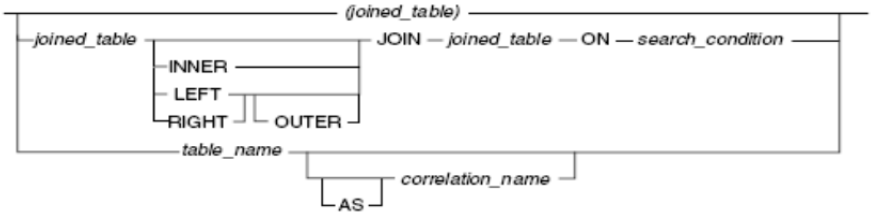


Creating a Join Index – Single Table

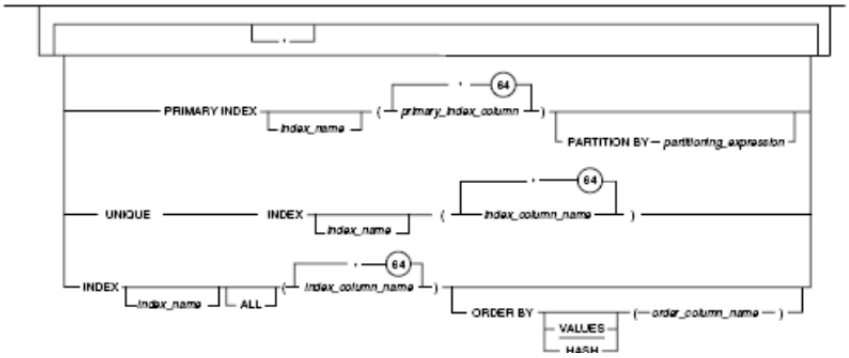
The CREATE JOIN INDEX syntax is shown below.



Joined_Table Excerpt



Indexes Excerpt



Creating a Join Index – Single Table

Compressed

```
CREATE JOIN INDEX  Orders_JI  AS
    SELECT              (custid),
                        (orderid,
                         orderstatus,
                         totalprice,
                         orderdate)
    FROM                Orders
    PRIMARY INDEX       (custid);
```

Non-Compressed

```
CREATE JOIN INDEX  Orders_JI2  AS
    SELECT              orderid,
                        custid,
                        orderstatus,
                        totalprice,
                        orderdate
    FROM                Orders
    PRIMARY INDEX       (custid);
```

The Orders base table is distributed across the AMPs based on the hash value of the orderid column (primary index of base table).

The Join Index (Orders_JI) effectively represents a subset of the Orders table (selected columns) and is distributed across the AMPs based on the hash value of the custid column.

The optimizer can use this Join Index to improve joins using the "customer id" to join with the Orders table.

Example 4 – Does the Join Index Help?

This EXPLAIN is **without** a Join Index.

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.O.
2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
3) We lock TFACT.O for read, and we lock TFACT.C for read.
4) We do an all-AMPs RETRIEVE step from TFACT.O by way of an all-rows scan with a condition of ("TFACT.O.orderstatus = 'O'") into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with high confidence to be 3,970 rows. The estimated time for this step is 0.41 seconds.
5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of a RowHash match scan, which is joined to TFACT.C by way of a RowHash match scan with no residual conditions. Spool 2 and TFACT.C are joined using a merge join, with a join condition of ("TFACT.C.custid = custid"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 3,970 rows. The estimated time for this step is 0.07 seconds.
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.49 seconds.

This EXPLAIN is **with** a Single Table Join Index (compressed single table join index).

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.ORDERS_JI.
2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
3) We lock TFACT.ORDERS_JI for read, and we lock TFACT.C for read.
4) We do an all-AMPs JOIN step from TFACT.C by way of a RowHash match scan with no residual conditions, which is joined to TFACT.ORDERS_JI by way of a RowHash match scan with a condition of ("TFACT.ORDERS_JI.orderstatus = 'O'"). TFACT.C and TFACT.ORDERS_JI are joined using a merge join, with a join condition of ("TFACT.C.custid = TFACT.ORDERS_JI.custid"). The input table TFACT.ORDERS_JI will not be cached in memory. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 3,970 rows. The estimated time for this step is 0.22 seconds.
5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.22 seconds.

Note: The EXPLAIN cost estimates were the same for both Orders_JI (compressed join index) and Orders_JI2 (non-compressed join index).

Example 4 – Does the Join Index Help?

List the valid customers who have open orders?

```
SELECT      C.custid, C.lname, O.orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          C.custid = O.custid
WHERE       O.orderstatus = 'O'
ORDER BY    1;
```

custid	lname	orderdate
1391	Poppy	2011-10-06
1906	Putman	2011-10-22
1969	Mitchell	2011-09-14
2916	Rotter	2011-10-24
2954	Agnew	2011-10-15
4336	Carson	2011-09-20
5396	Murphy	2011-10-21
:	:	:

SQL Query	Time
Without Join Index	.49 seconds
With Join Index	.22 seconds

- The rows of the customer table and the Join Index are located on the same AMP.
- A single table Join Index will help this query.

Why use Aggregate Join Indexes?

Summary Tables

Queries that involve counts, sums, or averages over large tables require processing to perform the needed aggregations. If the tables are large, query performance may be affected by the cost of performing the aggregations. Traditionally, when these queries are run frequently, users have built summary tables to expedite their performance. While summary tables do help query performance there are disadvantages associated with them as well.

Summary Tables Limitations

Require the creation of a separate table
Require initial population of the table
Require refresh of changing data, either via update or reload
Require queries to be coded to access summary tables, not the base tables
Allow for multiple versions of the truth when the summary tables are not up-to-date

Aggregate Indexes

The primary function of an aggregate join index is to provide the Optimizer with a performance, cost-effective means for satisfying any query that specifies a frequently made aggregation operation on one or more columns. The aggregate join index permits you to define a summary table without violating schema normalization.

Aggregate indexes provide a solution that enhances the performance of the query while reducing the requirements placed on the user. All of the above listed limitations are overcome with their use.

An aggregate index is created similarly to a join index with the difference that sums, counts and date extracts may be used in the definition. A denormalized summary table is internally created and populated as a result of creation. The index can never be accessed directly by the user. It is available only to the optimizer as a tool in its query planning.

Aggregate indexes do not require any user maintenance. When underlying base table data is updated, the aggregate index totals are adjusted to reflect the changes. While this requires additional processing overhead when a base table is changed, it guarantees that the user will have up-to-date information in the index.

Why use Aggregate Join Indexes?

Summary Tables

- Queries involving aggregations over large tables are subject to high compute and I/O overhead. Summary tables often used to expedite their performance.

Summary Tables Limitations

- Require the creation of a separate summary table.
- Require initial population of the summary table.
- Requires refresh of summary table.
- Queries must access summary table, not the base table.
- Multiple "versions of the truth".

Aggregate Join Indexes

- Aggregate join indexes enhance the performance of the query while reducing the requirements placed on the user.
- An aggregate join index is created similarly to a join index with the difference that sums, counts and date extracts may be used in the definition.

Aggregate Join Index Advantages

- Do not require any user maintenance.
- Updated automatically when base tables change (requires processing overhead)
- User will have up-to-date information in the index.

Aggregate Join Index Properties

Aggregate Indexes are similar to other Join Indexes in that they are:

Automatically kept up to date without user involvement.
Never accessed directly by the user.
Optional and provide an additional choice for the optimizer.
MultiLoad and FastLoad may not be used to load tables for which indexes are defined.

Aggregate Indexes are different from other Join Indexes in that they:

Use the SUM and COUNT functions.
Permit use of EXTRACT YEAR and EXTRACT MONTH from dates.

Define an aggregate join index as a join index that specifies SUM or COUNT aggregate operations. No other aggregate functions are permitted in the definition of a join index.

To avoid numeric overflow, the COUNT and SUM fields in a join index definition must be typed as FLOAT. If you do not assign a data type to COUNT and SUM, the system types them as FLOAT automatically. If you assign a type other than FLOAT, an error message occurs.

You must have one of the following two privileges to create any join index:

CREATE TABLE on the database or user which will own the join index,
or
INDEX privilege on each of the base tables.

Additionally, you must have this privilege:

DROP TABLE rights on each of the base tables.

Aggregate Join Index Properties

Aggregate Indexes are similar to other Join Indexes:

- Automatically kept up to date without user involvement.
- Never accessed directly by the user.
- Optional and provide an additional choice for the optimizer.
- MultiLoad and FastLoad may NOT be used to load tables for which indexes are defined.

Aggregate Indexes differ from other Join Indexes:

- Use the SUM and COUNT functions.
- Permit use of EXTRACT YEAR and EXTRACT MONTH from dates.

Privileges required to create any Join Index:

- CREATE TABLE in the database or user which will own the join index, or INDEX privilege on each of the base tables.

Additionally, you must have this privilege:

- DROP TABLE rights on each of the base tables.

Aggregation without an Aggregate Index

The facing page contains an example of aggregation and the base table does NOT have an aggregate index.

The Daily_Sales table has 35 item_ids and a row for every item for every day from 2002 through 2007.

The Daily_Sales table has 76,685 rows (2191 days x 35 items).

Note: Statistics were collected for all of the columns on the Daily_Sales table.

```
EXPLAIN SELECT
      item_id
     ,EXTRACT (YEAR FROM sales_date) AS Yr
     ,EXTRACT (MONTH FROM sales_date) AS Mon
     ,SUM (sales)
FROM   Daily_Sales
GROUP BY  1, 2, 3
ORDER BY  1, 2, 3;
```

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.Daily_Sales.
2) Next, we lock TFACT.Daily_Sales for read.
3) We do an all-AMPs SUM step to aggregate from TFACT.Daily_Sales by way of an all-rows scan with no residual conditions, and the grouping identifier in field 1. Aggregate Intermediate Results are computed locally, then placed in Spool 3. The input table will not be cached in memory, but it is eligible for synchronized scanning. The aggregate spool file will not be cached in memory. The size of Spool 3 is estimated with low confidence to be 57,514 rows. The estimated time for this step is 0.85 seconds.
4) We do an all-AMPs RETRIEVE step from Spool 3 (Last Use) by way of an all-rows scan into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 57,514 rows. The estimated time for this step is 0.39 seconds.
5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1.

Aggregation without an Aggregate Index

List the sales by Year and Month for every item.

```
SELECT    item_id
          ,EXTRACT (YEAR FROM sales_date) AS Yr
          ,EXTRACT (MONTH FROM sales_date) AS Mon
          ,SUM (sales)
FROM      Daily_Sales
GROUP BY  1, 2, 3
ORDER BY  1, 2, 3;
```

An all-rows scan of base table is required.
The base table has 76,685 rows.

EXPLAIN without an Aggregate Index (Partial Listing)

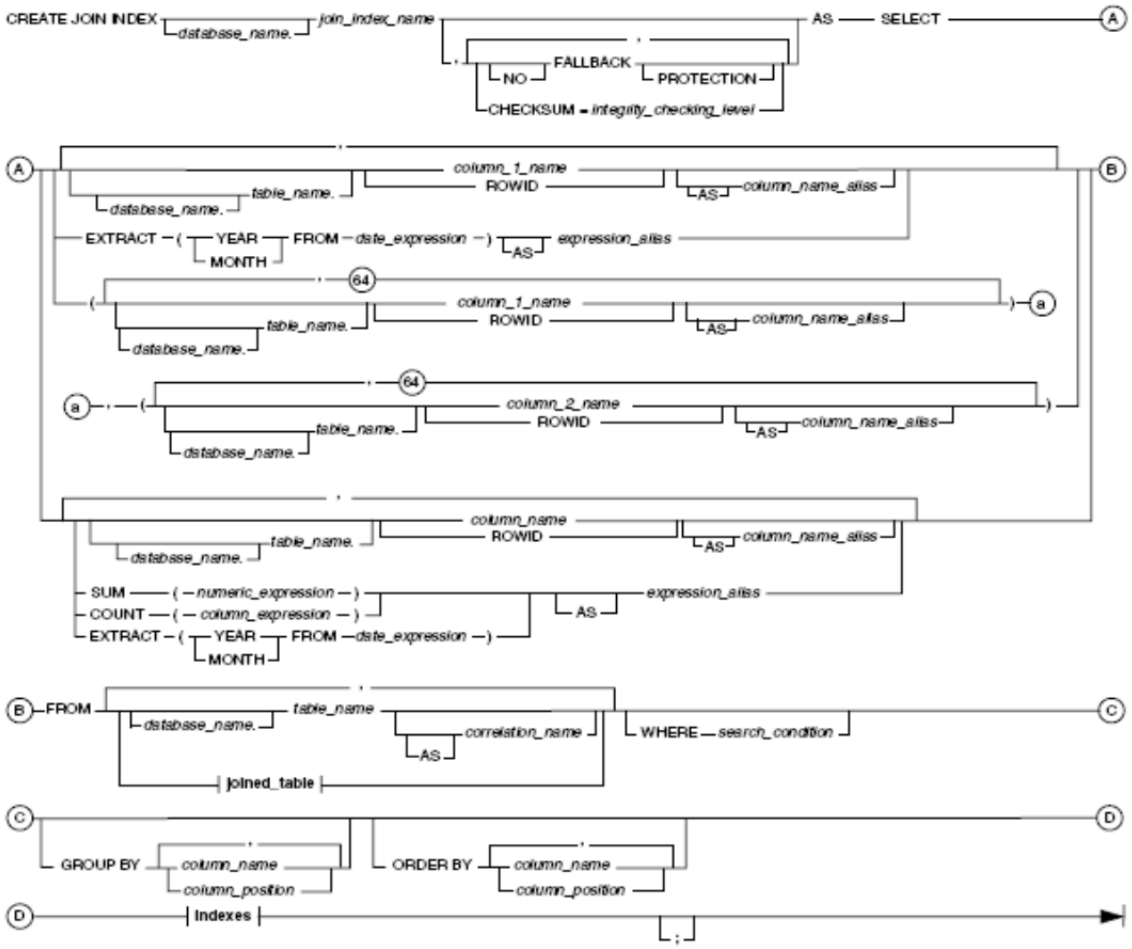
:

3) We do an all-AMPs SUM step to aggregate from TFACT.Daily_Sales by way of an all-rows scan with no residual conditions, and the grouping identifier in field 1. Aggregate Intermediate Results are computed locally, then placed in Spool 3. The input table will not be cached in memory, but it is eligible for synchronized scanning. The aggregate spool file will not be cached in memory. The size of Spool 3 is estimated with low confidence to be 57,514 rows. The estimated time for this step is 0.85 seconds.
4) We do an all-AMPs RETRIEVE step from Spool 3 (Last Use) by way of an all-rows scan into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 57,514 rows. The estimated time for this step is 0.39 seconds.

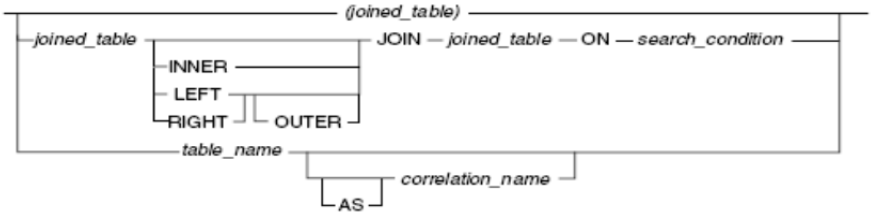
:

Creating an Aggregate Join Index

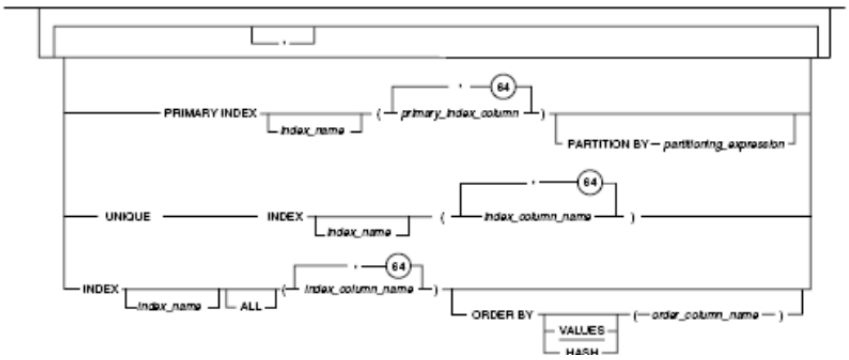
The CREATE JOIN INDEX syntax is shown below.



Joined_Table Excerpt



Indexes Excerpt



Creating an Aggregate Join Index

```
CREATE TABLE  Daily_Sales
    ( item_id          INTEGER NOT NULL
     ,sales_date       DATE FORMAT 'yyyy-mm-dd'
     ,sales            DECIMAL(9,2) )
PRIMARY INDEX (item_id);

CREATE JOIN INDEX  Monthly_Sales_JI  AS
SELECT
      item_id AS Item
     ,EXTRACT (YEAR FROM sales_date) AS Yr
     ,EXTRACT (MONTH FROM sales_date) AS Mon
     ,SUM (sales) AS Sum_of_Sales
FROM  Daily_Sales
GROUP BY 1, 2, 3;
```

```
COLLECT STATISTICS ON Monthly_Sales_JI COLUMN Item;
COLLECT STATISTICS ON Monthly_Sales_JI COLUMN Yr ;
COLLECT STATISTICS ON Monthly_Sales_JI COLUMN Mon;

HELP STATISTICS Monthly_Sales_JI;
```

Date	Time	Unique Values	Column Names
11/08/16	19:41:43	35	Item
11/08/16	19:41:43	10	Yr
11/08/16	19:41:43	12	Mon

Aggregation with an Aggregate Index

Execution of the following SELECT yields the result below:

```
SELECT  item_id
        , EXTRACT (YEAR FROM sales_date) AS Yr
        , EXTRACT (MONTH FROM sales_date) AS Mon
        , SUM (sales)
FROM    Daily_Sales
GROUP BY  1, 2, 3
ORDER BY  1, 2, 3;
```

item_id	Yr	Mon	Sum(Sales)
5001	2002	1	53987.47
5001	2002	2	45235.03
5001	2002	3	53028.29
5001	2002	4	47632.64
5001	2002	5	53592.29
5001	2002	6	51825.00
5001	2002	7	50452.64
5001	2002	8	53028.29
5001	2002	9	47841.75
5001	2002	10	74663.46
5001	2002	11	65094.86
5001	2002	12	74116.94
5001	2003	1	57433.45
5001	2003	2	46217.14
5001	2003	3	57013.05
5001	2003	6	55732.95
:	:	:	:

The complete EXPLAIN output of this SQL statement follows:

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.MONTHLY_SALES_JI.
2) Next, we lock TFACT.MONTHLY_SALES_JI for read.
3) We do an all-AMPs RETRIEVE step from TFACT.MONTHLY_SALES_JI by way of an all-rows scan with no residual conditions into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with high confidence to be 2,520 rows. The estimated time for this step is 0.04 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.04 seconds.

Aggregation with an Aggregate Index

List the sales by Year and Month for every item.

```
SELECT    item_id
         ,EXTRACT (YEAR FROM sales_date) AS Yr
         ,EXTRACT (MONTH FROM sales_date) AS Mon
         ,SUM (sales)
FROM      Daily_Sales
GROUP BY  1, 2, 3
ORDER BY  1, 2, 3;
```

- An all-rows scan of aggregate join index is used.
- The aggregate index consists of 2520 rows versus 76,685 rows in base table.

EXPLAIN with an Aggregate Index (Partial Listing)

:

3) We do an all-AMPs RETRIEVE step from TFACT.MONTHLY_SALES_JI by way of an all-rows scan with no residual conditions into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with high confidence to be 2,520 rows. The estimated time for this step is 0.04 seconds.

:

Sparse Join Indexes

Another capability of the join index allows you to index a portion of the table using the WHERE clause in the CREATE JOIN INDEX statement to limit the rows indexed. You can limit the rows that are included in the join index to a subset of the rows in the table based on an SQL query result. This is also referred to as a "Partial Covering" join index.

Any join index, whether simple or aggregate, multi-table or single-table, can be sparse.

Examples of how the Sparse Join Index may be used include:

Ignore rows that are NULL or are most common
Index rows whose Quantity < 100
Index a time segment of the table – rows that relate to this quarter

Customer Benefit

A sparse index can focus on the portion of the table(s) that is most frequently used.

Reduces the storage requirements for a join index
Makes access faster since the size of the JI is smaller

Like other index choices, a sparse JI should be chosen to support high frequency queries that require short response times. A sparse JI allows the user to:

Use only a portion of the columns in the base table.
Index only the values you want to index.
Ignore some columns, e.g., nulls, to keep access smaller and faster than before.
Avoid maintenance costs for updates

When the index size is smaller there is less work to maintain and updates are faster since there are fewer rows to update. A sparse JI contents can be limited by date, location information, customer attributes, or a wide variety of selection criteria combined with AND and OR conditions.

Performance

Better update performance on the base table when its indexes do not contain the most common value(s)
Smaller index size
Improved IO and storage
Collect statistics on the index even if it is only a single column

Limitations

Sparse Join Indexes follow the same rules as normal Join Indexes

Sparse Join Indexes

Sparse Join Indexes

- Allows you to index a portion of the table using the WHERE clause in the CREATE JOIN INDEX statement to limit the rows indexed.
- Any join index, whether simple or aggregate, multi-table or single-table, can be created as a sparse index.

Examples of how the Sparse Join Index may be used include:

- Ignore rows that are NULL or are most common
- Index rows whose Quantity < 100
- Index a time segment of the table – rows that relate to this quarter

Benefits

- A sparse index can focus on the portion of the table(s) that are most frequently used.
  - Reduces the storage requirements for a join index
  - Faster to create or build
  - Makes access faster since the size of the Join Index is smaller
  - Better update performance on the base table when its indexes do not contain the most common value(s)

Creating a Sparse Join Index

The facing page contains an example of creating a “Sparse Join Index”. The following EXPLAIN shows that the Sparse Join Index is used.

… (Locking steps)
3) We do an all-AMPs RETRIEVE step from TFACT.CUST_ORD_SJI by way of an all-rows scan with a condition of ("(TFACT.CUST_ORD_SJI.orderstatus = 'O') AND (((EXTRACT(DAY FROM (TFACT.CUST_ORD_SJI.orderdate)))= 24) AND ((EXTRACT(MONTH FROM (TFACT.CUST_ORD_SJI.orderdate)))= 08))") into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with no confidence to be 676 rows. The estimated time for this step is 0.06 seconds.
4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.06 seconds.

This following EXPLAIN shows that the Sparse Join Index is **not** used.

… (Locking steps)
4) We do an all-AMPs RETRIEVE step from TFACT.O by way of an all-rows scan with a condition of ("(TFACT.O.orderstatus = 'O') AND (TFACT.O.orderdate = DATE '2011-12-18')") into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with low confidence to be 23 rows. The estimated time for this step is 0.38 seconds.
5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of a RowHash match scan, which is joined to TFACT.C by way of a RowHash match scan with no residual conditions. Spool 2 and TFACT.C are joined using a merge join, with a join condition of ("TFACT.C.custid = custid"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with low confidence to be 23 rows. The estimated time for this step is 0.06 seconds.
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.44 seconds.

PERM Space Required

The amount of PERM space used by this sparse join index as compared to the full join index is listed below.

TableName	SUM(CurrentPerm)
Cust_Ord_JI	1,044,480
Cust_Ord_SJI	322,048

Creating a Sparse Join Index

```
CREATE JOIN INDEX   Cust_Ord_SJI AS
    SELECT          (C.custid, C.lname),
                    (O.orderid, O.orderstatus, O.orderdate)
    FROM            Customer C
    INNER JOIN      Orders O
    ON              C.custid = O.custid
    WHERE           EXTRACT (YEAR FROM O.orderdate) =
                                EXTRACT (YEAR FROM Current_Date)
    PRIMARY INDEX   (custid);
```

In this example, a sparse join index is created just for the year 2012.

```
SELECT      C.custid, C.lname, O.orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          C.custid = O.custid
WHERE       O.orderdate = '2012-08-24'
AND         O.orderstatus = 'O';
```

The join index will be used for this SQL and the EXPLAIN estimated cost is 0.06 seconds.

```
SELECT      C.custid, C.lname, O.orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          C.custid = O.custid
WHERE       O.orderdate = '2011-12-18'
AND         O.orderstatus = 'O';
```

The tables will have to be joined for this SQL and the EXPLAIN estimated cost is 0.44 seconds.

Creating a Sparse Join Index on a Partitioned Table

The facing page contains an example of creating a sparse join index on a partitioned table. The Sparse Join is only 130,560 bytes in size since it is only created for 3 months.

The following EXPLAIN shows the creation of the sparse join index on a set of partitions.

: (Locking Steps)
5) We execute the following steps in parallel.
    1) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.
    2) We do a single-AMP ABORT test from DBC.TVM by way of the unique primary index.
    3) We do an INSERT into DBC.TVFields (no lock required).
    4) We do an INSERT into DBC.TVFields (no lock required).
    5) We do an INSERT into DBC.TVFields (no lock required).
    6) We do an INSERT into DBC.TVFields (no lock required).
    7) We do an INSERT into DBC.TVFields (no lock required).
    8) We do an INSERT into DBC.Indexes (no lock required).
    9) We do an INSERT into DBC.TVM (no lock required).
6) We create the table header.
7) We create the index subtable on TFACT.Orders_PPI.
8) We lock DBC.TVM for write on a RowHash, and we lock DBC.Indexes for write on a RowHash.
9) We execute the following steps in parallel.
    1) We do an INSERT into DBC.Indexes.
    2) We do an INSERT into DBC.Indexes.
    3) We do an INSERT into DBC.Indexes.
    4) We do an INSERT into DBC.Indexes.
    5) We do an INSERT into DBC.Indexes.
    6) We do a single-AMP UPDATE from DBC.TVM by way of the unique primary index with no residual conditions.
    7) We do an all-AMPs RETRIEVE step from 3 partitions of TFACT.Orders_PPI with a condition of ("(TFACT.Orders_PPI.orderdate <= DATE '2012-12-31') AND (TFACT.Orders_PPI.orderdate >= DATE '2012-10-01')") into Spool 1 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 1 by row hash. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with high confidence to be 3,983 rows. The estimated time for this step is 0.05 seconds.
10) We do an all-AMPs MERGE into TFACT.Orders_PPI_JI from Spool 1 (Last Use).
11) We lock a distinct TFACT."pseudo table" for exclusive use on a RowHash to prevent global deadlock for TFACT.Orders_PPI.
12) We lock TFACT.Orders_PPI for exclusive use.
13) We modify the table header TFACT.Orders_PPI and update the table's version number.
14) We lock DBC.AccessRights for write on a RowHash.
15) We INSERT default rights to DBC.AccessRights for TFACT.Orders_PPI_JI.
16) We spoil the parser's dictionary cache for the table.
17) We spoil the parser's dictionary cache for the table.
18) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> No rows are returned to the user as the result of statement 1.

Creating a Sparse Join Index on a Partitioned Table

If the base table is partitioned, a sparse join index may be created for a partition or partitions – only the "partitions of interest" are scanned.

- Creation time of sparse join index is faster because of partition elimination.
- Partition scan of base table instead of full table scan.

```
CREATE SET TABLE Orders_PPI
   ( orderid          INTEGER NOT NULL,
     custid           INTEGER NOT NULL,
          :             :
     orderdate        DATE FORMAT 'YYYY-MM-DD' NOT NULL,
          :             :
     ordercomment     VARCHAR(79))
PRIMARY INDEX       (orderid)
PARTITION BY RANGE_N
   ( orderdate BETWEEN DATE '2003-01-01' AND DATE '2013-12-31'  EACH INTERVAL '1' MONTH ) ;
```

Orders_PPI table is partitioned by month.

```
CREATE JOIN INDEX Orders_PPI_JI AS
     SELECT          orderid, custid, orderstatus, totalprice, orderdate
     FROM            Orders_PPI
     WHERE           orderdate BETWEEN '2012-10-01' AND '2012-12-31'
     PRIMARY INDEX   (custid);
```

The sparse join index is created for the 4th quarter of 2012.

7) We do an all-AMPs RETRIEVE step from 3 partitions of TFACT.Orders_PPI with a condition of ("(TFACT.Orders_PPI.orderdate <= DATE '2012-12-31') AND (TFACT.Orders_PPI.orderdate >= DATE '2012-10-01')") into Spool 1 (all_amps), ...

3 partitions are scanned to build the JI.

ALTERing a Join Index to CURRENT

Staring with Teradata 13.10, you can now specify CURRENT_DATE and CURRENT_TIMESTAMP functions in a join index.

Also starting with Teradata 13.10, Teradata provides a new option with the ALTER TABLE statement to modify a join index that has been defined with a moving CURRENT_DATE (or DATE) or moving CURRENT_TIMESTAMP. This new option is called ALTER TABLE TO CURRENT.

When you specify CURRENT_DATE and CURRENT_TIMESTAMP as part of a partitioning expression for a join index, these functions resolve to the date and timestamp when you define the PPI. To partition on a new CURRENT_DATE or CURRENT_TIMESTAMP value, submit an ALTER TABLE TO CURRENT request.

The options WITH DELETE and WITH INSERT [INTO] *save_table* option are not available for a join index.

ALTERing a Join Index to CURRENT

This Teradata 13.10 option allows you to periodically resolve the CURRENT_DATE (or DATE) and CURRENT_TIMESTAMP of a join index to their current values.

Benefits include:

- **You do not have to use ALTER TABLE to change to partitioning (drop and/or add partitions) for a partitioned join index.**
- **To partition on a new CURRENT_DATE or CURRENT_TIMESTAMP value, simply submit an ALTER TABLE TO CURRENT request.**

  ALTER TABLE join_index_name TO CURRENT;

  Example: ALTER TABLE Cust_Ord_SJI TO CURRENT;

Considerations:

- **The options WITH DELETE and WITH INSERT INTO *save_table* are not available for a join index.**
- **If RANGE_N specifies CURRENT_DATE or CURRENT_TIMESTAMP in a partitioning expression, you cannot use ALTER TABLE to add or drop ranges for the join index. You must use the ALTER TABLE TO CURRENT statement to achieve this function.**

Partitioning a Sparse Join Index

The example on the facing page contains an example of creating a partitioned sparse non-compressed join index and also includes a NUSI on the same partitioned sparse non-compressed join index.

This example also illustrates the use of CURRENT_DATE in the sparse portion and in the partitioning expression of the join index.

Partitioning a Sparse Join Index

A sparse non-compressed join index can also be partitioned.

- This allows a partition scan of the join index instead of full join index.
- This example creates a sparse partitioned join index for the current year.
  - Teradata 13.10 allows the use of CURRENT_DATE (or DATE) and CURRENT_TIMESTAMP in a join index definition.
  - Use ALTER TABLE *join_index* TO CURRENT; to resolve the dates in the next year.
- This example also creates a NUSI on the join index.

```
CREATE JOIN INDEX Orders_PPI_JI2 AS
   SELECT   orderid, custid, orderstatus, totalprice, orderdate, clerkid
   FROM     Orders_PPI
   WHERE    EXTRACT (YEAR FROM orderdate) = EXTRACT (YEAR FROM Current_Date)

PRIMARY INDEX (custid)
   PARTITION BY RANGE_N (orderdate BETWEEN
        CAST((EXTRACT(YEAR FROM CURRENT_DATE) - 1900) * 10000 + 0101) AS DATE) AND
        CAST((EXTRACT(YEAR FROM CURRENT_DATE) - 1900) * 10000 + 1231) AS DATE)
        EACH INTERVAL '1' DAY)

INDEX (clerkid);
```

To resolve the dates in the next year, ALTER TABLE Orders_PPI_JI2 TO CURRENT;

Global (Join) Indexes

A Global Index is a term used to define a join index that contains the Row IDs of the base table rows. Some queries are satisfied by examining only the join index when all referenced columns are stored in the index. Such queries are said to be covered by the join index.

Other queries may use the join index to qualify a few rows, and then refer to the base tables to obtain requested columns that aren't stored in the join index. Such queries are said to be partially-covered by the index. This is referred to as a partially-covered global index.

Because the Teradata Database supports multi-table, partially-covering join indexes, all types of join indexes, except the aggregate join index, can be joined to their base tables to retrieve columns that are referenced by a query but are not stored in the join index. Aggregate join indexes can be defined for commonly-used aggregation queries.

A partial-covering join index takes less space than a covering join index, but in general may not improve query performance by as much. Not all columns that are involved in a query selection condition have to be stored in a partial-covering join index. The benefits are:

Disk storage space for the JI decreases when fewer columns are stored in the JI.
Performance increases when the number of selection conditions that can be evaluated on the join index increases.

When a Row ID is included in a Join Index, 10 bytes are used for the Row ID (Part # + Row Hash + Uniqueness Value). This is true even if the base table is not partitioned. Starting with Teradata 14.0, if the bas table is partitioned with > 65,535 partitions, the RowID in the Join Index is 16 bytes long.

Another use for a Global Join Index for a single table is that of a Hashed NUSI. This capability will be described in more detail later in the module.

Customer Benefit

Partial–Covering Global Join Indexes can provide:

Improved performance for certain class of queries.
Some of the query improvement benefits that join indexes offer without having to replicate all the columns required to cover the queries resulting in better performance.
Improved scalability for certain class of queries

Limitations

Aggregate JI does not support the partial-covering capability.
Global index is not used for correlated sub-queries.
Global index is not supported with FastLoad or MultiLoad.

Global (Join) Indexes

A Global Index is a term used to define a join index that contains the Row IDs of all of the tables in the join index.

Example:

- Assume that you are joining 2 tables (Table_A and Table_B) and each has 100 columns. The join index can include (at most) 64 columns from each base table.
- You can include the ROWID as part of the 64 columns for each table (ex., A.ROWID and B.ROWID). Each join index subtable row will include the Row IDs of the corresponding base table rows for Table_A and Table_B.
- The optimizer can build a plan that uses the join index to get most of the data and can join back to either or both of the tables for the rest of the data.

Another option – use a single table Global Index as a "Hashed NUSI" – the join index contains the "secondary index column" and the Row IDs in the join index.

- Useful when a column is used as a secondary index, but the typical number of rows for a value is much less than the number of AMPs in the system.
- For queries with an equality condition on a fairly unique column, it changes:
  - Table-level locks to row hash locks
  - All-AMP steps to group-AMP steps

Global Index – Multiple Tables

The facing page contains an example of creating a "global" multi-table join index.

The total number of columns in the Customer and Orders table is less than 64. Therefore, you could create a join index that includes all of the columns (16 in this case) from the two tables. However, if one table had 70 columns and the other table had 90 columns, you can only include a maximum of 64 columns from each table. Since the number of columns is more than 64, you would include the most frequently referenced columns from each table. Another reason to only include a subset of columns may be to minimize the size of the join index.

In this example, the join index has the most frequently referenced columns as well as the Row IDs of both tables. The join index subtable row will include the Row ID of the base table row for the Customer table and the Row ID of the base table for the Orders table. The optimizer may choose to use the join index to get most of the data and join back to either the Customer or the Orders table. The optimizer can build execution plans that can join back to either table.

The terminology used in EXPLAIN plans that indicates a join back is "… using a row id join …".

Join back simply means that the ROWID is carried as part of the join index. This permits the index to "join back" to the base row, much like a NUSI does. It is one of the features of Partial-Covering Join Indexes.

```
EXPLAIN  SELECT        C.custid, C.lname,
                       C.address, C.city, C.state,
                       O.orderdate
         FROM          Customer C
         INNER JOIN    Orders O
         ON            C.custid = O.custid
         WHERE         O.orderstatus = 'O'
         ORDER BY      1;
```

This EXPLAIN shows that a Global Join Index is used.

4) We do an all-AMPs RETRIEVE step from TFACT.CUST_ORD_GJI by way of an all-rows scan with a condition of ("TFACT.CUST_ORD_GJI.orderstatus = 'O'") into Spool 2 (all_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 2 by the sort key in spool field1. The size of Spool 2 is estimated with no confidence to be 482 rows. The estimated time for this step is 0.03 seconds.
5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.C by way of an all-rows scan with no residual conditions. **Spool 2 and TFACT.C are joined using a row id join,** with a join condition of ("Field_1 = TFACT.C.RowID"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with no confidence to be 482 rows. The estimated time for this step is 0.06 seconds.

Global Index – Multiple Tables

```
CREATE SET TABLE Customer
    ( custid          INTEGER NOT NULL,
      lname           VARCHAR(15),
      fname           VARCHAR(10),
      address         VARCHAR(50),
      city            VARCHAR(20),
      state           CHAR(2),
      zipcode         INTEGER)
UNIQUE PRIMARY INDEX (custid);
```

```
CREATE SET TABLE Orders
    ( orderid         INTEGER NOT NULL,
      custid          INTEGER NOT NULL,
      orderstatus     CHAR(1),
      totalprice      DECIMAL(9,2) NOT NULL,
      orderdate       DATE
                      FORMAT 'YYYY-MM-DD' NOT NULL,
      orderpriority   SMALLINT,
      clerkid         CHAR(16),
      shippriority    SMALLINT,
      ordercomment    VARCHAR(79))
UNIQUE PRIMARY INDEX (orderid);
```

```
CREATE JOIN INDEX  Cust_Ord_GJI  AS
      SELECT          (C.custid, C.lname, C.ROWID AS crid),
                      (orderid, orderstatus, orderdate, O.ROWID AS orid)
      FROM            Customer C
      INNER JOIN      Orders O
      ON              C.custid = O.custid
      PRIMARY INDEX   (custid);
```

- The Global Index contains those columns most frequently used in queries – effectively used as “covering join index”.
- The Global Index may be used to join back (via the Row ID) to the tables when columns are referenced that are not part of the join index.

Global Index as a “Hashed NUSI”

Another use for a Global Join Index for a single table is that of a Hashed NUSI. An actual NUSI accesses all AMPs, whereas this index only accesses the AMPs that have rows matching the value. The Global Join Index is hashed, and the system uses the hash to access a single AMP, and then uses the Row IDs in the subtable row to then access only those AMPs that have rows.

ODS (Operational Data Store) or tactical queries that involve an equality condition on a fairly unique column can use a global index which will change:

Table-level locks to row hash locks
All-AMP steps to group-AMP steps

Using a global index as a “Hashed NUSI” is similar to a single-table join index with one clear differentiation – it carries a pointer (Row ID) back to the base table, and is used as an alternative means to get to the base table row. It is not used to satisfy a query by itself.

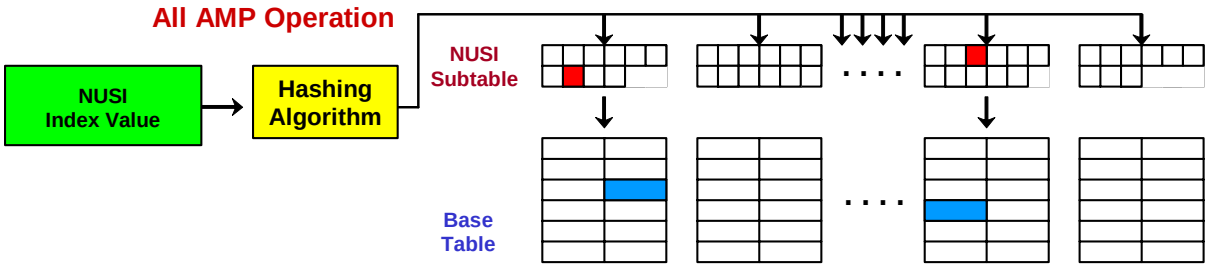
A “hashed NUSI” global index offers the advantages of a NUSI (it supports duplicate rows per value) combined with the advantages of a USI (its index rows are hash-distributed on the indexed value) and is often able to offer group AMP capabilities. As with all Teradata join indexes, it's use is transparent to the query and will be determined by the optimizer.

Its main benefit is for situations where you are only getting a few rows for one value, and you can avoid an all AMP operation that a NUSI always requires. This may not have a huge impact on a system with a modest number of AMPs. However, for very large systems, with hundreds or thousands of AMPs, a group AMP operation that engages a small percentage (e.g., only 1% of the total AMPs or less), when done often enough, may increase overall throughput of the platform, as well as faster query response.

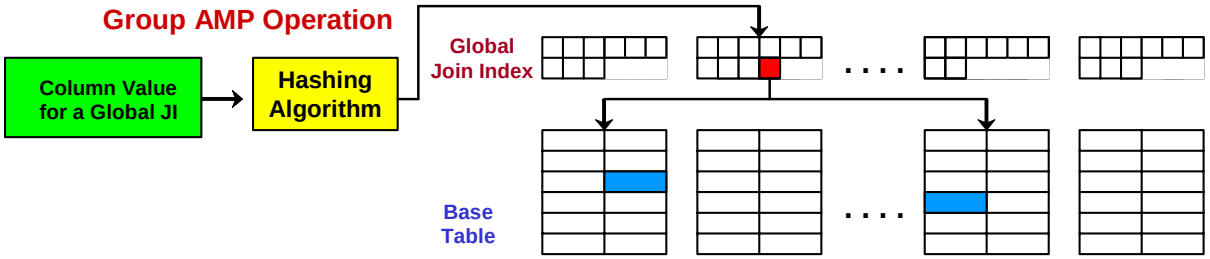
Global Index as a "Hashed NUSI"

Assume 200 AMPs and the typical rows per value is 2 for a column referenced in queries.

- **A NUSI on this column avoids a FTS, but it is an all-AMP operation and every AMP is required to look in the NUSI subtable for the hash value of the NUSI.**



- **A Global Join Index ("Hashed NUSI") on this column utilizes a Group AMP operation. The GJI subtable row contains the Row IDs of the base table rows.**



Creating a Global Index ("Hashed NUSI")

The facing page contains an example creating a Global Join Index as a "Hashed NUSI".

If the number of Row IDs within the Global Index subtable row is less than 50% of the number of AMPs, you will see an EXPLAIN plan with "group-AMPs" operations.

If the number of Row IDs within the Global Index subtable row is more than 50% of the number of AMPs, you will see an EXPLAIN plan with "all -AMPs" operations.

Note: When the Row ID is included in a Join Index, each Row ID is 10 bytes long. The partition number is included even if the base table is not partitioned. The partition number is 0 for non-partitioned tables.

Creating the Global Join Index without "Repeating Row Ids"

If the Global Join Index (that is to be used as a hashed NUSI) is created as follows – without the parenthesis, multiple Row IDs will not be included in a single join index subtable row.

```
CREATE JOIN INDEX     Orders_GI2
AS
SELECT                custid, ROWID
FROM                  Orders
PRIMARY INDEX         (custid);
```

Creating a Global Index ("Hashed NUSI")

```
CREATE JOIN INDEX    Orders_GI AS
      SELECT         (custid),
                     (ROWID)
      FROM           Orders
      PRIMARY INDEX  (custid);
```

Fixed portion of Join Index contains index value

Repeating portion of Join Index contains Row IDs

One Global Index row can contain multiple Row IDs.

- If the typical rows for a value is less than 50% of the number of AMPs, this global index will yield performance gains.

  This effectively means that the number of Row IDs in the subtable row is less than 50% of the number of AMPs.

  - EXPLAIN plan will indicate "group-AMPs" operation and "row hash locks".

- If the typical rows for a value is 50% or greater than the number of AMPs, this global index will result in an all-AMP operation (like a NUSI).

  This effectively means that the number of Row IDs in the subtable row is 50% or more than the number of AMPs.

  - EXPLAIN plan will indicate "all-AMPs" operation and "table level locks".

Example: Using a Global Index as a Hashed NUSI

The EXPLAIN of the SQL statement on the facing page is shown below.

: (locking steps)
3) We do a single-AMP RETRIEVE step from TFACT.ORDERS_GI by way of the primary index "TFACT.ORDERS_GI.custid = 1500" with no residual conditions into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by the sort key in spool field1. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 2 is estimated with high confidence to be 14 rows (564 bytes). The estimated time for this step is 0.00 seconds.
4) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.Orders by way of an all-rows scan with no residual conditions. Spool 2 and TFACT.Orders are joined using a row id join, with a join condition of ("Field_1 = TFACT.Orders.RowID"). The input table TFACT.Orders will not be cached in memory. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with index join confidence to be 14 rows (352 bytes). The estimated time for this step is 0.08 seconds.
5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.08 seconds.

Example: Using a Global Index as a Hashed NUSI

List the orders for a customer id of 1500.

```
SELECT    custid, orderid, orderstatus, orderdate, totalprice
FROM      Orders
WHERE     custid = 1500;
```

custid	orderid	orderstatus	orderdate	totalprice
1500	152400	C	2011-08-15	1150.00
1500	153237	C	2011-08-31	2149.00
1500	156842	O	2011-10-24	1207.50

EXPLAIN showing use of a Global Join Index as a Hashed NUSI (Partial Listing)

:

3) We do a single-AMP RETRIEVE step from TFACT.Orders_GI by way of the primary index "TFACT.Orders_GI.custid = 1500" with no residual conditions into Spool 2 (group_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by the sort key in spool field1. The size of Spool 2 is estimated with high confidence to be 14 rows (564 bytes). The estimated time for this step is 0.00 seconds.
4) We do a group-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.Orders. Spool 2 and TFACT.Orders are joined using a row id join, with a join condition of ("Field_1 = TFACT.Orders.RowID"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with index join confidence to be 14 rows (352 bytes). The estimated time for this step is 0.08 seconds.

:

Repeating Row Ids in Global Index

The SQL to create a Global Join Index with and without repeating Row IDs is shown on the facing page.

PERM Space Required

The amount of PERM space used by these global join indexes is listed below. Remember that these tables are quite small. Note that the global join index with parenthesis requires less space.

```
SELECT    TableName , SUM(CurrentPerm)
FROM      DBC.TableSize
WHERE     DatabaseName = USER
GROUP BY  1
ORDER BY  1;
```

TableName	SUM(CurrentPerm)	
Orders_GI	1,065,984	(with repeating Row IDs)
Orders_GI2	2,019,840	(without repeating Row IDs)

Repeating Row IDs in Global Index

What is the difference in creating a "Hashed NUSI" with or without repeating Row IDs?
Answer – the amount of PERM space needed.

```
CREATE JOIN INDEX  Orders_GI AS
      SELECT       (custid), (ROWID)
      FROM         Orders
      PRIMARY INDEX (custid);
```

Global Join Index

Join Index Row ID		Index Value	(Base Table) RowIDs		
RH		1500	RID1	RID2	RID3
RH		1501	RID4	RID5	

```
CREATE JOIN INDEX  Orders_GI2
      SELECT       custid, ROWID
      FROM         Orders
      PRIMARY INDEX (custid);
```

Global Join Index

Join Index Row ID		Index Value	(Base Table) RowID
RH		1500	RID1
RH		1500	RID2
RH		1500	RID3
RH		1501	RID4
RH		1501	RID5

```
SELECT      TableName,
            SUM(CurrentPerm) AS SumPerm
FROM        DBC.TableSize
WHERE       DatabaseName = USER
GROUP BY    1
ORDER BY    1;
```

TableName	SumPerm	
Orders_GI	1,065,984	(repeating Row IDs)
Orders_GI2	2,019,840	(w/o repeating Row IDs)

Hash Indexes

Hash Indexes are database objects that are user-defined for the purpose of improving query performance. They are file structures that contain properties of both secondary indexes and join indexes. Hash indexes were first introduced with Teradata V2R4.1.

Hash Indexes have an object type of N. Join Indexes have an object type of I.

The hash index provides a space-efficient index structure that can be hash distributed to AMPs in various ways.

The hash index has been designed to improve query performance in a manner similar to a single-table join index. In particular, you can specify a hash index to:

Cover columns in a query so that the base table does not need to be accessed.
Serve as an alternate access path to the base table row.

Example Tables

The same Customer and Orders table definitions are also used with Hash Index examples in this module.

```
CREATE SET TABLE Customer
    (custid              INTEGER NOT NULL,
     lname               VARCHAR(15),
     fname               VARCHAR(10),
     address             VARCHAR(50),
     city                VARCHAR(20),
     state               CHAR(2),
     zipcode             INTEGER)
UNIQUE PRIMARY INDEX (custid);

CREATE SET TABLE Orders
    (orderid             INTEGER NOT NULL,
     custid              INTEGER NOT NULL,
     orderstatus         CHAR(1),
     totalprice          DECIMAL(9,2) NOT NULL,
     orderdate           DATE  FORMAT 'YYYY-MM-DD' NOT NULL,
     orderpriority       SMALLINT,
     clerkid             CHAR(16),
     shippriority        SMALLINT,
     ordercomment        VARCHAR(79))
UNIQUE PRIMARY INDEX (orderid);
```

Note: Statistics have been collected on the primary index, any join columns, and on all hash indexes in these examples.

Hash Indexes

Hash Indexes may also be used to improve query performance. The hash index provides a space-efficient index structure that can be hash distributed to AMPs in various ways.

Similar to secondary indexes in the following ways:

- **Created for a single table only.**
- **The CREATE syntax is simple and very similar to a secondary index.**
- **May cover a query without access of the base table rows.**

Similar to join indexes in the following ways:

- **They "pre-locate" joinable rows to a common location.**
- **The distribution and sequencing of the rows is user specified.**
- **Very similar to single-table join index.**

Unlike join indexes in the following ways:

- **Automatically contains base table PI value as part of hash index subtable row.**
- **No aggregation operators are permitted.**
- **They are always defined on a single table.**
- **No secondary indexes may be built on the hash index.**
- **A trigger and a hash index cannot exist on a table – returns error message #3732.**

Hash Index – Example

The facing page includes an example of creating a hash index that can be used for joins.

It is not necessary to include the "order id" (orderid) in the hash index definition. It is included automatically as part of the hash index. If you include the primary index column(s) in the hash index row, Teradata does not include them a second time in the actual subtable row.

```
CREATE HASH INDEX Orders_HI3
      (orderid, custid, orderstatus, totalprice, orderdate)
ON                  Orders
BY                  (custid)
ORDER BY HASH       (custid);
```

The size of the subtable for Orders_HI1 (facing page) and the Orders_HI3 (above) is the same. The ORDER BY VALUES for **custid** is also a valid option (Orders_HI2) in this example.

Join Index Alternative Technique

A similar effect can be achieved with a single table join index **(STJI)** by adding an explicit ROWID to the join index definition. The ORDER BY VALUES for **custid** is not a valid option with a Join Index in this example.

```
CREATE JOIN INDEX Orders_JI3 AS
SELECT          orderid, custid, orderstatus, totalprice, orderdate,
                ROWID
FROM            Orders
PRIMARY INDEX   (custid);
```

PERM Space Required

The amount of PERM space used by these indexes is listed below. Remember that these tables are quite small.

```
SELECT          TableName, SUM(CurrentPerm)
FROM            DBC.TableSize
WHERE           DatabaseName = USER
GROUP BY 1
ORDER BY 1;
```

TableName	SUM(CurrentPerm)
Orders_HI1	3,606,528
Orders_HI2	3,606,528
Orders_HI3	3,606,528
Orders_JI3	3,028,992

Hash Index – Example

A Hash Index can be ordered by value or hash.

Create a hash index to facilitate joins between the "Orders" and "Customer" tables, based on the PK/FK relationship on "customer id".

```
CREATE HASH INDEX  Orders_HI1
                   (custid, orderstatus, totalprice, orderdate)
ON                 Orders
BY                 (custid)
ORDER BY HASH      (custid);
```

```
CREATE HASH INDEX  Orders_HI2
                   (custid, orderstatus, totalprice, orderdate)
ON                 Orders
BY                 (custid)
ORDER BY VALUES    (custid);
```

Characteristics of these Hash Indexes are:

- Hash index subtable rows are hash distributed by "custid" value.
- The BY option is required when ORDER BY HASH option is specified.

Hash Index – Example (cont.)

This EXPLAIN is executed against the tables **without** a Hash Index.

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.O.
2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
3) We lock TFACT.O for read, and we lock TFACT.C for read.
4) We do an all-AMPs RETRIEVE step from TFACT.O by way of an all-rows scan with a condition of ("(TFACT.O.custid >= 1870) AND ((TFACT.O.custid <= 1900) AND (TFACT.O.orderstatus = 'O'))") into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with low confidence to be 375 rows. The estimated time for this step is 0.39 seconds.
5) We do an all-AMPs JOIN step from TFACT.C by way of a RowHash match scan with a condition of ("(TFACT.C.custid <= 1900) AND (TFACT.C.custid >= 1870)"), which is joined to Spool 2 (Last Use) by way of a RowHash match scan. TFACT.C and Spool 2 are joined using a merge join, with a join condition of ("TFACT.C.custid = custid"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 200 rows. The estimated time for this step is 0.06 seconds.
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.45 seconds.

This EXPLAIN plan is executed with a **Hash Index** created on the table.

1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.ORDERS_HI2.
2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
3) We lock TFACT.ORDERS_HI2 for read, and we lock TFACT.C for read.
4) We do an all-AMPs JOIN step from TFACT.C by way of an all-rows scan with a condition of ("(TFACT.C.custid <= 1900) AND (TFACT.C.custid >= 1870)"), which is joined to TFACT.ORDERS_HI2 with a range constraint of ("(TFACT.ORDERS_HI2.custid >= 1870) AND (TFACT.ORDERS_HI2.custid <= 1900)") with a residual condition of ("(TFACT.ORDERS_HI2.custid >= 1870) AND ((TFACT.ORDERS_HI2.custid <= 1900) AND (TFACT.ORDERS_HI2.orderstatus = 'O'))"). TFACT.C and TFACT.ORDERS_HI2 are joined using a product join, with a join condition of ("TFACT.C.custid = TFACT.ORDERS_HI2.custid"). The input table TFACT.ORDERS_HI2 will not be cached in memory, but it is eligible for synchronized scanning. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with no confidence to be 267 rows. The estimated time for this step is 0.06 seconds.
5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.06 seconds.

Hash Index – Example (cont.)

List the customers with customer ids between 1870 and 1900 who have open orders?

```
SELECT      C.custid, C.lname,
            O.orderid, O.orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          C.custid = O.custid
WHERE       O.orderstatus = 'O'
AND         C.custid BETWEEN 1870 AND 1900
ORDER BY    1;
```

custid	lname	orderid	orderdate
1875	Porter	151105	2011-10-02
1876	Hengster	151166	2011-10-02
:	:	:	:

SQL Query	Time
Without Hash Index	.45 seconds
With Hash Index	.06 seconds

- The rows of the customer table and the Hash Index are located on the same AMP.
- This Hash Index utilizes the range constraint within the query.

EXPLAIN using Hash Index (Partial Listing)

4) We do an all-AMPs JOIN step from TFACT.C by way of an all-rows scan with a condition of ("(TFACT.C.custid <= 1900) AND (TFACT.C.custid >= 1870)"), which is joined to TFACT.ORDERS_HI2 with a range constraint of ("(TFACT.ORDERS_HI2.custid >= 1870) AND (TFACT.ORDERS_HI2.custid <= 1900)") with a residual condition of ("(TFACT.ORDERS_HI2.custid >= 1870) AND ((TFACT.ORDERS_HI2.custid <= 1900) AND (TFACT.ORDERS_HI2.orderstatus = 'O'))").

Summary

The facing page summarizes the key topics presented in this module.

Summary

Teradata provides additional index choices that can be used to improve performance for known queries.

Reasons to use a Join Index:

- May be used to pre-join multiple tables.
- May be used as an aggregate index.
- The WHERE clause can be used to limit the number of rows in the join index.
  - Referred to a "Sparse Index".
- Row ID(s) of table (or tables) can be included to create a "Global Index".
- May be used as a "Hashed NUSI".
- Secondary indexes can be created on a join index. Secondary indexes can be ordered by value or hash.

Reasons to use a Hash index (instead of a Join Index):

- Automatically includes the Primary Index value.
- The syntax is similar to secondary index syntax, thus simpler SQL to code.
- The Hash Index can be ordered by value or hash.

Module 30: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 30: Review Questions

Check the box if the attribute applies to the index.

	Compressed Join Index Syntax	Non-Compressed Join Index Syntax	Aggregate Join Index	Sparse Join Index	Hash Index
May be created on a single table					
May be created on multiple tables					
Requires the use of SUM or COUNT functions					
Requires a WHERE condition to limit rows stored in the index.					
Automatically updated as base table rows are inserted or updated					
Automatically includes the base table PI value as part of the index					

Lab Exercise 30-1

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Use the following SQL for this exercise.

To populate a table:

```
INSERT      INTO   new_tablename
SELECT  *   FROM  existing_tablename;
```

Lab Exercise 30-1

Lab Exercise 30-1

Purpose

In this lab, you will use Teradata SQL Assistant to create a join index and evaluate the Explains of various joins.

What you need

Populated PD tables and Employee, Department, and Job tables in your database

Tasks

1. Verify the number of rows in Employee, Job, and Department tables. If not correct, use INSERT/SELECT to populate the tables from the PD database.

Employee	Count = 1000
Department	Count = 60
Job	Count = 66

2. EXPLAIN the following SQL statement.

```
SELECT      Last_Name, First_Name, Dept_Name, Job_Desc
FROM        Employee E
INNER JOIN  Department D      ON  E.Dept_Number = D.Dept_Number
INNER JOIN  Job J             ON  E.Job_Code = J.Job_Code
ORDER BY    3, 1, 2;
```

What is estimated time cost for this EXPLAIN? 

Lab Exercise 30-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Use the following SQL for this exercise.

To create a join index:

```
CREATE JOIN INDEX join_index_name AS
SELECT .... ;
```

SUM of Perm space using the DBC.TableSizeV view.

```
SELECT      TableName (CHAR(15)), SUM(CurrentPerm)
FROM        DBC.TableSizeV
WHERE       DatabaseName = DATABASE
AND         TableName = 'join_index_name'
GROUP BY    1
ORDER BY    1;
```

Lab Exercise 30-1 (cont.)

3. Create a "non-compressed" join index which includes the following columns of Employee, Department, and Job. 

   Employee – Last_name, First_Name
   Department – Dept_Number, Dept_Name
   Job – Job_Code, Job_Desc

   Execute the HELP USER command. What is the object type of the Join Index? _____

4. EXPLAIN the following SQL statement (same SQL as step #2)

```
SELECT      Last_Name, First_Name, Dept_Name, Job_Desc
FROM        Employee E
INNER JOIN  Department D   ON  E.Dept_Number = D.Dept_Number
INNER JOIN  Job J          ON  E.job_code = J.job_code
ORDER BY    3, 1, 2;
```

   What is estimated time cost for this EXPLAIN? _  _____

   Is the join index used? _____

   How much space does the join index require (use the DBC.TableSizeV view)? ___  _____

Lab Exercise 30-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Lab Exercise 30-1 (cont.)

5. EXPLAIN the following SQL statement – Salary_Amount has been added as a projected column.

```
SELECT      Last_Name, First_Name, Dept_Name, Job_Desc, Salary_Amount
FROM        Employee E
INNER JOIN  Department D   ON  E.Dept_Number = D.Dept_Number
INNER JOIN  Job J          ON  E.Job_Code = J.Job_Code
ORDER BY    3, 1, 2;
```

   What is estimated time cost for this EXPLAIN? _____

   Is the join index used? _____ If not, why not? _____

6. Drop the Join Index.

7. (Optional) Create a new "non-compressed" join index similar to step #3 except include the Primary Index of Employee_Number for the Join Index.

   Employee – Last_name, First_Name, Employee_Number
   Department – Dept_Number, Dept_Name
   Job – Job_Code, Job_Desc

8. (Optional) EXPLAIN the SQL statement from step #5.

   Is the join index used? _____ If so, why? _____

9. (Optional) Drop the Join Index.

Notes

Module 31

TERADATA
Raising Intelligence

Miscellaneous SQL Features

After completing this module, you will be able to:

- **State the purpose and function of the session setting flags.**
- **Recognize differences in transaction modes for Teradata and ANSI.**
- **Distinguish between ANSI and Teradata case sensitivities.**
- **Describe 2 features of the System Calendar.**
- **Describe how space is allocated for volatile and global temporary tables.**

Notes

Table of Contents

Teradata SQL 31-4
Who is ANSI? 31-4
Teradata SQL and ANSI Differences 31-6
SQL Session Modes 31-8
Transaction Modes – Teradata 31-10
Transaction Modes – ANSI 31-12
Duplicate Rows 31-14
Transaction Mode Examples 31-16
Multi-Statement Requests 31-18
CASE Sensitivity Issues 31-20
Teradata Mode 31-20
ANSI Mode 31-20
Using ANSI Blind Test 31-20
Setting the SQL Flagger 31-22
SQLFLAG Example 31-24
HELP SESSION Command 31-26
BTEQ .SHOW Command 31-26
Why a System Calendar? 31-28
Calendar View Layout 31-30
One Row in the Calendar 31-32
Using the Calendar 31-34
Temporary Table Choices 31-36
Derived Tables Revisited 31-38
Volatile Tables 31-40
Secondary Indexes and Volatile Tables 31-40
Volatile Table Restrictions 31-42
Global Temporary Tables 31-44
Secondary Indexes and Global Temporary Tables 31-44
Creating Global Temporary Tables 31-46
Teradata 12.0 – Major Features 31-48
Teradata 13.0 – Major Features 31-50
Teradata 13.10 – Major Features 31-52
Teradata 14.0 – Major Features 31-54
Teradata Limits (Different Releases) 31-56
Module 31: Review Questions 31-58

Teradata SQL

SQL is a standard, open language without corporate ownership. The commercial acceptance of SQL was precipitated by the formation of SQL Standards committees by the American National Standards Institute and the International Standards Organization in 1986 and 1987. Various SQL standards have been released over the years.

SQL-89 (SQL1)
SQL-92 (SQL2)
SQL-1999 (SQL3)
SQL:2003
SQL:2006
SQL:2008

The existence of standards is important for the general portability of SQL statements.

Teradata SQL has evolved from a DB2 compatible syntax under V1 to an ANSI compliant syntax under V2 to an ANSI SQL:2008 compatible version. In every case, Teradata has always had its own extensions to the language. Current certification is at entry or core level SQL:2008 with Teradata extensions and some enhanced features implemented.

Teradata, in its historical development, has produced any number of innovative SQL language elements that do not conform to the ANSI SQL standard, a standard that did not exist when those features were conceived. The existing Teradata user base had invested substantial time, effort, and capital into developing applications using that Teradata SQL dialect.

Who is ANSI?

The American National Standards Institute is an administrator and coordinator of voluntary systems of standardization for the United States private sector. About 80 years ago a group of engineering societies and government agencies formed the institute to enhance the "quality of life by promoting and facilitating voluntary consensus standards and conformity." Today the Institute represents the interests of about 1,000 companies, organizations and government agencies. ANSI does not itself develop standards; rather it facilitates development by establishing consensus among qualified groups.

The American National Standards Institute (ANSI) defines a version of SQL that all vendors of relational database management systems support to a greater or lesser degree. The complete ANSI/ISO SQL:2008 standard is defined across nine individual volumes.

Acronym: NIST – National Institute of Standards and Technology

Teradata SQL

OS	Platform	SQL	Compatibilities
TOS	DBC/1012	DBC/SQL	DB2, SQL/DS, ANSI
TOS	3600	Teradata SQL (V1R5.1)	DB2, SQL/DS, ANSI (Outer Join added)
UNIX MP-RAS	5100	Teradata SQL (V2R1)	DB2, SQL/DS, ANSI (Similar to V1R5.1)
UNIX MP-RAS Windows 2003 Linux	52xx – 55xx 52xx – 56xx 54xx – 6xxx	Teradata SQL (V2R2 to Teradata 14.0)	ANSI (major syntax change)

Two levels of ANSI SQL:2008 compliance:
- Core or Entry level
- Enhanced level

Teradata SQL
- Teradata ANSI SQL:2008 compliant at the Core level
- Certified by US Government and NIST
- Includes many of the enhanced features

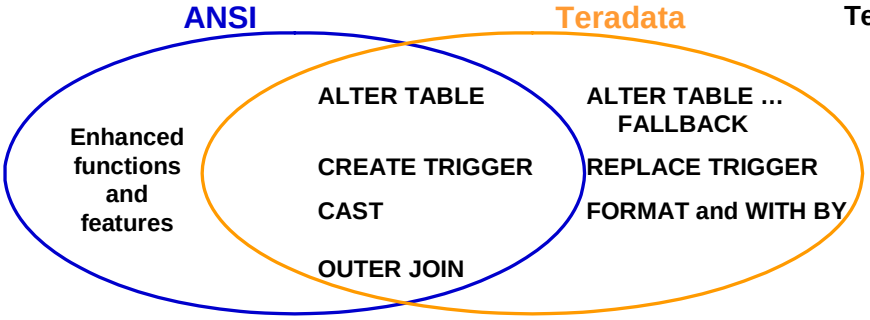
Teradata SQL and ANSI Differences

Teradata SQL meets ANSI SQL:2008 core standards and contains numerous extensions to the SQL:2008 standard. The SQL reference manuals identify those features which are extensions to SQL:2008.

Teradata sessions can operate in one of two modes: ANSI mode and Teradata (BTET) mode. The choice of mode determines the transaction protocol behavior, but also affects such things as case sensitivity defaults, collating sequences, data conversions and display functions. It is important to note that the exact same SQL statement might perform differently in each mode based on these considerations.

Regardless of mode selected, all syntax, whether ANSI compliant or not, is useable. The choice of mode does not inhibit any functionality.

Teradata SQL and ANSI Differences



Teradata SQL consists of:

- ANSI compatible ways to perform many Teradata features.
- Teradata extensions that are not ANSI standard.

Teradata allows for sessions to operate in either ...

- BTET (Teradata) mode
- ANSI mode

All syntax, both ANSI and Teradata extensions, is accepted in either mode.

The same syntax might function differently in each mode.

- Transaction protocol behavior
- CREATE TABLE SET or MULTISET default
- Case sensitivity and collating sequences
- Certain data conversion and display functions

SQL Session Modes

A session flag may be set for the transaction mode of the session. A session in Teradata mode will operate with BEGIN and END TRANSACTION protocols while a session in ANSI mode will operate with COMMIT protocol. There are other subtle differences in each mode's treatment of CREATE TABLE defaults, case sensitivity, collation sequences, data conversion and display.

A comparison summary chart follows:

Teradata Mode	**ANSI Mode**
The default is that a transaction is **implicit**. **Explicit** transactions are available using the **BT** and **ET** commands.	All transactions are **explicit** and a **COMMIT WORK** is required to successfully commit all completed work.
CREATE TABLE – defaults to SET table	CREATE TABLE – defaults to MULTISET table
Data comparison is NOT case specific.	Data comparison is case specific.
Allows truncation of display data.	Forbids truncation of display data.

SQL Session Modes

Transaction mode setting

BTET – uses standard Teradata mode (Begin Txn – End Txn mode)
ANSI – uses ANSI mode (Commit mode)

BTEQ Examples

.SET SESSION TRANSACTION BTET;
- requires neither for implicit transactions
- requires BT to start explicit transaction
- requires ET to end explicit transaction

.SET SESSION TRANSACTION ANSI;
- requires COMMIT to end transaction

Must be entered prior to LOGON. To change session mode, must LOGOFF first.

Session Mode affects:
- Transaction protocol
- CREATE TABLE defaults
- Default case sensitivities
- Data conversions

Transaction Modes – Teradata

Teradata mode is also referred to as **BTET mode** (Begin Transaction/End Transaction). It In this mode, all individual requests are treated as single implicit transactions. To aggregate requests into a single transaction requires the BEGIN and END TRANSACTION delimiters.

Transaction Modes – Teradata

.SET SESSION TRANSACTION BTET;

BTET mode characteristics:

- CREATE TABLE default – SET table
- A transaction is by definition implicit.
  - Each request is an implicit transaction.

BT / ET Statements

- BEGIN TRANSACTION (BT) and END TRANSACTION (ET) statements are used to create larger transactions out of individual requests.
- BT; – begins an explicit transaction
- ET; – commits the currently active transaction
- Locks are accumulated following a BT until an ET is issued.
- A DDL statement must be the last statement before an ET.
- A rollback occurs when any of the following occur:
  - ROLLBACK WORK - explicit rollback of active Txn
  - SQL statement failure - rollback of active Txn
  - Session abort - rollback of active Txn

Transaction Modes – ANSI

ANSI mode is also referred to as **COMMIT mode**. ANSI mode automatically aggregates requests until an explicit COMMIT command is encountered. Thus, all transactions in ANSI mode are by definition explicit.

When the session performing a macro is in ANSI mode, the actions of the macro are uncommitted until a commit or rollback occurs in subsequent statements unless the macro body ends with a COMMIT statement.

Note that all DDL statements must be immediately delimited by a COMMIT and also that macros containing DDL must contain only a single DDL statement and must also be followed by an immediate commit.

If a macro contains a data definition statement, it can include a COMMIT, but cannot contain other DML requests.

Transaction Modes – ANSI

.SET SESSION TRANSACTION ANSI;

ANSI mode characteristics:

- CREATE TABLE default – MULTISET table
- A transaction is committed only by an explicit COMMIT.
  - COMMIT WORK will commit the currently active Txn.
- Transactions are by definition explicit.
- Statement following a COMMIT automatically starts a new Txn.
- A DDL statement must be the last statement before a COMMIT.
- Locks are accumulated until a COMMIT is issued.
- A rollback occurs when any of the following occur:
  - ROLLBACK WORK - explicit rollback of active Txn
  - Session abort - rollback of active Txn
  - SQL statement failure - rollback current statement only

Duplicate Rows

A **duplicate row** is a row of a table whose column values are all identical to another row of the same table. If a designer adheres to the rule that a Primary Key must be unique, then it should preclude the possibility of having duplicate rows.

Having said that, the ANSI standard permits duplicate rows in order to satisfy the requirements of certain vendors who rely on them for certain types of auditing systems. For example, if I am loading a table from several different databases and the same record appears from three different places, I might want to know that it originated from those three places.

Even though this contradicts relational theory, the standard generously permits duplicate rows for these anomalous situations.

Teradata, adhering to the ANSI standard, permits duplicate rows by specifying that you wish to create a **MULTISET** table. In Teradata transaction mode, the default, however, is a **SET** table that does not permit duplicate rows.

When MULTISET is enabled, Teradata does not do a duplicate row check for new rows added.

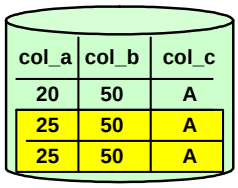
If the table is a SET table, it will only do a duplicate row check if the Primary Index is a NUPI and there are no other unique indexes on the table. If a unique index exists on the table, duplicate index check itself will suffice to ensure there are no duplicate rows.

Also, if MULTISET is enabled, it will be overridden by choosing a UPI as the Primary Index or by having a unique index (e.g., unique secondary) on another column(s) on the table. Doing this effectively disables the MULTISET.

Duplicate Rows

A duplicate row is a row of a table whose column values are all identical to another row in the same table.

Duplicate Rows



col_a	col_b	col_c
20	50	A
25	50	A
25	50	A

- **Because a PK uniquely identifies each row, ideally a relational table should not have duplicate rows!**
  - **The ANSI standard, however, permits duplicate rows for specialized situations, thus Teradata permits them as well. NO PRIMARY INDEX tables are MULTISET tables.**
- **You may select whether your table will or will not allow them.**

The Teradata default

```
CREATE SET TABLE table_A
        :
        :
```

Checks for * and disallows duplicate rows.

The ANSI default

```
CREATE MULTISET TABLE table_B
        :
        :
```

Doesn't check for and allows duplicate rows.

*** Note: If a UPI is selected on a SET table, the duplicate row check is replaced by a check for duplicate index values.**

Transaction Mode Examples

The facing page shows the various permutations of transaction modes and the expected results from success, failure and rollback.

Transaction Mode Examples

ANSI Mode	BTET Mode (explicit)	BTET Mode (implicit)
UPDATE A ... ; UPDATE B ... ; COMMIT; (Both commit)	BT; UPDATE A ... ; UPDATE B ... ; ET; (Both commit)	UPDATE A ... ; (A commits) UPDATE B ... ; (B commits)
UPDATE A ... ; UPDATE B ... ; (Fails) COMMIT; (A commits)	BT; UPDATE A ... ; UPDATE B ... ; (Fails) (Both rollback)	UPDATE A ... ; (A commits) UPDATE B ... ; (Fails) (Rollback B)
UPDATE A ... ; UPDATE B ... ; ROLLBACK ; (Both rollback)	BT; UPDATE A ... ; UPDATE B ... ; ROLLBACK ; (Both rollback)	(No explicit ROLLBACK in implicit Txn)
UPDATE A ... ; UPDATE B ... ; LOGOFF; (Both rollback)	BT; UPDATE A ... ; UPDATE B ... ; LOGOFF; (Both rollback)	UPDATE A ... ; (A commits) UPDATE B ... ; (B commits) LOGOFF;

Multi-Statement Requests

A multi-statement DML request is shown on the facing page. A semicolon at the end of a line defines the end of the request. These three UPDATE statements will be executed in parallel.

With SQL Assistant, you can use the Execute Parallel function to also group multiple DML statements into a single request.

As described on the facing page, requests have locks acquired up front in descending TableID order which minimizes the chance of deadlocks if the same request is executed by other users or if other requests using the same tables are executed.

The term **request** is used to refer to any of the following:

- A multi-statement request. Used only with DML (Data Manipulation Language) requests.
- A single statement request. Used with DDL (Data Definition Language) or DML requests.
- A macro. Used with multi-statement or single statement requests, following the above rules. A macro can contain multiple DML statements. A macro can contain a single DDL statement, but not a combination of DML and DDL statements.

The three types of requests above are also considered "implicit transactions" (or "system-generated" transactions). In fact, it is because these requests are transactions that their locks are held until the requests complete.

If locks are placed in separate requests, their order will be defined by the order of the requests. This is not recommended since this order may be different than the order that would be used in a single request. To prevent deadlocks, it is helpful to place all locks at the beginning of a transaction in a single request (especially for database and table-level locks).

Multi-Statement Requests

```
UPDATE Dept          SET Salary_Change_Date = CURRENT_DATE
; UPDATE Manager     SET Salary_Amt = Salary_Amt * 1.06
; UPDATE Employee    SET Salary_Amt = Salary_Amt * 1.04 ;
```

This is an example of 1 request – 3 statements. This one request is considered an "implicit transaction".

Notes:

- **A semi-colon at the end of a line defines the end of the request (BTEQ convention).**
- **You cannot mix DDL and DML within a single request.**

The 3 table-level write locks (in this example) will be:

- **Acquired in TID order.**
- **Held until done.**

Advantage: Minimizes deadlocks at the table level when many users execute requests on the same tables.

This applies for all types of requests:

- **Multi-statement requests (as above)**
- **Single-statement DDL or DML requests**
- **Macros**

CASE Sensitivity Issues

Teradata Mode

In **Teradata mode**, data is stored as entered unless an UPPERCASE attribute is specified for the column.

The default for character comparisons is NOT CASESPECIFIC unless either the CASESPECIFIC or UPPER/LOWER operators are specified as part of the comparison criteria.

ANSI Mode

ANSI mode always stores data as entered. The default mode for data comparison is always CASESPECIFIC unless the UPPER/LOWER operator is used as part of the comparison criteria.

Note the use of CASESPECIFIC and NOT CASESPECIFIC operators are non-ANSI compliant syntax.

Using ANSI Blind Test

Because ANSI does not permit use of CASESPECIFIC and NOT CASESPECIFIC as comparison operators or as column attributes, ANSI provides the UPPER operator as a means for doing a "case-blind" comparison of characters. Using this technique will allow a script to function compatibly in either ANSI or Teradata mode.

Teradata Mode

```
SELECT    first_name, last_name
FROM      Employee
WHERE     last_name LIKE  '%Ra%';
```

```
first_name     last_name
---------------  ---------------
Robert         Crane
James          Trader
I.B.           Trainer
Larry          Ratzlaff
Peter          Rabbit
```

ANSI Mode

```
SELECT    first_name, last_name
FROM      Employee
WHERE     UPPER(last_name)
     LIKE UPPER('%Ra%');
```

```
first_name     last_name
---------------  ---------------
Robert         Crane
James          Trader
I.B.           Trainer
Larry          Ratzlaff
Peter          Rabbit
```

CASE Sensitivity Issues

Column Attributes	Teradata Mode	ANSI Mode
Storage	As entered (default) UPPERCASE	None (As entered is default)
Comparisons	UPPER, LOWER CS (CASESPECIFIC) NOT CS (NOT CASESPECIFIC – Teradata Default)	UPPER, LOWER (CASESPECIFIC is ANSI default)

Teradata Mode – Default is NOT CS

```
SELECT      first_name, last_name
FROM        Employee
WHERE       last_name LIKE  '%Ra%';

first_name     last_name
---------- ----------
Robert     Crane
James      Trader
I.B.       Trainer
Larry      Ratzlaff
Peter      Rabbit
```

ANSI Mode – Default is CASESPECIFIC

```
SELECT      first_name, last_name
FROM        Employee
WHERE       last_name LIKE '%Ra%';

first_name     last_name
---------- ----------
Larry      Ratzlaff
Peter      Rabbit
```

ANSI Blind Test – an example of executing a "non case specific" compare in ANSI mode is provided on the facing page.

Setting the SQL Flagger

An additional BTEQ setting is available to affect the session mode. An SQLFLAG may be enabled to flag any syntax which is non-ANSI compliant. This flag does not inhibit the execution of any commands; rather it generates warning when any ANSI non-compliance is detected.

Setting the SQL Flagger

Teradata sessions have an additional selectable attribute to flag ANSI SQL non-compliance.

SQLFLAG setting

ENTRY	**– flags ANSI core incompatibilities**
NONE	**– turns off flagger**

BTEQ Example

```
.SET SESSION SQLFLAG ENTRY;      – flags non-core level ANSI syntax
```

Must be entered prior to LOGON. To change session mode, must LOGOFF first.

Affects:
- **Warnings generated for ANSI non-compliance**
- **No effect on command execution**

For example:

DATE is not ANSI standard. CURRENT_DATE is ANSI standard.

SQLFLAG Example

An example is shown of warnings generated by the SQLFlagger for a single SQL statement to select today's date. Note that following the warnings, the date is returned.

The following error codes are from the Teradata Messages manual.

5836 Token is not an entry level ANSI Identifier or Keyword.
Explanation: An identifier or keyword is not compliant with entry level ANSI rules.
Generated By: LEXER.
For Whom: User.
Remedy: If script is to be full ANSI compliant, change the indicated statement.

Note: This error is given because SELECT must be in uppercase.

5818 Synonyms for Operators or Keywords are not ANSI.
Explanation: A non-ANSI synonym has been used for a Keyword or Operator.
Generated By: SYN modules
For Whom: User.
Remedy: If script is to be full ANSI compliant, change the indicated statement.

Note: This error is given because SELECT must be fully spelled out.

5821 Built-in values DATE and TIME are not ANSI.
Explanation: These values are not supported in ANSI.
Generated By: SYN modules.
For Whom: User.
Remedy: If script is to be full ANSI compliant, change the indicated statement.

Note: This error is given because CURRENT_DATE must be used and in uppercase.

5804 A FROM clause is required in ANSI Query Specification.
Explanation: A query has been submitted that does not include a FROM clause.
Generated By: SYN modules.
For Whom: User.
Remedy: If script is to be full ANSI compliant, change the indicated statement.

SQLFLAG Example

```
.set session sqlflag entry;
.logon student130,**********;
sel date;

*** Query completed. One row found. One  column returned.
*** Total elapsed time was 1 second.

sel date;
$
*** SQL Warning 5836 Token is not an entry level ANSI Identifier or Keywor

sel date;
$
*** SQL Warning 5818 Synonyms for Operators or Keywords
    are not ANSI.

sel date;
         $
*** SQL Warning 5821 Built-in values DATE and TIME are not ANSI.

sel date;
         $
*** SQL Warning 5804 A FROM clause is required in ANSI Query
Specification.

 Current Date
    2012-02-24
```

```
.logoff
.set session sqlflag none;
.logon student130, ...
sel date;

*** Query completed. One row found.
*** One column returned.
*** Total elapsed time was 1 second.

 Current Date
    2012-02-24
```

HELP SESSION Command

There are new HELP features available with Teradata SQL.

Help at the session level shows whether Teradata (BTET) mode or COMMIT (ANSI) mode are invoked for the session.

BTEQ .SHOW Command

The **BTEQ .SHOW command** shows all settings enabled for a BTEQ invoked session of the Teradata DBC. Because BTEQ is primarily a client utility for report generation, many of the settings are reporting specifications. There are other settings that reflect BTEQ's import and export features as well.

The SHOW command displays session settings including the ANSI Flagger and the specified transaction mode.

```
.SHOW CONTROL


     :
[SET] SEPARATOR  = two blanks
[SET] SESSION CHARSET   = ASCII
[SET] SESSION RESPBUFLEN  = 8192
[SET] SESSION SQLFLAG   = NONE
[SET] SESSION TRANSACTION = BTET
[SET] SESSION TWORESPBUFS = ON
     :
```

HELP SESSION Command

HELP SESSION;

*** Help information returned. One row.
*** Total elapsed time was 1 second.

User Name	STUDENT130
Account Name	$M0+EDUC&S&D&H
Logon Date	12/02/24
Logon Time	10:48:14
Current DataBase	STUDENT130
Collation	ASCII
Character Set	ASCII
Transaction Semantics	Teradata
Current DateForm	IntegerDate
Session Time Zone	00:00
Default Character Type	LATIN
:	:
Default Date Format	YY/MM/DD
:	:
Currency Name	US Dollars
Currency	$
:	:
Default Timestamp format	YYYY-MM-DDBHH:MI:SS.S(F)Z
Current Role	TT_ACCESS_R
Logon Account	$M0+EDUC&S&D&H
Profile	Student_P
LDAP	N
:	:
Proxy User	
Temporal Qualifier	CURRENT VALIDTIME AND …
Default Number Format	FN9

BTEQ Note:

To produce this format in BTEQ, use these BTEQ settings:

```
.SET SIDETITLES
.SET FOLDLINE
```

To return to the default settings:

```
.SET DEFAULTS
```

Notes:

- The TD 14.0 HELP SESSION displays more parameters than previous releases.
- However, to see SQLFLAGGER setting, use SHOW CONTROL command.

Why a System Calendar?

Structured Query Language (SQL) permits a certain amount of mathematical manipulation of dates, however the needs of real world applications often exceed this innate capability. Implementing a system calendar is often necessary to answer time-relative business questions. Summarizing information based on a quarter of the year or on a specific day of the week can be onerous without the assistance of a system calendar.

As implemented for Teradata, the **System Calendar** is a high-performance set of nested views which, when executed, materialize date information as a dimension in a star schema. The system calendar is easily joined to other tables to produce information based on any type of time period or time boundary.

The underlying base table consists of one row for each day within the range of Jan 1, 1900 through Dec. 31, 2100. There is only one column, a date, in each row. Each level of view built on top of the base table adds intelligence to the date.

Why A System Calendar?

The Truth Is ...	**SQL has limited ability to do date arithmetic.** **There is a need for more complex, calendar-based calculations.**
I'd Like To Know …	***How does this quarter compare to same quarter last year?*** ***How many shoes do we sell on Sundays vs. Saturdays?*** ***During which week of the month do we sell the most pizzas?***
Some Good News	**Extends properties of DATE data type by joining to Calendar.** **Easily joined to other tables, i.e., dimension of a star schema.** **High performance - limited I/O.** **Has advantages over user-defined calendars.**
Standard Usage	**Statistics are created for materialized table for join planning.** **Only necessary rows are materialized for the calendar.**

Calendar View Layout

The views and the base table that make up the system calendar are contained in a database called ‘Sys_Calendar’. The contents of this database are easily seen with the help of the HELP DATABASE command.

HELP DATABASE Sys_Calendar;

*** Help information returned. 4 rows.
*** Total elapsed time was 1 second.

Table/View/Macro name	Kind	Comment
CALENDAR	V	?
CALENDARTMP	V	?
CALBASICS	V	?
CALDATES	T	?

The base table for the system calendar contains a row for each date between Jan 1, 1900 through Dec 31, 2100. Each row contains a single column that is a DATE data type. This is demonstrated using the SHOW TABLE command.

SHOW TABLE Sys_Calendar.Caldates;

*** Text of DDL statement returned.
*** Total elapsed time was 1 second.

```
------------------------------------------------------------
CREATE SET TABLE Sys_Calendar.Caldates, FALLBACK,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL
    (
     cdate DATE FORMAT 'YY/MM/DD')
UNIQUE PRIMARY INDEX ( cdate );
```

Calendar View Layout

Columns from the System Calendar:

calendar_date DATE UNIQUE (Standard Teradata date)
day_of_week BYTEINT, (1-7, where 1 = Sunday)
day_of_month BYTEINT, (1-31)
day_of_year SMALLINT, (1-366)
day_of_calendar INTEGER, (Julian days since 01/01/1900)
weekday_of_month BYTEINT, (nth occurrence of day in month)
week_of_month BYTEINT, (partial week at start of month is 0)
week_of_year BYTEINT, (0-53) (partial week at start of year is 0)
week_of_calendar INTEGER, (0-n) (partial week at start is 0)
month_of_quarter BYTEINT, (1-3)
month_of_year BYTEINT, (1-12)
month_of_calendar INTEGER, (1-n) (Starting Jan, 1900)
quarter_of_year BYTEINT, (1-4)
quarter_of_calendar INTEGER, (Starting Q1, 1900)
year_of_calendar SMALLINT, (Starting 1900)

System Calendar is a 4-level nested view of dates.

Underlying table is Sys_calendar.Caldates:

- Has one column called 'cdate' - DATE data type.
- Has one row for each date of calendar.
- Unique Primary Index is cdate.
- Each level of view adds intelligence to date.

Note:
System calendar includes Jan 1, 1900 through Dec. 31, 2100.

One Row in the Calendar

Four Levels of Calendar Views

Calendar — View which views Calendartmp

Calendartmp — View which adds:

- day_of_week
- weekday_of_month
- week_of_month
- week_of_year
- week_of_calendar
- month_of_quarter
- month_of_calendar
- quarter_of_year
- quarter_of_calendar

Calbasics — View which adds:

- calendar_date,
- day_of_calendar,
- day_of_month,
- day_of_year,
- month_of_year,
- year_of_calendar)

Caldates — Underlying table which contains a date column:

One Row in the Calendar

```
SELECT * FROM Sys_Calendar.Calendar WHERE calendar_date = '2012-02-17';
```

calendar_date	2012-02-17
day_of_week	6
day_of_month	17
day_of_year	48
day_of_calendar	40955
weekday_of_month	3
week_of_month	2
week_of_year	7
week_of_calendar	5850
month_of_quarter	2
month_of_year	2
month_of_calendar	1346
quarter_of_year	1
quarter_of_calendar	449
year_of_calendar	2012

February 2012

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29			

Note: SELECT CURRENT_DATE is the ANSI standard equivalent of SELECT DATE.

Using the Calendar

The daily sales table is used in the following example:

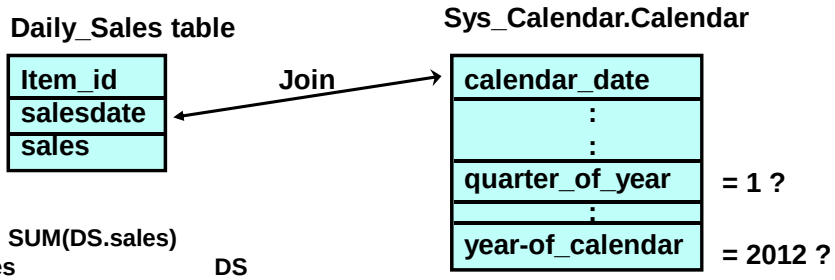
```
CREATE SET TABLE Daily_Sales,
     NO FALLBACK,
     NO BEFORE JOURNAL,
     NO AFTER JOURNAL,
     CHECKSUM = DEFAULT
          ( itemid          INTEGER,
            salesdate       DATE FORMAT 'YYYY/MM/DD',
            sales           DECIMAL(9,2) )
PRIMARY INDEX ( itemid );
```

The following is a non-ANSI standard way of performing the join on the facing page.

```
SELECT     DS.itemid
           ,SUM(DS.sales)
FROM       daily_sales                DS,
           Sys_Calendar.Calendar      SC
WHERE      DS.salesdate = SC.calendar_date
AND        SC.quarter_of_year = 1
AND        SC.year_of_calendar = 2012
AND        DS.itemid = 10
GROUP BY 1
;
```

Using the Calendar

Show total sales of item 10 reported in Q1 of 2012.



SQL:

```
SELECT      DS.itemid, SUM(DS.sales)
FROM        Daily_Sales                 DS
INNER JOIN  Sys_Calendar.Calendar       SC
ON          DS.salesdate = SC.calendar_date
AND         SC.quarter_of_year = 1
AND         SC.year_of_calendar = 2012
AND         DS.itemid = 10
GROUP BY    1;
```

Result:

itemid	Sum(sales)
10	4050.00

Salesdate is joined to the system calendar.

Calendar determines if this date meets this criteria:

- **Is it a Quarter 1 date?**
- **Is it a 2012 date?**

If yes on both, add this sales amount to result.

Temporary Table Choices

Generically speaking, there are three types of temporary tables now available with Teradata, any one of which will have advantages over traditional temporary table creation.

Derived tables were incorporated into Teradata under V2R2. Derived tables are always local to a specific query, as they are built with code within the query. The rows of the derived table are stored in spool and discarded when the query finishes. The data dictionary has no knowledge of derived tables.

Volatile Temporary tables (or **Volatile Tables**) are local to a session rather than a specific query, which means that the table may be used repeatedly within a session. Once the session ends, the volatile table is automatically discarded if it has not already been manually discarded. The data dictionary has no knowledge of volatile tables. Space for a volatile table comes from the user's Spool space.

Global Temporary tables (or **Temporary Tables**) are local to a session just as are volatile tables. Unlike volatile tables, global temporary tables are known by the data dictionary where a permanent definition is kept. Global temporary tables are materialized within a session, and then discarded when the session ends. Space for a global temporary table comes from the user's Temporary space.

In this module, we will be looking at the three types of temporary tables, how their implementations differ and when to use each.

Temporary Table Choices

Derived Tables

- Local to the query (table and columns are named within query)
- Incorporated into SQL query syntax (populated in query via SELECT in FROM)
- Materialized in SPOOL – Spool rows are discarded when query finishes
- No data dictionary involvement
- Commonly used with aggregation

Volatile Tables

- Local to a session – uses SPOOL space
- Uses CREATE VOLATILE TABLE syntax
- Discarded automatically at session end
- No data dictionary involvement

(Global) Temporary Tables

- Local to a session – uses TEMPORARY space
- Uses CREATE GLOBAL TEMPORARY TABLE syntax
- Materialized instance of table discarded at session end
- Creates and keeps table definition in data dictionary

Derived Tables Revisited

Derived tables were introduced into Teradata under V2R2. The creation of the derived table is local to the query. A query may have multiple derived tables. These tables may be joined or manipulated much as any other table would be.

The OLAP functions of SQL do not support standard aggregation functions due to their conflicting uses of the GROUP BY clause. This fact makes the OLAP functions excellent candidates for the use of derived tables, in particular when the requirement is to perform a statistical function on an aggregation.

We see in the facing page example that to find the top three selling items across all stores, we must first aggregate the sales by product-id using a derived table. Once we have this aggregation done in spool, we may apply the RANK function to answer the question.

Derived tables are useful, but only exist for the duration of the query. They are not a practical solution if the result is to be used in many follow-on queries. In this case, other types of temporary tables will be more appropriate.

Derived Tables Revisited

Get top three selling items across all stores:

```
SELECT      Prodid, Sumsales,  RANK(sumsales) AS "Rank"
FROM        (SELECT prodid, sum(sales) FROM Salestbl GROUP BY 1)
                AS tmp (prodid, sumsales)
QUALIFY RANK (sumsales) <= 3;
```

Result:

Prodid	Sumsales	Rank
A	170000.00	1
C	115000.00	2
D	110000.00	3

- Derived table name is “tmp”.
  - The table is required for this query but no others.
  - The query will be run only one time with this data.
- Derived column names are “prodid” and “sumsales”.
- Table is created in spool using the inner SELECT.
- SELECT statement is always in parenthesis following “FROM”.

Volatile Tables

Volatile tables have much in common with derived tables. They are materialized in spool and are unknown to the data dictionary. Unlike derived tables, volatile tables may be used repeatedly throughout a session. They may be dropped at any time manually or automatically at the session end.

Volatile tables require their own CREATE syntax. The table definition is kept in cache and not permanently written to disk. Volatile tables do not survive a system restart.

The **LOG** option indicates the desire for standard transaction logging of "before images" into the transient journal.

The **ON COMMIT DELETE ROWS** option specifies that at the end of a transaction, the table rows should be deleted. While this might seem a bit unusual, it is the default required by the ANSI standard. It may be appropriate in situations where a table is materialized only to produce an aggregation and the table rows are not needed beyond that purpose. (Typically this would occur in a multi-statement transaction.)

The **ON COMMIT PRESERVE ROWS** option provides the more normal situation where the table rows are kept following the end of the transaction.

Secondary Indexes and Volatile Tables

When initially creating a Volatile table, you can define secondary indexes as part of the CREATE VOLATILE TABLE statement.

You cannot add secondary indexes (via CREATE INDEX) to a Volatile table after it has been created. Secondary indexes have to be specified with the initial CREATE VOLATILE TABLE statement.

You cannot create a join index or a hash index on a VOLATILE table.

Volatile Tables

Similar to derived tables:

- Materialized in spool
- No Data Dictionary access or transaction locks
- Table definition kept in cache
- Designed for optimal performance

Different from derived tables:

- Is local to the session, not the query
- Can be used with multiple queries in the session
- Dropped manually anytime or automatically at session end
- Requires CREATE VOLATILE TABLE statement

Example:

```
CREATE VOLATILE TABLE vt_deptsal
, LOG
      (deptno        SMALLINT
      ,avgsal        DEC(9,2)
      ,maxsal        DEC(9,2)
      ,minsal        DEC(9,2)
      ,sumsal        DEC(9,2)
      ,empcnt        SMALLINT)
ON COMMIT PRESERVE ROWS;
```

CREATE Considerations:

- LOG indicates that a transaction journal is maintained.
- NO LOG allows for better performance.
- PRESERVE ROWS indicates keep table rows at TXN end.
- DELETE ROWS indicates delete all table rows at TXN end.
- Volatile tables do not survive a system restart.

Volatile Table Restrictions

Volatile tables must have names that are unique within the user's working database. Even though volatile tables are not known to the data dictionary, if names duplicating dictionary names were allowed, the system would not understand where to locate the requested named object if it could be found in two places.

Up to 1000 volatile tables are allowed on a single session. They must all have unique names. They also must be qualified by the User ID of the session, either explicitly or implicitly. A volatile table cannot belong to a database or a user; it can only belong to a user's session.

While **FALLBACK** is a selectable option, its value is limited for volatile tables. Because they cannot survive a system restart, making a table FALLBACK will not keep a table available following a restart. The only reason to make a volatile table FALLBACK would be to allow creation of the table in the event of a down AMP.

None of the following options are permitted with volatile tables:

- Permanent Journaling
- Referential Integrity
- CHECK constraints
- Column compression
- Column default values
- Column titles
- Named indexes

Volatile Table Restrictions

Restrictions:

- Up to 1000 volatile tables are allowed on a single session.
- Each must have a unique name.
- Volatile tables are always qualified by the session's userid.
- Secondary indexes are allowed on when VT is initially created
- Cannot be created with join and/or hash indexes.

Examples:

```
CREATE VOLATILE TABLE username.table1     (Explicit)
CREATE VOLATILE TABLE table1              (Implicit)
CREATE VOLATILE TABLE databasename.table1 (Error)
```

Multiple Sessions:

- Each session can use the same VT name (local to session).
- VT name cannot duplicate existing object name for this user.
  - Perm or Temp names, View names, etc.

FALLBACK:

Electable but not often useful for VTs. VTs don't survive a system reset.

Options not permitted:

- Permanent Journaling
- CHECK constraints
- Column default values
- Referential Integrity
- Column compression
- Column titles
- Named Indexes

Global Temporary Tables

Global Temporary Tables (also called **Temporary Tables**), unlike volatile and derived tables, have definitions stored in the Data Dictionary. The table itself is materialized by the first SQL DML statement that accesses the table, typically an INSERT SELECT or an INSERT.

Like volatile tables, global temporary tables are local to a session. The materialized instance of the table is not shareable with other sessions. The table instance may be dropped explicitly or it will be automatically dropped at the end of the session. The definition remains in the dictionary for future materialized instances of the table. The base definition may be dropped with an explicit DROP command.

The only privilege required by the user is the DML privilege necessary to materialize the table. Once materialized, no privileges are checked.

A special type of space called **"temporary space"** is used for global temporary tables. Like perm space, temporary space is sustained during a system restart. Global temporary tables are thus able to survive a system restart.

Up to 2000 materialized instances of global temporary tables may exist for a given session.

Two key reasons for the users of global temporary tables are

1. To simplify application code
2. Reduce the a large number of joins for specific tables

Note:

It is common to refer to volatile temporary tables as **"volatile tables"** and refer to global temporary tables simply as **"temporary tables"**.

Secondary Indexes and Global Temporary Tables

When initially creating a Global Temporary table, you can define secondary indexes as part of the CREATE GLOBAL TEMPORARY TABLE statement.

You can also add secondary indexes (via CREATE INDEX) to a Global Temporary table definition ONLY if it is NOT materialized.

- If a Global Temporary table is materialized by any user in the system, you cannot add additional secondary indexes to the Global Temporary table.

You cannot create a join index or a hash index on a Global Temporary table.

Global Temporary Tables

Global Temporary Tables	**Are created using CREATE GLOBAL TEMPORARY command.** **Require a base definition which is stored in the DD.** **Are materialized by first SQL DML statement to access table.**
Similarities to Volatile Tables:	**Each instance of global temp table is local to a session.** **Materialized tables are dropped automatically at session end.** **Have LOG and ON COMMIT PRESERVE/DELETE options.** **Materialized table contents aren't sharable with other sessions.** **Does not support join and/or hash indexes.**
Differences from Volatile Tables	**Base definition is permanent and kept in DD.** **Requires DML privileges necessary to materialize the table.** **Space is charged against an allocation of "temporary space" - CREATE USER TEMPORARY parameter.** **User can materialize up to 2000 global tables per session.** **Secondary indexes (CREATE INDEX) can be added to a Global Temporary Table as long it has not been materialized.** **Tables can survive a system restart.**

Creating Global Temporary Tables

Temporary tables are created using the CREATE GLOBAL TEMPORARY TABLE command. This stores the base definition of the table in the data dictionary. Like volatile tables, the defaults are to LOG transactions and ON COMMIT DELETE ROWS.

Temporary tables may be altered by the ALTER command to change any attributes of the table, similar to perm tables.

Once the table is accessed by a DML command, such as the INSERT SELECT seen on the facing page, the table is considered materialized and a row is entered into a dictionary view called DBC.Temptables.

Deleting all rows from a temporary table does not de-materialize the table. The instance of the table must be dropped or the session must be ended for the materialized table to be discarded.

Creating Global Temporary Tables

```
CREATE GLOBAL TEMPORARY TABLE gt_deptsal
      (deptno        SMALLINT
      ,avgsal        DEC(9,2)
      ,maxsal        DEC(9,2)
      ,minsal        DEC(9,2)
      ,sumsal        DEC(9,2)
      ,empcnt        SMALLINT);
```

Base table definition stored in DD/D
Default is ON COMMIT DELETE ROWS

```
ALTER TABLE gt_deptsal,
            ON COMMIT PRESERVE ROWS;
```

ALTER TABLE can be done to change defaults.

```
INSERT INTO gt_deptsal
SELECT      dept ,AVG(sal) ,MAX(sal) ,MIN(sal)
            ,SUM(sal) ,COUNT(emp)
FROM        emp
GROUP BY    1;
```

Table is now materialized.
Row is inserted in DBC.Temptables.

```
DELETE FROM gt_deptsal;
```

Table remains materialized until the rows are deleted in it.

Teradata 12.0 – Major Features

A list of key new Teradata 12.0 features is shown on the facing page.

Teradata 12.0 – Major Features

Performance
- Multi-Level Partitioned Primary Index
- OCES-3 (Optimizer Cost Estimation Sub-system)
- Enhanced Query Rewrite Capability
- Extrapolate Statistics Outside of Range
- Increase Statistics Intervals
- Collect Statistics for Multi-Column NULL Values
- Collect AMP Level Statistics Values
- Parameterized Statement Caching Improvements
- Windowed Aggregate Functions
- Hash Bucket Expansion

Active Enabled
- Online Archive
- Bulk SQL Error Logging Tables
- Full ANSI Merge-Into Capability
- Replication Scalability
- Restartable Scandisk
- Checktable Utility Performance
- Table Functions Without Join-Back

Cost, Quality, and Supportability
- Compression on Soft/Batch RI Columns
- Dispatcher Fault Isolation

Enterprise Fit
- Java Stored Procedures
- Restore/Copy Dictionary Phase
- Restore/Copy to Different Configuration Data
- Phase Performance
- Cursor Positioning for MSR
- UNICODE Support for password control
- Custom Password Dictionary Support
- New Password Encryption Algorithm

Ease of Use
- Queue Tables
- IN-List Processing
- LOB Enhancements to Stored Procedures
- External Stored Procedures
- UDF Table Function
- TASM: Query Banding, Traffic Cop, Global/Multiple Exceptions, Utility Management, and Open API's
- Enhanced Collection: DBQL & ResUsage
- Enhanced Explain Plan Details
- Stored Procedure Result Sets
- SQL Invocation via External Stored Proc.
- Index Wizard Support for PPI
- Dynamic Result Row Specification on Table Functions
- Normalized AMPUsage View for Coexistence

Teradata 13.0 – Major Features

A list of key new Teradata 13.0 features is shown on the facing page.

Teradata 13.0 – Major Features

Performance
- JI/AJI Enhancements (cost-based rewrites)
- Combined OLD/NEW Table Trigger
- Increased Max Number of AWTs per AMP
- Increase max value of dbscontrolCylinders Saved for PERM
- Large Object (LOB) Loader
- COUNT(*) Optimization
- Collect Statistics Optimization
- DPE for Inclusion/Exclusion Joins
- Increased Join/Subquery limits
- RESET WHEN Ordered Analytic
- Expanded Table Header (1 MB)
- Increase concrete step segment to 2MB

Active Enabled
- DDL Replication
- Identity Column and Trigger Replication
- LOB and UDTs Replication
- ResUsage Data by Workload Definition
- Simplified Query Capture Database
- DBQL XML Query Plan Logging

Cost, Quality, and Supportability
- Teradata Virtual Storage
- Preserve Column Compression in a Join Index

Enterprise Fit
- Java Stored Procedures
- Geospatial data types
- Teradata Trusted Sessions
- Add SSL/TLS support for LDAP
- ADAM (Active Directory Application Mode) Support
- Novell's directory for LDAP Mechanism
- Column Level GRANT/REVOKE
- Open LDAP Support
- Period Data Type
- Extended Usage of Scalar Subquery
- UDTs as parameters in Aggregate UDFs
- Recursive Query with UDF/SP
- Ordered Input Rows to UDFs
- Java User-Defined Functions
- Tunable UDF Memory Limit
- BAR for Join Indexes
- Java Stored Procedure Answer Sets
- Support Previous TTU on new DBMS
- Stored Procedure Access Rights

Ease of Use
- Queue Tables
- Automated TSET Data Acquisition
- DBQL Rules Enhancements
- Transfer Statistics

Teradata 13.10 – Major Features

A list of key new Teradata 13.10 features is shown on the facing page.

Teradata 13.10 – Major Features

Quality/Supporability
- AMP fault isolation
- Parser diagnostic information capture
- Dictionary cache re-initialization
- EVL fault isolation and unprotected UDFs

Performance
- FastExport without spooling
- Character-based PPI
- Timestamp Partitioning
- Merge data blocks during full table modify operations
- Statement independence
- TVS: Initial Data Temperature

Active Enable
- Read from Fallback
- TASM: Workload Designer
- TASM: Utilities Management
- TASM: Additional Workload Definitions
- TASM: Common Classifications

Ease of Use
- Teradata 13.10 Teradata Express Edition
- Domain Specific System Functions
- Moving current date in PPI
- Transparent Cylinder Packing
- Archive DBQL rule table

Enterprise Fit
- Algorithmic Compression for Character Data
- Multi-Value Compression for VARCHAR columns
- Block level compression
- Variable fetch size (JDBC)
- User Defined SQL Operators
  - Temporal Processing
  - Temporal table support
  - Period data type enhancements
  - Replication support
  - Time series Expansion support
- Enhanced trusted session security
- External Directory support enhancements
- Geospatial enhancements
- Statement Info Parcel Enhancements (JDBC)
- Support for IPv6
- Support unaligned row format for 64-bit platforms
- Enhanced hashing algorithm
- Large cylinder support
- User Defined Ordered Analytics

Teradata 14.0 – Major Features

A list of key new Teradata 14.0 features is shown on the facing page.

Teradata 14.0 – Major Features

Quality/Supporability
- Teradata uses non-root Linux user account
- Redesign RSS to Support an Application Data Pull Mode
- Selective Dump Capability

Performance
- Teradata Columnar
- Collect statistics enhancements
- Hash Join enhancements
- Increased Partition Limits
- Initial Data Temperature, Phase 2

Active Enable
- Active Fallback, Phase 2
- SLES 11: Improved workload management and priorities
- Online Reconfiguration Phase
- Teradata Unity

Ease of Use
- Expansion by Business Days
- New Embedded Services Functions
- New Fields in Workload Management APIs
- NUMBER data type
- SQL ARRAY/VARRAY data type
- TD_ANYTYPE Parameter data type

Enterprise Fit
- Block level compression Enhancements
- Encryption Enhancements
- Increased number of Vprocs
- Indexes on UDT columns
- Kerberos Authentication from Linux clients
- LDAP Authentication for Multiple Directory Services
- Multiple WITH/WITH RECURSIVE clauses
- Persistent standby nodes
- Provide Table with Teradata Reserved Query Band Names
- Restricted Creation of New Kanji1 Data
- Row-Level Security
- Separate LogonSource String
- Set AuditTrailId to Authcid for LDAP Authentication
- Temperature-Based Block-Level Compression
- Unicode 6.0 Update
- Unicode Support in LOWER Function
- User-Level Export Width
- Workload Management API Support for Extended Object Name (EON)

Teradata Limits (Different Releases)

The chart on the facing page highlights the system limits associated with different Teradata releases.

Teradata Limits (Different Releases)

	12.0	13.0	13.10	14.0
Maximum Vdisk Size	1.26 TB	1.26 TB	7.2 TB	7.2 TB
Maximum Block Size (sectors)	255	255	255	255
Maximum Table Header Size	128 KB	1 MB	1 MB	1 MB
Number of defined databases/users	4.2 Billion	4.2 Billion	4.2 Billion	4.2 Billion
Concrete Step Size	1 MB	2 MB	2 MB	2 MB
Maximum number of partitions (PPI)	65,535	65,535	65,535	9.2×10^{18}
Maximum number of partition levels	15	15	15	62
Number of spool tables per query	2048	2048	2048	2048
Maximum # of tables in a join	64	128	128	128
Maximum # of columns in a table	2048	2048	2048	2048
Maximum # of columns in an index	64	64	64	64
Maximum SQL Request Limit Size	1 MB	1 MB	1 MB	1 MB
Maximum SQL Response Buffer Size	1 MB	1 MB	1 MB	1 MB
Maximum # of Hash Bucket Entries	1 MB	1 MB	1 MB	1 MB
Maximum # of Vprocs (AMPs and PEs)	16,384	16,384	16,384	32,720

Module 31: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 31: Review Questions

1. Which BTEQ setting controls Teradata vs. ANSI mode? _____
2. Which commands will not work in ANSI mode? _____
3. True or False. The SQL Flagger is just a warning device and doesn't affect command execution.
4. True or False. Failure of an individual request in ANSI (or COMMIT) mode causes the entire transaction to be rolled back.
5. True or False. Logging off during an explicit transaction without either a COMMIT or ET will always result in a ROLLBACK.
6. True or False. HELP SESSION will show the session mode and the status of the SQL Flagger.
7. Where does a Volatile Temporary table get its space from? _____
8. Where does a Global Temporary table get its space from? _____

Notes

Module 32

Introduction to Application Utilities

After completing this module, you should be able to:

- **Identify the Application Utilities.**
- **Describe how the Application Utilities interface with the Teradata Database.**
- **State the advantage of using a utility over other access methods.**
- **Match Teradata Parallel Transporter operators with the corresponding Teradata utility.**

Notes

Table of Contents

Application Utilities 32-4
Application Utilities Environments 32-6
  Application Utilities on a Local Area Network 32-6
  Application Utilities on a Mainframe Host 32-6
Application Development 32-8
  Utility advantages 32-8
Transferring Large Amounts of Data 32-10
INSERT/SELECT: The Fast Path 32-12
Multi-Statement Insert/Select Example 32-14
DELETE (ALL): The Fast Path 32-16
AXSMOD, INMOD, or OUTMOD Routines 32-18
Teradata Parallel Transporter 32-20
Teradata Parallel Transporter Operators 32-22
Referential Integrity and Load Utility Issues 32-24
Honoring Table Free Space Percent 32-26
Application Utility Checklist 32-28
Application Utility Summary 32-30
Module 32: Review Questions 32-32

Application Utilities

The Teradata database provides several Application Utilities for processing large numbers of INSERTs, UPDATEs, and DELETEs in a batch environment.

Each utility exploits the capabilities provided by the Teradata parallel architecture for a specific data maintenance or batch-processing activity.

Teradata application utilities are supported on several hardware platforms including a wide range of channel-connected mainframes.

Regardless of the host platform, however, all access between the host and the Teradata database relies on the Call Level Interface (CLI), a series of callable subroutines that reside in the host's address space.

CLI is responsible for creating and managing the parcels that travel back and forth between Teradata and the host. It permits the host to send multiple tasks (sessions) to Teradata at the same time.

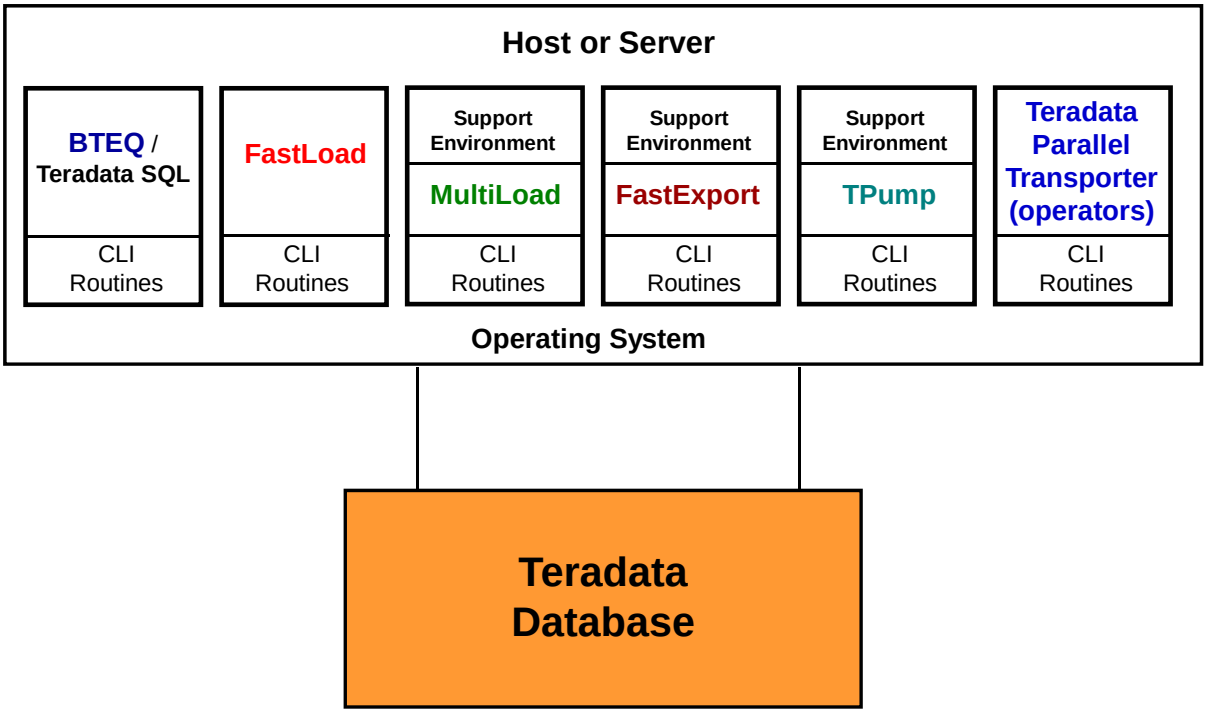
CLI is the vehicle that makes parallel access possible.

BulkLoad

The BulkLoad utility (not shown on facing page) is an older utility that executed on a channel–attached mainframe. BulkLoad was one of the original Teradata loader utilities and has been replaced by the more efficient utility, TPump. BulkLoad supported SELECT, INSERT, UPDATE, DELETE and submits SQL transactions at SQL speed.

In addition to the main processor being used on the host platform, most activities use special protocols on the database engine itself.

Application Utilities



Application Utilities Environments

Even though a single UNIX node employs an innovative architecture that uses the combined power of multiple tightly-coupled processors as a powerful UNIX mainframe, it is preferred that Application Utilities execute on a separate server, a different SMP node, or on a mainframe host. This server may be another node within the configuration or a separate server that is LAN connected. If the node is part of the configuration, then is communicates directly over the BYNET, thus avoiding performance constraints associated with a channel connection.

Application Utilities on a Local Area Network

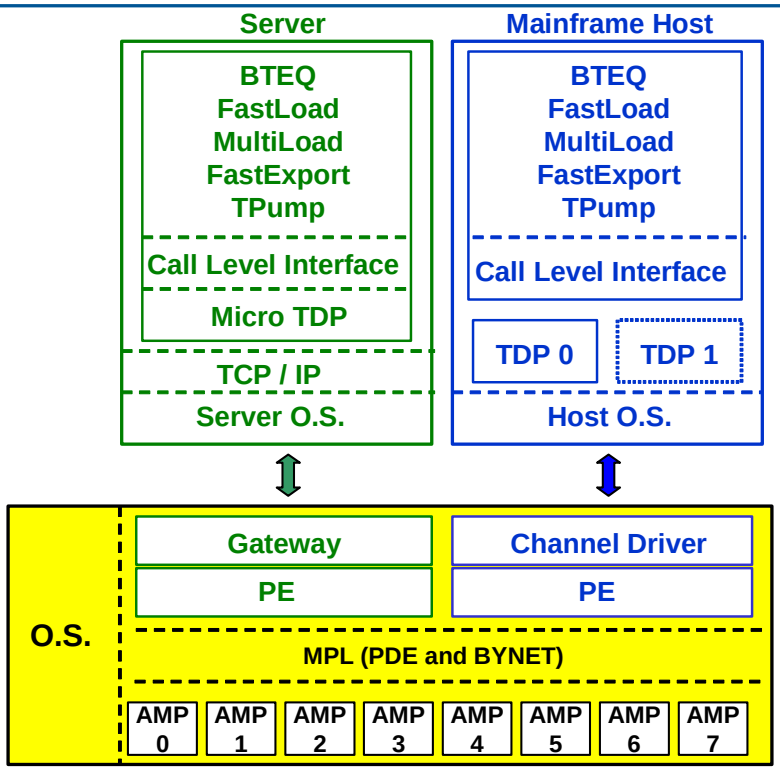
Each PC or workstation accessing the Teradata database over a LAN has a single-threaded version of TDP (Teradata Director Program), which is known as Micro TDP (MTDP).

Although the capacities of PCs and workstations are increasing rapidly, performance can be constrained by memory and disk limitations. For this reason PCs are less likely to handle the large amounts of data normally associated with the Application Utilities.

Application Utilities on a Mainframe Host

Application Utilities are frequently executed on mainframes hosts using the mainframe channel connections to load/unload data to/from Teradata. The TDP provides session control for the mainframe host.

Application Utility Environments



Application Development

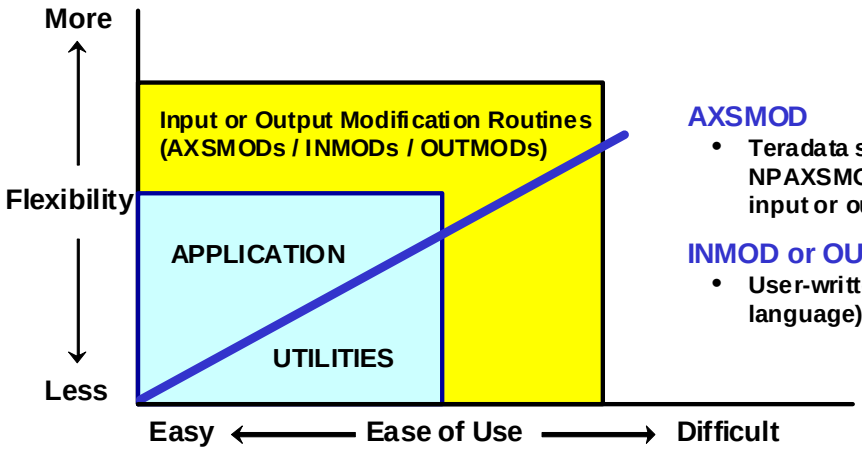
All access to the Teradata database is accomplished using the Call Level Interface. All the Teradata utilities are programs written in a 2nd or 3rd generation language. Depending upon the programmer's skill, programs written using the CLI are extremely flexible and can accomplish any function supported by the Teradata database. These types of programs are complex and difficult both to write and maintain.

Less complex, but still requiring a high level of programming ability, SQL statements can be imbedded in programs (example C) using the Teradata Preprocessor. Application design, coding and implementation are lengthy processes.

Utility advantages

The Teradata utilities represent an alternative to custom development. They are easy to use, simple to maintain, and quick to implement. Moreover, they have all been tested, documented, and are vendor-supported.

Application Development



AXSMOD

- Teradata supplied program (e.g., NPAXSMOD) that provides a specific input or output modification.

INMOD or OUTMOD

- User-written programs (e.g., C language) using Call-level interface.

Selection of the right vehicle can be crucial to the success of the application:

- How difficult is it to implement?
- How difficult is it to maintain?

Use the application utilities wherever possible:

- Offer the least complexity.
- Take full advantage of parallel processing.

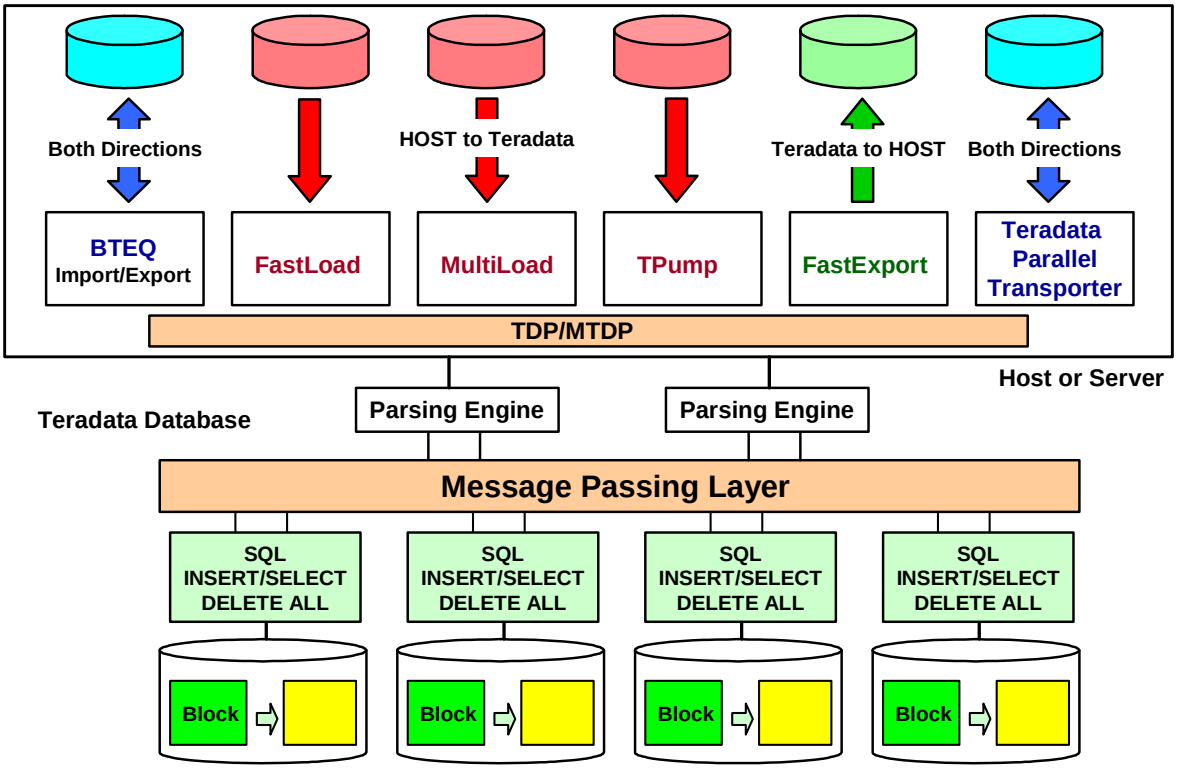
Transferring Large Amounts of Data

The Teradata application utilities reside in the host computer, whether it be the Application Processor, a mainframe, or a workstation.

- **BTEQ** supports all 4 DMLs: SELECT, INSERT, UPDATE and DELETE. BTEQ also supports IMPORT/EXPORT protocols.
- **FastLoad**, **MultiLoad**, and **TPump** transfer data from the host to Teradata.
- **FastExport** performs high volume SELECTs to export data from Teradata to the host.

Apart from the application utilities themselves, there is a special optimization of the **SQL INSERT/SELECT** and **DELETE** that deserve some attention.

Transferring Large Amounts of Data



INSERT/SELECT: The Fast Path

The Teradata INSERT/SELECT is optimized to populate one table in the Teradata database from another at high speed provided two conditions are true:

1. The tables must have the same Primary Index (PI) **and**
2. The target table must be empty.

In almost all computer systems, the disk is the slowest component and therefore defines a performance bottleneck that limits throughput. While you cannot reasonably avoid writing I/Os to disk, you can certainly try to keep them to a minimum by writing rows to disk *a block at a time*.

Teradata stores rows in data blocks sorted ascending by *row hash*. Teradata never mixes rows of different tables in the same block, and rows never span blocks.

If both the source table and the target table have the same Primary Index, they will be on the same AMP and already hashed, formatted, and sorted in data blocks. During this kind of INSERT/SELECT operation each AMP can locally assemble the blocks for the new table in memory, and, providing the target table is empty, write entire blocks to disk with a single I/O.

Multiple INSERT/SELECT operation

In addition to the above, BTEQ permits you to populate a single initially empty table from multiple source tables at high speed. To do this, use a multiple statement INSERT/SELECT. Again, *all* tables must have the same Primary Index.

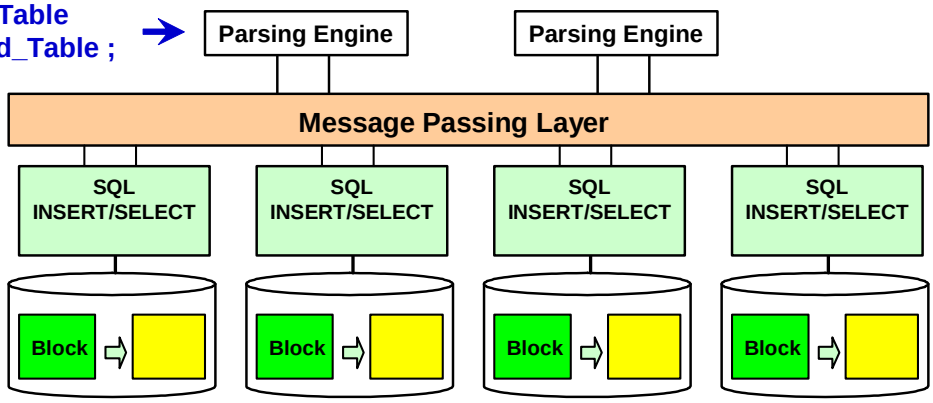
Example:

```
INSERT into T1 SELECT * FROM T2
; INSERT INTO T1 SELECT * FROM T3
; INSERT INTO T1 SELECT * FROM T4;
```

Each AMP selects the (presorted) rows from all source tables, builds the data blocks in memory, and writes the new table a block at a time.

INSERT/SELECT: The Fast Path

INSERT INTO New_Table
SELECT * FROM Old_Table ;



INSERT / SELECT achieves highest performance if:

- Target table is empty, AND
- Source and target tables have same Primary Index.

Advantages of using optimized INSERT / SELECT:

- One WRITE to the Transient Journal – instantaneous rollback for aborted Insert/Select statements.
- Data copied and written to disk a block at a time.
- No data redistribution over the BYNET.

Multi-Statement Insert/Select Example

Look at the example on the facing page. Using multiple Regional Sales History tables, a single summary table is built by combining summaries from the different regions.

A summarization may be done for each region. Summarizations then are inserted into a single table using a multi-statement Insert Select statement.

All multi-statement Insert Select statements output to the same spool table. The output is sorted and inserted into an empty table.

A multi-statement request is formed by semicolon placement in BTEQ, as shown on the facing page, or by placing statements in a single macro.

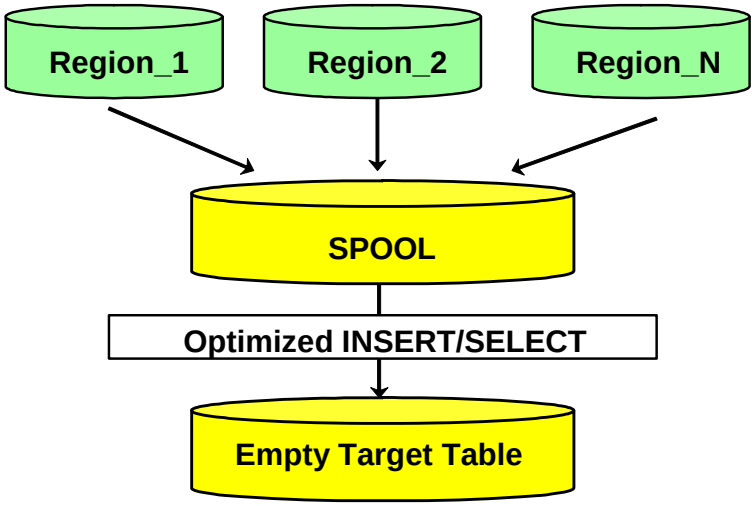
If the statements were executed separately, only the first statement is inserted into an empty table.

Multi-Statement INSERT/SELECT Example

```
INSERT INTO     Summary_Table
SELECT          store, region,
                SUM(sales),
                COUNT(sale_item)
FROM            Region_1
GROUP BY        1, 2

; INSERT INTO   Summary_Table
SELECT          store, region,
                SUM(sales),
                COUNT(sale_item)
FROM            Region_2
GROUP BY        1, 2
                . . .

; INSERT INTO   Summary_Table
SELECT          store, region,
                SUM(sales),
                COUNT(sale_item)
FROM            Region_N
GROUP BY        1, 2
;
```

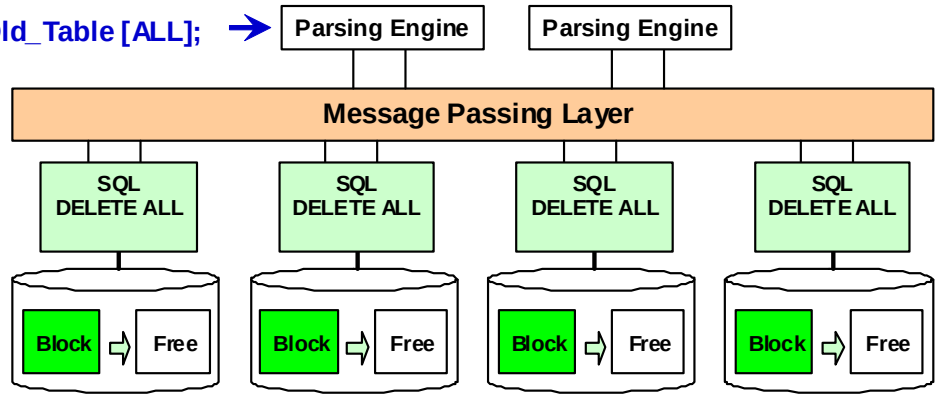


DELETE (ALL): The Fast Path

Like INSERT/SELECT, you can achieve high performance using DELETE (ALL).

DELETE (ALL): The Fast Path

DELETE [FROM] Old_Table [ALL];



DELETE achieves its highest performance when deleting all of the rows in a table.

High performance is achievable because:

- Transient Journal is not used to store before-images of deleted rows.
- DELETEs are done at the cylinder index / master index level.

ANSI Transaction Mode: In ANSI Transaction mode, to achieve the DELETE Fast Path performance, the COMMIT needs to be included as part of a multi-statement request.

```
DELETE Old_Table; COMMIT;
```

AXSMOD, INMOD, or OUTMOD Routines

Access modules are dynamically linked software components that provide input and output interfaces to different types of external data storage devices, OLE DB data sources, and message queuing software. Access modules import data from various data sources and return the data to a Teradata utility, which then stores the data in the data warehouse. Access modules are dynamically linked to one or more client utilities by the Teradata Data Connector Application Programming Interface (API).

Examples of Access Modules (AXSMOD) include

- OLE DB Access Module
- Named Pipe Access Module
- Teradata WebSphere MQ Access Module

The application utilities allow input data to be read or pre-processed by a user-written INMOD routine. The routines call the defined program module, which is responsible for delivering an input record.

An INMOD is a user exit routine used by utilities to supply or preprocess input records.

Major functions performed by an INMOD include:

- Generating records to be passed to the utility.
- Validating a data record before passing it to the utility.
- Reading data directly from one or more database systems such as Oracle.
- Converting fields in a data record before passing it to the utility.

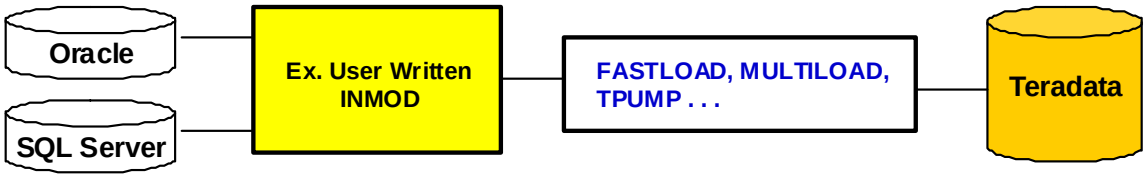
An OUTMOD is a user-written program that does post-processing of the data after the utility (e.g., FastExport) has retrieved the data. Export data to an Output Modification (OUTMOD) routine. You can write an OUTMOD routine to select, validate, and preprocess exported data.

AXSMOD, INMOD, or OUTMOD Routines

An INMOD is a user-written program that allows for pre-processing of the data before giving the data to the utility – possibly acting as a data filter to the utility.

An INMOD routine can perform various functions:

- Validate a data record, add or change data fields in the records.
- Read data directly from one or more database systems, allowing the creation of a composite input data record, and avoiding the need for an intermediate tape or disk.
- Select specific records for input to the Teradata Database.
- Perform data conversions not supported by the application utilities.



An OUTMOD is a user-written program that does post-processing of the data after the utility (e.g., FastExport) has retrieved the data.

An AXSMOD is a Teradata supplied program that can provide either a specific input or output modification.

Teradata Parallel Transporter

Teradata Parallel Transporter is the replacement for Teradata Warehouse Builder (TWB). This utility is effectively an object-oriented software system that executes multiple instances of data extraction, transformation, and load functions in a scalable, high-speed parallel processing environment.

Teradata Parallel Transporter is scalable and enables end-to-end parallelism. The previous versions of utilities (like FastLoad) allow you to load data into Teradata in parallel, but you still have a single input stream. Teradata Parallel Transporter allows you to run multiple instances of the extract, optional transformation, and load. You can have as many loads as you have sources in the same job. With multiple sources of data coming from multiple platforms, integration is important in a parallel environment.

Teradata Parallel Transporter eliminates the need for persistent storage. It stores data into data buffers so you no longer need to write data into a flat file. Since you don't need flat files, you no longer need to worry about a 2 GB file limit.

Teradata Parallel Transporter provides a single scripting language. You can do the extract, some transformation, and loads all in one SQL-like scripting language.

Once the dynamics of the language are learned, you can perform multiple tasks with a single script.

Teradata Parallel Transporter supports FastLoad INMODs, FastExport OUTMODs, and Access Modules to provide access to all the data sources you use today.

Teradata Parallel Transporter also provides a Direct API interface that can be used by Third Party partners. They can effectively write 'C' code to directly load/unload data to/from Teradata.

Teradata Parallel Transporter

Teradata Parallel Transporter (replacement for Teradata Warehouse Builder – TWB) can load data into and export data from the Teradata database.

It is effectively a parallel load and export utility. Characteristics of this utility include:

High performance

- **Parallel Export and Load operators eliminate sequential bottlenecks.**
- **Data Streams eliminate the overhead of intermediate (persistent) storage.**
- **Scalable**
- **End-to-end parallelism**

Easy to use

- **Single scripting language**
- **Access to various data sources**

Extensible

- **Direct API enables Third Party partners to write 'C' code to directly load/unload data to/from Teradata.**

Teradata Parallel Transporter Operators

Operators are the software components that provide the actual data extraction, transformation, and load functions in support of various data stores.

This utility supports different types of operators, where the operator type signifies the primary function of the operator:

- Producer Data extraction functions (e.g., Export operator):
  - Get data from the Teradata Database or from an external data store
  - Generate data internally
  - Pass data to other operators by way of the operator interface

- Consumer Data loading functions (e.g., Load operator):
  - Accept data from other operators by way of the operator interface
  - Load data into the Teradata Database or to an external data store

- Filter Data transformation functions such as:
  - Selection, validation, cleansing, and condensing

General notes about this utility and its operators.

- The FastLoad INMOD and FastExport OUTMOD operators support the current FastLoad and FastExport INMOD/OUTMOD features.

- The Data Connector operator is an adapter for the Access Module or non-Teradata files.

- The SQL Select and Insert operators submit the Teradata SELECT and INSERT commands.

- The Load, Update, Export and Stream operators are similar to the current FastLoad, MultiLoad, FastExport and TPump utilities, but built for the TWB parallel environment.

The INMOD and OUTMOD adapters, Data Connector operator, and the SQL Select/Insert operators are included when you purchase the Infrastructure. The Load, Update, Export and Stream operators are purchased separately.

To simplify these new concepts, the facing page compares the Teradata Parallel Transporter Operators with the classic utilities.

Teradata Parallel Transporter Operators

TPT Operator	Teradata Utility	Description
LOAD	FastLoad	A consumer-type operator that uses the Teradata FastLoad protocol. Supports Error limits and Checkpoint/ Restart. Both support Multi-Value Compression and PPI.
UPDATE	MultiLoad	Utilizes the Teradata MultiLoad protocol to enable job based table updates. This allows highly scalable and parallel inserts and updates to a pre-existing table.
EXPORT	FastExport	A producer operator that emulates the FastExport utility.
STREAM	TPump	Uses multiple sessions to perform DML transactions in near real-time.
DataConnector	N/A	This operator emulates the Data Connector API. Reads external data files, writes data to external data files, reads an unspecified number of data files.
ODBC	N/A	Reads data from an ODBC Provider.

Referential Integrity and Load Utility Issues

FastLoad and MultiLoad cannot be used to load data into tables that have standard or batch referential integrity constraints defined. FastLoad and MultiLoad can be used to load data into tables that have soft referential integrity constraints (REFERENCES WITH NO CHECK) defined.

First approach (probably easier in many situations):

1. Create the tables and define the Primary Keys. Primary Keys (referenced columns) must be NOT NULL and will be implemented as unique index (either primary or secondary).

2. Populate the tables.

3. Create the Foreign Key references.

   If any row in the referencing column violates the RI constraint, the Reference constraint is created and an error table (tablename_0) is automatically created.

Second approach:

1. Create the tables and define the Primary Keys. Primary Keys (referenced columns) must be NOT NULL and will be implemented as unique index (either primary or secondary).

2. Create the Foreign Key references.

3. Populate the tables.

   If you are populating the tables with INSERT/SELECT and using the second approach, when a foreign key violation is encountered, the INSERT/SELECT fails and the entire INSERT/SELECT is rolled back.

Referential Integrity and Load Utility Issues

Tables that have Standard or Batch reference constraints cannot be loaded with FastLoad, MultiLoad, or with TPT LOAD/UPDATE operators.

Standard RI – REFERENCES
Batch RI – REFERENCES WITH CHECK OPTION

Soft RI – REFERENCES WITH NO CHECK OPTION (can be loaded with FastLoad, MultiLoad, or Teradata Parallel Transporter (TPT) LOAD/UPDATE operators)

There are two different approaches to establishing RI and populating tables.

First Approach (recommended):

1. **Create the tables and define the Primary Keys.**
2. **Populate the tables.**
3. **Create the Foreign Key references.**

Second approach:

1. **Create the tables and define the Primary Keys.**
2. **Create the Foreign Key references.**
3. **Populate the tables with SQL or TPump.**

Honoring Table Free Space Percent

Because the system dynamically allocates free cylinder space for storage of inserted or updated data rows, leaving space for this during the initial load allows a table to expand with less need for cylinder splits and migrates. The system uses free space for inserted or updated rows.

You can specify the default value for free space left on a cylinder during certain operations on a table-by-table basis via the FREESPACE option in the CREATE TABLE and ALTER TABLE statements.

This allows you to select a different value for tables that are constantly modified versus tables that are only read after they are loaded.

To specify the global free space value, set the FreeSpacePercent (FSP) parameter with the DBS Control utility. If you do not expect table expansion, that is, the majority of tables are read-only, use the lowest value (0%) for free space percent.

If you set FSP to a value other than 0, tables are forced to occupy more cylinders than necessary when loading data. The extra space is not reclaimed until either you insert rows into the table, use the Ferret utility to initiate PACKDISK on a table, or until mini-cylinder packs are performed due to a lack of free cylinders.

Honoring Table Free Space Percent

Free Space Percent – how full to fill cylinders during load operations for a table.

- System default – DBSControl parameter: FreeSpacePercent
- CREATE or ALTER Table option – FreeSpace

Utility related operations that honor the Table Free Space Percent:

- SQL to add fallback – ALTER TABLE
- SQL to create a secondary index – CREATE INDEX
- Insert/Select into an empty table
- FastLoad
- MultiLoad (when loading an empty table)

Utility related operations that do not honor the Table Free Space Percent:

- SQL inserts and updates
- MultiLoad (when inserting or updating a populated table)
- TPump

Application Utility Checklist

We will complete the checklist on the opposite page as we discuss each utility. It will help you to evaluate capabilities and advantages.

As we discuss each one, it will become apparent that they have been developed over time to address evolving user needs.

Although not strictly a utility, BTEQ can be considered the grandparent of them all. BTEQ was initially developed as a means of sending the SQL to the Teradata database without having to write a complex program using the CLI for each and every query.

Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions					
DML Functions					
Multiple DML					
Multiple Tables					
Protocol Used					
Conditional APPLY					
Data Conversion					
Error Capture					
Error Limits					
User-written Routines					
Automatic Restart					
Max Load Limit					
Support Environment (SE)					

Application Utility Summary

The facing page summarizes some of the key points of this module.

Application Utility Summary

- BTEQ supports SELECT, INSERT, UPDATE, DELETE.
  - BTEQ INSERT/SELECT and DELETE (ALL) can provide a fast effective method to perform some tasks.
- FastLoad, MultiLoad, and TPump transfer data from the host to Teradata.
- FastExport transfers data from Teradata to the host.
- Utilities may offer the least complex solutions for an application, and can take advantage of parallel processing.
  - Utilities permit the use of AXSMODs, INMODs, and/or OUTMODs for pre- or post-processing data.
  - There is often more than one way to set up your application, but there may be one that is fastest or most effective.
- Teradata Parallel Transporter can load data into and export data from any accessible database object in the Teradata Database or other data store for which there exists an access operator.

Module 32: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 32: Review Questions

Answer True or False.

1. True or False. With MultiLoad, you can import and export data.
2. True or False. In Teradata mode, a BTEQ DELETE ALL function does not use the Transient Journal to store before-images of deleted rows.
3. True or False. An INSERT/SELECT of 1,000,000 rows into an empty table is only slightly faster than an INSERT/SELECT of 1,000,000 rows into a table with 1 row.

Match the Teradata Parallel Transporter operator with the corresponding Teradata utility.

1. ___ UPDATE	A. MultiLoad
2. ___ STREAM	B. FastLoad
3. ___ LOAD	C. FastExport
4. ___ EXPORT	D. TPump

Notes

Module 33

BTEQ

After completing this module, you will be able to:

- **Use .EXPORT to SELECT data from the Teradata database to another computer.**
- **State the purpose of the four types of BTEQ EXPORT.**
- **Use .IMPORT to process input from a host-resident data file.**
- **Use Indicator Variables to preserve NULLs.**
- **Describe multiple sessions, and how they make parallel access possible.**

Notes

Table of Contents

BTEQ 33-4
Using BTEQ Conditional Logic 33-6
BTEQ Error Handling 33-8
BTEQ EXPORT – Example 1 33-10
4 Types of BTEQ .EXPORT 33-12
    1. Field Mode (REPORT) 33-12
    2. Record Mode (DATA) 33-12
    3. INDICDATA 33-12
    4. Data Interchange Format (DIF) 33-12
    LIMIT 33-12
BTEQ Data Modes 33-14
BTEQ EXPORT – Example 2 33-16
  LINUX Variables in a BTEQ script 33-16
BTEQ EXPORT – Example 3 33-18
Indicator Variables 33-20
Determining the Logical Record Length with Fixed Length Columns 33-22
Determining the Logical Record Length with Variable Length Columns 33-24
Determining the Logical Record Length with .EXPORT INDICDATA 33-26
.IMPORT (for Network-Attached Systems) 33-28
  VARTEXT Notes 33-28
  AXSMOD Example 33-28
.IMPORT (for Channel-Attached Systems) 33-30
.PACK 33-32
.REPEAT 33-32
BTEQ IMPORT – Example 1 33-34
BTEQ IMPORT – Example 2 33-36
BTEQ IMPORT – Example 3 33-38
Multiple Sessions 33-40
.SET SESSIONS 33-40
Parallel Processing Using Multiple Sessions to Access Individual Rows 33-42
When Do Multiple Sessions Make Sense? 33-44
Application Utility Checklist 33-46
Module 33: Review Questions 33-48
Lab Exercise 33-1 33-50
Lab Exercise 33-2 33-54

BTEQ

BTEQ (Batch/Basic Teradata Query Language) was originally designed as a means to send SQL requests from a host file to the Teradata database and deliver the response in the format required.

BTEQ can be used for both exporting and importing data.

BTEQ is available on every platform supported by Teradata and is widely used.
Even though BTEQ is frequently used in a pseudo-interactive mode, this course concentrates on those features that render it a useful vehicle for batch or data maintenance operations.

A BTEQ session provides a quick and easy way to access a Teradata Database. In a BTEQ session, you can do the following:

- Enter Teradata SQL statements to view, add, modify, and delete data.
- Enter BTEQ commands.
- Enter operating system commands.
- Create and use Teradata stored procedures.

BTEQ

- **General purpose command-based utility for submitting SQL requests to the Teradata database.**
  - **BTEQ (Basic Teradata Query) operates in either a Batch or Interactive mode.**
  - **BTEQ is a CLI-based utility.**
- **Runs on every supported platform — laptop to mainframe.**
- **Exports data to a client system from the Teradata database:**
  - **As displayable characters suitable for reports, or**
  - **In native host format, suitable for other applications.**
- **Imports data from a host or server data file and can use that data within SQL statements (INSERT, UPDATE, or DELETE).**
- **Flexible and easy-to-use report writer.**
- **Limited ability to branch forward to a LABEL, based on a return code or an activity count.**
- **BTEQ does error reporting, not error capture.**
- **The .OS command allows the execution of operating system commands.**

Using BTEQ Conditional Logic

A feature of BTEQ that can be effectively used to improve application performance is its ability to branch forward in a script based on a test of either an error code or an activity count. While this is not a true loop function, it can be used to avoid unnecessary, time-consuming steps.

In the example on the facing page, the script is designed to delete a table. If the delete is successful, the return code is zero, and the system knows that the table already exists. It does not need to create it.

The example script also tests the number of rows that qualify for insertion into the table. Based on the result of the test, alternative subsequent actions can be performed.

Using BTEQ Conditional Logic

The Bank offers a number of special services to its Million-Dollar customers.

```
DELETE FROM Million_Dollar_Customer ALL;
.IF ERRORCODE = 0 THEN .GOTO  TableOK
CREATE TABLE Million_Dollar_Customer
      (Account_Number              INTEGER
      ,Customer_Last_Name          VARCHAR(20)
      ,Customer_First_Name         VARCHAR(15)
      ,Balance_Current             DECIMAL(9,2));
.LABEL  TableOK
INSERT INTO Million_Dollar_Customer
SELECT     A.Account_Number, C.Last_Name, C.First_Name, A.Balance_Current
FROM       Accounts A                INNER JOIN
           Account_Customer AC       INNER JOIN
           Customer C
ON         C.Customer_Number = AC.Customer_Number
ON         A.Account_Number = AC.Account_Number
WHERE      A.Balance_Current  GT 1000000;
.IF ACTIVITYCOUNT > 0 THEN .GOTO Continue
.QUIT
.LABEL Continue
```

DELETE all rows from the Million_Dollar_Customer table.

IF this results in an error (non-zero), **THEN** create the table, ELSE attempt to populate using INSERT/SELECT.

IF some rows are inserted (ACTIVITYCOUNT>0) **THEN** arrange services, ELSE terminate the job.

BTEQ Error Handling

The BTEQ error handling capability permits you to assign various severity values to specific types of errors. Use these values to make a decision to take a specific action based on the occurrence of either a specific type of error or a high-watermark value.

BTEQ Error Handling

```
.SET ERRORLEVEL   2168                SEVERITY 4,
                  (2173, 3342, 5262)  SEVERITY 8
.SET ERRORLEVEL   UNKNOWN             SEVERITY 16
SELECT            . . . . . . . . . . . .
FROM              . . . . . . . . . . . . ;
.IF ERRORLEVEL >= 16  THEN .QUIT 16 ;
```

You can assign an error level (SEVERITY) for each error code returned and make decisions based on the level you assign.

ERRORCODE → Tests last SQL statement only.
ERRORLEVEL → Set by user and retained until reset.

Capabilities:

- **Customize mapping from error code to ERRORLEVEL.**
- **.SET MAXERROR <integer> defines termination threshold.**

BTEQ EXPORT – Example 1

BTEQ and SQL commands are frequently maintained in the same file or script and may be submitted either in-stream or with a .RUN command.

BTEQ typically delivers a default response to all SQL queries that includes a helpful message along with diagnostic information about the elapsed time taken to perform the query. In its pseudo-interactive environment, this information is captured in the single default output file. This mixed output typically renders the data unsuitable for some purposes.

The .EXPORT feature of BTEQ, which names an output file, provides the ability to separate the report or output data from the accounting information.

The main difference between BTEQ .EXPORT to a LAN and a mainframe host is in the way the output file name is specified.

When identifying the destination for the output file, for BTEQ running on an IBM host, use the parameter, “DDNAME =”. For a LAN environment, use the expression “FILE =”.

Note that BTEQ statements are distinct from SQL statements; they begin with a period (.) and do not require a trailing semi-colon. (The trailing semi-colon is required for other application utilities.)

To export a file greater than 2 GB in a UNIX MP-RAS environment, use the key word AXSMOD as part of the .EXPORT command.

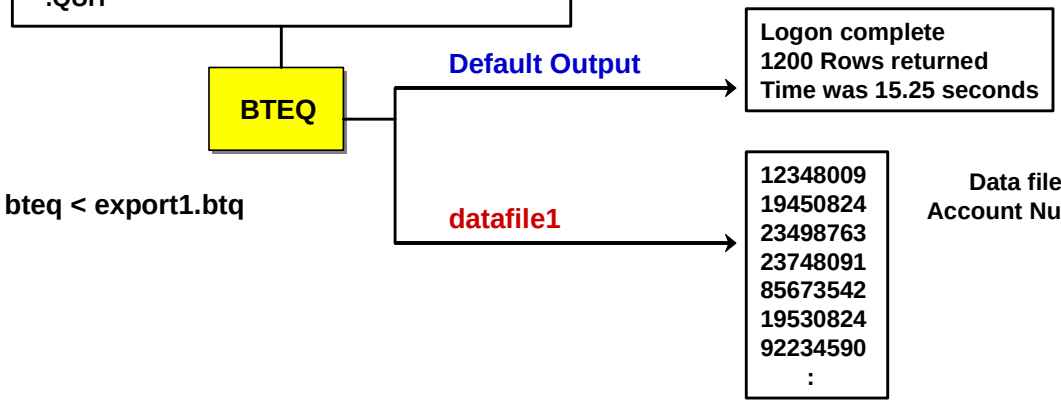
```
.LOGON tdp1/user1,passwd1
.OS rm /home/user1/largedatafile
.EXPORT DATA FILE=/home/user1/largedatafile AXSMOD
    SELECT      *
    FROM        Accounts;
.EXPORT RESET
.QUIT
```

BTEQ EXPORT – Example 1

export1.btq **BTEQ Script**

```
.LOGON tdp1/user1,passwd1
.OS rm /home/user1/datafile1
.EXPORT DATA FILE=/home/user1/datafile1
   SELECT    Account_Number
   FROM      Accounts
   WHERE     Balance_Current  LT 100;
.EXPORT RESET
.QUIT
```

Note: BTEQ will append data to an existing file. To avoid this, use the .OS "remove" command to ensure that the file doesn't exist.



bteq < export1.btq

4 Types of BTEQ .EXPORT

1. Field Mode (REPORT)

When submitting BTEQ requests to a Teradata database, you may have noted that output is always provided with column headings and underscores, with numeric values aligned to the right, characters to the left, and all output displayed in the center of the screen or report. This is Field mode, the default output of BTEQ (suitable for reports).

REPORT – output is truncated to 254 characters for mainframe and 75 for network.
REPORTWIDE – effectively sets width to 32,765 (only supported in some releases)

The .SET WIDTH can be used to set to width to a range of 20 – 65,531.

The REPORT format contains an option (not shown) of NOBOM or BOM (Byte Order Mark). This option identifies if the BOM is to be added or not when exporting Unicode data. This is associated with Unicode UTF text formatting. The default is to add the BOM.

2. Record Mode (DATA)

You might require output data in a flat-file format with binary data, no headings, etc. Request output in this format by using Data mode.

3. INDICDATA

Host computer systems rarely have the built-in capability to recognize or handle NULL data. You might need to use INDICDATA if the data contains NULL columns.

4. Data Interchange Format (DIF)

Use the DIF output option if you need data in a format suitable for PC-based applications such as Lotus 1-2-3 and Microsoft Excel. The DATALABELS option includes the column titles of the selection results as the first row in the DIF file.

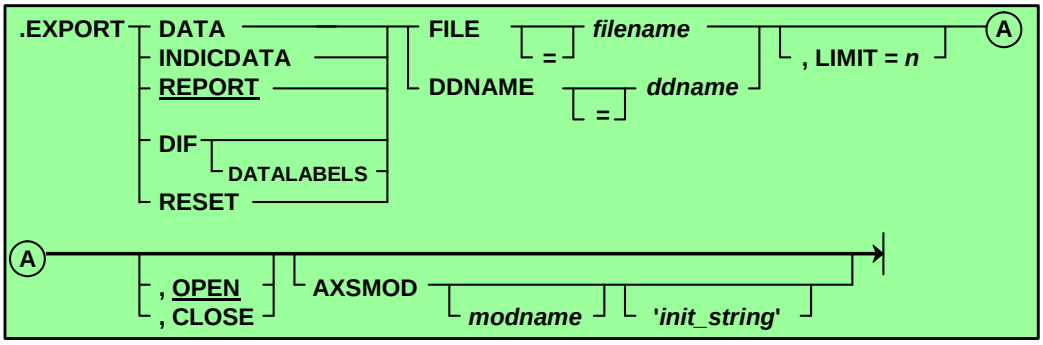
The DIF format also contains an option (not shown) of NOBOM or BOM (Byte Order Mark). This option identifies if the BOM is to be added or not when exporting Unicode data. This is associated with Unicode UTF text formatting. The default is to add the BOM.

LIMIT

The LIMIT feature of the EXPORT command is useful to application programmers who require a small subset of pre-existing data to test applications. Remember, however, that BTEQ is a host-resident utility, and that BTEQ commands are not seen by the Teradata database. For this reason, while the LIMIT function is observed by BTEQ, it is not seen by the Teradata database parser.

Note: The parameters must be on a single line. The LIMIT parameter must be on the same line as the .EXPORT.

BTEQ .EXPORT



.EXPORT DATA	Sends results to a host file in record mode.
.EXPORT INDICDATA	Sends query results that contain indicator variables to a host file. Bytes are included with bits that indicate if a column is NULL or has a value.
.EXPORT REPORT	Sends results to a host file in field mode.
	Data set contains column headings and formatted data. Exported records are truncated is they exceed width – default width is 254 for mainframe and 75 for network clients. Override with .SET WIDTH option.
.EXPORT DIF	Output converted to Data Interchange Format – used to transport data to various PC programs.
.EXPORT RESET	Reverses the effect of a previous .EXPORT and closes the output file.
LIMIT n	Sets a limit on number of rows exported.
OPEN/CLOSE	Output Data Set or File is either OPEN or CLOSEd during RETRY
AXSMOD	Access module used to export large file (> 2GB MP-RAS), tape device, etc.

BTEQ Data Modes

The terms “FIELD” and “RECORD” as mode-names for BTEQ may not be apparent until you consider the way parcels are sent by the Call Level Interface back and forth to the Teradata database.

If the application needs response data as formatted, displayable characters suitable for reports, specify FIELD Mode (the default) with an .EXPORT REPORT command. Field mode instructs the Teradata database to return formatted data parcels with a series of header parcels providing details of column headings, data types, and lengths. BTEQ then formats the responses prior to delivering them to the user. Each returning parcel contains a single FIELD of information.

If you require binary data for further activity, use .EXPORT DATA to provide a flat-file response. Each returning parcel will contain a complete RECORD.

INDICDATA mode is needed because host computer operating systems have no way of representing missing or unknown data (NULLs), but this functionality is required for Relational Database Systems.

AMPs provide output data consistent with the expectations of the particular type of host and convert output for fixed-length columns containing NULLs to zeros or spaces. Because zero is a number and a space is a valid character value, NULLs can be misunderstood by any application required to re-process the data. A flag is needed to indicate that, despite the values contained in this field, it should be treated as NULL.

Use INDICDATA mode to precede the output data record with the number of bytes representing individual bit settings for each field returned to the host. Teradata application utilities can then be instructed to observe this convention and thereby ensure complete integrity of the data.

BTEQ Data Modes

Field mode is set by : .EXPORT REPORT

Column A	Column B	Column C
1	2	3
4	5	6
7	8	9

Transfers data one column at a time with numeric data converted to character.

Record mode is set by : .EXPORT DATA

f1	f2	f3
f1	f2	f3
f1	f2	f3

Transfers data one row at a time in host format. Nulls are represented as zeros or spaces.

Indicator mode is set by: .EXPORT INDICDATA

Indic. Byte(s)	f1	f2	f3
Indic. Byte(s)	f1	f2	f3
Indic. Byte(s)	f1	f2	f3

Transfers data one row at a time in host format, sending an indicator variable for nulls. Nulls are represented as zeros or spaces.

BTEQ EXPORT – Example 2

The facing page displays a simple BTEQ .EXPORT script which limits the size of the output to 100 rows. Note the LIMIT parameter on the same line as the .EXPORT statement.

The **tee** command sends standard output (stdout) to the display device and to a specified file.

LINUX Variables in a BTEQ script

The following technique can be used to support variables in a LINUX script:

Example:

```
cat cr_bteq.sh

echo please enter the directory where all your files reside:
read in_dir

bteq << !
.run file = ${in_dir}/logon.btq;
.export data file = ${in_dir}/output_data.txt, limit=100
select * from au2.trans;
.export reset;
!
```

BTEQ EXPORT – Example 2

This example uses the LIMIT parameter to reduce the amount of exported data.

export2.btq

```
.LOGON tdp1/user1,passwd1
.EXPORT DATA FILE = datafile2, LIMIT=100, CLOSE
     SELECT      Account_Number
     FROM        Accounts
     WHERE       Balance_Current  < 500 ;
.EXPORT RESET
.QUIT
```

CLOSE If not used, BTEQ will append to an existing file.

To execute this script in a Linux environment:

```
$ bteq < export2.btq | tee export2.out

 *** Success, Stmt# 1 ActivityCount = 330
 *** Query completed. 330 rows found. 1 column returned.
 *** Total elapsed time was 1 second.
 *** Warning: RetLimit exceeded.
     Ignoring the rest of the output.
 *** Output returned to console
 $
```

datafile2 – contains 100 account numbers

export2.out – contains the output informational text sent to the console

BTEQ EXPORT – Example 3

The facing page displays a simple BTEQ .EXPORT script which exports a CSV (Comma Separated Value) file.

You have to specifically export the delimiter (or the comma in a CSV file) for each field that you are exporting. Integer data is cast as variable character data.

BTEQ EXPORT – Example 3

This script exports a data file with fields that are separated by a comma – referred to as a CSV data file (CSV – Comma Separated Value).

export3.btq

```
.LOGON tdp1/user1,passwd1
.OS rm custdata_csv
.EXPORT REPORT FILE = custdata_csv
    SELECT      CAST(Customer_Number    AS VARCHAR(11))    ||','||
                CAST(Last_Name          AS VARCHAR(30))    ||','||
                CAST(First_Name         AS VARCHAR(20))    ||','||
                CAST(Social_Security    AS VARCHAR(9))     (TITLE '')
    FROM        Customer
    SAMPLE      100;
.EXPORT RESET
.QUIT
```

Effectively exported as 1 concatenated field.

Examples of exported records are:

```
8062,Graham,Dennis,213629066
3168,Michelson,Mervin,296604596
2327,Green,Sharon,271600391
```

Note:

If data records exceed default width (ex., 75 characters, use the .SET WIDTH command to set the maximum record length (up to 65,531).

Note: Each record has an EOR terminator (hex '0A') that is automatically added.

Indicator Variables

Teradata invented the word INDICDATA for utilities that submit pure SQL as opposed to SQL-like utility statements. INDICDATA is used for BTEQ to instruct the software that these indicator bytes are present in the data file.

Note: The keyword INDICATORS is used for all the other utilities.

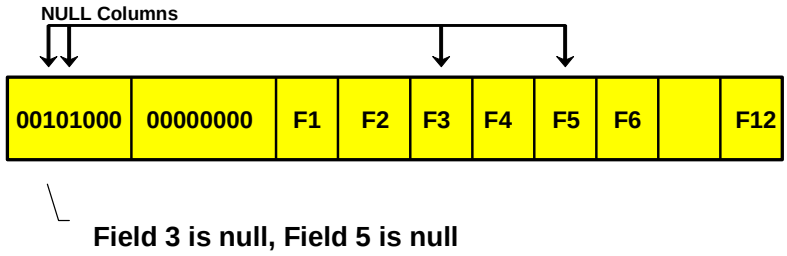
Indicator Variables

Indicator variables allow utilities to process records that contain NULL indicators.

.EXPORT INDICDATA
.[SET] INDICDATA [ON] } **BTEQ**

INDICATORS } **FastLoad, MultiLoad, FastExport, TPump**

INDICATORS causes leading n bytes of the record as NULL indicators instead of data.



Field 3 is null, Field 5 is null

Determining the Logical Record Length with Fixed Length Columns

Some host systems, such as IBM mainframes, require the correct LRECL (Logical Record Length) parameter in JCL, and will abend if the value is not correctly stated. Other types of hosts are less sensitive to this requirement.

If the input data contains only fixed-length columns, the Logical Record Length is relatively easy to calculate. Each record being treated has the same length. Fixed-length columns must accommodate the largest possible value, however, and frequently involve poor utilization of disk space.

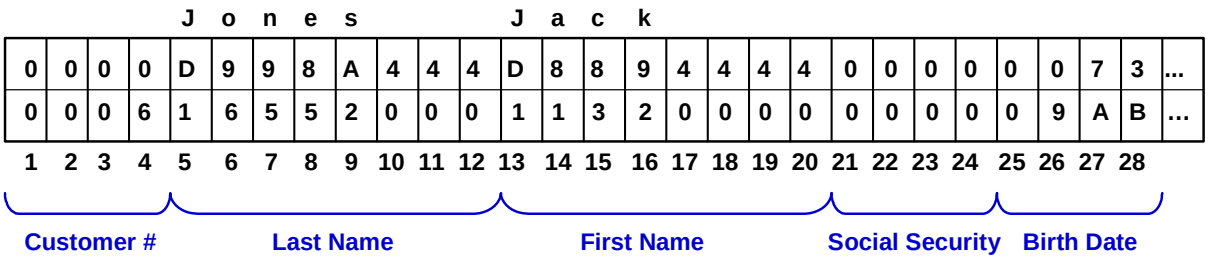
The example on the facing page is in EBCDIC, but even so the wastage of space involved with storage of non-functional blanks (HEX 40) is apparent.

Note: For convenience, HEX representations are provided in EBCDIC only.

Determining the Logical Record Length with Fixed Length Columns

```
                                                          Length
CREATE TABLE Customer,  FALLBACK
   (Customer_Number                 INTEGER                  4
   ,Last_Name                       CHAR(8)                  8
   ,First_Name                      CHAR(8)                  8
   ,Social_Security                 INTEGER                  4
   ,Birth_Date                      DATE                     4
   ,OD_Limit                        DECIMAL(7,2))            4
UNIQUE PRIMARY INDEX (Customer_Number);

                                                   Total    32
```



Determining the Logical Record Length with Variable Length Columns

By defining variable character columns as VARCHAR, you can frequently save a significant amount of storage space, but this does have a cost. Each variable-length column is required to provide a 2-byte leading length field, but you no longer have the cost of trailing blanks.

In calculating the correct LRECL parameter, you must allow not only for the 2-byte length field, but also for the *largest* column length accommodated. While the Logical Record Length may grow, the records themselves are typically shorter and of varying length.

Determining the Logical Record Length with Variable Length Columns

```
                                                          Length
CREATE TABLE Customer,  FALLBACK
   (Customer_Number                 INTEGER                  4
   ,Last_Name                       VARCHAR(8)              10
   ,First_Name                      VARCHAR(8)              10
   ,Social_Security                 INTEGER                  4
   ,Birth_Date                      DATE                     4
   ,OD_Limit                        DECIMAL(7,2))            4
UNIQUE PRIMARY INDEX (Customer_Number);

                                                   Total    36
```

Last_Name and First_Name redefined each as VARCHAR(8) reduces storage by 7 spaces, but adds two 2-byte length fields.

						J	o	n	e	s			J	a	c	k													
0	0	0	0	0	0	D	9	9	8	A	0	0	D	8	8	9	0	0	0	0	0	0	7	3	1	0	0	0
0	0	0	6	0	5	1	6	5	5	2	0	4	1	1	3	2	0	0	0	0	0	0	A	B	0	0	0	C
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Customer # | Last Name | First Name | Social Security | Birth Date | OD Limit

Determining the Logical Record Length with .EXPORT INDICDATA

If Indicator Variables are also used, then 1 byte will be allocated for every 8 columns or fields. For example, if there are 12 columns, then 2 bytes are needed for the NULL bits.

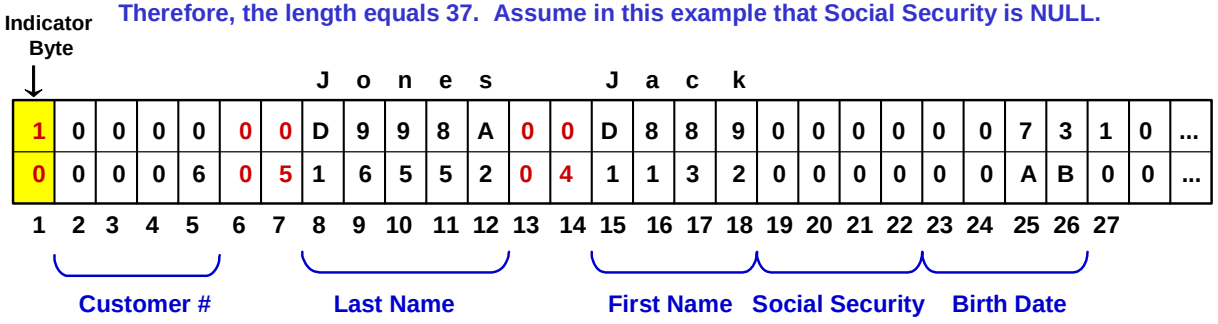
Determining the Logical Record Length with .EXPORT INDICDATA

```
                                                    Length
CREATE TABLE Customer,  FALLBACK
   (Customer_Number              INTEGER              4
   ,Last_Name                    VARCHAR(8)          10
   ,First_Name                   VARCHAR(8)          10
   ,Social_Security              INTEGER              4
   ,Birth_Date                   DATE                 4
   ,OD_Limit                     DECIMAL(7,2))        4
UNIQUE PRIMARY INDEX (Customer_Number);

                                                Total 37
```

Assume NULL for Social Security

All 6 columns are nullable adding 6 bits (1 byte) when using .EXPORT in INDICDATA mode. Therefore, the length equals 37. Assume in this example that Social Security is NULL.



.IMPORT (for Network-Attached Systems)

BTEQ is also useful when you want to IMPORT data from a network-attached server to Teradata as a series of INSERTs, UPDATEs, DELETEs, and "macro" transactions.

Since the Teradata database performs all necessary conversions from displayable characters to binary format, BTEQ .IMPORT has no need to support the concept of FIELD mode. BTEQ supports DATA, INDICDATA, REPORT, and VARTEXT formats for import. The VARTEXT record format is a Teradata V2R4 feature.

As .EXPORT permits the application programmer to limit the number of records written to the host file, .IMPORT allows you to skip a specified number of records at the beginning of the file. This allows you to derive a more typical sampling of transactions from the middle of the file.

VARTEXT Notes

The following rules apply to VARTEXT records:

1. The only acceptable data types are VARCHAR, VARBYTE, and LONG VARCHAR.

2. A delimiter at the end of the input record is optional.

3. The number of data items in the input record must be equal to the number of fields defined in the USING clause.

4. Two consecutive delimiter characters specify that the corresponding field should be "nulled". If the record starts with a delimiter, the first field will be "nulled".

5. You can use SET REPEATSTOP ON to stop inserting if an error is encountered during processing of a VARTEXT record. By default, it rejects the record and continues processing.

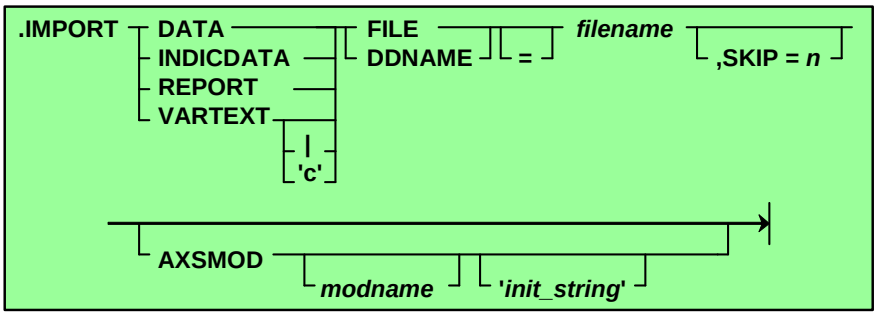
AXSMOD Example

The AXSMOD allows BTEQ to process a data file greater than 2 GB from a tape subsystem. The following two examples show an EXPORT/IMPORT pair with AXSMOD on a Windows 2000 platform using TNTBAR (Backup and Restore).

```
.EXPORT DATA FILE = 'temp' AXSMOD tntbar.dll 'device=tape0 tapeidbteq1 volset=vol1'
.IMPORT DATA FILE = 'temp' AXSMOD tntbar.dll 'device=tape0 tapeidbteq1 volset=vol1'
```

.IMPORT
(for Network-Attached Systems)

IMPORT loads data from a server to the Teradata database with a USING clause.



DATA	**imports data from the server to Teradata with a USING clause.**
INDICDATA	**import records contain NULL bits.**
REPORT	**imports Teradata "report" data. Data expected in BTEQ EXPORT REPORT format.**
VARTEXT	**record format as variable length character fields. Default delimiter is | or specify with field delimiter within single quotes.**
AXSMOD	**access module used to import from tape, named pipes, etc.**

.IMPORT (for Channel-Attached Systems)

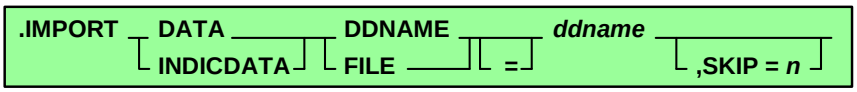
BTEQ is also useful when you want to IMPORT data from a channel-attached host to Teradata as a series of INSERTs, UPDATEs, DELETEs, and “macro” transactions.

Since the Teradata database performs all necessary conversions from displayable characters to binary format, BTEQ .IMPORT has no need to support the concept of FIELD mode. It only supports DATA (flat-file input) and INDICDATA modes for Channel-Attached systems.

As .EXPORT permits the application programmer to limit the number of records written to the host file, .IMPORT allows you to skip a specified number of records at the beginning of the file. This allows you to derive a more typical sampling of transactions from the middle of the file.

.IMPORT
(for Channel-Attached Systems)

IMPORT loads data from a mainframe host to the Teradata database with a USING clause.



.IMPORT DATA	**Reads a host file in record mode.**
.IMPORT INDICDATA	**Reads data in host format using indicator variables in record mode to identify nulls.**

DDNAME	**Name of MVS JCL DD statement or CMS FILEDEF.**
FILE	**Name of input data set in all other environments.**
SKIP = n	**Number of initial records from the data stream that should be skipped before the first row is transmitted.**

.PACK

The PACK command provides the capability of sending multiple IMPORT data file records along with an SQL request. BTEQ uses this factor to indicate an upper limit when determining how many records to pack into the USING data buffer sent with the SQL request.

A pack factor is established by using the PACK command or by using a PACK clause for the REPEAT command.

.REPEAT

Submits the next Teradata SQL request a specified number of times.

The REPEAT command is typically used with Teradata SQL requests that contain a USING clause. Each time the request is submitted, it uses the next data row from the input data stream. The REPEAT command is cancelled if the input file cannot be accessed.

Options include:

n	How many times you want to submit the next request.
*	The next request is to be submitted continuously until the import file runs out of data.
RECS *r*	Where *r* is an integer in the range of 1 .. 2251636603879500
PACK *p*	The REPEAT command's PACK clause overrides the PACK command setting for the duration of the repeat. Once the repeat is over, the pack factor returns to the value associated with the PACK command.

Example1: **.REPEAT * PACK 100**

This equates to repeat the request as many times as possible before reaching EOF (or max n) and pack up to 100 records with each request. The number of records actually sent is determined at REPEAT completion. The pack factor actually used may vary for each request sent.

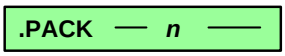
Example 2: **.REPEAT RECS 200 PACK 100**

If you need to ensure an exact number of records are transferred, use the RECS clause version for the repeat factor to compensate for any "reduced" requests. For example,

This equates to repeat the request as many times as necessary to read up to 200 records and pack a maximum of 100 records with each request.

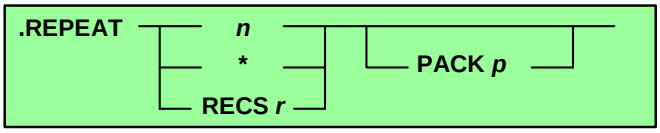
.PACK and .REPEAT

.PACK – Specifies how many records to pack into the USING data buffer sent with the SQL request.



The PACK command provides the capability of sending multiple IMPORT data file records along with an SQL request. BTEQ uses this factor to indicate an upper limit.

.REPEAT – Submits the next Teradata SQL request a specified number of times.



The REPEAT command's PACK clause overrides the PACK command setting for the duration of the repeat.

BTEQ IMPORT – Example 1

The facing page displays a sample BTEQ .IMPORT script with several interesting features:

- Elimination of individual accounting output (.QUIET ON)
- .REPEAT

.QUIET ON is used to avoid individual statement accounting in the default output file. If multiple sessions are used, all individual row accounting is eliminated, and final statistics are provided. This method avoids a great deal of performance-limiting I/O on the host.

The .REPEAT statement causes BTEQ to continue reading input values until reaching the limit specified. If this command is omitted, BTEQ will perform the required action only once.

When using .REPEAT:

- The asterisk (*) causes the next request to be submitted continuously until the import file runs out of data.
- You can specify a number to indicate how many times you want to submit the next request.

Note: You can use either of the syntax examples on the facing page. However, if you attempt to combine both into the same script, you will get the following error message.

*** Failure 3706 Syntax error: expected something between the word 'in_Custno' and 'C'

BTEQ IMPORT – Example 1

(loading data from a Linux Server)

import1.btq

```
.LOGON tdp1/user1,passwd1
.IMPORT DATA FILE = datafile1a;
.QUIET ON
.REPEAT  *
USING    in_CustNo    (INTEGER)
         , in_SocSec   (INTEGER)
         , Filler      (CHAR(30))
         , in_Lname    (CHAR(20))
         , in_Fname    (CHAR(10))
INSERT INTO  Customer
         ( Customer_Number
         , Last_Name
         , First_Name
         , Social_Security )
         VALUES
         (  :in_CustNo
         , :in_Lname
         , :in_Fname
         , :in_SocSec) ;
.QUIT
```

import1a.btq

```
.LOGON tdp1/user1,passwd1
.IMPORT DATA FILE = datafile1a;
.QUIET ON
.REPEAT  *
USING  ( in_CustNo    INTEGER
         , in_SocSec   INTEGER
         , Filler      CHAR(30)
         , in_Lname    CHAR(20)
         , in_Fname    CHAR(10) )
INSERT INTO  Customer
         ( Customer_Number
         , Last_Name
         , First_Name
         , Social_Security )
         VALUES
         (  :in_CustNo
         , :in_Lname
         , :in_Fname
         , :in_SocSec) ;
.QUIT
```

.QUIET ON

Limits output to reporting only errors and request processing statistics.

.REPEAT *

Causes BTEQ to read records until EOF.

USING

Defines the input data from the server.

Note the syntax difference with the USING option.

To execute:

```
bteq < import1.btq | tee import1.out
```

BTEQ IMPORT – Example 2

On a Linux platform, **bteq** can be called in either its pseudo-interactive mode or by reference to an input script. To specify the script files, use the "<" for input redirection and the ">" for output redirection.

BTEQ IMPORT – Example 2

(using imported data as update data)

```
bteq
Enter your BTEQ Command:
.RUN FILE = c:\td_scripts\import2.btq
   or
bteq < c:\td_scripts\import2.btq
```

This example shows execution of a BTEQ script on a Windows server and the optional use of the .RUN command.

.RUN – processes the Teradata SQL requests and BTEQ commands from the specified run file.

import2.btq

```
. LOGON  tpd1/user1,passwd1
. IMPORT DATA FILE = c:\td_datafiles\datafile2
. QUIET ON
. REPEAT  * PACK 10
 USING          in_CustNo          (INTEGER)
               , in_SocSec         (INTEGER)
 UPDATE        Customer
    SET        Social_Security    = :in_SocSec
    WHERE      Customer_Number    = :in_CustNo
 ;UPDATE       Customer_History
    SET        Social_Security    = :in_SocSec
    WHERE      Customer_Number    = :in_CustNo ;
.QUIT;
```

.REPEAT * PACK 10

Causes BTEQ to read records until EOF. Up to 10 import records are sent along with the SQL request.

Note: Multi-statement request

Allows BTEQ to use imported data with multiple tables.

BTEQ IMPORT – Example 3

The facing page displays a simple BTEQ .IMPORT script which imports a CSV (Comma Separated Value) file.

You have to specify the delimiter (the comma in a CSV file) since the default is the |.

BTEQ IMPORT – Example 3

(importing data from a CSV file)

This script effectively imports data from a file with fields that are separated by a comma; referred to as a CSV data file (CSV – Comma Separated Value).

export3.btq

```
.LOGON tdp1/user1,passwd1
.IMPORT VARTEXT  ','  FILE = custdata_csv
.QUIET ON
.REPEAT *
     USING ( in_custno       VARCHAR(11),
             in_lname        VARCHAR(30),
             in_fname        VARCHAR(20),
             in_ssn          VARCHAR(9) )
     INSERT INTO Customer
     VALUES (:in_custno, :in_lname, :in_fname,:in_ssn);
.QUIET OFF
SELECT COUNT(*) FROM Customer;
.QUIT
```

To execute:

```
bteq < export3.btq | tee export3.out
```

Multiple Sessions

A session might be defined as a “logon to the Teradata database that permits the user to single-thread one or more sequential transactions that continue until a LOGOFF statement is sent."

Teradata interprets multiple sessions as a number of users logging on (with the same user ID and password), with each one sending sequential transactions until it submits a LOGOFF statement.

Multiple sessions permit the Teradata database to work on multiple tasks in parallel. This capability requires special software in the (sequential processing) host to enable it to send *multiple requests at the same time*. This special software is the Call Level Interface.

For multiple sessions to be effective in parallel, the system uses *row hash locks* rather than *full table locks*. Since the only type of access that uses row hash locking is the Primary or Unique Secondary Index request, multiple sessions are useful only in connection with UPI, NUPI or USI transactions. All other access requires a full table lock and results in sequential access to tables.

.SET SESSIONS

The .SET SESSIONS command must appear prior to the .LOGON statement since it directs BTEQ to initialize the appropriate number of LOGONs. The SESSIONS parameter tells BTEQ how many times to log on to the Teradata database for parallel access.

The sessions are then logged on by TDP or the Teradata GATEWAY and logged on to the dedicated PEs to distribute the workload as evenly as possible.

If only a single session (no .SET SESSION statement) is used, BTEQ will still provide the elapsed time associated with each DML.

The limit on the number of sessions that you can request with BTEQ depends on the operating environment.

Typically, BTEQ Imports will take advantage of multiple sessions and BTEQ Exports will not.

Multiple Sessions

- **Session:**
  - Logical connection between host and Teradata database.
  - Work stream composed of a series of requests between the host and the database.
- **Multiple sessions:**
  - Allow tasks to be worked on in parallel.
  - Require row hash locking for parallel processing: UPI, NUPI, USI transactions.
  - Too few degrade performance.
  - Too many will not improve performance.

Specified before .LOGON

```
.SET SESSIONS 8
.LOGON tdp1/user1,passwd1
.IMPORT DATA FILE = datafile4
.QUIET ON
.REPEAT *
USING     ( in_CustNo          INTEGER
          , in_SocSec          INTEGER
          , Filler             CHAR(30)
          , in_LName           CHAR(20)
          , in_FName           CHAR(10) )
               :

.QUIT
```

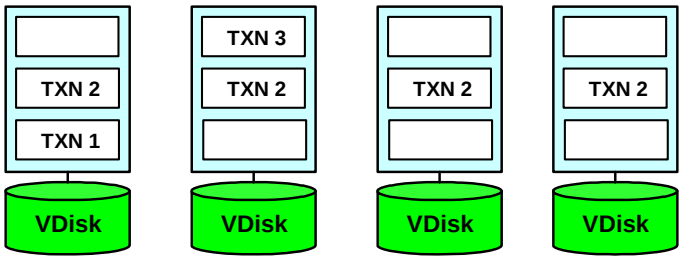
Parallel Processing Using Multiple Sessions to Access Individual Rows

It is important to remember that multiple sessions only benefit transactions that do *not* use a full table lock.

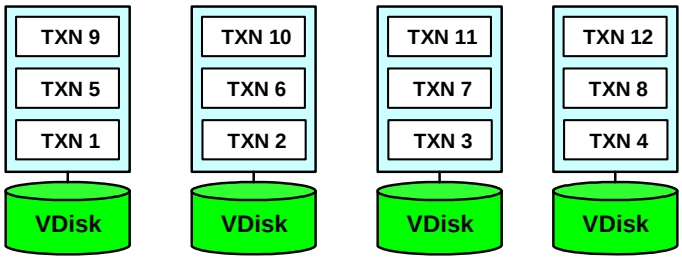
If multiple sessions are specified where a full table lock is required, not only will they execute sequentially but performance will be inhibited by the system resources needed for management.

Parallel Processing Using Multiple Sessions to Access Individual Rows

Although a single row can reside on 1 AMP, it might require a full table scan to locate it. All AMPs are required to participate.



If the location of the row is known, only the applicable AMP needs to be involved. Other AMPs can work on other tasks — multiple sessions are useful.



Multiple transactions execute in parallel, provided that:

- **Each transaction uses fewer than all AMPs.**
- **Enough are sent to keep ALL AMPs busy.**
- **Each parallel transaction has a unique internal ID.**

When Do Multiple Sessions Make Sense?

Assuming there are no fallback tables:

- Primary Index transactions use 1 AMP and a Row Hash lock.
- Unique Secondary Index transactions use 2 AMPs and Row Hash locks.
- Non-unique Secondary Index transactions and full table scans use all AMPs and table-level locks.

You should now be in a position to respond to users who complain that since they were using multiple sessions to perform BTEQ updates and that the job ran faster with one session, therefore BTEQ was broken.

Try to complete the table on the facing page to determine which type of requests would benefit from multiple sessions.

When Do Multiple Sessions Make Sense?

Multiple sessions improve performance ONLY for SQL requests that impact fewer than ALL AMPs.

TRANS_HISTORY

Trans_Number	Trans_Date	Account_Number	Trans_ID	Amount
PK		FK,NN		
USI		NUPI		
NUSI				

Which of the following batch requests would benefit from multiple sessions?

1. INSERT INTO Trans_History
   VALUES (:T_Nbr, DATE, :Acct_Nbr, :T_ID, :Amt);
2. SELECT * FROM Trans_History
   WHERE Trans_Number=:Trans_Number;
3. DELETE FROM Trans_History
   WHERE Trans_Date < DATE - 120;
4. DELETE FROM Trans_History
   WHERE Account_Number= :Account_Number;

Trans Type	Table or Row Lock	Multiple Sessions Useful or Not?

Application Utility Checklist

The checklist on the facing page summarizes BTEQ functions. It will be completed for each utility as it is discussed.

While BTEQ efficiently restarts from a failure of the Teradata database system, it must rollback in the event of a host failure and restart from the first record.

Clearly this can be a distinct disadvantage. While BTEQ can be very useful and performs reasonably well, your applications probably also require the ability to restart from a host failure without rolling back to the first record.

For this reason, many of the other application utilities have a restart capability built in.

Automatic Restart – If the Teradata server restarts, the utility will retry to connect to the Teradata database automatically and restart.

Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions	ALL				
DML Functions	ALL				
Multiple DML	Yes				
Multiple Tables	Yes				
Protocol Used	SQL				
Conditional APPLY	No				
Data Conversion	Yes				
Error Capture	No				
Error Limits	No				
User-written Routines	No				
Automatic Restart	No				
Max Load Limit	No				
Support Environment (SE)	No				

Module 33: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 33: Review Questions

Answer True or False.

1. **True or False.** **With BTEQ you can import data from the host to Teradata AND export from Teradata to the host.**

2. **True or False.** **.EXPORT DATA sends results to a host file in field mode.**

3. **True or False.** **INDICDATA is used to preserve nulls.**

4. **True or False.** **With BTEQ, you can use conditional logic to bypass statements based on a test of an error code.**

5. **True or False.** **It is useful to employ multiple sessions when ALL AMPS will be used for the transaction.**

6. **True or False.** **With .EXPORT, you can have output converted to a format that can be used with PC programs.**

Lab Exercise 33-1

After creating the data file named **data33_1**, you may want to check its size to ensure that you have created it correctly. An easy way to do this is in Linux is to use the Linux **ls –l** command.

The size of this file (data33_1) should be 312,000 bytes.

A technique that can be used to create Linux scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where Linux is running and …

3. **cat > lab33_14.btq** (or whatever filename you wish)

   Use the mouse to choose the **Edit ➔ Paste** function

   To exit the cat command, press either the DELETE key or CNTL C.

Lab Exercise 33-1

Lab Exercise 33-1

Purpose

In this lab, you will use BTEQ to perform imports with different numbers of sessions. You will move selected rows from the AP.Accounts table to your personal Accounts table and from a data file to your table. You will repeat tasks using different numbers of sessions.

What you need

Populated AP.Accounts table and your empty Accounts table.

Tasks

1. INSERT/SELECT all rows into your Accounts table (userid.Accounts from the populated AP.Accounts table. Note the timing and verify that you have the correct number of rows.

   Time: Number of rows:

2. Export 4000 rows to a data file (data33_1).
3. Delete all rows in your "Accounts" table.
4. Import the rows from your data set (data33_1) to your empty Accounts table. Note the time and verify the number of rows.

   Time: Number of rows:

5. Delete all the rows in your "userid.Accounts" table again.

Lab Exercise 33-1 (cont.)

Recommendation: Include the PACK option as part of the .REPEAT command.

Lab Exercise 33-1 (cont.)

6. **Specify 8 sessions and import the rows from your data set to your empty Accounts table. Note the time and verify the number of rows.**

   Time: **Number of rows:**

7. **Delete all the rows from your "Accounts" table again.**

8. **Specify 8 sessions and use a PACK of 10 and import the rows from your data set to your empty Accounts table. Note the timing and verify the number of rows.**

   Time: **Number of rows:**

9. **What are your conclusions based on the tasks you have just performed?**

   _____

   _____

Lab Exercise 33-2

The size of data33_2 should be 3,500 bytes.

Note: If your data file size is 30,500 bytes, you exported all of the columns from the Customer table instead of just the Customer Number.

Lab Exercise 33-2

Lab Exercise 33-2

Purpose

In this exercise, use BTEQ to export 500 customer numbers (.EXPORT DATA) from the AP.Customer table. This data file should contain 500 customer numbers representing the customers with the highest social security numbers. Using this as input to access the Customer table, generate a report file (.EXPORT REPORT).

What you need

Populated AP.Customer table

Tasks

1. Using AP.Customer table, export to a data file (data33_2) the 500 customer numbers for the customers that have the highest Social Security numbers. (Hint: Use the descending order option (DESC) for Social Security numbers and only export the customer numbers.)
2. Using the 500 customer numbers (in data33_2), select the 500 appropriate rows from AP.Customer and export a report file named "report33_2". In the report, include these fields: Customer_Number, Last_Name, First_Name, Social_Security.

   Hint: You will .IMPORT DATA from data33_2 and use .EXPORT REPORT to report33_2.
3. View your report using the Linux *more* command. The completed report should look like this:

Customer_Number	Last_Name	First_Name	Social_Security
2001	Smith	John	123456789

4. What are highest and lowest Social Security numbers in your report?

   Highest: _____ Lowest: _____

Notes

Module 34

FastLoad

After completing this module, you will be able to:

- **Describe the two phases of FastLoad.**
- **Prepare a FastLoad script.**
- **Partition a data load over successive runs.**
- **Restart an interrupted FastLoad.**

Notes

Table of Contents

FastLoad 34-4
FastLoad Acquisition Phase (Phase 1) 34-6
FastLoad End Loading Phase (Phase 2) 34-8
FastLoad and NoPI Tables 34-10
FastLoad – NoPI Table 34-12
A Sample FastLoad Script 34-14
Converting the Data 34-16
Data Conversion Chart 34-18
NULLIF 34-20
  FastLoad INMODs 34-20
FastLoading Zoned Decimals and Time Stamps 34-22
FastLoad BEGIN LOADING Statement 34-24
BEGIN LOADING Statement 34-24
FastLoad Error Tables 34-26
Error Recovery 34-28
CHECKPOINT Option 34-30
END LOADING Statement 34-32
RECORD Statement 34-34
INSERT Statement 34-36
Staged Loading of Multiple Data Files 34-38
FastLoad Fails to Complete 34-40
Restarting FastLoad (Output) 34-42
Restarting FastLoad – Summary 34-44
Additional FastLoad Commands 34-46
FastLoad with Additional Options 34-48
Invoking FastLoad 34-50
Application Utility Checklist 34-52
Summary 34-54
Module 34: Review Questions 34-56
Lab Exercise 34-1 34-58
Lab Exercise 34-2 34-60

FastLoad

The facing page describes the capabilities of FastLoad.

FastLoad

Features

- Purpose – insert large amounts of data into a single empty table at high speed.
- Available on Teradata nodes, servers, channel, or network-attached hosts.
  - Teradata 12 Driver for the JDBC Interface provides FastLoad support.
- Load data in stages – input data may be loaded from multiple separate batches
- Can be executed in batch or interactive mode.
- Supports INMOD routines.
- Input data that fails to load is saved in error tables.
- Input data error limits may be set.
- Checkpoints can be taken for automatic restarts.

Restrictions

- The target table must initially be empty.
- The target table can NOT have Secondary Indexes (USI/NUSI), Join Indexes, or Hash Indexes.
- Referential Integrity constraints are not supported, however Soft RI is supported.
- The target table can NOT have Enabled Triggers.
- Duplicate rows are NOT loaded into a target table (even if the table is MULTISET).
- If an AMP goes down, FastLoad can NOT be restarted until the AMP is back online.

FastLoad Acquisition Phase (Phase 1)

FastLoad with a PI table has two phases. The Acquisition Phase is also called Phase 1 and the End Loading Phase is Phase 2

For each FastLoad job, there are two SQL sessions, one for handling SQL requests and the other for handling log table restart-related operations. There are also load sessions established for each FastLoad job that can be specified in a FastLoad script via the SESSIONS command.

Note: The blocks can be up to 63.5 KB (default).

The illustration on the facing page shows the process used in Phase 1.

Step 1 – FastLoad is executed on a host and establishes two Parsing Engine SQL sessions and one or more Load Session on AMPs (depending on the SESSIONS parameter).

As an overview, depending on the number of sessions requested, blocks of data will be distributed to different AMPs in the system. If the number of sessions requested is less than the number of AMPs, then blocks will be distributed to those AMPs for which "load" sessions have been established.

Step 2 – FastLoad sends blocks of records to Teradata.

Step 3 – The AMP receives a block of records in memory. When a FastLoad session is established, a well-known mailbox of the AMP vproc that is associated with the session is sent back to the Client. During the data acquisition phase, data rows that are sent from the FastLoad Client through that session will go to the mailbox. The Load Control Task (with the AMP) first picks up the data rows and then forwards them to the local AMP deblocking task.

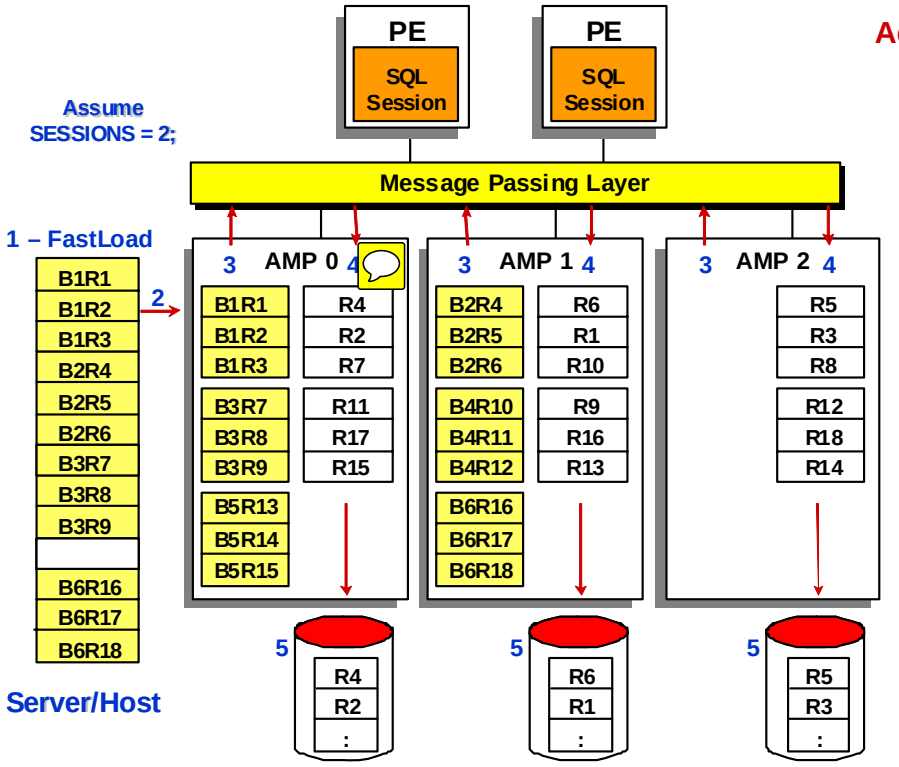
The AMP deblocking task hashes each record in the block and redistributes each row to the Message Passing Layer (PDE and BYNET). The Message Passing Layer (MPL) delivers rows to the appropriate AMP based on row hash value.

Step 4 – Every AMP will have a receiving task which collects the rows from the MPL.

Step 5 – When enough rows are collected to fill a block in memory, the AMP writes the block to disk.

At this point, the AMP has the rows it should have, but they are not in row hash sequence.

FastLoad Acquisition Phase (Phase 1)



Acquisition Phase

- FastLoad uses one SQL session to define AMP steps and another SQL session for log table restart operations
- FastLoad sends a block of records directly to each AMP which has a "Load" session.
- The deblocking task within the AMP hashes each record and redistributes the rows.
- Every AMP's receiving task collects the rows and writes a block of unsorted data rows to disk.
- At the end of this phase, each AMP has the rows it should have, but the rows are not in row hash sequence.

FastLoad End Loading Phase (Phase 2)

The second phase of FastLoad has each AMP (in parallel) reading the data blocks from disk, sorting the data rows based on row hash, and writing the blocks back out to PERM space.

The illustration on the facing page shows the process used in Phase 2.

Step 6 – FastLoad receives the END LOADING; statement. FastLoad sends a request to the Parsing Engine to indicate the start of Phase 2.

Step 7 – The PE broadcasts the start of Phase 2 to all AMPs.

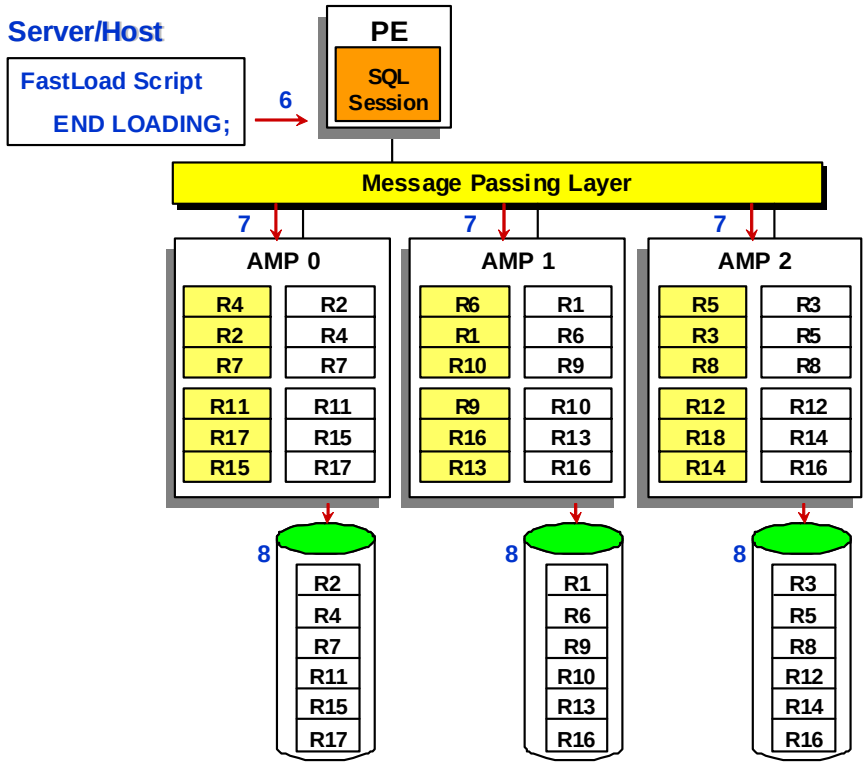
Step 8 – Each AMP reads its blocks in from disk.

Step 9 – Each AMP sorts its data rows based on row hash sequence.

Step10 – Each AMP writes the sorted blocks back to disk.

If the table is Fallback protected, then the Fallback copy of data is created at this time. The table is made available for user access.

FastLoad End Loading Phase (Phase 2)



End Loading Phase

- When the FastLoad job receives END LOADING, FastLoad starts the End Loading Phase.
- Each AMP sorts the target table, puts the rows into blocks, and writes the blocks to disk.
- Duplicate rows are removed.
- Fallback rows are then generated if required.
- Table data is available when this phase completes.

FastLoad and NoPI Tables

The facing page describes some considerations of using FastLoad with a NoPI table.

There are several areas that work differently with FastLoad on NoPI tables.

Acquisition Phase (Phase 1)

- Since the number of FastLoad sessions is often less than the number of AMPs in the system (common with large systems), there will be some AMPs that do not have a session (deblocker) task. These AMPs will not receive any data from FastLoad.

- Therefore (to avoid skewing), each block of data received by a deblocker task will be redistributed to all of the AMPs in a round robin fashion. The AMP that receives the block will simply append it to the target table.

End Loading or Sort Phase (Phase 2)

- There is no sorting of the rows on a NoPI table so this portion of phase 2 is completely eliminated for a NoPI table.

- With a PI table, duplicate rows are discarded in this phase. Duplicate rows are NOT discarded with a NoPI table.

- With PI tables, error table 2 is used to record UPI violations. This error table not used with NoPI tables, but it is created and dropped at job end**.**

- If a table has Fallback protection, then Fallback is added as part of Phase 2.

FastLoad and NoPI Tables

A NoPI Table is useful as a staging table. Loading data into a NoPI staging table will be faster than when compared to the same table that has a primary index.

- Data can be loaded into a staging table (NoPI table) quickly using FastLoad freeing up client resources earlier for other work.

FastLoad

- The data-redistribution processing in the acquisition phase is done more efficiently by using bigger blocks to distribute the rows between AMPs (4 KB versus 64 KB).
- The End Loading or Sort phase is eliminated.
- Since a NoPI table load is faster, the staging table is available sooner.

FastLoad Acquisition Phase Differences

- Since the number of FastLoad sessions is often less than the number of AMPs in the system (common with large systems), there will be some AMPs that do not have a session (deblocker) task. These AMPs will not receive any data from FastLoad.

  Therefore (to avoid skewing), each block of data received by a deblocker task will be redistributed to all of the AMPs in a round robin fashion. The AMP that receives the block will simply append it to the target table.

FastLoad – NoPI Table

FastLoad with NoPI also has two phases. The Acquisition Phase is also called Phase 1 and the End Loading Phase is Phase 2

For each FastLoad job (whether a PI or NoPI table), there are two SQL sessions, one for handling SQL requests and the other for handling log table restart-related operations. There are also load sessions established for each FastLoad job that can be specified in a FastLoad script via the SESSIONS command.

The illustration on the facing page shows the process used in the Acquisition Phase.

Step 1 – FastLoad is executed on a host and establishes two Parsing Engine SQL sessions and one or more Load Session on AMPs (depending on the SESSIONS parameter).

As an overview, depending on the number of sessions requested, blocks of data will be distributed to different AMPs in the system. If the number of sessions requested is less than the number of AMPs, then blocks will be distributed to those AMPs for which "load" sessions have been established.

Step 2 – FastLoad sends blocks of records to Teradata.

Step 3 – The AMP receives a block of records in memory. With a NoPI table, an AMP will distribute blocks of data to other AMPs in a round robin fashion. Blocks are not broken into rows.

Rows are sent from the FastLoad client to the AMP load sessions in round robin fashion. For example, assume a system has 100 AMPs but there are only 10 load sessions. Therefore, rows are sent to only 10 AMPs from the client (FastLoad). Teradata then redistributes the blocks from those 10 AMPs to all of the AMPs in a round robin fashion.

With the new NoPI hash map, hash bucket is selected from this NoPI hash map for NoPI table. It is the same for FastLoad or SQL.

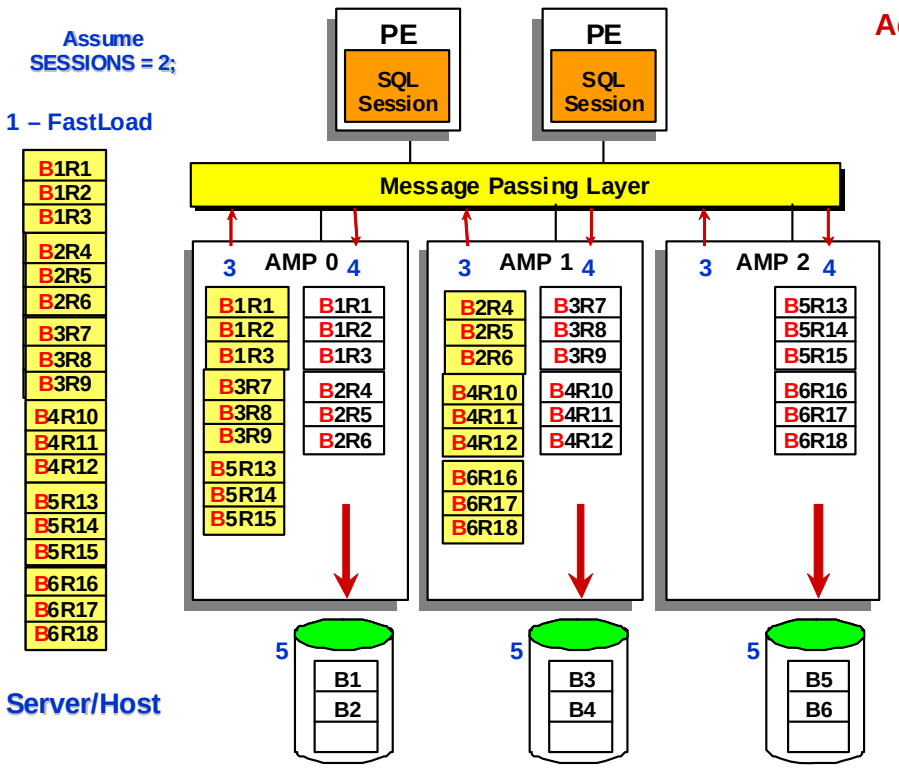
Step 4 – Every AMP will have a receiving task which collects the block from the MPL.

Step 5 – The AMP writes the block to disk.

At this point, the AMP has the rows it should have and new blocks are simply appended to existing blocks in a NoPI table.

The sort function in the End Loading or Sort Phase (Phase 2) is not done with a NoPI table. However, if the table is Fallback protected, then the Fallback subtable is created in phase 2.

FastLoad Acquisition Phase (NoPI Table)



Acquisition Phase

- FastLoad uses one SQL session to define AMP steps and another SQL session for log table restart operations.
- FastLoad sends a block of records directly to each AMP which has an assigned "Load" session.
- With a NoPI table, the AMP distributes blocks of data between the AMPs in a round robin fashion.
- Every AMP has a receiving task which collects the blocks and writes the block to disk.

A Sample FastLoad Script

The job on the facing page first cleans up and prepares the environment for use. It could be performed using BTEQ.

The first step is to make sure that the table is empty and to remove old error tables before starting the FastLoad job.

Often it is better to perform set-up steps outside the FastLoad script so that the FastLoad operation can be isolated to perform loading tasks. If a restart is necessary, you will not need to change your FastLoad script to remove the unneeded statements.

The load job on the facing page is being run on a Linux system as noted by the “FILE=” parameter. If this job was going to be run on a MVS system, then use the “DDName=” parameter instead of the “File” parameter.

A Sample FastLoad Script

SETUP
Create the table, if it doesn't already exist.

```
LOGON tdpid/username,password;
DROP TABLE Acct;
DROP TABLE Acct_e1;
DROP TABLE Acct_e2;

CREATE TABLE Acct, FALLBACK (
    Acct_Num        INTEGER
   ,Street_Num      INTEGER
   ,Street          CHAR(25)
   ,City            CHAR(25)
   ,State           CHAR(2)
   ,Zip_Code        INTEGER)
UNIQUE PRIMARY INDEX (Acct_Num);
LOGOFF;
```

```
LOGON tdpid/username,password;

BEGIN LOADING Acct
   ERRORFILES Acct_e1, Acct_e2
   CHECKPOINT 100000;


DEFINE  in_AcctNum (INTEGER)
          ,in_Zip       (INTEGER)
          ,in_Nbr       (INTEGER)
          ,in_Street    (CHAR(25))
          ,in_State     (CHAR(2))
          ,in_City      (CHAR(25))
  FILE=datafile1;

INSERT INTO Acct VALUES (
           :in_AcctNum
          ,:in_Nbr
          ,:in_Street
          ,:in_City
          ,:in_State
          ,:in_Zip);

END LOADING;
LOGOFF;
```

Start the utility. Error files must be defined.

Checkpoint is optional.

DEFINE the input; must agree with host data format.

INSERT must agree with table definition. Phase 1 begins. Unsorted blocks are written to disk.

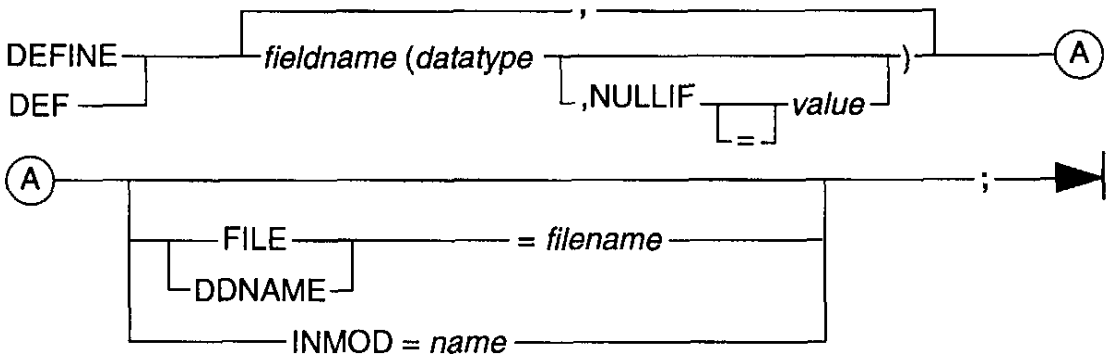
Phase 2 begins with END LOADING. Sorting and writing blocks to disk.

Converting the Data

This example shows two examples of conversions you can perform in the FastLoad environment.

Each input data field (DEFINE) must undergo a conversion to fit in the database field (Create Table).

All are valid conversions and are limited to **one per column**.



Valid data types for table that can be loaded with FastLoad are:

BIGINT
BYTE
BYTEINT
CHARACTERS(n)
DATE
DECIMAL(x) OR DECIMAL(x,y)
FLOAT
GRAPHIC(n)
INTEGER
LONG VARCHAR
LONG VARGRAPHIC
SMALLINT
VARBYTE(n)
VARCHAR(n)
VARGRAPHIC(n)

Converting the Data

```
LOGON educ2/user14,ziplock;
DROP TABLE Accounts;
DROP TABLE Accts_e1;
DROP TABLE Accts_e2;
CREATE TABLE Accounts, FALLBACK (
        Account_Number  INTEGER
       ,Account_Status  CHAR(15)
       ,Trans_Date      DATE
       ,Balance_Forward DECIMAL(5,2)
       ,Balance_Current DECIMAL(7,2) )
  UNIQUE PRIMARY INDEX
       (Account_Number);
LOGOFF;
```

```
LOGON educ2/user14,ziplock;
BEGIN LOADING Accounts
      ERRORFILES Accts_e1, Accts_e2 ;
DEFINE  in_Acctno       (CHAR(9))
       , in_Trnsdate    (CHAR(10))
       , in_Balcurr     (CHAR(7))
       , in_Balfwd      (INTEGER)
       , in_Status      (CHAR(10))
   FILE = datafile2;
INSERT INTO Accounts
       (Account_Number
       ,Account_Status
       ,Trans_Date
       ,Balance_Forward
       ,Balance_Current)
VALUES (
        :in_Acctno
       ,:in_Status
       ,:in_Trnsdate  (Format 'YYYY-MM-DD')
       ,:in_Balfwd
       ,:in_Balcurr);
END LOADING;
LOGOFF;
```

Notes:

- FastLoad permits conversion from one data type to another, *once* for each column.
- Including optional column names with the INSERT statement provides script documentation which may aid in the future when debugging or modifying the job script.

Data Conversion Chart

On the facing page is a comprehensive chart of possible conversions, showing the old data type and the new data type, and sample data for each.

An INVALID result comes from an unsupported conversion.

An overflow output is the result of too much data for the receiving field.

General Notes:

- The “target table” can have range constraints” defined at the column level when the table is created and these will be checked as part of Phase 1 with FastLoad.

- If the input field has leading or trailing spaces (blanks) and you are converting to a numeric field (e.g., INTEGER), leading/trailing spaces are ignored.

Ex.	**converted to**
'0001'	**0001**
' 001'	**0001**
'001 '	**0001**
' 01 '	**0001**
' '	**0000**
'1 '	**0001**
'1 0'	**conversion error**

Data Conversion Examples

FROM:	TO:	ORIGINAL DATA:	STORED AS:
CHAR(13)	VARCHAR(5)	ABCDEFHIJKLM	ABCDE
CHAR(5)	INTEGER	ABCDE	invalid
CHAR(5)	INTEGER	12345	0000012345
CHAR(13)	INTEGER	12345bbbbbbbb	0000012345
CHAR(13)	INTEGER	1234567890123	overflow
CHAR(13)	DATE	92/01/15bbbbb	920115
CHAR(13)	DATE	920115bbbbbbb	invalid
CHAR(13)	DATE	01/15/92bbbbb	invalid
CHAR(6)	DEC(5,2)	123.50	123.50
CHAR(6)	DEC(5,2)	12350	overflow
VARCHAR(5)	CHAR(13)	ABCDE	ABCDEbbbbbbbb
BYTEINT	INTEGER	123	0000000123
SMALLINT	INTEGER	12345	0000012345
INTEGER	SMALLINT	0000012345	12345
INTEGER	SMALLINT	1234567890	invalid
INTEGER	BYTEINT	0000000123	123
INTEGER	BYTEINT	0000012345	invalid
INTEGER	DATE	0000920115	920115
INTEGER	CHAR(8)	0000012345	bbb12345
DECIMAL(3,2)	INTEGER	1v23	0000000001
DECIMAL(3,2)	CHAR(5)	1v23	b1.23
DECIMAL(3,2)	CHAR(3)	1v23	1.2
DATE	INTEGER	0000920115	0000920115
DATE	SMALLINT	0000920115	invalid
DATE	CHAR(8)	0000920115	92/01/15
DATE	CHAR(6)	0000920115	92/01/

NULLIF

NULLIF is used for special conversions. When another system uses a special combination to represent unknown data values, you can test for them and convert them to NULL in the Teradata database.

If you attempt to place a zero (0) into a DATE field, you will get the following error:

#3520 A constant value in a query is not valid for column *colname*.

FastLoad INMODs

An INMOD is an exit routine that can precondition data and pass it on to the loader. You can write INMODs to pre-screen the input data being provided to FastLoad.

The INMOD and FastLoad use a return code value to communicate with each other.

You can write INMODs as restartable routines so that they synchronize with the loader's checkpoints.

Use INMODs to perform unusual conversions of data, for example, adding a sequenced column to the data, or reading data from a non-standard input file format.

Note: When there is a serious problem, and the job must be terminated, the INMOD function must return a 4-byte integer.

To reference an INMOD routine, use the INMOD option with the DEFINE statement.

```
DEFINE      fieldname1    (INTEGER,
            fieldname2    (CHAR(4)),
            …
INMOD = name_of_inmod_routine;
```

NULLIF

```
DEFINE      in_Acctno           (CHAR(9))
           ,in_Status           (CHAR(10))
           ,in_Trnsdate         (CHAR(10),   NULLIF = '0000-00-00')
           ,in_Balfwd           (INTEGER)
           ,in_Balcurr          (CHAR(7))
FILE = datafile3;

INSERT INTO Accounts VALUES (
            :in_Acctno
           ,:in_Status
           ,:in_Trnsdate        (FORMAT 'YYYY-MM-DD')
           ,:in_Balfwd
           ,:in_Balcurr);
```

- NULLIF allows you to specify that if an input field contains a specified value, it should be treated as NULL.
  - You can only include one NULLIF value with the NULLIF option.
- One common example occurs when dates are entered as zeroes; they may cause a fault since they are not in the expected format.

FastLoading Zoned Decimals and Time Stamps

On the facing page are two common conversion situations that you may encounter.

"Packed Decimal" is a mainframe data type that can be converted to decimal in the Teradata database.

Dates and/or Time Stamps are often presented to the loader in display form or character format. To convert them into acceptable dates or time stamps in the database, you must identify the input form with a "format" statement.

The following are acceptable in DATE FORMAT statements:

- yyyy (four digit year)
- yy (year in two digits)
- mmm (three character abbreviation of month)
- mm (two digit month)
- ddd (day of the year)
- dd (day of the month)
- a series of punctuation characters

  . decimal
  , comma
  - dash
  b blank (space)
  / slash

If the input file has a field with a TimeStamp(0) in it, then

DEFINE input field as CHARACTER (19);
on the INSERT use (FORMAT 'YYYY-MM-DDbHH:MI:SS')

Note: TimeStamp(0) indicates that there are no decimal digits associated with the seconds portion of the time stamp.

FastLoading Zoned Decimals and Time Stamps

- If EBCDIC unpacked decimal values are presented for loading into a decimal-type column, an error is returned:

Unpacked	F1	F2	F3	F4	F5	F6	F7	C8
Decimal (8)	1	2	3	4	5	6	7	H
Packed decimal	01	23	45	67	8C			

2679 Format or data contains a bad character.

Loads into decimal-defined column with no errors.

- Use the following script structure if a signed zoned decimal data representation is required or if time stamps are in character format.

```
CREATE TABLE Trans
          ( Tr_Num          DECIMAL(8)
          , Tr_DateTIme     TIMESTAMP(0)
          . . . . .             );
DEFINE     in_TNbr          (CHAR(8))
          ,in_TDateTime     (CHAR(19))
FILE = datafile4 ;
INSERT INTO TRANS VALUES
          (:in_TNbr         (FORMAT '9(8)S')
          ,:in_TDateTime    (FORMAT 'YYYY-MM-DDbHH:MI:SS')
          . . . . .   );
```

Define unpacked decimal as CHAR data and FORMAT.

FastLoad BEGIN LOADING Statement

The general syntax for the FastLoad statement to begin loading is shown on the facing page.

The BEGIN LOADING statement must reference a table, not a view.

BEGIN LOADING Statement

Use the BEGIN LOADING statement to identify the table that is to be loaded with FastLoad. The table may be in the user's default database (shown), or, by prefixing the name with a *databasename* and a dot (.), you may specify another database.

Use the BEGIN LOADING statement to specify (and create) the two error tables for the operation. You can specify error tables in the same database as the data table or in a different database.

You can also define a checkpoint interval for the job. It is specified as a number of input records. The checkpoint will be rounded to the nearest number of records that can fit within a block.

The FastLoad utility can recognize the indicator bytes placed in front of the data record by the application that created the input. These bytes identify the fields in the input record that should be NULL in the created record of the table and can be automatically generated by either BTEQ Export (INDICDATA) or FastExport (INDICATORS).

BEGIN LOADING Statement

```
BEGIN LOADING [dbname.]table_name
      ERRORFILES    [dbname.]Err_Table_1,
                    [dbname.]Err_Table_2
      [ CHECKPOINT integer ]
      [ INDICATORS ] ;
```

- Name of empty table (not a view).
- Name of two error tables (required).
- Optional checkpoint interval.
- Allows NULLs to be preserved.

A userid needs the following privileges in order to execute the FastLoad utility.

- For *Target_table_name*: SELECT and INSERT (CREATE and DROP or DELETE, if using those functions)
- For *Error_tables:* CREATE TABLE
- Required privileges for the user PUBLIC on the restart log table (SYSADMIN.FASTLOG):
  - SELECT INSERT UPDATE DELETE
- There will be a row in the FASTLOG table for each FastLoad job that has not completed in the system.

FastLoad Error Tables

FastLoad requires two error tables that you specify in the Begin Loading statement. Error tables capture data and duplication errors.

FastLoad discards empty error tables at the successful conclusion of the loader job. If they contain data, error tables are maintained for you to use in analyzing errors.

You must remove the error tables before you re-run the same load job or it will terminate in an error condition.

If you must restart a FastLoad job, the error tables must already exist.

FastLoad Error Tables

ErrorTable1

Contains one row for each row which failed to be loaded due to constraint violations or translation errors. The table has three columns:

Column_Name	Datatype	Content
ErrorCode	Integer	The Error Code in DBC.ErrorMsgs.
ErrorFieldName	VarChar(30)	The column that caused the error.
DataParcel	VarByte(64000)	The data record sent by the host.

ErrorTable2

For non-duplicate rows, captures those rows that cause a UPI duplicate violation.

Notes

- Duplicate rows are counted and reported but not captured.
- Error tables are automatically dropped if empty upon completion of the run.
- Performance Note: Rows are written into error tables one row at a time. Errors slow down FastLoad.

Error Recovery

The facing page contains some sample output from a FastLoad job. It describes the error situations encountered during the previous FastLoad job.

Because there are errors in both of the error tables, they will be retained by the system for analysis.

You must remember to analyze these error tables and remove them from the system prior to running this same FastLoad script again.

Error Recovery

Output report from FastLoad

A	Total Records Read	=	35000
B	Total ErrorTable 1	=	1250 (Not loaded due to error)
C	Total ErrorTable 2	=	30 (Duplicate UPIs only)
D	Total Inserts Applied	=	33700
E	Total Duplicate Rows	=	20

(A = B + C + D + E)

Investigating the failed rows

```
SELECT DISTINCT ErrorCode, ErrorFieldName FROM Error_Table_1;
```

Investigating the duplicate index violations

```
SELECT * FROM Error_Table_2;
```

CHECKPOINT Option

The CHECKPOINT option defines points in a job where FastLoad pauses to record that Teradata has processed a specified number of input records. When you use checkpoints, you do not have to rerun the entire FastLoad job if it stops before completion. FastLoad will use the checkpoint information in the restart log table to determine the restart location.

If you are going to use the CHECKPOINT option, the Reference Manual recommendation is:

For smaller Teradata Database systems,

If records < 4K, then use CHECKPOINT 100,000
If records ≥ 4K, then use CHECKPOINT 50,000

For larger Teradata Database systems, increase the CHECKPOINT value.

Because checkpoints slow down FastLoad processing, it is also recommended to set the CHECKPOINT value to effectively take a checkpoint every 10 to 15 minutes. Frequently, this means setting the CHECKPOINT value to a much larger value.

CHECKPOINT Option

BEGIN LOADING . . .
CHECKPOINT *integer*;

- **Used to verify that rows have been transmitted and processed.**
- **Specifies the number of rows transmitted before pausing to take a checkpoint and verify receipt by AMPs.**
- **If the CHECKPOINT parameter is not specified, FastLoad takes checkpoints as follows:**
  - **Beginning of Phase 1**
  - **Every 100,000 input records**
  - **End of Phase 1**
- **FastLoad can be restarted from previous checkpoint.**
- **Performance Note: Checkpoints slow down FastLoad processing – set the CHECKPOINT large enough that checkpoints are taken every 10 to 15 minutes. Usually, this requires a CHECKPOINT value much larger than 100,000.**

END LOADING Statement

The End Loading statement signifies to the loader that all data has been acquired. At this time the loader can wrap up Phase One and get started with Phase Two.

With a NoPI table, FastLoad will start the END LOADING phase, but complete it very quickly because there is no data to sort.

While a NoPI target table is being loaded with FastLoad, users can view the table content with an ACCESS lock.

This is possible because rows are always appended at the end of a NoPI table. This is not allowed on a PI target table until the data has been sorted which does not happen until the end of Phase 2.

Example: A NoPI table was loaded without including the END LOADING statement.

SELECT * FROM Orders_nopi;

SELECT Failed. 2652: Operation not allowed: DS.Orders_nopi is being Loaded.

LOCKING ROW FOR ACCESS SELECT * FROM Orders_nopi;

Orderid	Custid	Orderstatus	Amount	Date	O_Priority	O_Clerk
100002	1,002	C	1,010.00	2009-01-02	10	Jack Snow
100019	1,019	C	1,095.00	2009-01-07	10	Jack Snow
100021	1,021	C	1,105.00	2009-01-07	10	Jack Snow

END LOADING Statement

END LOADING ;

PI Table

- Indicates that all data rows have been transmitted.
- Begins Phase 2 or Sort Phase processing.
- Omission implies:
  - The load is incomplete and will be restarted later.
  - This causes the table that is being loaded to become "FastLoad paused."
  - If you attempt to access a table (via SQL) that is in a "FastLoad paused" state, you will get the following error.

    Error #2652 Operation Not Allowed *tablename* is being loaded

NoPI Table

- There is no Sort in Phase 2 for a NoPI table. Phase 2 is very fast for a NoPI table.
- While a NoPI target table is being loaded, users can view the table with an ACCESS lock. This is possible because rows are always appended at the end of a NoPI table.
- The following SELECT will succeed for a NoPI table.

```
LOCKING ROW FOR ACCESS SELECT * FROM Orders_nopi;
```

RECORD Statement

Use the RECORD statement to override any default positioning assumed by the loader.

You can specify the record of the input file to begin with, and optionally, the record to end with.

The RECORD statement is a separate statement that follows the BEGIN LOADING statement and is specified before the DEFINE statement.

RECORD Statement

RECORD [*integer*] [*THRU integer*];

- If you do not use a RECORD command, FastLoad reads from the first record in the data source to the last record (or from the last CHECKPOINT).
- RECORD allows control over which input records are to be brought in for loading.
- RECORD is a separate statement used before the DEFINE statement.

Examples:

```
RECORD 1 THRU 1000;
```

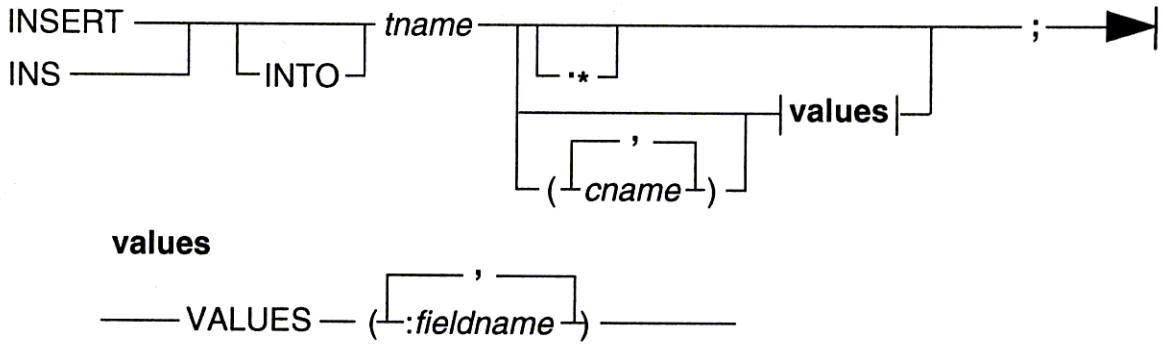
1st through the 1,000th record

```
RECORD 2;
```

2nd through the last record; ignore first record

INSERT Statement

INSERT is a Teradata SQL statement that inserts data records into the rows of the FastLoad table.



During the insert operation, field values are inserted in the table in the order in which the columns are listed in the CREATE TABLE statement. If field values in the input data are stored in the same order as columns are defined in the CREATE TABLE statement for the FastLoad table, you do not need to specify a list of column names in the INSERT statement.

When you use the wildcard format of the INSERT statement, a list of field names is constructed from the definition of the table. During the insert operation, field names and their data types are taken from the CREATE TABLE statement and used to define the table.

The field name definitions are established in the order in which columns are defined in the CREATE TABLE statement. So, the fields in each data record must be in the same order as the columns in the definition of the table.

When using the second form of the INSERT statement, you still need to use the DEFINE command to specify the name of the input data source or INMOD routine used in the FastLoad job.

If you enter a DEFINE command that defines one or more fields before the INSERT statement, the FastLoad utility appends the field definitions to the definitions constructed from the INSERT statement.

The command example on the following page defines the "**datafile5**" input data source and fields in each record (Account_Number, Number, Street, City, State, and Zip_Code).

INSERT Statement

```
DEFINE          Account_Number      (INTEGER)
               ,Street_Number       (INTEGER)
               ,Street              (CHAR(25))
               ,City                (CHAR(25))
               ,State               (CHAR(2))
               ,Zip_Code            (INTEGER)
FILE = datafile5;
INSERT INTO Accounts
               (Account_Number
               ,Street_Number
               ,Street
               ,City
               ,State
               ,Zip_Code)
VALUES
               (:Account_Number
               ,:Street_Number
               ,:Street
               ,:City
               ,:State
               ,:Zip_Code);
```

The "wildcard" format may be used to construct the names in the INSERT statement. The field names are constructed from the DD/D table definition.

```
DEFINE FILE = datafile5 ;
INSERT INTO Accounts.* ;
```

Staged Loading of Multiple Data Files

The example on the facing page illustrates using FastLoad to load two different data sets into a single table.

You might be tempted to create a single script/job that attempts to load the Customer table with starting LOGON and LOGOFF statements. This will not work. When Fastload (as a batch utility) encounters a LOGOFF statement, it completes and never attempts to do the second LOGON statement.

Note that the DEFINE statements for different data files may be different in the two scripts. However, the BEGIN LOADING statement must identify the same table name and the same error table names.

Staged Loading of Multiple Data Files

load_US.fld **fastload < load_US.fld**

```
LOGON tdpid/username,password;
BEGIN LOADING Customer
    ERRORFILES Cust_e1, Cust_e2;

DEFINE   in_CustNum      (INTEGER)
        ,in_Lname        (CHAR(20))
        ,in_Fname        (CHAR(15))
        ,in_Postalcode   (CHAR(10))
FILE=US.dat;

INSERT INTO Customer VALUES (
 :in_CustNum
,:in_Lname
,:in_Fname
,:in_Postalcode);


LOGOFF;
```

No END LOADING statement. Table is in "FastLoad Paused" state.

load_Int.fld **fastload < load_Int.fld**

```
LOGON tdpid/username,password;
BEGIN LOADING Customer
    ERRORFILES Cust_e1, Cust_e2;

DEFINE   in_CustNum      (INTEGER)
        ,in_Lname        (CHAR(30))
        ,in_Fname        (CHAR(25))
        ,in_Postalcode   (CHAR(10))
FILE=International.dat;

INSERT INTO Customer VALUES (
 :in_CustNum
,:in_Lname
,:in_Fname
,:in_Postalcode);

END LOADING;
LOGOFF;
```

END LOADING; indicates no more data and to start Phase 2.

FastLoad Fails to Complete

The output on the facing page provides an example of a FastLoad job that ran out of space in the database where the table was being loaded.

You can also see the list of each 100,000 rows as they are encountered.

The FastLoad job is shown below:

```
LOGON tdt5b/tfact01,tfact01;
BEGIN LOADING DS.Sales ERRORFILES DS.sales_1, DS.sales_2  INDICATORS;
DEFINE FILE=sales.dat;
INSERT INTO DS.Sales.*;
END LOADING;
LOGOFF;
```

The table definition is:

```
CREATE SET TABLE DS.Sales, NO FALLBACK,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL
    (
     store_id INTEGER NOT NULL,
     item_id INTEGER NOT NULL,
     sales_date DATE FORMAT 'YYYY-MM-DD',
     total_revenue DECIMAL(9,2),
     total_sold INTEGER,
     note VARCHAR(256) CHARACTER SET LATIN NOT CASESPECIFIC)
UNIQUE PRIMARY INDEX ( store_id, item_id, sales_date );
```

FastLoad Fails to Complete

```
0001 LOGON tdt5b/tfact01, ;
          :
**** 09:42:20 Number of FastLoad sessions connected = 26
**** 09:42:20 FDL4808 LOGON successful
          :
0002 BEGIN LOADING DS.Sales ERRORFILES DS.sales_e1, DS.sales_e2 INDICATORS;
          :
0003 DEFINE FILE=sales.dat;
**** 09:42:27 FDL4803 DEFINE statement processed
          :
0004 INSERT INTO DS.Sales.*;
          :
**** 09:42:27 Number of recs/msg: 228
**** 09:42:27 Starting to send to RDBMS with record 1
**** 09:42:28 Starting row 100000
**** 09:42:28 Starting row 200000
**** 09:42:28 RDBMS error 2644: No more room in database DS.
**** 09:42:28 Increase database size and restart FastLoad
**** 09:42:28 Logging off all sessions

**** 09:42:33  Total processor time used = '1.26667 Seconds'
   .       Start:   Tue Feb 14 09:42:17 2012
   .       End :   Tue Feb 14 09:42:33 2012
   .       Highest return code encountered = '12'.
**** 09:42:33 FastLoad Paused
```

Restarting FastLoad (Output)

Prior to restarting the job shown on the facing page, you must give the database more space to accommodate the load.

In the output example, you can see that:

- The restart log and error tables remain.
- The Loader repositions itself at the last checkpoint so that it can pick up loading from that point.
- After completing the restart, error tables are dropped and loading is complete.

Restarting FastLoad (Output)

```
0001 LOGON tdt6-1/tfact01, ;
          :
**** 09:53:17 Number of FastLoad sessions connected = 26
          :
0002 BEGIN LOADING DS.Sales ERRORFILES DS.sales_e1, DS.sales_e2 INDICATORS;
0003 DEFINE FILE=sales.dat;
          :
0004 INSERT INTO DS.Sales.*;
          :
**** 09:53:20 Starting row 200000
**** 09:53:21 Starting row 300000
**** 09:53:22 Starting row 400000
**** 09:53:22 Sending row 483350
**** 09:53:22 Finished sending rows to the RDBMS

0005 END LOADING;

**** 09:53:41 END LOADING COMPLETE

   Total Records Read        = 483350
   Total Error Table 1       = 0  ---- Table has been dropped
   Total Error Table 2       = 0  ---- Table has been dropped
   Total Inserts Applied     = 483350
   Total Duplicate Rows      = 0

  Start:  Tue Feb 14 09:53:24 2012
  End :   Tue Feb 14 09:53:41 2012          Note:  These times apply to Phase 2.

0006 LOGOFF;
```

Restarting FastLoad – Summary

You may occasionally need to restart the FastLoad utility after already having started to load the table.

On the facing page are four scenarios you might encounter that would cause you to restart FastLoad. The scenarios also provide the steps you should take to execute the restart process.

Note on condition 4: If you original script completes successfully and you resubmit the original script with END LOADING; then the same data will be loaded a second time. The duplicate rows will be removed as part of phase 2 (assuming a PI table, not a NoPI table).

Restarting FastLoad – Summary

Condition 1: Abort in Phase 1 – data acquisition incomplete.

Solution: Resubmit the script. FastLoad will begin from record 1 or the first record past the last checkpoint.

Condition 2: Abort occurs in Phase 2 – data acquisition complete.

Solution: Submit only BEGIN and END LOADING statements; restarts Phase 2 only.

Condition 3: Normal end of Phase 1 (paused) – more data to acquire, thus there is no 'END LOADING' statement in script.

Solution: Resubmit the adjusted script with new data file name. FastLoad will be positioned to record 1 of the new input file and will continue loading the new file.

Condition 4: Normal end of Phase 1 (paused) – no more data to acquire, no 'END LOADING' statement was in the script.

Solution: Submit BEGIN and END LOADING statements; restarts Phase 2 only.

Additional FastLoad Commands

Some additional FastLoad commands include:

Use **AXSMOD** to specify an access module (e.g., WebSphere MQ) that provides data to the FastLoad utility on network-attached client systems.

Use SESSIONS *max min* **to specify the number of sessions;** placed before the LOGON.

> ***(max*** **=**maximum number of sessions that will be logged on; the *max* specification must be greater than zero). If you specify a *max* value larger than the number of available AMPs, FastLoad limits the sessions to one per working AMP.
>
> The default maximum is one session for each AMP. Using the asterisk as the *max* specification logs on for the maximum number of sessions — one for each AMP.
>
> ***min*** is optional; the minimum number of sessions required to run the job. The *min* specification must be greater than zero; default is 1.
>
> Using the asterisk as the *min* specification logs on for at least one session, but less than or equal to the max specification.

Use **ERRLIMIT** to control a runaway error condition, such as an incorrect definition of the input data. Specify the maximum number of error records you want to occur before the system issues an ERROR and terminates the load.

Use **TENACITY** to specify the number of hours FastLoad will try to establish a connection. Default is no tenacity. The statement must be placed before LOGON.

Use **SLEEP** to specify the number of minutes FastLoad waits before retrying a logon. Default is 6 minutes. The statement must be placed before LOGON.

Use **DELETE** when you must empty an existing table prior to loading. It must precede the BEGIN LOADING statement. (It must be removed from the script prior to a restart.) Specify the table name and use ALL to indicate that you want all rows deleted. The option requires DELETE access right or privilege.

Use **DROP TABLE** in conjunction with the **CREATE TABLE** command to remove an old table and reestablish it. Use DROP TABLE to remove old error tables. These commands must precede the BEGIN LOADING statement and must be removed before a restart. This option requires DROP access right or privilege.

Use **HELP TABLE** to automatically acquire a DEFINE statement when the input data is an exact map of the table that you are loading. The **HELP TABLE** uses the current Data Dictionary definitions to generate the DEFINE.

Use **DATEFORM** to specify the form of the DATE data type for the job. This option is placed before LOGON.

Additional FastLoad Commands

AXSMOD	*name* ["*init-string*"] ;	
SESSIONS	*max [min]* ;	
ERRLIMIT	*max rejected records* ;	
TENACITY	*hours* ;	(default is no TENACITY)
SLEEP	*minutes* ;	(default is 6 minutes)
DELETE FROM	*tablename* [ALL] ;	
DROP TABLE	*tablename* ;	
HELP TABLE	*tablename* ;	
NOTIFY	*OFF* | *LOW* | *MEDIUM* | *HIGH* . . . ;	
DATEFORM	*INTEGERDATE* | *ANSIDATE* ;	
SET SESSION CHARSET	"*charsetname*" ;	
SET RECORD	*FORMATTED*, *BINARY, TEXT, UNFORMATTED or VARTEXT "c"* ;	

Specifies the record format of the import file – existence of record header and/or record trailer.

FORMATTED	– includes both the record length indicator (LI) and an EOR indicator
BINARY	– includes a record length indicator and no EOR indicator
TEXT	– no record length indicator and an EOR indicator
UNFORMATTED	– no record length indicator and no EOR indicator
VARTEXT	– no record length indicator and an EOR indicator; fields are in CSV format

Import Record Layout

LI	Indicator Bytes	Data format is dependent on how the data was exported.	EOR
2	0 – n		x'0A'

If the imported data has indicator bytes, include the INDICATORS option on BEGIN LOADING.

FastLoad with Additional Options

For the FastLoad job on the facing page, the script uses the redirect process to acquire the input script (<) and direct the printed output of the FastLoad job (>).

The example also contains several parameters (SESSIONS, TENACITY, and SLEEP) that must be specified before the LOGON statement. Another option, DATEFORM, if used, must be placed before the LOGON statement.

The ERRLIMIT parameter may be specified before or after the LOGON statement.

FastLoad with Additional Options

```
fastload < job1.fld | tee job1.out
```

```
SESSIONS 4;
TENACITY 4;
SLEEP 3;
LOGON educ2/bank,bkpasswd;
ERRLIMIT 1000;
BEGIN LOADING Customer
   ERRORFILES Cust_e1, Cust_e2;
DEFINE      in_CustNum      (INTEGER)
           ,in_SocSec       (INTEGER)
           ,Filler          (CHAR(40))
           ,in_Lname        (CHAR(30))
           ,in_Fname        (CHAR(20))
   FILE = datafile6
INSERT INTO CUSTOMER VALUES (
            :in_CustNum
           ,:in_Lname
           ,:in_Fname
           ,:in_SocSec);
END LOADING;
LOGOFF;
```

→ Input script file name & output file (report) name.

→ These options must be specified before LOGON.

→ Maximum number of error records before terminating.

↘ Start Phase 1.

→ Start Phase 2. If omitted, FastLoad will pause.

Invoking FastLoad

The facing page displays the commands you can use to invoke the FastLoad utility in batch mode. The parameters for each command are listed in the three-column table.

Invoking FastLoad

Network Attached Systems:	**fastload [PARAMETERS] < *scriptname* >*outfilename***
Channel-Attached MVS Systems:	**// EXEC TDSFAST FDLOPT= [PARAMETERS]**
Channel-Attached VM Systems:	**EXEC FAST [PARAMETERS]**

Channel Parameter	Network Parameter	Description
BUFSIZE=kb	-b *kb*	Specifies input buffer size; maximum is 63 KB (default)
CHARSET=*charsetname*	-c *charsetname*	Specify a character set or its code. Examples are EBCDIC, ASCII, or Kanji sets
ERRLOG=*filename*	-e *filename*	Alternate file specification for error messages; produces a duplicate record.
INMODETYPE=SAS_C	N/A	Specifies that the job will use an INMOD routine written in SAS/C.
SLEEP=m*inutes*	-s *minutes*	Number of minutes that FastLoad pauses before retrying a logon.
TENACITY=*hours*	-t *hours*	Number of hours that FastLoad will continue trying to logon when the maximum number of load jobs are already running.
	< *scriptname*	Name of file that contains FastLoad commands and SQL statements.
	> *outfilename*	Name of output file for FastLoad messages.

Application Utility Checklist

The facing page adds the FastLoad capabilities to the checklist.

Whether or not a Teradata FastLoad job restarts automatically because of a Teradata Database failure depends on the operational configuration of the Teradata Database when it returns to service. While Teradata is not operational, the FastLoad job is paused.

- If the configuration of the restarted Teradata Database is *exactly* the same as it was when you invoked Teradata FastLoad, then Teradata FastLoad restarts the job automatically.
  - If the Teradata FastLoad job was paused in the end loading phase, the Teradata Database resumes processing at the same place it was stopped.
  - If the Teradata FastLoad job was paused in the loading phase, the Teradata Database resumes processing:
    - At the last checkpoint if the BEGIN LOADING command specified the checkpoint option.
    - At the beginning if the BEGIN LOADING command did not specify the checkpoint option.
- If the configuration of the restarted Teradata Database is different from the way it was when you invoked Teradata FastLoad (ex., an AMP is down), then Teradata FastLoad does not restart the job. In this case, to restart and continue with the paused Teradata FastLoad job, you must reestablish the original configuration of the Teradata Database.

Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions	ALL	LIMITED			
DML Functions	ALL	INSERT			
Multiple DML	Yes	No			
Multiple Tables	Yes	No			
Protocol Used	SQL	FASTLOAD			
Conditional APPLY	No	No			
Data Conversion	Yes	1 per column			
Error Capture	No	Yes			
Error Limits	No	Yes			
User-written Routines	No	Yes			
Automatic Restart	No	Yes*			
Max Load Limit	No	Yes			
Support Environment (SE)	No	No			

Summary

The facing page summarizes some important concepts regarding the FastLoad utility.

Summary

FastLoad Features and Characteristics:

- Excellent utility for loading new or empty tables from a host or server.
- The empty table cannot have secondary indexes, join indexes, hash indexes, or Referential Integrity.
- Can reload previously emptied tables
  - Remove referential integrity or secondary indexes prior to using FastLoad.
- Full Restart capability
- Has two phases – creates an error table for each phase.
  - Error Limits and Error Tables, accessible using SQL

Module 34: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 34: Review Questions

Match the item in the first column to a corresponding statement in the second column.

1. ___ Phase 1	a. Might be used if a zero date causes an error
2. ___ CHECKPOINT	b. Table status required for loading with FastLoad
3. ___ ERRORTABLE1	c. Records written in unsorted blocks
4. ___ ERRORTABLE2	d. Records rows with duplicate values for UPI
5. ___ Empty Table	e. Not permitted on table to be loaded with FastLoad
6. ___ Secondary Index	f. Points FastLoad to a record in an input file
7. ___ Conversion	g. Can be used to restart loading from a given point
8. ___ NULLIF	h. Records constraint violations
9. ___ RECORD	i. Builds the actual table blocks for the new table
10. ___ Phase 2	j. Transform one data type to another, once per column

Lab Exercise 34-1

The BTEQ syntax to create the two data files for this exercise is:

```
.LOGON      …;

.EXPORT DATA FILE = data34_1, CLOSE;
 EXEC AP.Lab34_1_1;
.EXPORT RESET

.EXPORT DATA FILE = data34_2, CLOSE;
 EXEC AP.Lab34_1_2;
.EXPORT RESET

.LOGOFF;
```

These macros have WHERE clauses that SELECT specific data rows. These macros limit the number of rows that are selected; therefore there is no need to include the LIMIT parameter with the BTEQ .EXPORT statement.

After creating these data files, you may want to check their size to ensure that you have created them correctly. An easy way to do this is to use the Linux **ls –l** command.

The size of **data34_1** should be 244,000 bytes.
The size of **data34_2** should be 183,000 bytes.

A technique that can be used to create Linux scripts without using vi or vim is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where Linux is running and …

3. **cat > lab34_12.fld** (or whatever filename you wish)

   Use the mouse to choose the **Edit ➔ Paste** function
   To exit the cat command, press either the CNTL C or DELETE key.

Lab Exercise 34-1

Lab Exercise 34-1

Purpose

In this lab, you will set up a restartable FastLoad operation.

What you need

You need to create two data input sets and use your empty Customer table. The input data sets will get their data from the AP.Customer table.

Tasks

1. Using two separate BTEQ EXPORT commands, create two source data sets, data34_1 and data34_2. The SQL for selecting the appropriate rows is contained in the macros AP.LAB34_1_1 (for data34_1) and AP.LAB34_1_2 (for data34_2).

   Note: data34_1 has 4000 records and data34_2 has 3000 records

2. Create a FastLoad script that loads the first 4000 records (data34_1 file) into your Customer table and do not include the END LOADING statement in this script.

3. Create a FastLoad script that loads the additional 3000 records (data34_2) into your Customer table and complete the FastLoad.

4. Check the result. (Your Customer table should contain 7000 rows.)

Lab Exercise 34-2

The BTEQ syntax to create the data file for this exercise is:

```
.LOGON      …;
.EXPORT DATA FILE = data34_3, CLOSE;
 EXEC AP.Lab34_2;
.EXPORT RESET
.LOGOFF;
```

After creating this data file, you may want to check its size to ensure that you have created it correctly. An easy way to check the size in Linux is to use the Linux **ls –l** command.

The size of **data34_3** should be 495,000 bytes.

Note: FastLoad requires that DECIMAL be spelled out (DEC causes a syntax error).

A technique that can be used to create Linux scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where Linux is running and …

3. **cat > lab34_22.fld** (or whatever filename you wish)

   Use the mouse to choose the **Edit ➔ Paste** function
   To exit the cat command, press either the CNTL C or DELETE key.

Lab Exercise 34-2

Lab Exercise 34-2

Purpose

In this lab, you will create a FastLoad script to load data into a TRANS table and convert incoming dates with a value of '0000-00-00' and converts these dates to NULL.

What you need

An empty TRANS table and the macro AP.Lab34_2.

Tasks

1. Use BTEQ EXPORT and the macro AP.Lab34_2 to create a source data file (data34_3). This macro outputs the DATE in character format and the year is output as 4 characters.

2. FastLoad the data from the file data34_3 to your empty TRANS table. In this exercise, the default format for a date is character (10) with a format of YYYY-MM-DD. The data file has dates set to 0 (zero) that must be converted to NULL.

   (Hint: Use a FORMAT 'YYYY-MM-DD' on the INSERT and a NULLIF='0000-00-00' on the DEFINE. You must define incoming DATE as a character field.)

3. How many rows in your table have a NULL Trans_Date? _____

Notes

Module 35

TERADATA
Raising Intelligence

The Support Environment for FastExport, MultiLoad, and TPump

After completing this module, you will be able to:

- **Explain the elements of the Support Environment.**
- **Use the Support Environment to invoke a utility.**
- **Process parameter input from a host file.**
- **Use system and user-defined variables.**
- **Write messages to an output file.**

Notes

Table of Contents

Support Environment 35-4
Setting Up the Support Environment 35-6
Invoking Utilities 35-8
Support Environment Commands 35-10
Initializing the Log Table 35-12
Initialization and Wrap Up Commands 35-14
User-defined and System Variables 35-16
.ACCEPT – Environment or File Variable 35-18
.DISPLAY and .ROUTE Commands 35-20
Example: Using Variables in a Script 35-22
Working with Control Logic 35-24
Support Environment Example – Input 35-26
Support Environment Example – Output 35-28
Teradata SQL Support 35-30
Script – Example Input 35-32
Script – Example Output 35-34
Summary 35-36
Module 35: Review Questions 35-38
Lab Exercise 35-1 (optional) 35-40

Support Environment

To ensure consistency for application utilities, Teradata provides the Support Environment, a sophisticated utility platform that makes *fully automatic* restarts available. Without needing to know the reason for the failure and without needing to change the script, you can restart a job by resubmitting the script.

The Support Environment supports a complete range of SQL commands (except SELECT), and permits conditional processing of SQL and utility commands with an easy-to-use .IF, .THEN, .ELSE, and .ENDIF facility.

Support Environment

- **Provides a common environment – language, functions, flexibility, etc. – for utilities such as FastExport, MultiLoad, and TPump.**
- **Provides a fully-nested .RUN file facility.**
- **Interprets utility commands and provides error reporting.**
- **Supports system-defined and user-defined variables.**
- **Allows for conditional processing of commands.**
- **Supports a wide range of DDL and DML commands.**
- **Allows logic to be applied both before and after the utility executes.**
- **Provides recovery management from a Teradata or host failure.**

Setting Up the Support Environment

Utilities are not called by the user directly, but are invoked by the Support Environment after initial housekeeping tasks are complete.

The setup process requires the naming of a restart log table, which then governs the operation completely.

You can follow the setup commands with SQL statements for preparing the job.

Setting Up the Support Environment

- **Utilities are invoked by the Support Environment after the preparatory statements have been executed:**

```
.LOGTABLE   logtable_name;
.LOGON      tdpid/username,password;
```

- **These may be followed by statements to:**
  - **Statements to SET variables**
  - **Accept variables from outside sources (a file or the operating system)**
  - **SQL statements to CREATE and DROP tables, secondary indexes, etc.**
  - **SQL statements to set the default database, INSERT into tables, etc.**
  - **Commands to invoke the utility function (e.g., .BEGIN EXPORT)**
- **Comments may be included with scripts.**

```
/* This job is used to ….. */
```

Invoking Utilities

The facing page identifies the commands you can use to invoke the utilities that use the Support Environment.

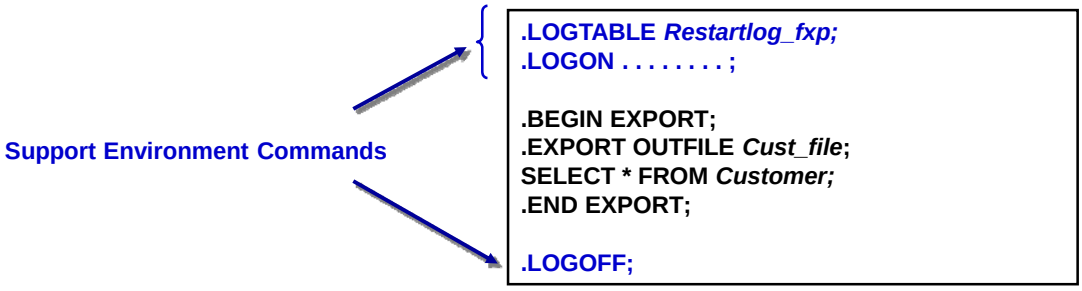
If from its interrogation of the restart log, the Support Environment determines the present operation to be a restart, the Support Environment accepts responsibility for not submitting a previously successful operation a second time.

The facing page also shows an example of job code that calls the Support Environment.

Note: The script on the facing page runs the FastExport utility. FastExport is discussed in more detail in a later module.

Invoking Utilities

- FastExport is invoked with: .BEGIN EXPORT
- MultiLoad is invoked with: .BEGIN [IMPORT] MLOAD
  or
  .BEGIN DELETE MLOAD
- TPump is invoked with: .BEGIN LOAD
- Upon initialization, the Support Environment accesses the Restart Log Table to determine whether or not this is a restart.
- If it is, the Support Environment will not submit a previously successful statement a second time.



Support Environment Commands

Note that *all* Support Environment and utility commands are preceded with a period (.). Any statement NOT preceded by a period is presumed to be SQL and is sent to the Teradata database for processing.

Support Environment Commands

.LOGTABLE	Acquires or creates the Restart Log Table.
.LOGON	Connects multiple sessions to Teradata.
.LOGOFF	Terminates the utility operation.
.DATEFORM	Specify INTEGERDATE or ANSIDATE.
.ACCEPT	Input parameters to Support Environment.
.RUN	Specifies an external script file.
.IF … THEN [.ELSE]	Identifies statements to be executed if certain conditions are true or false
.ENDIF	Required to terminate a .IF condition.
.DISPLAY	Writes messages to a specific destination.
.ROUTE	Specifies output file other than SYSPRINT or standard output.
.SET	Assigns a data type and value to a variable.
.SYSTEM	Submits an operating system command to the client environment.

Initializing the Log Table

Utilities use information in the Restart Log Table to restart jobs halted because of a Teradata or client system failure.

If a utility completes with a return code of zero, the Restart Log Table is automatically dropped.

Initializing the Log Table

- **The .LOGTABLE command is required.**
- **.LOGTABLE and .LOGON commands must be the first statements processed (either directly or via the .RUN command).**
- **.LOGTABLE creates a new table or acquires an existing Log Table.**
- **Privileges required for the Log Table:**
  - **CREATE TABLE (to create a new table)**
  - **INSERT**
  - **UPDATE**
  - **SELECT**
- **By default, the log table is created in your default database which requires PERM space.**
- **The format of the Log Table is unique to each of the utilities (FastExport, MultiLoad, and TPump).**

Initialization and Wrap Up Commands

The facing page displays initialization and wrap up commands for the Support Environment.

Utilities deliver to the Host a “high watermark” return code. If this value is zero, all work tables, empty error tables, and the restart log table are dropped. Any return code of 8 or greater indicates an aborted job.

Initialization and Wrap Up Commands

.LOGON [*tdpid* /] *username* , [*password* [,'*acctid*']] ;

Utility will run any startup string defined for the user.

.LOGOFF [*retcode*] ;

Permits user to specify return-code; otherwise, high watermark is returned.

.RUN FILE *fileid*

[IGNORE
{ *charpos1* | *charpos* 1 THRU | THRU *charpos2* | *charpos1* THRU *charpos2* }] ;

Identifies external source of control statements and optionally ignores extraneous data - nest up to 16 levels of .RUN.

When the Utility terminates, it returns to the HOST with a "high watermark" return code:

00	Successful completion
04	Warning
08	User error
12	Severe internal error
16	No output message available

User-defined and System Variables

The facing page displays some of the variables you can use within the Support Environment. See the reference manuals for the complete set of variables.

The .SET command has to precede the .BEGIN EXPORT (MLOAD or LOAD) command.

Note: The Return Code is the return code from the last Teradata Database command.

Miscellaneous notes on Time and Date variables:

- &SYSDATE – returns 8-character date in *yy/mm/dd f*ormat
- &SYSDATE4 – returns 10-character date in *yyyy/mm/dd* format
- &SYSDAY – returns 3-character uppercase day of week specification: MON, TUE, WED, THU, FRI, SAT or SUN
- &SYSTIME – returns 8-character time in *hh:mm:ss* format

For these 4 variables, the original values are maintained after a utility restart operation.

Note that because the values are all character data types, you should not reference them in numeric operations.

User-Defined and System Variables

.SET *var* [TO] *expression*; ***Permits a variable to be set or reset to an expression or a pre-existing variable.***

Example: **.SET dbase TO 'student130';** **Reference this variable with &dbase.**
.SET tname TO 'customer'; **Reference this variable with &tname.**

System Variables	*Description*	*Format*	*Example*
&SYSDATE	System Date	YY/MM/DD	12/04/17
&SYSDATE4	System Date	YYYY/MM/DD	2012/04/17
&SYSTIME	System Time	HH:MI:SS	09:11:53
&SYSDAY	Day of Week	X(3)	TUE
&SYSOS	Client Operating System	X(5)	UNIX, Win32, Linux
&SYSUSER	Client System User Id		student130
&SYSRC	Return Code		0

MultiLoad Specific Variables (not complete list)

&SYSDELCNT_	Delete Count	Ex., &SYSDELCNT1, &SYSDELCNT2, ...
&SYSINSCNT_	Insert Count	Ex., &SYSINSCNT1, &SYSINSCNT2, ...
&SYSUPDCNT_	Update Count	
&SYSETCNT_	Error Table Count	
&SYSUVCNT_	Uniqueness Violation Count	
&SYSRCDCNT_	Count of import records read	
&SYSRJCTCNT_	Count of records rejected from import file	Note: n is 1 to 5

.ACCEPT – Environment or File Variable

The ACCEPT command can...

- Accept from a single data record from an external source, and use it to set one or more utility variables.

- Accept from an operating system variable and use it to set a utility variable.

You can treat input values for the Support Environment in whole or in part. The IGNORE function of the .ACCEPT statement permits the ACCEPTed data to contain filler data. For example, you can also use the IGNORE to ignore sequence numbers in the first 6 columns.

When ACCEPTing from a **fileid**, the **fileid** can be any of the following:

- with VM, a FILEDEF name
- with MVS, a DDNAME
- with UNIX and Windows, a pathname for a file
- an * which represents the system console or standard input (*stdin*) device.

Multiple values in the input record are space separated. Character values must be delimited with single quotes. For example,

'Los Angeles' 90210	is valid
Los Angeles 90210	is not valid

If the input file contains multiple records, the ACCEPT command will only accept from the first record.

If the number of variables is greater than the number of values in the input record, then unused variables are undefined or NULL.

If the number of values in the input record is greater than the number of variables, you will receive a warning message.

.ACCEPT – Environment or File Variable

The ACCEPT command can ...

- **accept from a environment variable and use the variable throughout the script/job.**
- **accept from a single data record from an external file, and use it to set one or more utility variables.**

```
.ACCEPT var  [FROM]     ENVIRONMENT VARIABLE env_var ;
                        ENV VAR

.ACCEPT var, . .  [FROM] FILE fileid
              [ IGNORE
                { charpos1                          }
                { charpos1  THRU                    }
                { THRU      charpos2                } ];
                { charpos1  THRU  charpos2          }
```

For display or evaluation, variable names must be preceded with an ampersand (&).

Utility variables will be replaced by their values before text is displayed.

If the value is a character, the variable name must be enclosed in single quotes.

.DISPLAY and .ROUTE Commands

When DISPLAYing to a **fileid**, the **fileid** can be any of the following:

- with VM, a FILEDEF name
- with MVS, a DDNAME
- with UNIX and Windows, a pathname for a file
- an * which represents the system console or standard output (*stdout*) device.

In UNIX, **/dev/tty** references the user's terminal device directly. **/dev/tty** is not the same as standard output.

The DISPLAY command creates a new file or replaces an existing file; it does **not** append to an existing file. However, multiple DISPLAY commands to the same filename in the same script are all placed into the same file.

The ECHO function on the .ROUTE command permits messages to be sent to multiple destinations.

.DISPLAY and .ROUTE Commands

.DISPLAY ' *text* ' [TO] **FILE** *filename*;

Used to write messages to specified filename.

.ROUTE MESSAGES [TO] **FILE** *filename*

([WITH] ECHO [TO] FILE *filename* | [WITH] ECHO [OFF]) ;

Changes routing of default output.

'ECHO' permits routing to default and a second copy anywhere in script.

Examples:

```
.DISPLAY 'Run Date - &SYSDATE4' TO FILE /dev/tty;

   Run Date - 2012/04/17

.ACCEPT home FROM ENV VAR HOME;
.DISPLAY 'The $HOME directory is &home' TO FILE *;

   The $HOME directory is /home/student130
      (* - the output is directed to standard output device.)

.ROUTE MESSAGES TO FILE  /tmp/mldrun1.out  WITH ECHO TO FILE /dev/tty;
```

Example: Using Variables in a Script

The facing page contains an example of using variables in a FastExport Script.

Notes:

- The .SYSTEM command is the Linux remove file command with the –f or force option. The –f option removes the file without prompting the user.
- The script on the facing page runs the FastExport utility. FastExport is discussed in more detail in a later module.
- The two periods (..) between the &dbase and &tname are needed to represent a single period. If a single period was used, the support environment would interpret the text immediately after the single period as a command.
- If an ampersand is needed in the script, use &&.

Example: Using Variables in a Script

Example:

```
.LOGTABLE Custlog_fxp;
.LOGON . . . . . ;

.ACCEPT home FROM ENV VAR HOME;

.SET dbase          TO 'student130';
.SET tname          TO 'customer';
.SET expname        TO '&home/cust_file';

.SYSTEM 'rm -f &expname';

.DISPLAY 'Exporting data file &expname' TO FILE /dev/tty;

.BEGIN EXPORT;

.EXPORT OUTFILE &expname;
 SELECT * FROM &dbase..&tname;
.END EXPORT;

.LOGOFF;
```

Output:

Data file saved in Linux:
/home/student130/cust_file

Output to terminal screen:
Exporting data file /home/student130/cust_file

Portion of FastExport output:

```
0011 SELECT * FROM &dbase..&tname;
**** 16:23:10 UTY2402 Previous statement modified to:
0012 SELECT * FROM student130.customer;
```

Working with Control Logic

The facing page describes the use of .IF, .ELSE, and .ENDIF statements to apply conditional logic to your job.

The *conditional expression* is an expression that can be evaluated as either true or false.

When evaluation of the expression returns a numeric result:

- Zero is interpreted as false
- Nonzero results are interpreted as true

The Support Environment utilities (MultiLoad, FastExport, and TPump) support the nesting of .IF commands up to 100 levels.

Working with Control Logic

.IF *conditional expression* **THEN;** *if condition is true,* *then execute these statement(s)* ;	*.IF is followed by a conditional expression that initiates execution of subsequent commands and statements.*
[.ELSE;] *if condition is false,* *then execute these statement(s)* ;	*.ELSE is followed by commands and statements which execute when the preceding IF command is false.*
.ENDIF;	*.ENDIF delimits the group of commands and statements subject to previous IF or ELSE commands.*

Note:

The Support Environment utilities support the nesting of .IF commands up to 100 levels.

Support Environment Example – Input

The example on the facing page demonstrates a number of the features of the Support Environment, including:

- The .RUN facility
- The .IF/.ENDIF function
- Using system variables
- Displaying messages to an output file
- Initializing MultiLoad (MLOAD)

Note: The Support Environment is case-sensitive for variables and input data.

Support Environment Example – Input

```
.LOGTABLE CustLog_mld;

.RUN FILE /home/ks186001/logon;

.IF '&SYSDAY'  NE 'FRI' THEN;
   .DISPLAY 'This job runs on Friday'
         TO FILE /tmp/display_out;
   .LOGOFF;
.ENDIF;


.BEGIN IMPORT MLOAD
  . . .
```

Create or Acquire Restart Log Table.

Run commands in file logon.

Check Day of Week. Write a message and terminate Job if not 'FRI' (Case-specific).

Invoke utility.

/home/ks186001/logon

```
.LOGON tdt5b/KS186001,amber96;
```

Support Environment Example – Output

The Support Environment performs a *preliminary* syntax check of all *utility* statements prior to calling the utility. It also resolves all variables and writes messages to output files as directed. The resolutions are *not* dynamic. Once a variable has been resolved, it remains resolved across application restarts.

Thus, if **&SYSDAY** has once been resolved to **'FRI',** and the job later aborts, upon restart, &SYSDAY remains resolved to 'FRI' even though the actual day of the week may have changed.

Support Environment Example – Output

MultiLoad Utility Output

Logon / Connection

```
0001        .LOGTABLE CustLog_mld ;
0002        .RUN FILE /home/ks186001/logon;
0003        .LOGON tdt5b/KS186001, ;
17:29:43    FRI APR 13, 2012
UTY6211     A successful connect was made to the DBC.
17:29:44    FRI APR 13, 2012
UTY6211     Logtable 'KS186001.CustLog_mld' has been created.
```

Processing Control Statements

```
0004        .IF '&SYSDAY' NE 'FRI' THEN ;
17:29:44    FRI APR 13, 2012
UTY2402     Previous statement modified to:
0005        .IF 'FRI' NE 'FRI' THEN;
0006        .DISPLAY 'This job runs on Friday'
            TO FILE /tmp/display_out;
0007        .LOGOFF;
0008        .ENDIF;
0009        .BEGIN IMPORT MLOAD
```

Teradata SQL Support

The Support Environment supports a full range of SQL functionality, except for SELECT.

Note:

Specifying any DML statements (INSERT /UPDATE/DELETE) before specifying the utility BEGIN command (e.g., BEGIN MLOAD) will use the non-fast path and processing will be done as normal SQL statements and the Transient Journal will be used as needed. This may be very slow depending on the SQL statement. Specifying the DML statement after the utility BEGIN command (e.g., BEGIN MLOAD) will use the fast path. For example, in MultiLoad, the processing will be done in the utility transaction phase which is very fast.

Teradata SQL Support

- **The Support Environment supports:**
  - **Utility operations.**
  - **DML and DDL functions for preparatory tasks in the same job-step, avoiding multiple utility invocations.**

- **Examples of TERADATA SQL statements that can be used include:**

CREATE DATABASE	ALTER TABLE
MODIFY DATABASE	DROP TABLE, VIEW, MACRO, INDEX
DELETE DATABASE	CREATE TABLE, VIEW, MACRO, INDEX
DROP DATABASE	REPLACE VIEW, MACRO
	RENAME TABLE, VIEW, MACRO
DATABASE	
CHECKPOINT	INSERT (INSERT/SELECT is accepted)
COLLECT STATISTICS	UPDATE
COMMENT ON	DELETE
SET SESSION COLLATION	GRANT
	REVOKE
RELEASE MLOAD	GIVE

Note: User-generated transactions (BT, ET) and SELECT are not supported.

Script – Example Input

Multiple input variables from a file treated by the .ACCEPT command are separated by a space. Text values must be enclosed in single quotes.

The Control_Table has the following columns:

ID	Status
2	Text

Script – Example Input

Host File: parfile1

0	0	0	2		'	T	e	x	t	'

The script is for setting up a MultiLoad operation. The database table is named Control_Table. It has columns named Status and ID.

```
.LOGTABLE rlog2_mld;
.LOGON tdt5b/student130,password;
.ACCEPT num, name FROM FILE parfile1;
.SET nbrin TO 2;
.IF &num = &nbrin THEN;
UPDATE Control_Table SET Status = '&name'   WHERE ID = &num;
.DISPLAY 'Update of record &nbrin successful'
    TO FILE /tmp/display2_out;
.ENDIF;
.LOGOFF;
```

Script – Example Output

Notice how the output shows the resolution of these values before the utility is called.

Script – Example Output

Host File: parfile1

0	0	0	2		'	T	e	x	t	'

MultiLoad Utility

```
                17:45:18  Processing start date FRI APR 13, 2012
                Logon/connection
0001            .LOGTABLE rlog2_mld;
0002            .LOGON tdt5b/student130, ;
UTY6211         A successful connect was made to the DBC
UTY6217         Logtable 'STUDENT130.rlog2_mld' has been created.
                Processing Control Statements
0003            .ACCEPT num, name FROM FILE parfile1;
0004            .SET nbrin TO 2;
0005            .IF &num = &nbrin THEN;
UTY2402         Previous statement modified to:
0006            .IF 2=2 THEN;
0007            UPDATE Control_Table SET Status = '&name' WHERE ID = &num;
UTY2402         Previous statement modified to:
0008            UPDATE Control_Table SET Status = 'Text' WHERE ID = 2;
UTY1016         'UPDATE' request successful
0009            .DISPLAY 'Update of record &nbrin successful' TO FILE /tmp/display2_out;
UTY2402         Previous statement modified to:
0010            .DISPLAY 'Update of record 2 successful' TO FILE /tmp/display2_out;
0011            .ENDIF;
0012            .LOGOFF;
```

Summary

The facing page summarizes some of the important concepts regarding the Support Environment.

Summary

Support Environment:

- Common environment for utilities such as MultiLoad, FastExport, and TPump.
- Provides error reporting.
- Supports a wide range of DDL and DML commands for one-step jobs.
- Allows for conditional processing.
- Supports system- and user-defined variables.
- Provides recovery management from a Teradata or host failure.

Module 35: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 35: Review Questions

Match the item in the first column to its corresponding statement in the second column.

_____ 1. .LOGTABLE	a. Connects sessions to Teradata
_____ 2. .LOGON	b. Uses a single data record to set one or more utility variables
_____ 3. .ACCEPT	c. System variable
_____ 4. UPDATE	d. Identifies the log to create or acquire
_____ 5. &SYSDATE	e. Teradata SQL statement permitted by Support Environment

Lab Exercise 35-1 (optional)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Lab Exercise 35-1 (optional)

Lab Exercise 35-1 (optional)

Purpose

In this lab, you will use the Support Environment to accept data from an input record (data35_1) and insert it into a row in your customer table.

What you need

AP.Customer and your Customer Table.

Tasks

1. Create a file or data set (data35_1) and enter the following data for INSERT into the Customer table:

Customer_Number	10001
Last_Name	'YourLastName'
First_Name	'YourFirstName'
Social_security	333445555

Use the format:

10001 'YourLastName' 'YourFirstName' 333445555 (items separated by spaces)

2. Prepare a Support Environment script that defines the record to the Support Environment, using the ACCEPT to read the record and use the SET command to dynamically modify the table name in your INSERT statement. Use FastExport to execute this script.

3. Test the result: SELECT * FROM Customer WHERE Customer_Number = 10001;

Notes

Module 36

TERADATA
Raising Intelligence

FastExport

After completing this module, you will be able to:

- **State FastExport capabilities.**
- **Describe how sorted output is produced from a multiple-session SELECT.**
- **Prepare a FastExport script.**

Notes

Table of Contents

FastExport 36-4
.BEGIN and .END EXPORT 36-6
SESSIONS *max min* 36-6
TENACITY and SLEEP 36-6
SPOOLMODE 36-6
.END EXPORT 36-6
.EXPORT 36-8
A FastExport Script 36-10
The SELECT Request 36-12
FastExport without Spooling 36-12
Impact of Requesting Sorted Output 36-14
The SORT Procedure 36-16
Multiple Exports in one FastExport Job 36-18
Invoking FastExport 36-20
FastExport and Variable Input 36-22
Selection Controls 36-22
A FastExport Script with ACCEPT 36-24
A FastExport Script with LAYOUT 36-26
Application Utility Checklist 36-28
Summary 36-30
Module 36: Review Questions 36-32
Lab Exercise 36-1 36-34
Lab Exercise 36-2 36-36

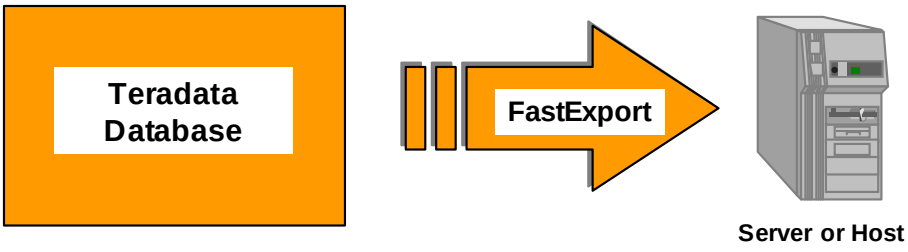
FastExport

FastExport is designed to outperform BTEQ .EXPORT in the transfer of large amounts of data from the larger Teradata database systems to the host using multiple sessions.

FastExport is NOT designed to make the Teradata Database perform faster. It is designed to make greater use of multiple Parsing Engines and AMPs as well as multiple channels.

FastExport

- Exports large volumes of formatted data from Teradata to a host file or user-written application.
- Takes advantage of multiple sessions.
- Export from multiple tables.
- Uses Support Environment.
- Fully automated restart.
- Uses one of the "Loader" slots.
- Teradata 13.10 Feature – FastExport without Spooling improves performance.
  - Data block export begins immediately.
  - The contents of a table are read in one pass and exported while data blocks are being read into memory buffers.
  - SELECT statements cannot have ORDER BY, HAVING, WITH, Joins, SUM, etc.



.BEGIN and .END EXPORT

The BEGIN EXPORT command signifies the beginning of an export task and sets the specifications for the task sessions with the Teradata Database.

SESSIONS *max min*

max is maximum number of FastExport sessions that will be logged on when you enter a LOGON.

- The *max* specification must be greater than zero.
- If you specify a SESSIONS *max* value larger than the number of available AMPs, the utility limits the sessions to one per working AMP.
- The default is 4 for Linux and networked systems.
- Using the asterisk character as the max specification logs on for the maximum number of sessions — one for each AMP.
- ***min*** is optional, the minimum number of sessions required to run the job. The *min* specification must be greater than zero. The default minimum, if you do not use the SESSIONS option or specify a min value, is 1.
- Using the asterisk character as the *min* specification logs on for at least one session, but less than or equal to the max specification.

SESSIONS * * has the effect of not using the SESSIONS parameter at all.

TENACITY and SLEEP

Tenacity specifies the number of hours that the FastExport utility tries to log on to the Teradata Database.

When the FastExport utility tries to log on for a new task, and the Teradata Database indicates that the maximum number of utility import/export sessions are already running, the FastExport utility:

1. Waits for six minutes, by default, or for the amount of time specified by the SLEEP option.
2. Then it tries to log on to the Teradata Database again.
3. The FastExport utility repeats this process (steps 1 and 2) until it has either logged on for the required number of sessions or exceeded the TENACITY *hours* time period.

SPOOLMODE

- SPOOL - Tells FastExport to spool the answer set. This is the default.
- NOSPOOL - Tells FastExport to try to use the NoSpool method. If the NoSpool method is not supported, FastExport issues a warning and then uses the Spool method
- NOSPOOLONLY - Tells FastExport to use the NoSpool method only. If the NoSpool method is not supported, then terminate the job with an error.

.END EXPORT

The **.END EXPORT** command indicates that the Support Environment has completed its syntax check and housekeeping activities and instructs FastExport to send the SELECT(s) to the Teradata database.

.BEGIN and .END EXPORT

```
.BEGIN EXPORT   [ SESSIONS      max [min]
                  TENACITY      hours
                  SLEEP         minutes
                  SPOOLMODE     SPOOL | NOSPOOL | NOSPOOLONLY
                  NOTIFY        OFF | LOW | MEDIUM | HIGH  ...     ];
```

SESSIONS

- Maximum, and optionally, minimum number of sessions to request – defaults to 4 for Linux.
- The utility will log on 2 *additional* SQL sessions: one for the Restart Log and one for the SELECT.

TENACITY and SLEEP

- Tenacity – # of hours FastExport will try to establish a connection to the system; default is 4.
- Sleep – # of minutes that FastExport will wait between logon attempts; default is 6.

SPOOLMODE (13.10)

- Specifies if FastExport should use a spool file or not.

NOTIFY

- Parameter for specifying the notify user exit option
- The FastExport manual specifies in detail which events are associated with each level.

```
.END EXPORT;
```

- Delimits a series of commands that define a single EXPORT action.
- Causes the utility to send the SELECT(s) to the Teradata Database.

.EXPORT

The .EXPORT statement permits the definition of the output data file and optionally an AXSMOD and/or OUTMOD routine. The reference manual contains the details on using the AXSMOD and OUTMOD options.

Only RECORD and INDICATOR output modes are permitted, since FastExport performance relies heavily on large amounts of data being returned (maybe millions of rows) and this is considered unsuitable for generation of reports. Therefore, there is no FIELD mode with FastExport. INDICATOR is the default.

The BLOCKSIZE parameter defaults to 63.5 KB.

The FORMAT option only applies to non-mainframe systems. The options for FORMAT are:

FASTLOAD a two-byte integer, ***n***, followed by ***n*** bytes of data, followed by an end-of-record marker, either X '0A' (FastExport as Linux client) or X '0D0A' (FastExport as Windows client) systems.

BINARY a two-byte integer, n, followed by n bytes of data

TEXT an arbitrary number of bytes followed by an end-of-record marker, either X '0A' (FastExport as Linux client) or X '0D0A' (FastExport as Windows client) systems.

UNFORMAT exported as it is received from CLI without any client modifications.

In summary the FORMAT option has these characteristics.

FORMAT	Length Indicator (2 bytes)	End-of-Record Marker (Hex '0A' or Hex '0A0D')
FastLoad	Y	Y
Binary	Y	N
Text	N	Y
Unformat	N	N

Note: The Length Indicator does not include itself or the Hex '0A' or '0D0A'. Indicator bytes follow the Length Indicator or they are at the start of the record.

MLSCRIPT this option causes FastExport to generate a MultiLoad script that can be used to load the exported data back into Teradata.

.EXPORT

```
.EXPORT   OUTFILE  fileid  [ AXSMOD  name  [ 'init-string' ] ] [ OUTMOD  module_name ]
    [ MODE        RECORD | INDICATOR ]
    [ BLOCKSIZE   integer ]
    [ FORMAT      FASTLOAD | BINARY | TEXT | UNFORMAT ]
    [ OUTLIMIT    record_count ]
    [ MLSCRIPT    fileid ] ;
```

MODE If RECORD, then indicator bytes for NULLs are not included in exported data.
If INDICATOR, then indicator bytes for NULLs are included in exported data.

BLOCKSIZE Defines the maximum block size to be used in returning exported data. Default (and maximum) is 63.5 KB.

FORMAT Record format of the exported file – this option impacts the record header and trailer.

- FASTLOAD – includes both the record length indicator (LI) and an EOR indicator
- BINARY – includes a record length indicator and no EOR indicator
- TEXT – no record length indicator and an EOR indicator
- UNFORMAT – no record length indicator and no EOR indicator

LI	Indicator Bytes	Data (format is totally dependent on the SELECT)	EOR
2	0 – n		x'0A'

OUTLIMIT Defines the maximum number of records to be written to the output host file.

MLSCRIPT FastExport generates a MultiLoad script that can be used later to load the exported data back into a Teradata system.

A FastExport Script

FastExport is called from the Support Environment using the initialization procedure. FastExport requires a Restart Log Table that must be identified with the .LOGTABLE statement.

A FastExport Script

Define Restart Log	.LOGTABLE RestartLog1_fxp;
	.RUN FILE logon;
	.SET City TO 'Los Angeles'; .SET ZipCode TO 90066;
Specify number of sessions	.BEGIN EXPORT;
Destination file	.EXPORT OUTFILE custacct_data;
Via a SELECT, specify the columns and rows to export.	SELECT A.Account_Number , C.Last_Name , C.First_Name , A.Balance_Current FROM Accounts A INNER JOIN Accounts_Customer AC INNER JOIN Customer C ON C.Customer_Number = AC.Customer_Number ON A.Account_Number = AC.Account_Number WHERE A.City = '&City' AND A.Zip_Code = &ZipCode ORDER BY 1;
Send request.	.END EXPORT;
Terminate sessions	.LOGOFF;

The SELECT Request

FIELD MODE (formatted output for reports), is not available with FastExport. The WITH and WITH BY operators, which provide sub-totals and grand totals, are not supported.

FastExport permits multiple statement SELECTs.

FastExport without Spooling

This Teradata 13.10 Feature (FastExport without Spooling) allows table data to be exported without an entire table being read into a spool file.

FastExport without Spooling improves performance as follows:

- Data block export begins immediately.
- The contents of a table are read in one pass and exported while data blocks are being read into memory buffers.

The FastExport command .begin export SPOOLMODE NOSPOOL exports the data without spooling for SELECT statements with the following:

- A single retrieve or sampling step (simple statements)
- String functions
- Arithmetic operators
- CASE expressions
- Exports from PPI tables

Spooling is required when:

- The statement contains ORDER BY, HAVING, or WITH clauses
- There is a JOIN, SUM, or statistics step
- The SELECT statement has multiple retrieve steps
- The SELECT statement has both a retrieve and a sampling step

If you use the NOSPOOL option and the SELECT requires spooling, FastExport performs *with* spool instead of returning an error message.

The SELECT Request

- Defines the data to be exported to the host, server, or client workstation.
- The job may consist of multiple SELECT statements.
- Applies normal transaction locks (READ lock) which are fully automatic.
  - These locks are normally held by the utility until all response rows have been moved to AMP spool, and then are released.
  - Supports the "LOCKING ROW (or TABLE *tablename*) FOR ACCESS" modifier to request an "access lock".
- FastExport Restrictions – you cannot use SELECT (in FastExport) with the following:
  - Non-data tables (e.g. CURRENT_DATE, ...)
  - Equality condition for a Primary Index or USI
  - WITH option to generate total or subtotal response rows.
  - The USING modifier to submit data parameters as a constraint to the SELECT.
- Teradata 13.10 Feature – FastExport without Spooling. Data blocks can be exported directly, avoids use of spool. The SELECT restrictions with NOSPOOL are:
  - The statement contains ORDER BY, HAVING, or WITH clauses
  - There is a JOIN, SUM, or statistics step
  - The SELECT statement has multiple retrieve steps
  - The SELECT statement has both a retrieve and a sampling step

Impact of Requesting Sorted Output

The facing page describes the sort process.

Sorting exported data adds additional overhead to the FastExport job. Only sort the exported data rows if it is necessary.

Impact of Requesting Sorted Output

A special FastExport "sort protocol" is used to take advantage of multiple sessions. Each session transfers data a block at a time from multiple AMPs.

This protocol includes the following steps:

- **The SELECT request is fully processed in the normal way using DBC/SQL protocol.**
- **At this point, response data is maintained in spool, sorted locally by the AMPs.**
- **Two further distributions between the AMPs (using the BYNET) are required to complete the sort.**

Sort notes:

- **Requesting sorted data adds additional work (overhead and time) to Teradata.**
- **If the exported rows are to be loaded back into a Teradata DB (e.g., MultiLoad), there probably is no need to sort the exported rows.**

The SORT Procedure

Response rows are initially placed in the AMP local spool and sorted. They are then redistributed over the BYNET in such a way that all values from the first sort value are placed on the first logical AMP (which is randomly selected); values from the second sort value are placed on the next physical AMP and so on, round-robin until all data is sorted.

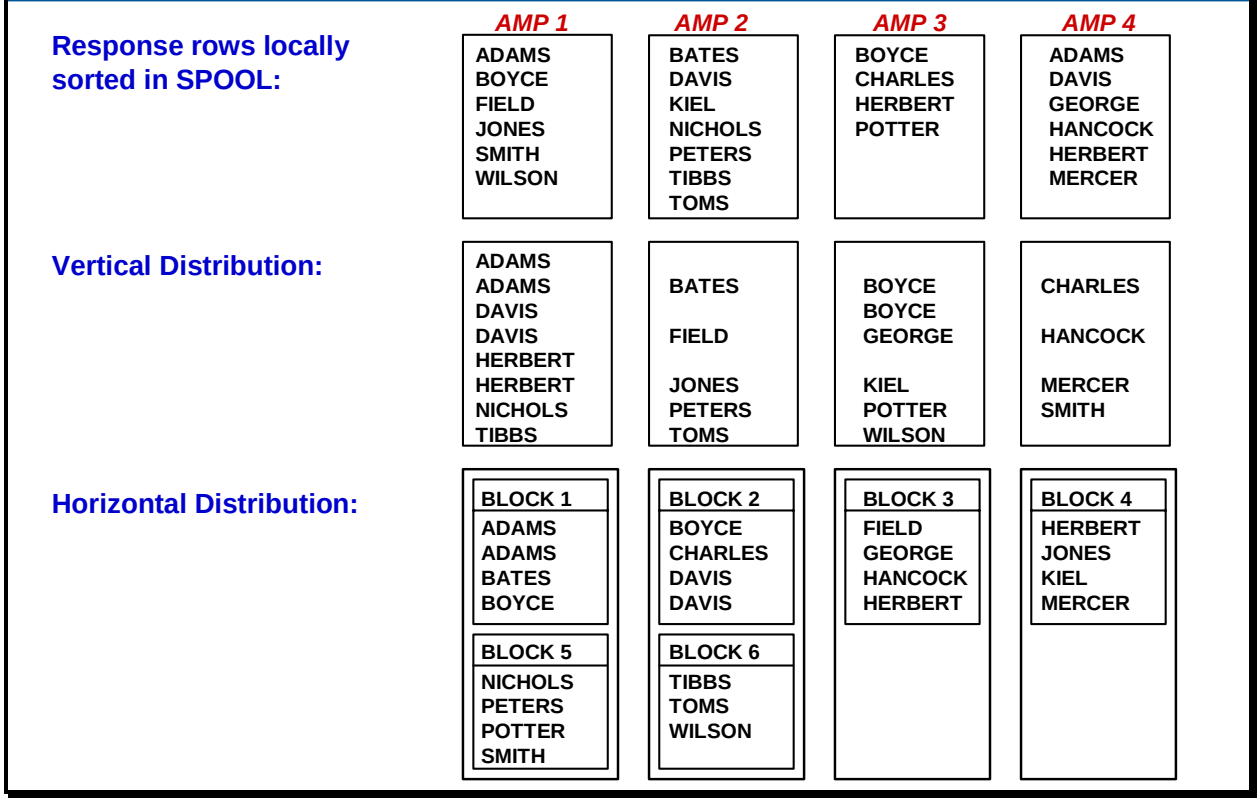
This process is known as the VERTICAL distribution.

HORIZONTAL distribution takes place as blocks of data are built taking all values for the first sort value from the first AMP, all values for the second sort value from the next AMP and so on, round-robin until the block is full.

Multiple sessions are then used to return the sorted blocks in sequence to the host.

Unlike a normal data sort, this procedure is comparatively resource-intensive. It is counter-balanced by the improved performance possible, with multiple sessions and block transfer to the host.

The SORT Procedure



Multiple Exports in one FastExport Job

A FastExport job can contain multiple .BEGIN EXPORT; and .END EXPORT; pairs as shown on the facing page.

Notes:

If the script was modified as:

```
.BEGIN EXPORT SESSIONS 8 4 SPOOLMODE NOSPOOL;
.EXPORT OUTFILE Cust_file;
```

Then the following is true:

- 8 Sessions (in Linux) would be used for the first BEGIN EXPORT if 8 sessions were available.
- A spool file is used because there were two SELECTs from two tables.
- The second .BEGIN EXPORT will use 4 sessions, the Linux default.

If the script was modified as:

```
.BEGIN EXPORT SESSIONS 8 4 SPOOLMODE NOSPOOLONLY;
.EXPORT OUTFILE Cust_file;
```

Then the following is true:

- 8 Sessions (in Linux) would be used if 8 sessions were available.
- The script will abort with the first BEGIN EXPORT. The second BEGIN EXPORT is not executed.

Multiple Exports in one FastExport Job

cust_trans.fxp

```
.LOGTABLE RestartLog2_fxp ;
.LOGON . . . . . ;
.DISPLAY 'Exporting Cust_file - &SYSDATE4' TO FILE /dev/tty;
.BEGIN EXPORT;
.EXPORT OUTFILE Cust_file;
  SELECT * FROM Customer_1;
  SELECT * FROM Customer_2;
.END EXPORT;
.DISPLAY 'Exporting Trans_file - &SYSDATE4' TO FILE /dev/tty;
.BEGIN EXPORT;
.EXPORT OUTFILE Trans_file;
  SELECT * FROM Transactions;
.END EXPORT;
.LOGOFF ;
```

To execute: fexp < cust_trans.fxp > cust_trans.out

Exported data file: Cust_file Output to screen: Exporting Cust_file - 2012/02/28
Exported data file: Trans_file Output to screen: Exporting Trans_file - 2012/02/28

Invoking FastExport

The facing page displays the commands you can use to invoke the FastExport utility in batch mode. The parameters for each command are listed in the three-column table.

If you want to use FastExport in interactive mode, enter the command **fexp** at your system command prompt.

Optionally, when FastExport starts, it can read a configuration file to establish defaults for the FastExport job. On network-attached systems (e.g., Linux and Windows), FastExport can read configuration parameters from a file named **fexpcfg.dat**. This file is located either in the current directory or the directory referenced by the variable FEXPLIB.

On channel-attached systems, the DD statement for the MultiLoad configuration file must be labeled FEXPCFG.

There are 7 parameters you can set in this file.

- CHARSET=character-set-name
- ERRLOG=filename
- BRIEF=on/off
- MAXSESS=max-sessions
- MINSESS=min-sessions
- STATUS=ON/OFF
- DATAENCRYPTION=ON/OFF

The values that you specify in the FastExport configuration file override the internal utility default values for these parameters. Configuration file parameters can be overridden with runtime parameters. The order of preference (highest to lowest) for these parameters is:

1 – Runtime parameters
2 – FastExport script parameters
3 – Configuration file parameters
4 – FastExport default values

The FastExport utility automatically checks for a configuration file each time you invoke the utility. Upon locating a configuration file, the utility sets the defaults as specified, produces the appropriate output messages and begins processing your FastExport job.

If the configuration file cannot be opened, or if the FastExport utility encounters syntax errors in the file, the utility produces an output message, disregards the error condition and begins processing your FastExport job. An invalid configuration file entry does not abort your FastExport job.

If there is no configuration file, the utility begins processing your FastExport job without an error indication. The configuration file is an optional feature of the FastExport utility, and its absence is not considered to be an error condition.

Invoking FastExport

Network Attached Systems: fexp [PARAMETERS] < *scriptname* >*outfilename*

Channel-Attached MVS Systems: // EXEC TDSFEXP FEXPPARM= [PARAMETERS]

Channel-Attached VM Systems: EXEC FASTEXPT [PARAMETERS]

Channel Parameter	Network Parameter	Description
BRIEF	-b	Reduces print output runtime to the least information required to determine success or failure.
CHARSET=*charsetname*	-c *charsetname*	Specify a character set or its code. Examples are EBCDIC, ASCII, or Kanji sets.
ERRLOG=*filename*	-e *filename*	Alternate file specification for error messages; produces a duplicate record.
"*fastexport command*"	-r '*fastexport cmd*'	Signifies the start of a FastExport job; usually a RUN FILE command that specifies the script file.
MAXSESS=*max sessions*	-M *max sessions*	Maximum number of FastExport sessions logged on.
MINSESS=*min sessions*	-N *min sessions*	Minimum number of FastExport sessions logged on.
	< *scriptname*	Name of file that contains FastExport commands and SQL statements.
	> *outfilename*	Name of output file for FastExport messages.

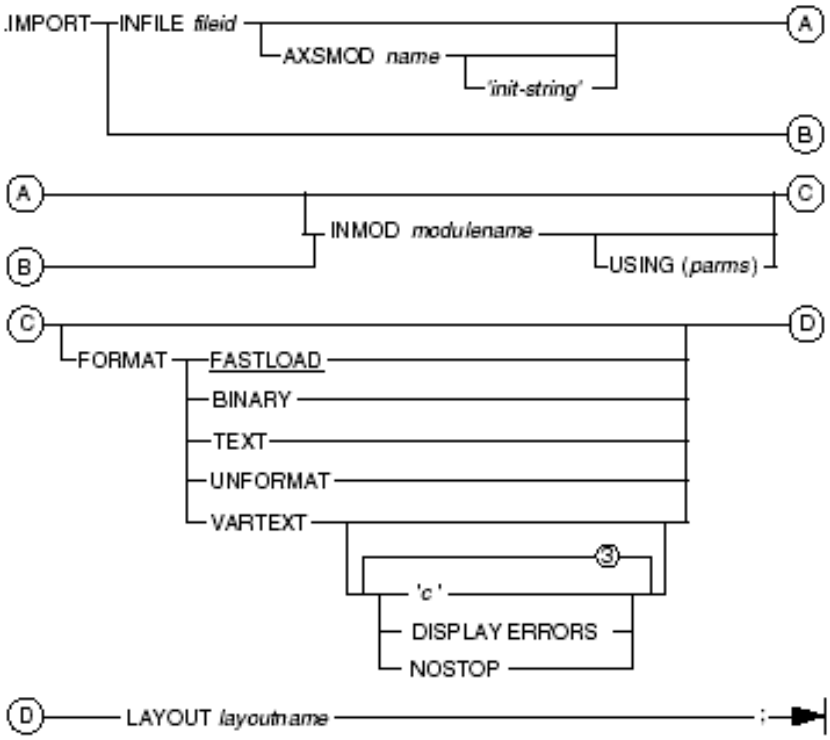
FastExport and Variable Input

The facing pages provide additional information regarding FastExport. Variable input to FastExport can either be accepted from a parameter file or you can IMPORT from a data file.

Selection Controls

This page defines the optional use of an IMPORT data set. Data from this data set can be used as dynamic variables in the SELECT statement(s) of the FastExport.

This allows you to run the same SELECT statement multiple times, each with a different dynamic value. The results of all of the occurrences of the SELECT statement are sent to a single FastExport data set.



The FORMAT options are similar to these described earlier for .EXPORT. VARTEXT specifies that each record is variable length text record format, with each field separated by a delimiter character.

FastExport and Variable Input

Selection Controls

- There are two techniques that can be used to provide variable input to FastExport.
  - ACCEPT from a parameter file; only accept from a single record.
  - IMPORT from a data file; each import record is applied to every SELECT.
- Read input variables from a host input data file described by the .LAYOUT command.
- Apply *each* input variable value to *every* SELECT in the exact order listed in the FastExport script before reading the next.
- Defines a host file as the source of the data values required for the SELECT REQUEST.
- Permits the use of a user-written INMOD routine to (optionally) read and (always) process the input record before passing it to the utility.

A FastExport Script with ACCEPT

The script shown on the facing page adds an ACCEPT command to the script shown earlier.

A FastExport Script with ACCEPT

parmfile1 'Los Angeles' 90066

city zipcode

ACCEPT variables from input record.

Reference accepted variables with an &.

```
.LOGTABLE  RestartLog1_fxp;

.RUN       FILE logon ;

.ACCEPT    city, zipcode FROM FILE parmfile1;

.BEGIN     EXPORT   SESSIONS 4 ;

.EXPORT    OUTFILE  custacct_data;

SELECT       A.Account_Number
           , C.Last_Name
           , C.First_Name
           , A.Balance_Current
FROM       Accounts A                    INNER JOIN
           Accounts_Customer AC          INNER JOIN
           Customer C
ON         C.Customer_Number = AC.Customer_Number
ON         A.Account_Number = AC.Account_Number
WHERE      A.City          = '&city'
AND        A.Zip_Code   =  &zipcode
ORDER BY   1 ;

.END EXPORT ;

.LOGOFF     ;
```

A FastExport Script with LAYOUT

The script shown on the facing page adds LAYOUT commands to the script shown earlier.

A FastExport Script with LAYOUT

city_zip_infile

Los Angeles	90066
San Diego	90217

city zipcode

IMPORT fields from input records.

Reference imported fields with a :

```
.LOGTABLE  RestartLog1_fxp;

.RUN       FILE logon ;
.BEGIN     EXPORT   SESSIONS 4 ;

.LAYOUT    Record_Layout  ;
.FIELD     city              1  CHAR(20) ;
.FIELD     zipcode           *  CHAR(5) ;

.IMPORT    INFILE city_zip_infile   LAYOUT Record_Layout ;

.EXPORT    OUTFILE  cust_acct_outfile2 ;
SELECT      A.Account_Number
           , C.Last_Name
           , C.First_Name
           , A.Balance_Current
FROM       Accounts A                    INNER JOIN
           Accounts_Customer AC          INNER JOIN
           Customer C
ON         C.Customer_Number = AC.Customer_Number
ON         A.Account_Number = AC.Account_Number
WHERE      A.City          = :city
AND        A.Zip_Code      = :zipcode
ORDER BY   1 ;

.END EXPORT ;

.LOGOFF ;
```

Application Utility Checklist

The facing page adds the FastExport capabilities to the checklist.

Automatic Restart – If the Teradata server restarts, FastExport will retry to connect to the Teradata database automatically and restart automatically.

Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions	ALL	LIMITED	Yes (SE)		
DML Functions	ALL	INSERT	SELECT		
Multiple DML	Yes	No	Yes		
Multiple Tables	Yes	No	Yes		
Protocol Used	SQL	FASTLOAD	EXPORT		
Conditional APPLY	No	No	No		
Data Conversion	Yes	1 per column	Yes		
Error Capture	No	Yes	N/A		
Error Limits	No	Yes	N/A		
User-written Routines	No	Yes	Yes		
Automatic Restart	No	Yes*	Yes		
Max Load Limit	No	Yes	Yes		
Support Environment (SE)	No	No	Yes		

Summary

Remember that FastExport is not designed to make the Teradata database perform faster. It *is* designed to take full advantage of multiple Parsing Engines, mainframe channels, and the LAN.

The facing page summarizes some important concepts regarding the FastExport utility.

Summary

- **Best choice for exporting large amounts of data from the Teradata database to a host file using multiple sessions.**
- **Fully automatic restart capability.**
- **Specialized processing of output data can be handled using an OUTMOD routine.**
- **The MaxLoadTasks and MaxLoadAWT parameters determine the maximum number of utility jobs that can execute at a given time.**
  - **Teradata can accommodate not more than a combined total of 60 utility jobs at any one time (FastLoad, MultiLoad, FastExport).**
  - **Up to 30 of these can be FastLoad, MultiLoad.**
  - **The FastExport limit is 60 minus the number of active FastLoad and MultiLoad jobs.**

Module 36: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 36: Review Questions

Answer True or False.

1. True or False. FastExport requires the use of a PI or USI in the SELECTs.
2. True or False. The number of FastExport sessions (for a Linux server) defaults to the number of AMPs.
3. True or False. The maximum block size you can specify with FastExport is 128 KB.
4. True or False. You can export from multiple tables with FastExport.
5. True or False. You can use multiple SELECTs in one FastExport job.
6. True or False. The default lock for a SELECT in a FastExport job is a table level ACCESS lock.

Lab Exercise 36-1

The facing page describes the tasks of this lab exercise.

The size of the data file (data36_1) should be 1,005,000 bytes.

With Linux systems, to view the output text from FastExport on your screen and place the output text into a file, you can use the following option:

fexp < scriptname.fxp | tee scriptname.out

To append the text to a file, use the –a (append option)

fexp < scriptname.fxp | tee –a scriptname.out

A technique that can be used to create Linux scripts without using vi or vim is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where Linux is running and …

3. **cat > lab36_11.fxp** (or whatever filename you wish)

   Use the mouse to choose the **Edit ➔ Paste** function (or right-click to paste in Linux)
   To exit the cat command, press either the CNTL C or DELETE key.

Lab Exercise 36-1

Lab Exercise 36-1

Purpose

In this lab, you will use FastExport to create an export file that contains one record for each transaction. You will have to join columns from two different tables in order to create the export file.

What you need

Populated AP.Accounts and AP.Trans tables.

Tasks

1. Create a FastExport script that outputs to file data36_1. For each transaction in the AP.Trans table, include the transaction_number, account_number, street_number, street, city, state, and the zip_code of the associated account (AP.Accounts).
2. Run the script.
3. Test the result by using the Linux ls -l command.

Lab Exercise 36-2

The facing page describes the tasks of this lab exercise. The exported file should have 439 rows. The following Linux command will provide the number of rows in the output report.

```
wc –l report36_2
```

Output should look like:

```
20024001   Los Angeles $233.00    Below MIN
20024002   Los Angeles $244.65    Below MIN
    :             :       :            :
20024797   Los Angeles $9,506.40  Above MAX
20024798   Los Angeles $9,518.05  Above MAX
```

Note: By default, a literal is exported as variable character data (preceded by 2 byte length indicator). This length indicator causes binary characters in the output. To avoid this, convert or cast the literal to fixed length character output. CAST can also be used to format the output.

The CASE can be used to generate the Below MIN or Above MAX literal for the output.

```
SELECT      CAST (Account_Number                    AS CHAR(12)),
            CAST (City                              AS CHAR(12)),
            CAST (CAST (Balance_Current
                  AS FORMAT '$,$$$,$$9.99')         AS CHAR(12)),
            CAST ((CASE
                  WHEN Balance_Current < &LoVal THEN ' Below MIN'
                  WHEN Balance_Current > &HiVal  THEN ' Above MAX'
                END)                                AS CHAR(10))
FROM   AP.Accounts
WHERE …
```

One way to create Linux scripts without using vi or vim is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where Linux is running and …

3. **cat > lab36_22.fxp** (or whatever filename you wish)

   Use the mouse to choose the **Edit ➔ Paste** function or right-click in Linux.
   To exit the cat command, press either the CNTL C or DELETE key.

Lab Exercise 36-2

Lab Exercise 36-2

Purpose

In this lab, you will use FastExport to accept input values as a parameter or read input data from a data file, and export a report to another data file. In order to produce readable output, all selected data should be converted to character data as outlined below:

What you need

Populated AP.Accounts table.

Tasks

Use FastExport to only export a report that contains a list of Accounts which either fall below a minimum Balance_Current or exceed a maximum value AND are from the city in the input data file.

1. Create an input file named data36_2 with 1 line of input: 'Los Angeles'
2. Prepare a FastExport script which does the following:
   a. Treats this as a parameter file and ACCEPT from it. Treat this data as variable input for the SELECT.
   b. Uses the .SET command to initialize two variables: LoVal 500 and HiVal 9499
   c. Includes a SELECT statement that projects ACCOUNT NUMBER, CITY, BALANCE CURRENT, and a character string of either BELOW MIN or ABOVE MAX and sorts by Account_Number. Simply display 'BELOW MIN' or 'ABOVE MAX' as a literal with the SELECT. Cast all of the numeric columns to character data.
   d. Creates an output file named *report36_2*. Note: Include MODE RECORD and FORMAT TEXT.
3. Run the test and view the result using the Linux *more* command.

Notes

Module 37

TERADATA
Raising Intelligence

MultiLoad

After completing this module, you will be able to:

- **Describe the capabilities of MultiLoad.**
- **Name the five phases of MultiLoad and state the main function of each.**
- **Create a MultiLoad script.**
- **Run a script to update/load table(s) using MultiLoad.**
- **Explain the advantages of using MultiLoad.**

Notes

Table of Contents

What is MultiLoad? 37-4
MultiLoad Limitations 37-6
How MultiLoad Works 37-8
Advantages of MultiLoad 37-10
Basic MultiLoad Statements (for Import Tasks) 37-12
Sample MultiLoad IMPORT Task 37-14
IMPORT Task 37-16
  Other Import Options 37-16
    INMOD 37-16
    AXSMOD 37-16
5 Phases of IMPORT Task 37-18
Phase 1: Preliminary 37-20
Phase 2: DML Transaction 37-22
Phase 3: Acquisition 37-24
Phase 3: Acquisition – a Closer Look 37-26
Phase 4: Application 37-28
Phase 4: Application – a Closer Look 37-30
Phase 5: Cleanup 37-32
    Execute END MLOAD processing as an explicit transaction 37-32
    MLOAD Session Logoff 37-32
Sample MultiLoad DELETE Tasks 37-34
DELETE Task Differences from IMPORT Task 37-36
A Closer Look at DELETE Task Application Phase 37-38
MultiLoad Locks 37-40
    Utility locks 37-40
Restarting MultiLoad 37-42
RELEASE MLOAD Statement 37-44
Invoking MultiLoad 37-46
Application Utility Checklist 37-48
Summary 37-50
Module 37: Review Questions 37-52
Lab Exercise 37-1 37-54

What is MultiLoad?

MultiLoad is a batch mode utility that runs on the host system. It is used for loading, updating or deleting data to and from populated tables, typically with batch inputs from a host file.

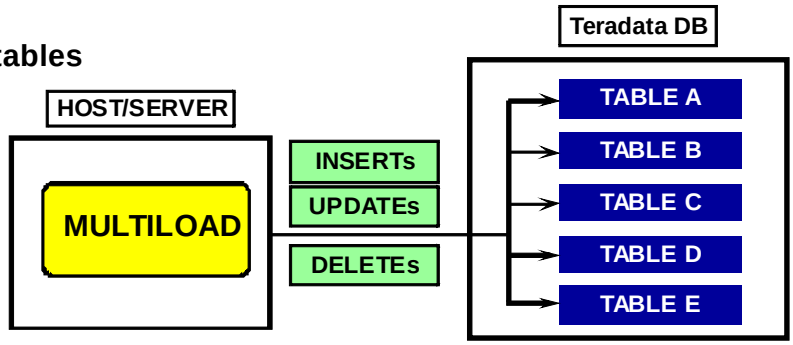
MultiLoad has many features that make it appealing for maintaining large tables:

- Uses FastLoad-like technology to accomplish TPump-like functionality.
- Support for up to five tables per import task.
- Tables may contain pre-existing data, but cannot have Unique Secondary Indexes nor can it have Referential Integrity.
- Ability to perform multiple maintenance operations with one pass of input data files.
- Ability to perform conditional maintenance based on 'apply' condition.
- Ability to do INSERTs, UPDATEs, DELETEs and UPSERTs (UPDATE if exists, else INSERT).
- Each affected data block is written only once.
- Host and LAN support.
- Full Restart capability using a Log file, even with AMPs down.
- Programmable error limits.
- Error reporting via error tables.
- Support for INMODs to customize data being loaded—although less likely.

The DBSControl parameter MaxLoadTasks defines the maximum number of utilities (FastLoad, FastExport, and MultiLoad) that can run on the system at one time.

What is MultiLoad?

- Batch mode utility that runs on a server or host system.
- FastLoad-like technology – TPump-like functionality
- Supports up to five populated tables
- Multiple operations with one pass of input files
- Conditional logic for applying changes
- Supports INSERTs, UPDATEs, DELETEs and UPSERTs; typically with batch inputs from a host file.
- Affected data blocks only written once
- Host and LAN support
- Full Restart capability
- Error reporting via error tables
- Support for INMODs



MultiLoad Limitations

MultiLoad is a very powerful and flexible utility. Some **MultiLoad** restrictions are:

- No data retrieval capability (i.e., no **SELECT** statement).
- Concatenation of input data files is not allowed.
- Host (APPLY clause) will not process arithmetic functions (i.e., ABS, LOG, etc.).
- Host will not process exponentiation or aggregate operators (i.e., AVG, SUM, etc.).
- Cannot process tables with Unique Secondary Indexes USIs, Join Indexes, or Hash Indexes defined.
- Import tasks require use of Primary Index.

If any of the above limitations are significant to your ability to load a table, you might want to consider alternatives:

- Write an INMOD for use with MultiLoad.
- Use TPump.
- Use FastLoad.

MultiLoad Limitations

- No data retrieval capability.
- Concatenation of input data files is not allowed.
- Host will not process arithmetic functions.
- Host will not process exponentiation or aggregates.
- Cannot process tables defined with USI's, Referential Integrity, Join Indexes, Hash Indexes, or Triggers.
  - Soft Referential Integrity is supported
- Import tasks require use of Primary Index.
  - Loads to a No Primary Index table are NOT allowed.

Alternatives:

- Write an INMOD for use with MultiLoad.
- Use TPump.
- Use FastLoad.

How MultiLoad Works

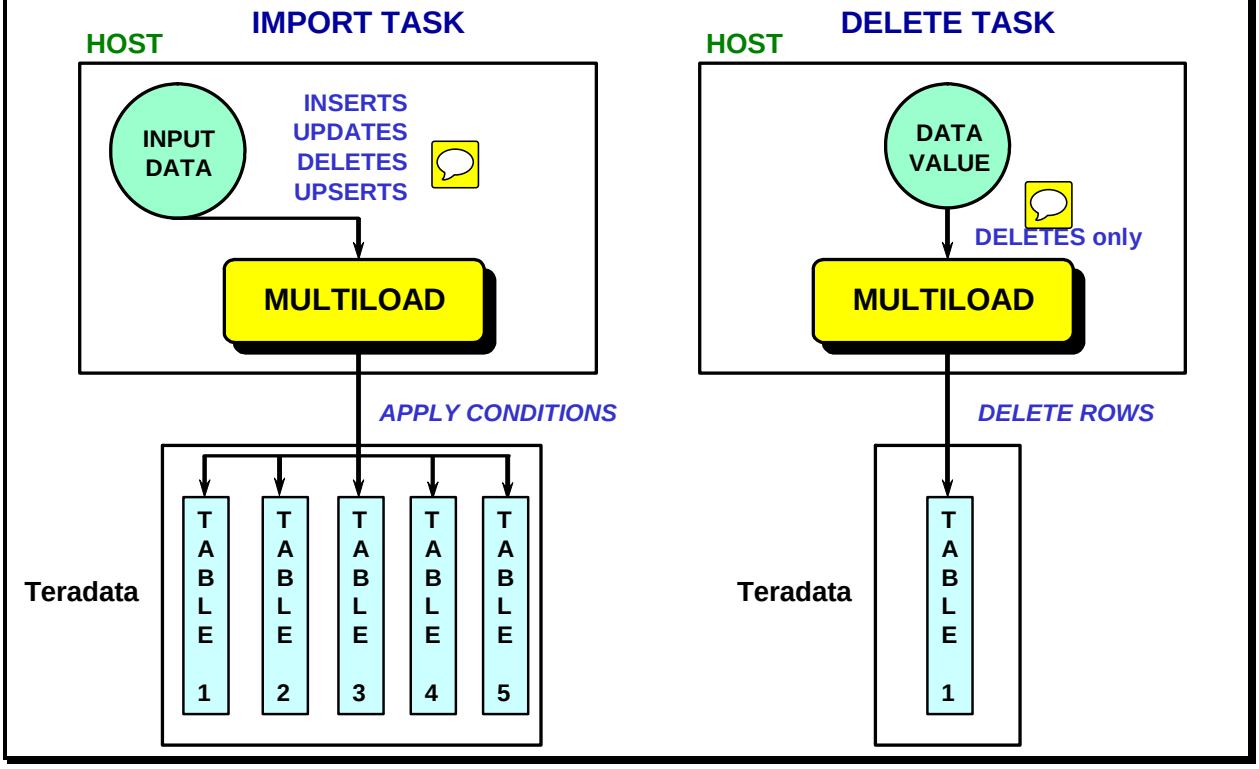
MultiLoad typically uses an input file that is read to run batch-like maintenance actions against data on the Teradata database. It allows **INSERT, DELETE, UPDATE** and **UPSERT** operations against up to five tables per import task.

There are two distinct types of tasks that MultiLoad can perform:

IMPORT task Intermix a number of different SQL/DML statements and apply them to up to five different tables depending on the APPLY conditions.

DELETE task Execute a single **DELETE** statement on a single table, often with a single value as a condition for the deletion (i.e., DELETE FROM TABLE_1 WHERE COL_A > 921000;). This value may be hard-coded or supplied from a host file.

How MultiLoad Works



Advantages of MultiLoad

The advantages of MultiLoad are listed on the facing page.

Advantages of MultiLoad

- Minimizes the use of the PEs.
- Gets input data to the AMPs as quickly as possible.
- Uses multiple-AMP sessions.
- Uses the parallelism of the AMPs to apply changes.
- Keeps BYNET activity low with AMP-local processing.
- Avoids Transient Journaling overhead.
- Allows Checkpoint/Restartability even with down AMPs.
- Prevents lengthy rollbacks of aborted jobs.
- Allows for maximum access to table during processing.
- Posts errors to special error tables.
- Provides extensive processing statistics.

Basic MultiLoad Statements (for Import Tasks)

The following is an explanation of common components of a MultiLoad IMPORT script:

.LOGTABLE defines the table name of the Restart Log.

.LOGON defines username, which will own the sessions.

.BEGIN MLOAD TABLES defines the tables, which will participate in the MultiLoad.

.LAYOUT defines the layout of the incoming record(s).

.FIELD defines the name of an input field, its position in the record, and its data type. (Absolute positioning may be done with a number, or relative positioning with an asterisk, "*".)

.FILLER defines input data that will not be sent to the database table. A .FILLER statement allows a name and requires a starting position or asterisk, and the data type.

.DML LABEL defines a set of DML instructions, which will be applied if conditions are met.

.IMPORT INFILE references the name of the input file.

FROM *m* FOR *n*	optionally defines starting # and ending #
THRU *k*	of records to process from input file.

FORMAT options – FASTLOAD
BINARY
TEXT
UNFORMAT
VARTEXT ','

LAYOUT references previously defined **LAYOUT.**

APPLY references **LABEL** to be applied and conditions under which to do so.

.END MLOAD; defines end of MultiLoad script.

.LOGOFF; terminate the sessions.

Basic MultiLoad Statements
(for Import Tasks)

```
.LOGTABLE  [ logtable_name ] ;
.LOGON  [ tdpid/userid, password ] ;
.BEGIN MLOAD TABLES  [ tablename1, ... ] ;
.LAYOUT  [ layout_name ] ;
      .FIELD ..... ;
      .FILLER ..... ;
.DML  LABEL  [ label ] ;
      INSERT (or UPDATE or DELETE) statements;
.IMPORT  INFILE  [ filename ]
      [ FROM m ]  [ FOR n ]  [ THRU k ]
      [ FORMAT FASTLOAD | BINARY | TEXT | UNFORMAT |  VARTEXT 'c' ]
      LAYOUT  [ layout_name ]
      APPLY  [ label ]  [ WHERE condition ] ;
.END  MLOAD ;
.LOGOFF ;
```

```
.FIELD  fieldname { startpos datadesc } || fieldexp          [ NULLIF  nullexpr ]
    [ DROP {LEADING / TRAILING } { BLANKS / NULLS }
    [ [ AND ] {TRAILING / LEADING }  { NULLS / BLANKS } ] ] ;
.FILLER  [ fieldname ] startpos datadesc ;
```

Sample MultiLoad IMPORT Task

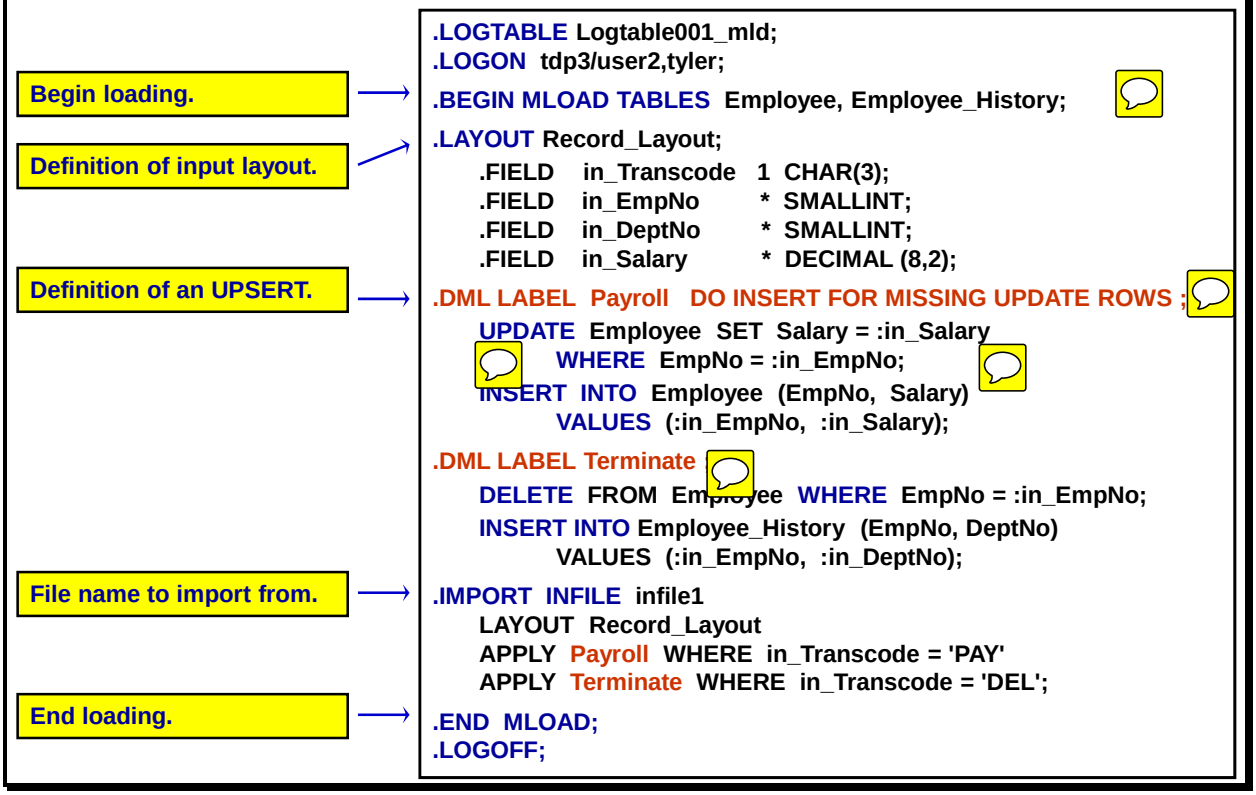
The script on the facing page updates, inserts, and deletes, depending on the conditions for the employee table.

Each import task can include multiple INSERT, UPDATE, and DELETE statements, and the multiple DML operations can be conditionally applied to as many as five tables with a single pass of the client file.

The key words "**DO INSERT FOR MISSING UPDATE ROWS**" indicate an UPSERT operation. An UPSERT requires consecutive UPDATE and INSERT statements following the .DML LABEL statement.

If the UPDATE statement fails because the target table row does not exist, MultiLoad automatically executes the INSERT statement, completing the operation in a single pass instead of two.

Sample MultiLoad IMPORT Task



IMPORT Task

IMPORT tasks are used to do multiple combinations of **INSERTs, DELETEs, UPDATEs** and **UPSERTs** to one or up to five tables. Updates that change the value of a table's primary index are not permitted. You may change the value of a column based on its current value (i.e., COL = COL + 10).

IMPORT tasks cannot be done on tables with Unique Secondary Indexes.

Other Import Options

INMOD

An INMOD is an exit routine that can precondition data and pass it on to the loader. You can write INMODs to pre-screen the input data being sourced into MultiLoad.

INMOD and MultiLoad use a return code value to communicate with each other.
You can write INMODs as restartable routines so that they can synchronize with the loader's checkpoints.

When an INMOD-connected loader restarts, both the utility and the INMOD can be repositioned to the last checkpoint.

Use INMODs to perform unusual conversions of data, for example, adding a sequenced column to the data, or reading data from a non-standard input file format.

AXSMOD

Another option for the IMPORT command is to include the specification of an AXSMOD.

AXSMOD is used to specify an access module file that imports data from a file.

The AXSMOD option is not required for importing:

- Disk files on either network- or channel-attached systems
- Magnetic tape files on channel-attached client systems.

It is *required* for importing magnetic tape and other types of files on *network-attached* client systems.

IMPORT Task

- INSERTs, DELETEs, UPDATEs and UPSERTs allowed.
- Up to a maximum of five tables:
  - Empty or populated.
  - NUSIs permitted.

- MultiLoad Import task operations are always primary index operations - however, you are not allowed to change the value of a table's primary index.
- Change the value of a column based on its current value.
- Permits non-exclusive access to target tables from other users except during Application Phase.
- Input error limits may be specified as a number or percentage.
- Allows restart and checkpoint during each operating phase.
- IMPORT tasks cannot be done on tables with USI's, Referential Integrity, Join Indexes, Hash Indexes, or Triggers.
  - IMPORT tasks can be done on tables defined with "Soft Referential Integrity".

5 Phases of IMPORT Task

IMPORT consists of five separate phases of processing. They are:

Preliminary phase	Basic setup
DML phase	Get DML steps down on AMPS
Acquisition phase	Send the input data to the AMPS and sort it
Application phase	Apply the input data to the appropriate target tables
End phase	Basic clean up

5 Phases of IMPORT Task

Preliminary	Basic set up
DML Transaction	Send the DML steps to the AMPs
Acquisition	Send the input data to the AMPs
Application	Apply the input data to appropriate table(s)
Cleanup	Basic clean up

Details

Phase 1: Preliminary

The first of **IMPORT**'s five phases is the **Preliminary**. It performs the following tasks:

Validate all statements	All MultiLoad and SQL statements are validated and syntax checked.
Start all sessions	Typically, one MLOAD session per AMP plus two control sessions. One is for handling the SQL and logging and the second is an alternate logging session.
Create work tables	Work tables are created on each AMP for each target table. They will hold the DML steps to be performed as well as the input data to be applied.
Create error tables	Two error tables are created for each target table. One is for general errors and one is for uniqueness violations.
Create Restart log	Create the log for this run which will allow restarts.
Apply locks to tables	Utility locks are applied to the target table headers. This lock disallows any DDL to the table (except DROP).

Phase 1: Preliminary

IMPORT Phases

Validate all statements	→	MultiLoad and SQL
Start all sessions	→	#AMPS + 2 
Create work tables	→	One per target table
Create error tables	→	Two per target table
Create Restart log	→	One per IMPORT run
Apply locks to target tables	→	Prevent DDL

Phase 2: DML Transaction

The second of IMPORT's five phases is the DML/Transaction. It performs the following:

Send prototype DML to the DBC

All DML statements (minus data) are sent from host to DBC where they are parsed. Steps are generated and stored on each AMP in the work table for the affected target table.

In Phase 1, work tables were created on each AMP for each target table. In this phase, the DML steps to be performed will be placed into the work tables.

Add a USING modifier to the request

Each request is submitted with a USING clause, with host data to be filled in at execution time.

Add a "Match Tag" to the request

Because it will be necessary to know which DML is to be associated with which incoming record (this is what the APPLY clause decides), we will use a "match tag" to link DML requests with input records.

Phase 2: DML Transaction

IMPORT Phases

Send prototype DML to the Teradata Database

- Store DML steps in work tables

Add a USING modifier to the request

- Host data to be filled in from input file

Add a "Match Tag" to the request

- Allows link between DML and transaction record

Phase 3: Acquisition

The third of **IMPORT's** five phases is the Acquisition phase. It performs the following steps:

Get host data to the appropriate AMP worktables

Work tables only, not target tables, are involved in the Acquisition phase. The host reads the input file and tests for the APPLY conditions. A copy of the input record is made for every successful APPLY. The appropriate "match tag" information is also built into the record and the records are bunched into blocks. They are sent, round robin to the AMPs using a "quickpath," that is, they go through but are never processed by the PEs. The AMPs will have started "deblocker" tasks that will read the individual records from the block, hash on primary index value, and send that row to the AMP that holds the target row. The AMPs will also have started "receiver" tasks that pick up the incoming records with the correct hash value. These records are accumulated in the work table of the appropriate target table and reblocked. Records are built for the FALLBACK subtables as well.

Sort the reblocked records in the work tables

Access locks are placed on the target tables. Records are sorted according to the hash value and the sequence in which they will be applied to the target table.

Set up transition to the Application phase

The Access lock on the target tables is upgraded to a Write lock. Utility locks are applied to the table headers indicating the Application phase is about to begin. An End Transaction statement commits all header changes for all target tables across all AMPs.

Phase 3: Acquisition

IMPORT Phases

- **Get the data from host and apply it to appropriate AMP worktables.**
  - Duplicate “input records” record for each successful APPLY.
  - Add “Match Tag” information to record.
  - Make blocks and send “quickpath” to AMPs.
  - Deblock and resend record to “correct” AMP.
- **Reblock and store in worktable of target table.**
  - Sort the reblocked records in the work tables. 
  - Sort by hash value and sequence to be applied.
- **Set up transition to the Application phase.**
  - Upgrade locks on target tables to Write.
  - Set table headers for Application phase.
  - This is effectively the “point of no return”.

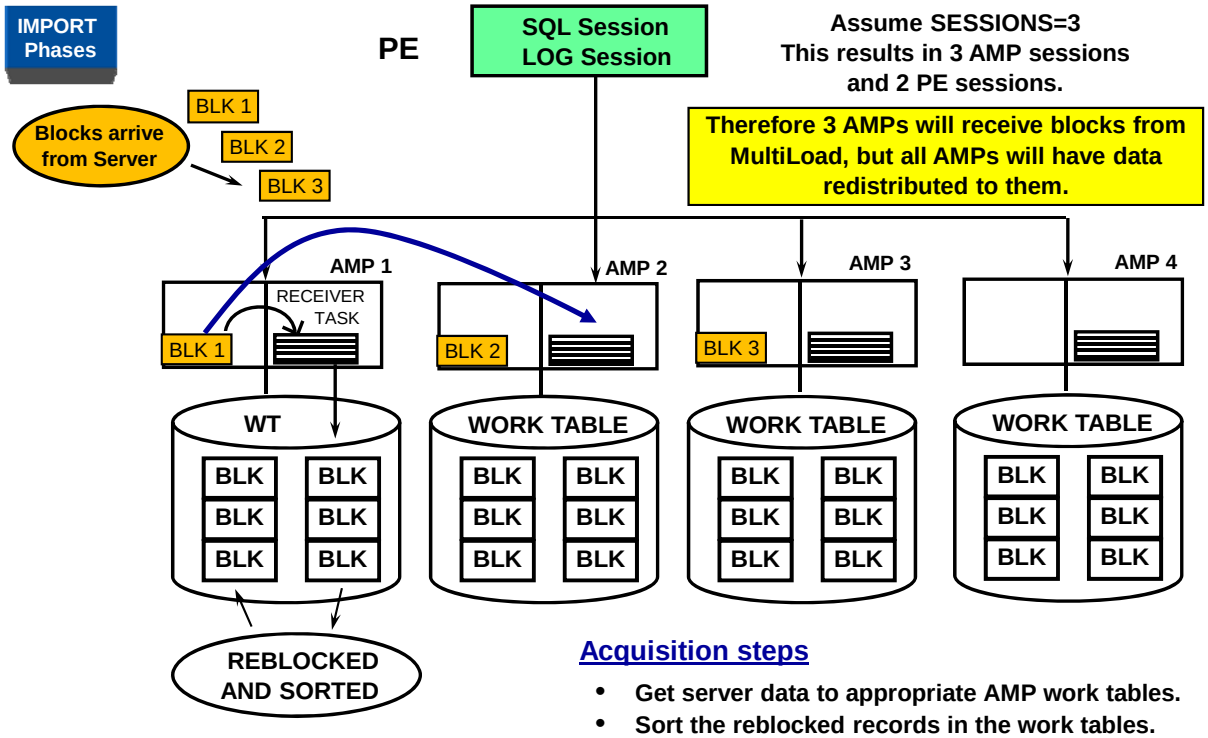
Notes:

- Errors that occur in this phase go into the Acquisition Error Table (default name is ET_tablename).
- There is no acquisition phase activity for a DELETE Task.

Phase 3: Acquisition – a Closer Look

The diagram on the facing page shows data movement from the host to the deblockers on to the appropriate receivers (based on hash code) then to the work table where finally it is sorted and reblocked.

Phase 3: Acquisition – a Closer Look



Acquisition steps

- Get server data to appropriate AMP work tables.
- Sort the reblocked records in the work tables.
- Set up transition to the Application phase.

Phase 4: Application

The fourth of **IMPORT**'s five phases is the **Application phase**. It performs the following steps:

Execute MLOAD for each target table as a single multi-statement request

There is no further interaction with the host until the end of the phase. There is a separate execution of MLOAD for each target table, which means that the AMPs may independently and asynchronously apply changes to target tables. Because all EXEC MLOADs (up to five) are submitted as a multi-statement request, they are looked upon as a single transaction. If the transaction fails, changes are not rolled back, and the transaction is restartable at the point of failure. This eliminates the need for transient journaling.

Apply work subtable changes to target subtables

Each target table block requiring change is read and written only once. After reading the target block, that part of the work table (called a "work unit") having matching hash codes is also read. Changes are applied to the target rows of the block according to the **DML** operation identified by that row's match tag.

If an error results from applying a row, that row is inserted into the UV error table associated with the target table for which that row was intended. Duplicate rows, missing update, or delete rows may also be inserted into this error table according to options specified by the user.

Because of the possibility of UPSERT processing and/or missing rows, it may be necessary to sweep the block more than once. A bit map is maintained showing which changes in the work unit have been applied and which have not. Once all processing has been done, the block is written out and a checkpoint is written to the work table.

After applying all changes to the target tables, **NUSI** changes, both primary and fallback, are applied to the target **NUSI** subtables. If the target table has permanent journaling, a Private Permanent Journal is maintained by MLOAD, and is then transferred to the true Permanent Journal.

Phase 4: Application

IMPORT Phases

- **Execute MLOAD for *each* target table as a *single* multi-statement request.**
  - **End of host interaction until end of phase.**
  - **AMPs independently apply changes to target tables.**
  - **Executed as a single transaction without rollback.**
  - **Restartable based on last checkpoint.**
  - **No transient journal needed.** 

Note:

- **Errors that occur in this phase go into the Application Error Table (default name is UV_tablename).**

Phase 4: Application – a Closer Look

The diagram on the facing page is intended to show how an individual AMP applies changes from the work subtables to the target tables during the **Application phase**.

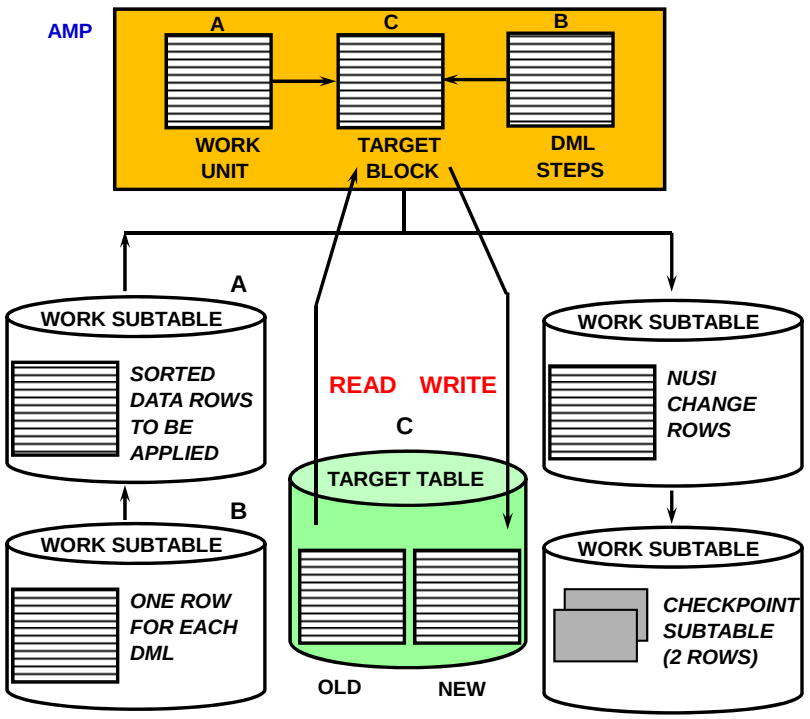
There will be one MLOAD task in each AMP for each target table. Two **MultiLoads** running, each against three target tables will result in each AMP having six MLOAD tasks running.

Phase 4: Application – a Closer Look

IMPORT Phases

Apply work subtable changes to target subtables:

- Affected blocks read/written only once.
- Changes applied based on matching row-hash.
- Errors written to appropriate error table.
- Checkpoint after writing each block.
- NUSI subtable changes applied.



Phase 5: Cleanup

The fifth **IMPORT** phase is the **End** or **Cleanup phase**, which performs the following steps:

Execute END MLOAD processing as an explicit transaction

After all changes have been applied to the target tables, many housekeeping chores remain before the utility is finished. All locks, both utility and DBC locks, must be released. All table headers must be restored to their original status across all AMPs. All Work Tables and any empty Error Tables are dropped. The dictionary cache for Target Tables is spoiled. Statistics are reported and a final error code is returned to the user. If the error code is zero, the Log Table is dropped.

MLOAD Session Logoff

A LOGOFF request is sent to each Load Control Task on an AMP that owns a session.

Phase 5: Cleanup

IMPORT Phases

- **Execute END MLOAD processing as a series of transactions performed by the host utility:**
  - All locks are released.
  - Table headers are restored across all AMPs.
  - Dictionary cache of Target Tables is spoiled.
  - Statistics are reported.
  - Final Error Code is reported.
  - Target tables are made available to other users.
  - Work Tables are dropped.
  - Empty Error Tables are dropped.
  - Log Table is dropped (if Error Code = 0).
- **MLOAD Session Logoff:**
  - LOGOFF request is sent to each AMP with a session.

Sample MultiLoad DELETE Tasks

The following is an explanation of the components of a **MultiLoad DELETE** Task script:

.LOGTABLE defines the name of the Restart Log.

.LOGON defines username that will own the sessions.

.BEGIN DELETE MLOAD TABLES defines table that will participate in the MultiLoad.

.LAYOUT defines the layout of the incoming record.

.FIELD defines the name of an input field, its position in the record, and its data type. (Absolute positioning may be done with a number, or relative positioning with an asterisk "*.")

DELETE FROM standard **SQL DELETE** statement.

.IMPORT INFILE references **DDNAME** of the input file.

LAYOUT references previously defined **LAYOUT.**

.END MLOAD; defines end of MultiLoad script.

.LOGOFF; terminate the sessions.

A **DELETE** task is simpler than most **IMPORT** tasks. Note also that a **DELETE** task has no **.DML** and no **APPLY** clauses, because the single imported data record is unconditionally applied by the single **DELETE** statement.

Sample MultiLoad DELETE Tasks

Hard code the values of rows to be deleted.

```
.LOGTABLE  Logtable002_mld;
.LOGON     tdp3/user2,tyler;
.BEGIN  DELETE  MLOAD TABLES Employee;
DELETE FROM  Employee WHERE Term_date  >  0;
.END  MLOAD;
.LOGOFF;
```

Pass a single row containing value(s) to be used.

```
.LOGTABLE Logtable003_mld;
.LOGON  tdp3/user2,tyler;
.BEGIN DELETE MLOAD TABLES Employee;
.LAYOUT Remove;
     .FIELD in_Termdate   *  INTEGER;
DELETE FROM  Employee  WHERE Term_date  >  :in_Termdate;
.IMPORT  INFILE  infile2
     LAYOUT  Remove;
.END  MLOAD;
.LOGOFF;
```

DELETE Task Differences from IMPORT Task

DELETE tasks operate very similarly to **IMPORT** tasks with some differences.

Differences include:

- Deleting based on equality of a Unique Primary Index access is not permitted.
- A single **DML DELETE** statement is sent to each AMP with a match tag parcel.
- There is no **Acquisition phase** because there are no varying input records to apply.
- The **Application phase** reads each target block and deletes qualifying rows.
- Altered blocks are written back to disk.
- All other aspects of task are similar to **IMPORT** task.

DELETE Task Differences from IMPORT Task

DELETE tasks operate very similarly to IMPORT tasks with some differences:

- **Deleting based on a equality UPI value is not permitted.**
  - **An inequality (e.g., >) test of a UPI value is permitted.**
  - **An equality (e.g., =) test of a NUPI value is permitted.**
- **A DML DELETE statement is sent to each AMP with a match tag parcel.**
- **No Acquisition phase because no variable input records to apply.**
- **Application phase reads each target block and deletes qualifying rows.**
- **All other aspects similar to IMPORT task.**

Why use MultiLoad DELETE (versus SQL DELETE)?

- **MultiLoad DELETE is faster and uses less disk space and I/O (no Transient Journal).**
- **MultiLoad DELETE is restartable.**
  - **If SQL DELETE is aborted, Teradata applies Transient Journal rows. SQL DELETE can be resubmitted, but starts from beginning.**

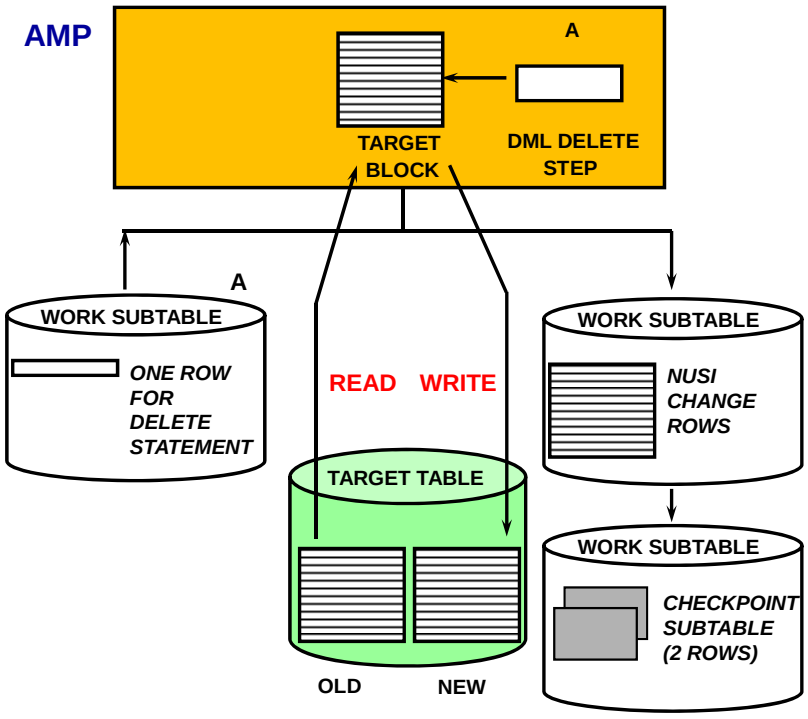
A Closer Look at DELETE Task Application Phase

The accompanying diagram attempts to show the movement of data in the Application phase of a **DELETE** task. Notice the absence of a Work Table carrying imported rows. There is no work table because the same **DML DELETE** statement will be applied to every row in the table.

Using MultiLoad **DELETE** tasks give you an advantage over using traditional utilities to accomplish a similar **DELETE** since there is no use of a transient journal, and no rollback in the event of failure. Because of the restart capabilities of MultiLoad, no completed work needs to be reapplied.

A Closer Look at DELETE Task Application Phase

- Note absence of Work Table for Import rows.
- Faster than traditional SQL DELETE due to:
  - Lack of transient journaling
  - No rollback of work
  - Restartable from checkpoint



MultiLoad Locks

MultiLoad uses several different locks at various stages of the operation. These are intended to insure maximum availability of the target tables during **MultiLoad** processing as well as restartability of various phases of the utility.

Utility locks

Utility locks are placed in the table headers to indicate to utilities such as Reconfig and Rebuild that an MLOAD is in progress and to do special processing. Utility locks are the minimum level of lock required when an MLOAD is invoked even when it is not currently running.

There are two types of utility locks: **Acquisition locks** and **Application locks.** They are defined below:

Acquisition locks	prevent any **DDL** activity against the table with the exception of the **DROP** command, but does allow all **DML** command access.
Application locks	allow concurrent access-lock **SELECT** access and the **DROP DDL** statement, but reject all other **DML** and **DDL** statements.

There are two important points to bear in mind in understanding the locking strategy of MultiLoad. First, there are never any **Exclusive locks** used on the target tables. This means that **Access locks** are always useable against target tables throughout the MultiLoad execution.

Secondly, each of the five stages of MultiLoad is treated as one or more DBC transactions. This means that the end of each stage is also the end of a transaction and that all locks associated with that stage are thereby released. New locks are applied with the beginning of the next phase/transaction. This permits the ability of other transactions, outside of MultiLoad, to effect the target table rows during a MultiLoad execution.

For example, at the end of the **Acquisition phase**, the access lock on the target table is released. As the **Application phase** begins, a "new write lock" is applied to the target table as a part of the new transaction. Between these two locks, other external requests may "get in" to the target table, thus preventing them from having to wait in the queue until MultiLoad completes.

> **Note:** If you need to examine the logtable, the work tables, or the error tables at any time during the execution of MultiLoad, you MUST use an ACCESS lock to access them in order to prevent MultiLoad from abnormally terminating due to locking problems.

MultiLoad Locks

Utility locks: Placed in table headers to alert other utilities that a MultiLoad is in session for this table. They include:

- **Acquisition lock**
  - **DML — allows all**
  - **DDL — allows DROP only**
- **Application lock**
  - **DML — allows SELECT with ACCESS only**
  - **DDL — allows DROP only**

Restarting MultiLoad

MultiLoad contains a number of features that allow for recovery from any host or DBC failure. It does this with minimal requirements for job resubmission or continuation. Upon restart, MultiLoad will check the restart log table and resume operations from where it had previously left off.

If a **DBC restart** occurs during MLOAD, the host program will reinitiate MLOAD after DBC recovery and continue from where it left off with no user interaction required.

If a **host restart** occurs during MLOAD, or the job is aborted, the user may resubmit the script as-is, and MLOAD will determine its stopping point and begin again. No script alteration is required.

If an MLOAD task is stopped during the Application phase, it must be resubmitted and allowed to run to completion.

Restarts are initiated based on checkpoint information in the Logtable. Because MLOAD does not do transient journaling, a traditional rollback operation is not performed when a failure occurs. MLOAD is designed with a check pointing feature that allows for restart of the job with minimal loss of work. The following principles guide the MultiLoad check pointing strategy:

Acquisition phase check pointing is performed according to user specification as specified in the **.BEGIN MLOAD** statement. This checkpoint can be based on time or on number of records processed. The default check point interval is fifteen minutes.

Application phase check pointing is performed each time a data block is written to the target table. Each block is written one time.

Sort phase(s) sort operations do their own internal check pointing that overrides the higher level checkpoints.

Restarting MultiLoad

Teradata Restart

- MLOAD reinitiated automatically after Teradata recovery.
- Continue from checkpoint without user interaction.

Host restart

- Resubmit the original script.
- MLOAD determines its stopping point and restarts.

MultiLoad Checkpointing Strategy

Acquisition phase

- Checkpointing is performed according to user.
- Checkpoint based on time or on number of records.
- Default checkpoint interval is fifteen minutes.

Application phase

- Checkpointing done after each write of data block.
- Each block is written at most only one time.

Sort phase(s)

- Sort operations do their own internal checkpointing.

RELEASE MLOAD Statement

Once MultiLoad execution has begun, table headers are updated in the target tables indicating that a MLOAD is in progress. Even if the MLOAD doesn't successfully complete, target tables are still considered under the control of the MLOAD and access to them will be restricted accordingly.

The **RELEASE MLOAD** statement provides a way to return tables to general availability where there is no desire to restart the MLOAD. If the specified table is in the Preliminary, DDL or the early part of the Acquisition phase, the **RELEASE MLOAD** statement makes the table completely accessible and prevents any attempt to restart the MLOAD. If the MLOAD had proceeded into the Application phase, the **RELEASE MLOAD** statement is rejected and the job must be restarted.

Once a lock has been applied to the target table, the **RELEASE MLOAD** statement will not be effective until the transaction with the lock completes. Even if the transaction completes, **RELEASE MLOAD** may be rejected if the point of no return has occurred. In a **DELETE** task, because there is no Acquisition phase, the point of no return is the point in the DML phase when the **DELETE** statement is sent to the DBC. In an **IMPORT** task, the actual point of no return is the point at which the Acquisition phase ends.

To successfully complete a **RELEASE MLOAD**, the following procedure must be followed:

1. Make sure MLOAD is not running; abort it if it is. (Note: MLOAD is still in a restartable state if aborted. If it is past the point of no return, go to step 4.)

2. Enter **RELEASE MLOAD**.

3. If successful, drop the work and error tables. (You may wish to examine any errors in the error tables before dropping them.)

4. If not successful, determine if past point of no return. If so, either restart MLOAD and let it complete, or drop target, work, and error tables. Otherwise, handle errors as appropriate.

RELEASE MLOAD Statement

```
RELEASE MLOAD Employee, Job, Department;
```

- Returns target tables to general availability.
- Prevents any attempt to restart MultiLoad.
- Cannot be successful in all cases.
- Cannot override a target table lock.
- IMPORT — possible before Application phase.
- DELETE — possible during Preliminary phase.

To successfully complete a RELEASE MLOAD:

1. Make sure MLOAD is not running; abort if it is. (If it is past the point of no return, go to step 4.)
2. Enter RELEASE MLOAD.
3. If successful, drop the log, work, and error tables.
4. If not successful:
   a.) restart MLOAD and let it complete, or
   b.) drop target, work, and error tables, or
   c.) handle error as appropriate.

Invoking MultiLoad

The facing page displays the commands used to invoke the MultiLoad utility in batch mode. The parameters for each command are listed in the three-column table.

Optionally, when MultiLoad starts, it can read a configuration file to establish defaults for the MultiLoad job. On network-attached systems (e.g., Linux and Windows), MultiLoad can read configuration parameters from a file named **mloadcfg.dat**. This file is located either in the current directory or the directory referenced by the variable MLOADLIB.

On channel-attached systems, the DD statement for the MultiLoad configuration file must be labeled MLOADCFG.

There are 7 parameters you can set in this file.

- CHARSET=character-set-name
- ERRLOG=filename
- BRIEF=on/off
- MAXSESS=max-sessions
- MINSESS=min-sessions
- MATCHLEN=ON/OFF
- DATAENCRYPTION=ON/OFF

Note: When you enable MATCHLEN, MultiLoad verifies that the record length of the import data is the same as the layout record length specified by the IMPORT command.

The values that you specify in the MultiLoad configuration file override the internal utility default values for these parameters. Configuration file parameters can be overridden with runtime parameters. The order of preference (highest to lowest) for these parameters is:

1 – Runtime parameters
2 – MultiLoad script parameters
3 – Configuration file parameters
4 – MultiLoad default values

The MultiLoad utility automatically checks for a configuration file each time you invoke the utility. Upon locating a configuration file, the utility sets the defaults as specified, produces the appropriate output messages and begins processing your MultiLoad job.

If the configuration file cannot be opened, or if the MultiLoad utility encounters syntax errors in the file, the utility produces an output message, disregards the error condition and begins processing your MultiLoad job. An invalid configuration file entry does not abort your MultiLoad job.

If there is no configuration file, the utility begins processing your MultiLoad job without an error indication. The configuration file is an optional feature of the MultiLoad utility, and its absence is not considered to be an error condition.

Invoking MultiLoad

Network Attached Systems: mload [PARAMETERS] < *scriptname* >*outfilename*

Channel-Attached MVS Systems: // EXEC TDSMLOAD MLPARM= [PARAMETERS]

Channel-Attached VM Systems: EXEC MLOAD [PARAMETERS]

Channel Parameter	Network Parameter	Description
BRIEF	-b	Reduces print output runtime to the least information required to determine success or failure.
CHARSET=*charsetname*	-c *charsetname*	Specify a character set or its code. Examples are EBCDIC, ASCII, or Kanji sets.
ERRLOG=*filename*	-e *filename*	Alternate file specification for error messages; produces a duplicate record.
"*multiload command*"	-r '*multiload cmd*'	Signifies the start of a MultiLoad job; usually a RUN FILE command that specifies the script file.
MAXSESS=*max sessions*	-M *max sessions*	Maximum number of MultiLoad sessions logged on.
MINSESS=*min sessions*	-N *min sessions*	Minimum number of MultiLoad sessions logged on.
	< *scriptname*	Name of file that contains MultiLoad commands and SQL statements.
	> *outfilename*	Name of output file for MultiLoad messages.

Application Utility Checklist

The facing page adds the MultiLoad capabilities to the checklist.

Automatic Restart – If the Teradata server restarts, MultiLoad will retry to connect to the Teradata database automatically and will automatically restart.

Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions	ALL	LIMITED	Yes (SE)	Yes (SE)	
DML Functions	ALL	INSERT	SELECT	INS/UPD/DEL	
Multiple DML	Yes	No	Yes	Yes	
Multiple Tables	Yes	No	Yes	Yes	
Protocol Used	SQL	FASTLOAD	EXPORT	MULTILOAD	
Conditional APPLY	No	No	No	Yes	
Data Conversion	Yes	1 per column	Yes	Yes	
Error Capture	No	Yes	N/A	Yes	
Error Limits	No	Yes	N/A	Yes	
User-written Routines	No	Yes	Yes	Yes	
Automatic Restart	No	Yes*	Yes	Yes	
Max Load Limit	No	Yes	Yes	Yes	
Support Environment (SE)	No	No	Yes	Yes	

Summary

The facing page summarizes some of the important concepts regarding this module.

Summary

- **Batch mode utility.**
  - **Supports up to five populated tables.**
  - **Multiple operations with one pass of input files.**
  - **Conditional logic for applying changes.**
- **Supports INSERTs, UPDATEs, DELETEs and UPSERTs; typically with batch inputs from a host or server data file.**
  - **Affected data blocks only written once.**
- **Full Restart capability.**
- **Error reporting via error tables.**
- **Support for INMODs.**

Module 37: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 37: Review Questions

Answer True or False.

1. True or False. With MultiLoad, you can import data from the host into populated tables.
2. True or False. MultiLoad cannot process tables with USIs or Referential Integrity defined.
3. True or False. MultiLoad allows changes to the value of a table's primary index.
4. True or False. MultiLoad allows you to change the value of a column based on its current value.
5. True or False. MultiLoad permits non-exclusive access to target tables from other users except during Application Phase.

Match the MultiLoad Phase in the first column to its corresponding task in the second column.

1. ___ Preliminary	a. Acquires or creates Restart Log Table.
2. ___ DML Transaction	b. Locks are released.
3. ___ Acquisition	c. Applies (loads) data to the work tables.
4. ___ Application	d. Execute mload for each target table as a single multi-statement request.
5. ___ Cleanup	e. Stores DML steps in work tables

Lab Exercise 37-1

To create the **data37_1** file, execute the following statements in BTEQ.

```
.EXPORT DATA FILE=data37_1, CLOSE;
 EXEC AP.Lab37_1;
.EXPORT RESET;
```

The size of data37_1 should be 2400 bytes long after executing the macro AP.Lab37_1.

The format of data37_1 is:

Char(1)	Integer
A	PI_value for Accounts table
C	PI_value for Customer table
T	PI_value for Trans table

Example of data in data37_1:

Table_code	PI_value
A	20024001
:	:
C	2001
:	:
T	20024002
T	20024003
:	:

The control letter (A, C, or T) must be in upper-case letters in the APPLY statements. The Integer value represents the Primary Index Value that you will use to delete a row in the appropriate table. If the input record contains a code of A, delete a row from the Accounts table using the Integer value as the PI value. Likewise, if the input record contains a code of C, delete a row from the Customer table. If the input record contains a code of T, delete a row from the Trans table.

A technique that can be used to create Linux scripts without using vi or vim is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where Linux is running and …

3. **cat > lab37_13.mld** (or whatever filename you wish)

   Use the mouse to choose the **Edit ➔ Paste** function
   To exit the cat command, press either the DELETE key or CNTL C.

Lab Exercise 37-1

Lab Exercise 37-1

Purpose

In this lab, you will use MultiLoad to delete rows from your three tables. An input file will be created which will contain a control letter (A - Accounts, C - Customer, and T - Trans) followed by a primary index value for the appropriate table.

What you need

Your three tables with two hundred (200) rows in each.

Tasks

1. Prepare the data file by executing the macro AP.Lab37_1. Export your data to a file called *data37_1*.
2. Prepare your tables by doing the following:
   a. Issue a Delete All command on each of your tables.
   b. Execute the following commands which will load 200 rows into each of the tables:

```
INSERT INTO Accounts   SELECT * FROM AP.Accounts   WHERE Account_Number < 20024201;
INSERT INTO Customer   SELECT * FROM AP.Customer   WHERE Customer_Number < 2201;
INSERT INTO Trans      SELECT * FROM AP.Trans      WHERE Account_Number < 20024201;
```

3. Prepare your MultiLoad script to Delete Rows from each of the tables depending on the incoming code (A, C, or T) from *data37_1*. This job should delete 100 rows from each of the three tables.
4. Check your results by doing a SELECT COUNT(*) on each of your tables.

Notes

Module 38

A MultiLoad Application

After completing this module, you will be able to:

- **Describe the tables involved in a MultiLoad job.**
- **Set error limits as a record value or as a percentage of loaded rows.**
- **Specify a checkpoint interval.**
- **Redefine input record layout.**

Notes

Table of Contents

New Accounts Application – Description 38-4
New Accounts Application Script (1 of 3) 38-6
New Accounts Application Script (2 of 3) 38-8
New Accounts Application Script (3 of 3) 38-10
.BEGIN IMPORT Task Command 38-12
Work Tables 38-14
Error Tables 38-16
ERRLIMIT 38-18
CHECKPOINT 38-20
More .BEGIN Parameters 38-22
SESSIONS *max min* 38-22
More .BEGIN Parameters: AMPCHECK 38-24
DELETE Task Command 38-26
.LAYOUT and .TABLE 38-28
.LAYOUT Parameters — CONTINUEIF 38-30
.LAYOUT Parameters — INDICATORS 38-32
.FIELD and .FILLER 38-34
Performance Considerations 38-34
.LAYOUT Command — Examples 38-36
Redefining the Input – Example 38-38
The .DML Command Options 38-40
The .DML Command Options (cont.) 38-42
MultiLoad Statistics 38-44
Summary 38-46
Module 38: Review Questions 38-48
Lab Exercise 38-1 38-50
Lab Exercise 38-2 38-52

New Accounts Application – Description

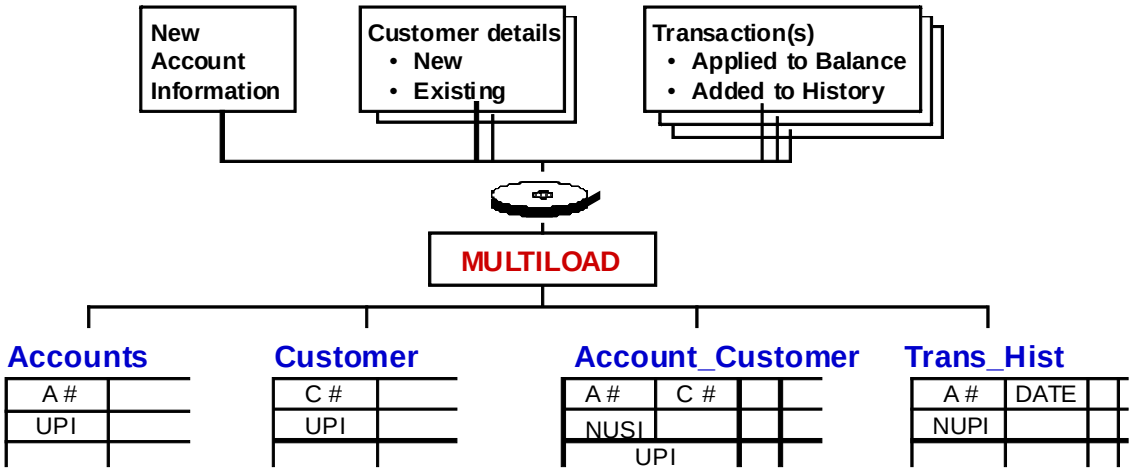
The facing page diagrams an example of a bank procedure for customers opening new accounts. The application must be able to handle:

- New customers with new accounts
- Existing customers who open new accounts.

You need to do an UPSERT (an UPDATE or an INSERT). The tasks are to:

- Execute an UPDATE to a Customer Row. (This UPDATE actually only checks for the existence of the Customer Row.)
- If this fails, INSERT a new Customer Row.

New Accounts Application – Description



- Each New Account requires an *INSERT* into the Accounts table and an *INSERT* into Account_Customer.
- An Account can be opened for new or pre-existing customer(s) – *UPSERT* to the Customer table.
- Each New Account will probably require an opening transaction – *UPDATE* to Accounts (balance) and *INSERT* to Trans_History.

New Accounts Application Script (1 of 3)

The entire input script for this application is shown on the next few pages. The first page shows the Support Environment statements to define the LOGTABLE and LOGON.

The .BEGIN IMPORT statement defines the tables used in this example.

The .LAYOUT statement is followed by the definition of .FIELD or .FILLER statements.

These statements are discussed in more detail later in this module.

New Accounts Application Script (1 of 3)

```
.LOGTABLE  ACT_Logtable1_mld;
.LOGON tdpid/username, password;
.BEGIN IMPORT MLOAD TABLES Accounts, Account_Customer, Customer, Trans_Hist ;
.LAYOUT    Record_Layout;
.FILLER    in_Field_Indicator        1          CHAR(1) ;
.FIELD     in_Account_Number         2          INTEGER;
.FIELD     in_Number                 *          INTEGER;
.FIELD     in_Street                 *          CHAR(25);
.FIELD     in_City                   *          CHAR(20);
.FIELD     in_State                  *          CHAR(2);
.FIELD     in_Zip_Code               *          INTEGER;
.FIELD     in_Balance_Forward        *          INTEGER;
.FIELD     in_Balance_Current        *          INTEGER;
.FIELD     in_Customer_Number        2          INTEGER;
.FIELD     in_Last_Name              *          CHAR(25);
.FIELD     in_First_Name             *          CHAR(20);
.FIELD     in_Social_Security        *          INTEGER;
.FIELD     in_AC_Account_Number      2          INTEGER;
.FIELD     in_AC_Customer_Number     *          INTEGER;
.FIELD     in_Trans_Number           2          INTEGER;
.FIELD     in_Trans_Account_Number   *          INTEGER;
.FIELD     in_Trans_ID               *          INTEGER;
.FIELD     in_Trans_Amount           *          DECIMAL(10,2);
```

New Accounts Application Script (2 of 3)

The facing page shows all the .DML statements that define labels for the one or more DML commands that follow.

The sequence in which these commands are performed is indicated by the APPLY clauses in the .IMPORT statement. APPLY clauses may contain conditions to be met before the command is applied to the input data. The .IMPORT statement also defines the name of the file that contains the input data and the LAYOUT for that file. The LAYOUT is matched to a defined .LAYOUT statement by the logical name that was associated with it.

The MultiLoad script is terminated with the .END MLOAD statement.

New Accounts Application Script (2 of 3)

```
.DML  LABEL  New_Accounts;
   INSERT  INTO  Accounts VALUES
      (:in_Account_Number, :in_Number, :in_Street, :in_City ,:in_State, :in_Zip_Code,
       :in_Balance_Forward, :in_Balance_Current );
.DML  LABEL  New_Acct_Customer;
   INSERT  INTO  Account_Customer VALUES
      (:in_AC_Account_Number, :in_AC_Customer_Number);
.DML  LABEL  Upsert_Customer  DO INSERT FOR MISSING UPDATE ROWS;
   UPDATE  Customer  SET Last_Name = :in_Last_Name
      WHERE Customer_Number = :in_Customer_Number;
   INSERT  INTO Customer VALUES
      (:in_Customer_Number, :in_Last_Name, :in_First_Name, :in_Social_Security);
.DML  LABEL  Trans_Update;
   INSERT  INTO Trans_Hist VALUES
      (:in_Trans_Number, DATE, :in_Trans_Account_Number, :in_Trans_ID, :in_Trans_Amount );
   UPDATE  Accounts
      SET     Balance_Current =Balance_Current  + :in_Trans_Amount
      WHERE   Account_Number = :in_Trans_Account_Number;
.IMPORT INFILE datafile4  LAYOUT Record_Layout
   APPLY New_Accounts          WHERE in_Field_Indicator = 'A'
   APPLY New_Acct_Customer     WHERE in_Field_Indicator = 'B'
   APPLY Upsert_Customer       WHERE in_Field_Indicator = 'C'
   APPLY Trans_Update          WHERE in_Field_Indicator = 'T';
.END MLOAD;
```

New Accounts Application Script (3 of 3)

The facing page shows the support environment commands to check MultiLoad support environment counts and COLLECT STATISTICS on the tables as part of the utility job.

System variables that are available with MultiLoad include:

&SYSDELCNT(n)	Delete Count	Ex., &SYSDELCNT1, …
&SYSINSCNT(n)	Insert Count	Ex., &SYSINSCNT1, ...
&SYSUPDCNT(n)	Update Count	
&SYSETCNT(n)	Error Table Count	
&SYSUVCNT(n)	Uniqueness Violation Count	
&SYSRCDCNT(n)	Count of import records read	
&SYSRJCTCNT(n)	Count of records rejected from import file	

Note: n is 1 to 5

A .LOGOFF; statement terminates the Support Environment session.

New Accounts Application Script (3 of 3)

```
.IF (&SYSINSCNT1 > 10000 OR &SYSUPDCNT1 > 10000) THEN;
   COLLECT STATISTICS ON Accounts;
.ENDIF;

.IF (&SYSINSCNT2 > 10000) THEN;
   COLLECT STATISTICS ON Account_Customer;
.ENDIF;

.IF (&SYSINSCNT3 > 5000 OR &SYSUPDCNT3 > 5000) THEN;
   COLLECT STATISTICS ON Customer;
.ENDIF;

.IF (&SYSINSCNT4 > 100000) THEN;
   COLLECT STATISTICS ON Trans_Hist;
.ENDIF;

.LOGOFF;
```

MultiLoad environment variables can be checked to optionally COLLECT STATISTICS as part of the job.

&SYSDELCNT_	where _ is 1 to 5
&SYSINSCNT_	"
&SYSUPDCNT_	"
&SYSETCNT_	"
&SYSUVCNT_	"
&SYSRCDCNT_	"
&SYSRJCTCNT_	"

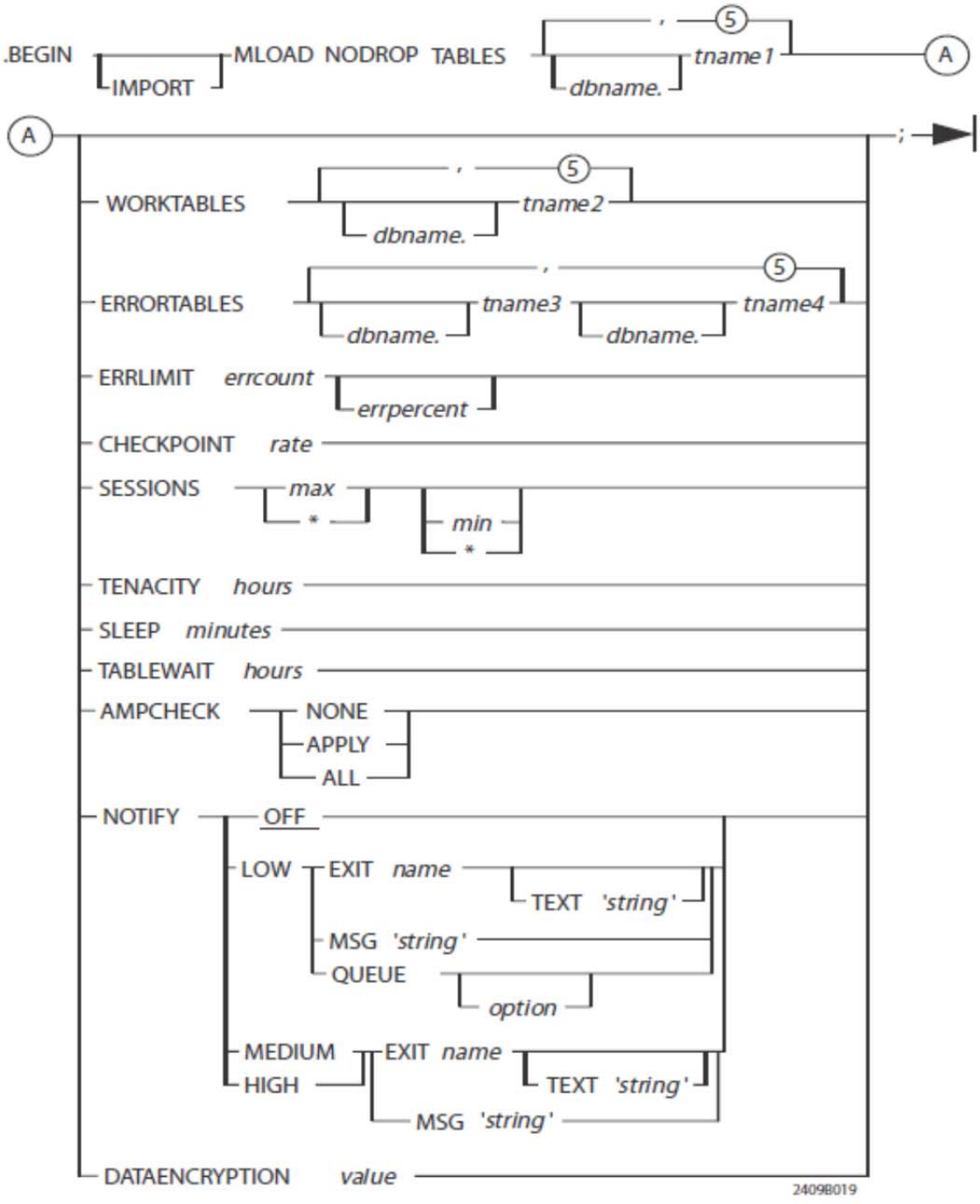
Note:

```
.BEGIN IMPORT MLOAD TABLES Accounts, Account_Customer, Customer, Trans_Hist ;
```

1st table, 2nd table, 3rd table, 4th table

.BEGIN IMPORT Task Command

The complete format of the .BEGIN IMPORT MLOAD command is:



TABLEWAIT option – This option (*specified in hours*) is the number of hours that Teradata MultiLoad continues trying to start Teradata MultiLoad when one of the target tables is being loaded by some other job (e.g., FastLoad or MultiLoad).

.BEGIN IMPORT Task Command

- Specifies the tables and optionally the work and error tables used in this MultiLoad job.
- Also used to specify miscellaneous MultiLoad options such as checkpoint, sessions, etc.

```
.BEGIN [IMPORT] [NODROP] MLOAD
    TABLES          tname1, tname2, ...
    WORKTABLES      wt_table1, wt_table2, …
    ERRORTABLES     et_ table1 uv_table1, et_ table2 uv_table2, et_ table3 uv_table3, …
    ERRLIMIT        errcount  [errpercent]
    CHECKPOINT      rate                          (Default – 15 min.)
    SESSIONS        limit                         (Default – 1 per AMP + 2)
    TENACITY        hours                         (Default – 4 hours)
    SLEEP           minutes                       (Default – 6 minutes)
    AMPCHECK        NONE | APPLY | ALL
    NOTIFY          OFF | LOW | MEDIUM | HIGH . . .
    ;
.END MLOAD;                                       Note: tname = [dbname.]tablename
```

NODROP (Teradata 14.0 option)

NODROP – MultiLoad will NOT drop error tables even if they are empty at the end of the job.

Work Tables

There must be one work table for each data table.

If WORKTABLES are not defined, they are created with the table name prefixed by 'WT_'.

If the data table is Fallback protected, then the associated Work table is Fallback protected.

If the data table is not Fallback protected, then the associated Work table is not Fallback protected.

Work Tables

IMPORT Task: WORKTABLES *wt_table1, wt_table2, …*

DELETE Task: WORKTABLES *wt_table1*

.BEGIN parameters

- Default worktable is create in user's default database and work table is named *WT_TableName*.
- Alternative may be specified as *DataBaseName.WorkTableName*.
- There must be one work table defined for each data table.

This example utilizes variables to define the database and table names to use with MultiLoad and a different database for the work and error tables.

Example:

```
.SET DBASE          TO   'Payroll';
.SET DBASE_UTIL     TO   'UtilityDB';
.SET TABLE1         TO   'Employee';
.SET TABLE2         TO   'Paycheck';

.BEGIN MLOAD
    TABLES          &DBASE..&TABLE1, &DBASE..&TABLE2
    WORKTABLES      &DBASE_UTIL..WT_&TABLE1
                    ,&DBASE_UTIL..WT_&TABLE2
    ERRORTABLES     &DBASE_UTIL..ET_&TABLE1  &DBASE_UTIL..UV_&TABLE1
                    ,&DBASE_UTIL..ET_&TABLE2  &DBASE_UTIL..UV_&TABLE2
    . . . ;
```

For Employee, the work table is resolved to UtilityDB.WT_Employee.
For Paycheck, the work table is resolved to UtilityDB.WT_Paycheck.

Error Tables

ERRORTABLES default to an 'ET_' or 'UV_' prefix and the Table name. There will be two error tables for each user table.

The “ET_tablename” error table is referred to as the Acquisition Phase Error Table. Errors that occur in the Acquisition Phase are placed in this table.

The “UV_tablename” error table is referred to as the Application Phase Error Table. Errors that occur in the Application Phase are placed in this table.

Regardless if the data table is Fallback or No Fallback, the ET and UV error tables are automatically Fallback protected.

Error Tables

ERRORTABLES *et_table1 uv_table1, et_table2 uv_table2, ...*

.BEGIN parameters

- **Error table (ET)**
  - default is the user's database and the table is named ***ET_Tablename***.
  - contains any errors that occur in the **Acquisition Phase**.
  - contains primary index overflow errors that occur in the Application phase.
- **Uniqueness Violation (UV) table (effectively also an error table)**
  - default is the user's database and the table is named ***UV_Tablename***.
  - contains **Application Phase** errors (uniqueness violations, constraint errors, and overflow errors on columns other than the primary index.

Example:

```
.SET DBASE          TO   'Payroll';
.SET DBASE_UTIL     TO   'UtilityDB';
.SET TABLE1         TO   'Employee';
.SET TABLE2         TO   'Paycheck';

.BEGIN MLOAD
    TABLES          &DBASE..&TABLE1, &DBASE..&TABLE2
    WORKTABLES      &DBASE_UTIL..WT_&TABLE1
                   ,&DBASE_UTIL..WT_&TABLE2
    ERRORTABLES     &DBASE_UTIL..ET_&TABLE1  &DBASE_UTIL..UV_&TABLE1
                   ,&DBASE_UTIL..ET_&TABLE2  &DBASE_UTIL..UV_&TABLE2
    . . . ;
```

For Employee, the error tables are resolved to UtilityDB.ET_Employee and UtilityDB.UV_Employee.
For Paycheck, the error tables are resolved to UtilityDB.ET_Paycheck and UtilityDB.UV_Paycheck.

ERRLIMIT

The ERRLIMIT option allows you to specify an error count and, optionally, an error percent.

The specification of an *error count* indicates the approximate number of errors (not uniqueness violations) that should cause the MultiLoad to stop processing (and abort).

The additional definition of an *error percent* indicates that you wish to stop processing when a percentage of errors has been detected after an approximate number of records have been transmitted.

ERRLIMIT

.BEGIN parameters

ERRLIMIT *ErrCount*

Without ERRPERCENT:

- Specifies approximate number of data errors permitted during Acquisition.
- Does not count Uniqueness violations.

ERRLIMIT *ErrCount ErrPercent*

With ERRPERCENT:

- Specifies a percentage of data errors after an approximate number of records has been transmitted.

Example: **ERRLIMIT 10000 5**

In this example, after processing 10,000 input records, the system looks for an error rate of 5%.

CHECKPOINT

The CHECKPOINT option defines the checkpoint rate as either the *number of records* or a *time interval*.

If you specify a CHECKPOINT *rate* of 60 or more, a checkpoint operation occurs after each multiple of that number of records is processed.

If you specify a CHECKPOINT *rate* of less than 60, a checkpoint operation occurs at the specified frequency, in minutes.

The default is for MultiLoad to perform a CHECKPOINT every 15 minutes. If you do not use the CHECKPOINT *rate* specification, the MultiLoad utility performs a checkpoint operation at the default rate — every 15 minutes and at the end of each phase.

Note: Specifying a CHECKPOINT rate of 0 inhibits the checkpoint function—the MultiLoad utility does not perform any checkpoint operations during the import task.

CHECKPOINT

.BEGIN parameters

- Rate may be specified in the Acquisition Phase of a complex IMPORT task as:
  - A *number of incoming records* (exact count; not less than 60)
  - A *time interval in minutes* (approximate; less than 60)
- If no CHECKPOINT value is specified MultiLoad will checkpoint every 15 minutes, and at the end of each Phase. The default is 15 minutes.

Example 1: **CHECKPOINT 30**

In this example, a 30-minute time interval is specified.

Example 2: **CHECKPOINT 100000**

In this example, a checkpoint after 100,000 input records is specified.

More .BEGIN Parameters

The facing page specifies additional parameters you can use with the .BEGIN statement.

SESSIONS *max min*

max maximum number of MultiLoad sessions that will be logged on when you enter a LOGON.

- The *max* specification must be greater than zero.
- If you specify a SESSIONS *max* value that is larger than the number of available AMPs, the MultiLoad utility limits the sessions to one per working AMP.
- The default maximum is one session for each AMP.
- Using the asterisk character as the *max* specification logs on for the maximum number of sessions—one for each AMP.

min optional, the minimum number of sessions required to run the job.

- The *min* specification must be greater than zero.
- The default minimum, if you do not use the SESSIONS option or specify a min value, is 1.
- Using the asterisk character as the *min* specification logs on for at least one session, but less than or equal to the max specification.

More .BEGIN Parameters

.BEGIN parameters

SESSIONS

SESSIONS 48 32

- Used to specify the maximum, and optionally, minimum sessions generated by MultiLoad.

TENACITY

TENACITY 10

- Number of hours MultiLoad will try to establish a connection to the system.
- The default is 4 hours.

SLEEP

SLEEP 3

- Number of minutes MultiLoad waits before retrying a logon; must be greater than 0.
- The default is 6 minutes.

NOTIFY

NOTIFY OFF

Note: The MultiLoad manual specifies in detail which events are associated with each level.

- NOTIFY LOW for initialize event and CLIv2 errors.
- NOTIFY MEDIUM for the most significant events.
- NOTIFY HIGH for every MultiLoad event that involves an operational decision point.
- NOTIFY OFF suppresses the notify option.

More .BEGIN Parameters: AMPCHECK

Use the AMPCHECK option to specify what you want to occur in the event of an AMP being down.

The default (APPLY) specifies that if AMPs are offline, you want MultiLoad to continue processing every phase except the Application phase as long as all tables involved in the MultiLoad job have been defined with FALLBACK protection.

More .BEGIN Parameters: AMPCHECK

AMPCHECK NONE | APPLY | ALL

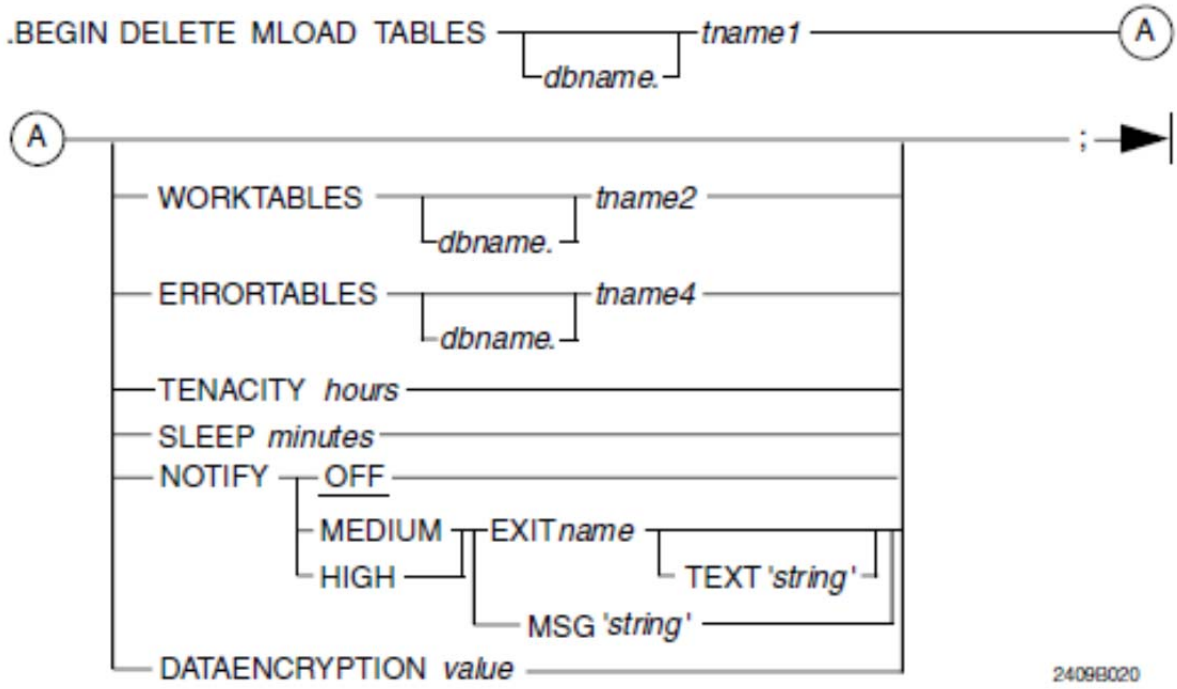
.BEGIN parameters

- **NONE**
  - MultiLoad will not perform an AMPCHECK.
  - It *will proceed* if AMPs are offline, provided all target tables are FALLBACK.
- **APPLY**
  - MultiLoad will continue in all phases *except* the Application phase with AMPs offline, provided all target tables are FALLBACK.
  - This is the default.
- **ALL**
  - MultiLoad *will not proceed* with down AMPs, regardless of the protection-type of the target tables.
  - Most conservative option.

DELETE Task Command

The facing page summarizes the options for the DELETE task. Note that it has many of the same options as the IMPORT task.

The complete format of the .BEGIN DELETE MLOAD command is:



DELETE Task Command

- Specifies the table and optionally the work and error tables used in this MultiLoad Delete task.
- Also used to specify miscellaneous MultiLoad options such as tenacity, sleep, etc.

```
.BEGIN  DELETE  MLOAD
        TABLES          tname1
        WORKTABLES      wt_table1
        ERRORTABLES     et_ table1
        TENACITY        hours                   (Default – 4 hours)
        SLEEP           minutes                 (Default – 6 minutes)
        NOTIFY          OFF | LOW | MEDIUM | HIGH . . .
        ;
.END MLOAD;
```

.LAYOUT and .TABLE

The layout is given a name that is referenced in the .IMPORT statement.

The .LAYOUT statement can be used multiple times within the same MultiLoad job to indicate files with different fields. The layout name specifies the layout that should be used for the APPLY clauses which follow.

The .LAYOUT statement is always followed by either a .TABLE or .FIELD/.FILLER statement. If .TABLE is used, the input data type and fields are the same as an existing database table definition.

.LAYOUT and .TABLE

.LAYOUT

- Must appear between the .BEGIN IMPORT MLOAD command and the applicable .IMPORT command.
- Must be immediately followed by .TABLE, .FIELD, or .FILLER commands.

.TABLE

- The input fields are defined with the same name, data type, and order of an existing table.

Format:

```
.LAYOUT   layoutname
          ( CONTINUEIF position = variable )   ;
          ( INDICATORS                     )

.TABLE    tableref  ;
```

.LAYOUT Parameters — CONTINUEIF

Input data records may be concatenated if the record to be concatenated follows the initial data portion. For example, a data set or file contains records that are currently divided into two or three parts (records), but need to be treated as one record for MultiLoad. The CONTINUEIF option is used to indicate the value to check for when concatenating records.

The CONTINUEIF option must be of the type CHARACTER and may be multiple characters.

The general format is:

CONTINUEIF *position = value*

position

an unsigned integer (never an asterisk) that specifies the starting character position of the field of every input record that contains the continuation indicator. **Note:** The position is relative to the first character position of the input record or input record fragment, which is always position 1.

value

the continuation indicator specified as a character constant or a string constant. The MultiLoad utility uses the length of the constant as the length of the continuation indicator field.

Miscellaneous Notes:

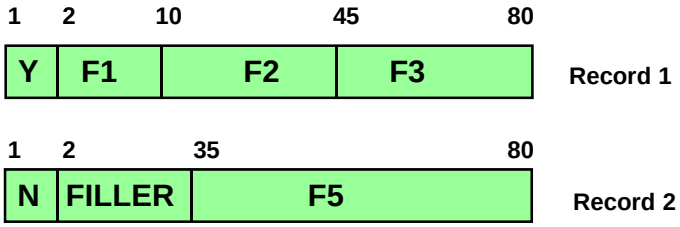
- The ***value*** is case sensitive – always specify the correct character case for this parameter.
- If the conditional phrase is “true,” then the MultiLoad utility forms a single record to be sent to the Teradata RDBMS by concatenating the next input record at the end of the current record. (The current record is the one most recently obtained from the external data source.)
- If the conditional phrase is “false”, then the MultiLoad utility sends the current input record to the Teradata RDBMS either by itself or as the last of a sequence of concatenated records.
- **Note:** Regardless of whether the condition evaluates to true or false, the MultiLoad utility removes the tested string (the continuation indicator field) from each record.

.LAYOUT Parameters — CONTINUEIF

CONTINUEIF

- Record 1 and the immediately following Record 2 each represent a part of the input record.
- Continuation Character Field is defined beginning in character position 1.
- This option may be needed with legacy systems where input records are limited to 80 characters (carryover from punched card era of computing).

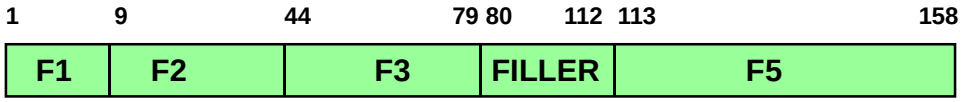
Physical Input Records



MultiLoad .LAYOUT Definition

```
.LAYOUT Record_Layout   CONTINUEIF 1= 'Y';
```

Logical Input Transaction



Note: F1 begins in character position 1 and Filler immediately follows F3.

.LAYOUT Parameters — INDICATORS

Use the INDICATORS parameters to handle nulls.

Miscellaneous Notes:

- When you use the INDICATORS specification, the MultiLoad utility sends *all of* the FIELD commands, including redefines, to the Teradata Database.
- **Caution:** Inappropriate INDICATORS specifications can corrupt the target table on the Teradata Database.
  - If INDICATORS is specified in the LAYOUT command and the data file does not contain indicator bytes in each record, the target table is loaded with incorrect data.
  - Conversely, if INDICATORS is not specified and the data file contains indicator bytes in each record, the target table also is corrupted.
- Always make sure that your INDICATORS specifications match the mode of the data you are sending to the Teradata Database.
- **Note:** INDICATORS processing is done only after any CONTINUEIF processing is completed for a record.

.LAYOUT Parameters — INDICATORS

INDICATORS

- The INDICATORS contain bits that, when equal to 1, represent a null data field.

Physical Input Records

MultiLoad .LAYOUT Definition

```
.LAYOUT Record_Layout   INDICATORS;
```

.FIELD and .FILLER

When .FIELD is used, you specify a name for the field, the starting position or asterisk (*), the data type, and possibly options governing the handling of the input data.

By specifying a .FILLER, you define input data that will not be sent to the database table. A .FILLER statement requires a name, a starting position or asterisk, and the data type.

Performance Considerations

The majority of client processing during a MultiLoad job occurs when it is processing its input rows. The most efficient means of sending the row to the Teradata Database would be a bulk move of the input row to the output row.

However, there are many cases where fields need to be evaluated and field data may need to be individually moved from the input to the output row. Note, however, that performance is affected whenever a field needs to be evaluated or individually moved.

The need for moving individual field data from the input to the output row occurs for any of the following scenarios:

- DROP syntax on FIELD statements
- FILLER fields
- Concatenated fields
- Complex layout (first field is variable-length field, redefinition of field positions)

In addition to the above scenarios, variable length fields, NULLIF in the layout, and APPLY WHERE clauses might require additional CPU consumption.

If possible, try to avoid using the above options to improve MultiLoad performance.

.FIELD and .FILLER

```
.FIELD  fieldname { startpos datadesc } || fieldexp
    [ NULLIF  nullexpr ]
    [ DROP {LEADING / TRAILING } { BLANKS / NULLS }
    [ [ AND ] {TRAILING / LEADING }  { NULLS / BLANKS } ] ] ;

.FILLER  [ fieldname ] startpos datadesc ;
```

.FIELD

- Input fields supporting redefinition and concatenation.

  Startpos identifies the start of a field relative to 1.

  Fieldexpr specifies a concatenation of fields in the format:

  fieldname1 || fieldname2 [|| fieldname3 …]

  The option DROP LEADING / TRAILING BLANKS / NULLS is applicable only to character datatypes, and is sent as a VARCHAR with a 2-byte length field.

.FILLER

- Identifies data NOT to be sent to the Teradata database.

.LAYOUT Command — Examples

The facing page shows two examples using .LAYOUT: one with .TABLE and one with .FIELD and .FILLER.

.LAYOUT Command — Examples

Example 1:

```
.LAYOUT transrecord
   CONTINUEIF 7 = 'ABC'
   INDICATORS;

.FIELD     field1   1   char(5)     NULLIF field1 = 'AAAAA' DROP LEADING BLANKS;
.FILLER    field2   *   char(1);
.FIELD     field3   *   char(3);
.FIELD     field4                  field2 || '&' || field3 ;
```

Example 2:

```
.LAYOUT emp_record;
.TABLE Employee;
```

Note:

Employee is an existing table whose column names and data descriptions are assigned, in the order defined, to fields of the input data records.

Redefining the Input – Example

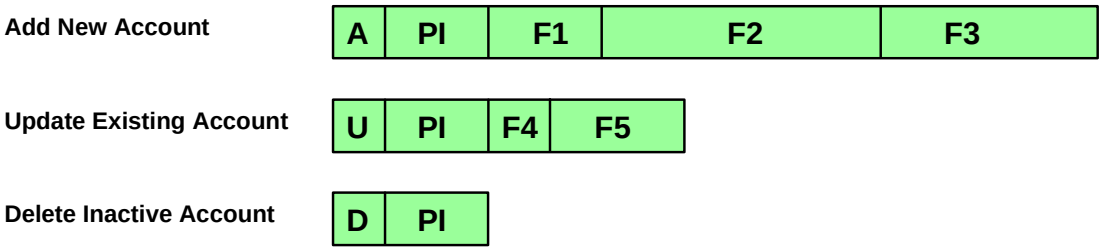
The input file may contain different types of records. Fields within these records start from the beginning of the record (position 1). Subsequent fields are indicated by an asterisk or by a starting character position.

By indicating the starting position of the field within the record, you can redefine the record. Remember, the asterisk (*) refers to the next position after the preceding field.

In the example on the following page, F4 starts in position 6 (immediately following the '.FILLER trans_type' and '.FIELD Pl'). F5 follows F4.

Redefining the Input — Example

- A bank loads daily transactions sequentially on a tape for batch processing by MultiLoad.
- An input data record might be an Add, Update or Delete, each of which has a different length and contains different fields, as illustrated:




```
.LAYOUT   Record_Layout ;
.FILLER   trans_type      1    CHAR(1)  ;
.FIELD    PI              2    INTEGER  ;
.FIELD    F1              *    INTEGER  ;
.FIELD    F2              *    CHAR(25) ;
.FIELD    F3              *    CHAR(20) ;
.FIELD    F4              6    CHAR(2)  ;
.FIELD    F5              *    INTEGER  ;
```

Note:
FILLER data is not placed into ML work tables.

The .DML Command Options

The DML command provides labels for one or more DML statements (INSERT, UPDATE, or DELETE). The format of this command is shown next along with the options on this command.

You have already seen examples of how some of these options can be used to direct the processing of an UPSERT. The MARK option indicates that duplicate or missing rows should be recorded in one of the error tables.

MARK is the default for INSERT, UPDATE, and DELETE. If an error occurs in the Application Phase (e.g., uniqueness violation) with MARK specified, then the duplicate row is placed in to the UV_errtable. If a row is missing with an UPDATE or DELETE, then that transaction is also placed in the UV_errtable.

IGNORE is the default for UPSERT if the UPSERT is successful. If an UPSERT fails because of a constraint violation, the error is placed in the UV_errtable.

For import tasks, you can specify as many as five distinct error treatment options with one DML LABEL command. For example:

```
.DML LABEL COMPLEX
  IGNORE DUPLICATE INSERT ROWS
  MARK DUPLICATE UPDATE ROWS
  MARK MISSING UPDATE ROWS
  MARK MISSING DELETE ROWS
  DO INSERT FOR MISSING UPDATE ROWS;
```

Notes:

- If a uniqueness violation occurs with MARK specified, the duplicate rows go to the uniqueness violation table.
- IGNORE DUPLICATE ROWS does not apply if there are any unique indexes in the table.
- In the case of an *upsert* operation, both the insert and update portions must fail for an error to be recorded. If MARK MISSING UPDATE ROWS, then "marked" rows for the missing update operations then have nulls for the target table columns.
- If you do not specify either INSERT or UPDATE with DUPLICATE, then the MARK or IGNORE specification applies to both insert and update operations.
- Similarly, if you do not specify either UPDATE or DELETE with MISSING, then the MARK or IGNORE specification applies to both update and delete operations.
- MARK is the default for all actions except MISSING UPDATE for an *upsert* operation.

The .DML Command Options

Defines Labels along with Error Treatment conditions for one or more following INSERTs, UPDATEs or DELETEs to be applied under various conditions:

```
.DML LABEL  Labelname

[ { MARK | IGNORE } DUPLICATE [INSERT  | UPDATE ] ROWS
  { MARK | IGNORE } MISSING   [UPDATE | DELETE ]

    DO INSERT FOR  [MISSING UPDATE] ROWS ] ;
```

MARK or IGNORE
Whether or not to record duplicate or missing INSERT, UPDATE, OR DELETE rows into the UV_error_table and continue processing.

Operation:	Default if MARK/IGNORE is not used:
INSERT (Duplicate violation)	Marked in UV_tablename
UPDATE (Duplicate row violation)	Marked in UV_tablename
UPDATE (Fails - missing row)	Marked in UV_tablename
DELETE (Fails - missing row)	Marked in UV_tablename
UPSERT (If successful)	Ignored
UPSERT (Fails)	Mark failure of INSERT in UV_tablename

Example of UPSERT failure:

1. PI value doesn't exist, so UPDATE can't occur.
2. INSERT fails because of check violation - e.g., can't put character data in a numeric field.

The .DML Command Options (cont.)

The syntax of the .DML Label command is repeated on the facing page for your convenience.

The default for an UPSERT operation is to not mark missing update rows.

When the MARK MISSING UPDATE ROWS is used with an UPSERT, this will list (in the UV_errtable) data rows that can't be updated (the row doesn't exist with the PI value). If the insert also fails (e.g., constraint violation), the insert record is also marked in the UV_errtable. In this case, one record can cause 2 rows to go into the UV_errtable – one for the missing update and one for the insert failure.

The .DML Command Options (cont.)

```
.DML LABEL  Labelname
[{MARK | IGNORE} DUPLICATE  [INSERT  | UPDATE]  ROWS
 {MARK | IGNORE} MISSING    [UPDATE | DELETE]
    DO INSERT FOR  [MISSING UPDATE] ROWS ] ;
```

DO INSERT FOR MISSING UPDATE — Key statement that indicates an UPSERT. An SQL UPDATE followed by an SQL INSERT is required.

Example 1:

```
.DML LABEL Action1    DO INSERT FOR MISSING UPDATE ROWS;
```

The default for an UPSERT operation is to not mark missing update rows.

Example 1:

```
.DML LABEL Action2    MARK MISSING UPDATE ROWS
                      DO INSERT FOR MISSING UPDATE ROWS;
```

When the MARK MISSING UPDATE ROWS is used with an UPSERT, this will place (in the UV_table) data rows that can't be updated (no PI). If the insert also fails, the insert record is also marked in the UV_table.

MultiLoad Statistics

Statistics indicating the number of records processed by each DML for *each table* is reported at the end of the Application Phase. The facing page shows an example of this output.

MultiLoad Statistics

At the end of the application phase, MultiLoad provides statistical information.

```
****06:06:41 UTY1803 Import Processing Statistics
                                                  Total
                                     Import 1     Thus far
Candidate Records considered . . . .  200000      200000
Apply conditions satisfied  . . . . .  200000      200000

****06:18:34 UTY0818 Statistics for table ACCOUNTS:
          Inserts:          50000
          Updates:          50000
          Deletes:          0
****06:18:35 UTY0818 Statistics for table ACCOUNT_CUSTOMER:
          Inserts:          50000
          Updates:          0
          Deletes:          0
****06:18:35 UTY0818 Statistics for table CUSTOMER:
          Inserts:          25000
          Updates:          25000
          Deletes:          0
****06:18:35 UTY0818 Statistics for table TRANS_HIST:
          Inserts:          50000
          Updates:          0
          Deletes:          0
```

Summary

The facing page summarizes some of the important concepts regarding the MultiLoad utility.

Summary

- On the .BEGIN statement, optionally, the names of work and error tables can be specified.
- You can:
  - Specify error limits and checkpoints.
  - Limit sessions.
  - Designate time allowed for connection.
  - Specify retry time limit.
  - Designate the level of notification you prefer.
  - Designate how MultiLoad will proceed if AMPs are offline.
- .LAYOUT parameters define the data format.
- .DML commands define Labels and Error Treatment conditions for one or more operations.
- You can use FastLoad or MultiLoad INMODs.

Module 38: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 38: Review Questions

1. **Complete the BEGIN statement to accomplish the following:**
   - **Specify an error limit count of 200,000 and an error percentage of 5%.**
   - **Specify a checkpoint at 500,000 records.**
   - **Request 16 sessions, but allow the job to run with only 8.**
   - **Set the number of hours to try to establish connection as 6.**

```
.LOGTABLE RestartLog_mld;
.LOGON _____________________;
.BEGIN  [IMPORT]  MLOAD TABLES  Trans_Hist




;
.END MLOAD ;
```

Lab Exercise 38-1

The size of data38_1 should be 26,700 bytes.

The execution of the macro AP.Lab38_1 will cause 300 rows to be created in data38_1. 100 of these records will start with an A, 100 will start with a C, and 100 will start with a T.

The format of data38_1 is:

Char(1)	Multiple columns
A	Data row for Accounts table
C	Data row for Customer table
T	Data row for Trans table

Example of data in data38_1:

Char(1)	Data
A	20024001 279 Beach Blvd Los Angeles CA 90066 394.00 233.00
:	:
C	2001 Atchison Roger 213598761
:	:
T	1 2002-02-08 20024002 413 -0.08
T	2 2002-02-08 20024003 414 -0.15
:	:

The control letter (A, C, or T) must be in upper-case letters in the APPLY statements. If the input record contains a code of A, insert a row into the Accounts table. If the input record contains a code of C, insert a row into the Customer table. If the input record contains a code of T, insert a row into the Trans table.

Important note: The Trans_Date is output as a date in ANSI date ('YYYY-MM-DD') format of CHAR(10). Therefore, define the Tran_Date as CHAR(10) for input.

A technique that can be used to create Linux scripts without using vi or vim is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where Linux is running and …

3. **cat > lab38_12.mld** (or whatever filename you wish)

   Use the mouse to choose the **Edit ➔ Paste** function
   To exit the cat command, press either the DELETE key or CNTL C.

Lab Exercise 38-1

Lab Exercise 38-1

Purpose

In this lab, you will use MultiLoad to insert the data rows you deleted in a lab from Module 37. The Accounts, Trans, and Customer tables each should have 100 rows in them (from Lab 37-1).

What you need

Populated tables and macro AP.Lab38_1.

Tasks

1. Export data to a file called *data38_1* by executing the macro AP.Lab38_1. Submit the following commands in BTEQ:

```
.EXPORT DATA FILE = data38_1, CLOSE;
 EXEC AP.LAB38_1;
.EXPORT RESET;
```

   Note: The Trans_Date is exported with an ANSI Date Format of 'YYYY-MM-DD' and a data type of CHAR(10).

2. Prepare a MultiLoad script which inserts rows into one of three different tables using the redefinition feature of MultiLoad. The input record contains a mixture of records which are to be inserted into the Accounts table if the first record byte is an "A", the Customer table if the first byte is a "C" and the Trans table if the first byte is a "T".

3. Check your tables for 200 rows per table. Your MultiLoad job should have a final return code of zero and should evidence 100 rows inserted into each of the three tables.

Lab Exercise 38-2

The size of data38_2 should be 15,600 bytes.

The execution of the macro AP.Lab38_2 will cause 200 rows to be created in data38_2. 100 of these records will be used to update 100 existing rows in the Accounts table and 100 of the records will be used to add 100 new rows (accounts) to the Accounts table.

There is no code (A) since all of the records apply to the Accounts table.

Each record in the data38_2 file has the same data as the columns in the Accounts table.

Data row for Accounts table
Data row for Accounts table
:

A technique that can be used to create Linux scripts without using vi or vim is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where Linux is running and …

3. **cat > lab38_23.mld** (or whatever filename you wish)

   Use the mouse to choose the **Edit ➔ Paste** function
   To exit the cat command, press either the DELETE key or CNTL C.

Lab Exercise 38-2

Lab Exercise 38-2

Purpose

In this lab, you will prepare and execute a MultiLoad script which performs an 'UPSERT' operation (INSERT MISSING UPDATE) on your Accounts table as a single operation.

What you need

Populated tables and macro AP.Lab38_2.

Tasks

1. Delete all rows from the Accounts Table and use the following INSERT/SELECT to create 100 rows of data in your table.

   ```
   INSERT INTO Accounts  SELECT * FROM  AP.Accounts  WHERE Account_Number < 20024101 ;
   ```

2. Export data to a file *data38_2* using the macro AP.Lab38_2.

   ```
   .EXPORT DATA FILE = data38_2, CLOSE;
    EXEC AP.Lab38_2;
   .EXPORT RESET;
   ```

3. Prepare and execute a MultiLoad script which performs an 'UPSERT' operation (INSERT MISSING UPDATE) on your Accounts table as a single operation. Use the data from *data38_2* as input to the MultiLoad 'UPSERT' script. If the row exists, UPDATE the Balance_Current with the appropriate incoming value. If not, INSERT a row into the Accounts table.

4. Validate your results. MultiLoad should have performed 100 UPDATES and 100 INSERTS with a final return code of zero.

Notes

Module 39

TPump

After completing this module, you will be able to:

- **State the capabilities and limitations of TPump.**
- **Describe TPump commands and parameters.**
- **Prepare and execute a TPump script.**

Notes

Table of Contents

TPump 39-4
ATOMIC UPSERT 39-4
TPump Limitations 39-6
.BEGIN LOAD Statement 39-8
Notes on CHECKPOINT 39-8
Notes on ERRLIMIT 39-8
TPump Specific Parameters 39-10
SERIALIZE 39-10
PACK 39-10
RATE 39-10
LATENCY 39-10
NOMONITOR 39-10
ROBUST 39-10
MACRODB 39-10
.BEGIN LOAD – PACK and RATE 39-12
.BEGIN LOAD – SERIALIZE OFF 39-14
.BEGIN LOAD – SERIALIZE ON 39-16
Latency 39-16
.BEGIN LOAD – ROBUST ON 39-18
Sample TPump Script (1 of 2) 39-20
Sample TPump Script (2 of 2) 39-22
Optional INMOD 39-22
TPump Compared with MultiLoad 39-24
Economy of Scale and Performance 39-24
Multiple Statement Requests 39-24
Macro Creation 39-24
Locking and Transactional Logic 39-24
Additional TPump Statements 39-26
DATABASE 39-26
EXEC(UTE) 39-26
Invoking TPump 39-28
TPump Statistics 39-30
TPump Monitor 39-32
Checking Status of a Job 39-32
Changing the Statement Rate as a Job Runs 39-32
Application Utility Checklist 39-34
Summary 39-36
Module 39: Review Questions 39-38
Lab Exercise 39-1 39-40

TPump

TPump (**Teradata Parallel Data Pump**) provides an excellent utility for low-volume batch maintenance of large Teradata databases. It enables acquisition of data from the client with low processor utilization. TPump is as flexible as BulkLoad (an older Teradata utility that is no longer supported), which it is has replaced.

The functionality of TPump is enhanced by the Support Environment. In addition to coordinating activities involved in TPump tasks, it provides facilities for managing file acquisition, conditional processing, and certain DML (Data Manipulation Language) and DDL (Data Definition Language) activities on the Teradata Database. The Support Environment enables an additional level of user control over TPump.

TPump uses row-hash locks, making concurrent updates on the same table a possibility.

TPump has a built-in resource governing facility that allows the operator to specify how many updates occur (the statement rate) minute by minute, and then change the statement rate while the job continues running. This utility can be used to increase the statement rate during windows when TPump is running by itself, and then decrease the statement rate later on if users log on for ad-hoc query access.

TPump can always be stopped and all its locks can be dropped with no ill effect.

The facing page identifies the principal features of the TPump utility.

ATOMIC UPSERT

TPump has support for ATOMIC UPSERT. This enhances active warehouse transactions by allowing TPump to perform PACKed UPSERT operations without the cost of rollbacks that were incurred in previous UPDATE then INSERT transactions. UPSERT is a composite of UPDATE and INSERT operations.

The one-pass UPSERT is called atomic to emphasize that both the UPDATE and the INSERT are grouped together and executed as a single SQL statement. The syntax has been modified to allow an optional ELSE in the UPDATE statement. ATOMIC UPSERT makes inserting faster because it requires only one lock, no one can change the table during the ATOMIC UPSERT, and the client application sends and receives fewer packets during inserts, which improves performance.

TPump

- Allows near real-time updates from transactional systems into the warehouse.
- Performs INSERT, UPDATE, and DELETE operations, or a combination, from the same source. Up to 128 DML statements can be included for one IMPORT task.
- Alternative to MultiLoad for low-volume batch maintenance of large tables.
- Allows target tables to:
  - Have secondary indexes, join indexes, hash indexes, and Referential Integrity.
  - Be MULTISET or SET.
  - Be populated or empty.
  - Tables can have triggers - invoked as necessary
- Allows conditional processing (via APPLY in the .IMPORT statement).
- Supports automatic restarts; uses Support Environment.
- No session limit — use as many sessions as necessary.
- No limit to the number of concurrent instances.
- Uses row-hash locks, allowing concurrent updates on the same table.
- Can always be stopped and locks dropped with no ill effect.
- Designed for highest possible throughput.
  - User can specify how many updates occur minute by minute; can be changed as the job runs.

TPump Limitations

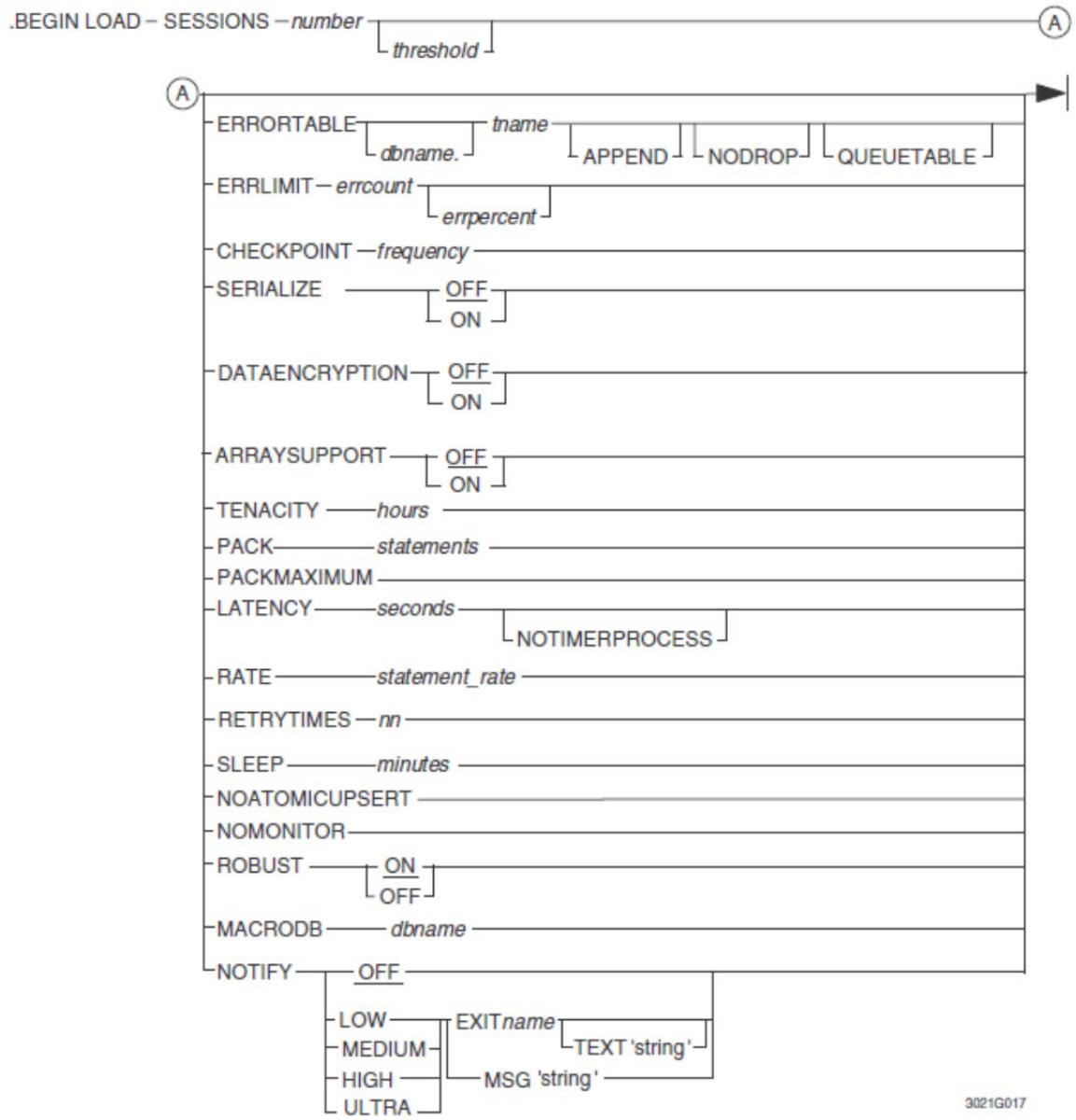
The facing page lists some TPump limitations you should be aware of.

TPump Limitations

- Use of SELECT is not allowed.
- Concatenation of data files is not supported.
- Exponential operators are not allowed.
- Aggregate operators are not allowed.
- Arithmetic functions are not supported.
- There is a limit of four IMPORT commands within a single TPump "load" task.
- In using TPump with dates before 1900 or after 1999, the year portion of the date must be represented by four numerals (yyyy).
  - The default of two numerals (yy) to represent the year is interpreted to be the 20th century.
  - The correct date format must be specified at the time of table creation.

.BEGIN LOAD Statement

The format of the .BEGIN LOAD statement is:



Notes on CHECKPOINT

Note that for TPump, only the *frequency* option is used for checkpoints. If you specify a CHECKPOINT *frequency* of more than 60, TPump terminates the job. Specifying a CHECKPOINT *frequency* of zero bypasses the check pointing function.

Notes on ERRLIMIT

In extreme cases (all records have errors), if the number of statements in each request (PACK factor) is greater than the ERRLIMIT, the job can stop due to exceeding the ERRLIMIT, producing no error table rows. To avoid this, set the ERRLIMIT greater than the PACK factor.

.BEGIN LOAD Statement

Many of the .BEGIN parameters are similar to those for MultiLoad.

.BEGIN LOAD		
SESSIONS	*max* [*min*]	(required)
ERRORTABLE	*tablename*	(defaults to jobname_ET)
ERRLIMIT	*errcount*	[*errpercent*]
CHECKPOINT	*frequency*	(default is 15 minutes)
TENACITY	*hours*	(default is 4)
SLEEP	*minutes*	(default is 6)
NOTIFY	OFF [LOW, …]	(default is OFF)

However, TPump has numerous parameters on the .BEGIN LOAD statement that are unique to TPump.

SERIALIZE	*ON* | *OFF*	(default ON if UPSERT)
PACK	*number*	(default is 20, max is 600)
PACKMAXIMUM	(use maximum pack factor)	
RATE	*number*	(default is unlimited)
LATENCY	*number*	(range is 10 – 600 seconds)
NOMONITOR		(default is monitoring on)
ROBUST	*ON* | *OFF*	(default is ON)
MACRODB	*dbname*	(default is logtable dbase) ;

TPump Specific Parameters

There are a number of parameters specific to TPump.

SERIALIZE

If ON, this options guarantees that operations on a row occur serially. If SERIALIZE is specified without ON or OFF, the default is ON. If SERIALIZE is not specified, the default is OFF unless the job contains an UPSERT operation which causes SERIALIZE to default to ON. This feature is meaningful only when a primary index for the table is specified by using the KEY option with the FIELD command.

Ex. .FIELD ACCTNUM * INTEGER KEY;

PACK

Specifies the number of statements to pack into a multiple-statement request. Packing improves network/channel efficiency by reducing the number of sends and receives between the application and Teradata.

RATE

Specifies the initial maximum rate at which statements are sent to the Teradata RDBMS per minute. If the statement rate is either zero or unspecified, the rate is unlimited. If the statement rate is less than the statement packing factor, TPump sends requests smaller than the packing factor. If the TPump monitor is in use, the statement rate can be changed later.

LATENCY

Allows TPump to commit to Teradata any data sitting in the buffer longer than the LATENCY value. Allows TPump to become a multi -threaded application in the event that control of the main thread is awaiting input from an access module. The range is 10-600 seconds.

NOMONITOR

Prevents TPump from checking for statement rate changes from or update status information for the TPump Monitor application.

ROBUST

The OFF parameter signals TPump to use “simple” restart logic. In this case, restarts cause TPump to begin where the last checkpoint occurred in the job. Any processing that occurred after the checkpoint is redone. This method does not have the extra overhead of additional database writes in the robust logic. Note, that certain errors may cause reprocessing, resulting in extra error table rows due to re-executing statements (attempting to re-insert rows) which previously resulted in the errors.

In Robust Mode, Teradata writes 1 database row in the Restart Log table for every SQL transaction.

MACRODB

Specifies the database to contain any macros used by TPump. If not specified, the default database that TPump uses to place create macros is the same database as the Restart Log table.

TPump Specific Parameters

Specific TPump .BEGIN LOAD parameters are:

SERIALIZE	***ON*** **|** ***OFF***	**ON guarantees that operations on a given key combination (row) occur serially. Used only when a primary index is specified. KEY option must be specified if SERIALIZE ON.**
PACK	***statements***	**Number of statements to pack into a multiple-statement request.**
PACKMAXIMUM		**Number of statements to pack into a multiple-statement request.**
RATE	***statement rate***	**Initial maximum rate at which statements are sent per minute. If the statement rate is zero or unspecified, the rate is unlimited.**
LATENCY	***seconds***	**# of seconds before a partial buffer is sent to the database.**
NOMONITOR		**Prevents TPump from checking for statement rate changes from or update status information for the TPump Monitor.**
ROBUST	***ON*** **|** ***OFF***	**OFF signals TPump to use "simple" restart logic; TPump will begin where the last checkpoint occurred.**
MACRODB	***dbname***	**Indicate a database to contain any macros used by TPump.**

.BEGIN LOAD – PACK and RATE

PACK specifies the number of statements to pack into a multi-statement request. This improves network/channel efficiency by reducing the number of sends and receives between the application and Teradata. Up to a maximum of 600 statements may be specified. If the TPump parser issues a warning that it has reduced the requested packing rate to a value, change your script to this value. This will reduce the overhead caused by dynamic adjusting of TPump.

The Teradata Database enforces a maximum column limit with TPump jobs. This limit is 2550.

Example – if you issue a TPump job as a multi-statement request, with a USING clause that has 64 columns, you could divide 2560 by 64 to give you the maximum PACK factor you could use – but note that this will most likely not provide the best performance level, and will most certainly blow out the (unknown) plastic steps limit. The other restrictions that must be considered are:

- 64K message size limit
- TPump limit of 2430 statements
- Teradata USING limit of 2560 columns
- Plastic Steps limit

In order to determine the best PACK rate for a TPump job, you need to experiment with various numbers. As you increase the PACK rate, the throughput improvement is great at first, then falls off and gets worse. Going from a PACK rate of 1 to 2 could provide huge performance gains, and going from 2 – 4 could be just as beneficial, but moving from 8 to 16 might cause a performance drop. You could run tests at 2, 4, 8, 16, 32 and 64 and graph the results. One goal is to find the "sweet spot" that provides the best performance for your job, and another goal is to find the maximum PACK rate that your job can use.

PACKMAXIMUM can be used to determine what the MAX PACK can be. Do not run PACKMAXIMUM against productions jobs, because it determines the PACK factor by trial and error. Do this once, in a test situation, to determine the maximum PACK factor you could employ.

NOTE: Clean Data

If your data has errors in it, larger packs can hurt – this is because of the way TPump handle errors. If you have a few hundred statements and an error occurs, Teradata rolls back the entire request, and TPump has to remove the statement and the data, and then reissue the entire statement. It is very important to have CLEAN data when using TPump.

.BEGIN LOAD – PACK and RATE

- **PACK specifies the number of statements to pack into a multi-statement request.**
  - **Improves network/channel efficiency by reducing the number of sends and receives between the application and Teradata.**
  - **Increasing the PACK rate improves throughput performance – to a certain level.**
- **Restrictions to consider:**
  - **64K message size limit**
  - **TPump limit of 2430 statements**
  - **Teradata USING clause limit of 2560 columns**
  - **Teradata Plastic Steps limit**
- **PACKMAXIMUM – use in testing to help establish a PACK value; do not use in a production job.**
- **RATE sets the initial maximum rate at which statements are sent per minute.**
  - **Defaults to unlimited; specify a value to control the number of amount of work sent to Teradata.**
  - **Example: RATE 12000 PACK 20**

    12000 20 = 600 packets/minute or approximately 10 packets/second

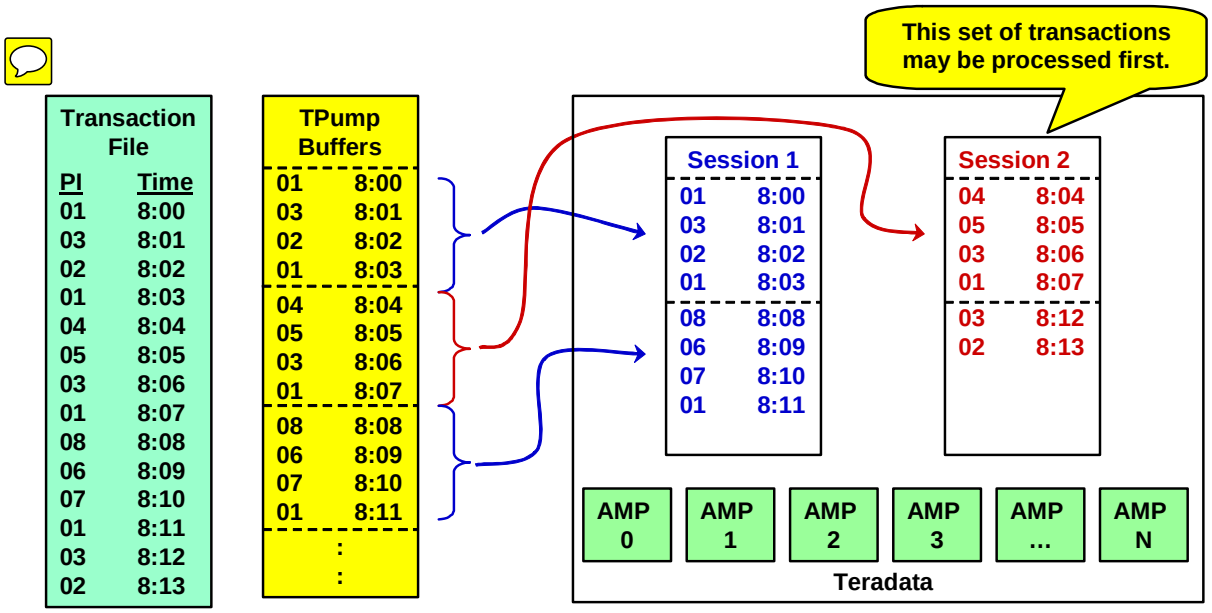
.BEGIN LOAD – SERIALIZE OFF

With SERIALIZE OFF, statements are executed on the first session available; hence, operations may occur out of order.

The example on the facing page illustrates how transactions are processed by TPump.

.BEGIN LOAD – SERIALIZE OFF

- With SERIALIZE OFF, transactions are processed in the order they are encountered and placed in the first available buffer. Buffers are sent to PE sessions and different PEs process the data independently of other PEs.
- SERIALIZE OFF does not guarantee the order in which transactions are processed.



.BEGIN LOAD – SERIALIZE ON

SERIALIZE ON tells TPump to partition the input records across the number of sessions it is using, ensuring that all input records that touch a given target table row (or that contain the same non-unique primary index value, for example) are handled by the same session.

SERIALIZE ON is useful for two reasons:

1. The order that the updates are applied is important in this application.
2. There is a possibility that rows with the same primary index value will be inserted through different buffers at the same time.

This second point has performance implications, as SERIALIZE ON can eliminate the lock delays or potential deadlocks caused by primary index collisions.

This guarantees both input record order and all the records with the same primary index value will be handled in the same session, and possibly the same buffer. The way SERIALIZE guarantees input order is to partition on the columns you have specified in your TPump script as KEY fields. Usually this will be the primary index of the table being updated, but it may be a different column or set of columns. It could, for example, be the primary index column(s) of a join index built upon the table being loaded.

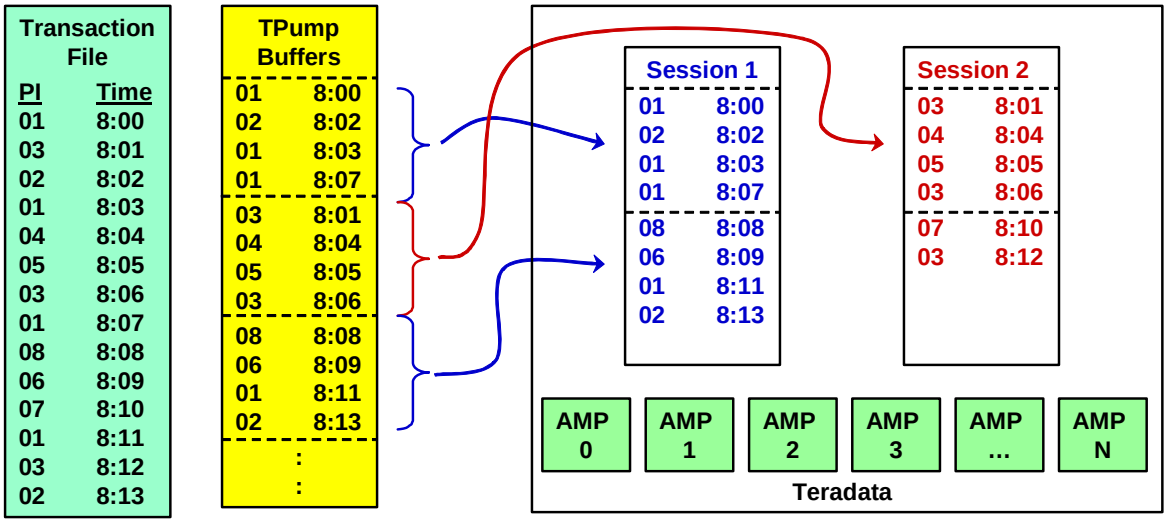
SERIALIZE reduces row blocking which could lead to deadlocks between buffers within the same TPump job, when rows with non-unique primary index values are being processed. Manual partitioning is required to do the same if the input data is divided between multiple TPump jobs.

Latency

Latency (seconds) is the number of seconds to use as a threshold for flushing stale buffers, based on the number of seconds the oldest record is in the buffer. The range is from 10 to 600 seconds. If serialization is off, only the current buffer can be stale. If serialization is on, the number of stale buffers can range from zero to the number of sessions.

.BEGIN LOAD – SERIALIZE ON

- SERIALIZE ON can eliminate lock delays or potential deadlocks caused by primary index collisions, improving performance.
- SERIALIZE guarantees both input record order and all records with the same PI value will be handled in the same session. It is recommended to specify the PI in the statement column(s) as KEY.
- KEY Fields determine the PE session in which TPump send the transaction to.



.BEGIN LOAD – ROBUST ON

ROBUST ON is the default for all TPump jobs. By inserting some extra information about the rows just processed into the database log table at the completion of each buffer's request, TPump has a way to avoid re-applying rows that have already been processed in the event of a restart.

The ROBUST ON variable causes a row to be written to the log table each time a buffer has successfully completed its updates. These mini-checkpoints are deleted from the log when a checkpoint is taken, and are used during a restart to identify which rows have already been successfully processed since the most recent checkpoint was taken, then bypassing them on a restart. The larger the TPump pack factor, the less overhead involved in this activity.

ROBUST ON is particularly important if re-applying rows after a restart would cause either a data integrity problem or have an unacceptable performance impact. ROBUST ON is advisable for these specific conditions:

1. INSERTs into multi-set tables, as such tables will allow reinsertion of the same rows multiple times.
2. When updates are based on calculations or percentage increases
3. If pack factors are large, and applying and rejecting duplicates after a restart would be unduly time-consuming.
4. If data is time-stamped at the time it is inserted into the database.

ROBUST ON is always a good idea for TPump jobs that read from queues, but is particularly important if timestamps are used to record the time of insertion into the database. The original rows that were inserted before the restart will carry a timestamp that reflects their insertion time. If a reapply of a row is attempted, the reapplied row will carry a timestamp that is different, even though all of the other data is identical. To Teradata, this will appear as a different row with the same primary index value, so duplicate row checking will not prevent the duplicate insertion. ROBUST ON is needed to keep duplicates from being added to the table being loaded in the case of restart.

.BEGIN LOAD – ROBUST ON

- ROBUST ON is the default and recommended option for all TPUMP jobs.
  - This option avoids re-applying rows that have already been processed in the event of a restart.
  - Causes a row to be written to the log table each time a buffer has successfully completed its updates.
  - The larger the TPump PACK factor, the less overhead involved in this activity.
- These rows are deleted from the log when a checkpoint is taken.
- ROBUST ON is recommended for these specific conditions:
  - INSERTS into multi-set tables, as such tables will allow re-insertion of the same rows multiple times.
  - When UPDATEs are based on calculations or percentage increases.
  - If PACK factors are large, and applying and rejecting duplicates after a restart would be time-consuming.
  - If data is time-stamped at the time it is inserted into the database.
- ROBUST ON is always a good idea for TPump jobs that read from queues. It keeps duplicates from being re-inserted into the table in the event of a restart.

Sample TPump Script (1 of 2)

The next few pages provide an example of a TPump script.

Miscellaneous TPump Notes:

- With TPump, all required data is imported; none is obtained from tables already in the Teradata Database.
- No statement of an IMPORT task may make any reference to a table or row other than the one affected by the statement.
- TPump rejects UPDATEs that try to change to Primary Index value.
- .DML UPDATE requires a WHERE clause.
- .DML DELETE must not contain any joins
- .DML DELETE cannot have an OR (alternative is to use 2 separate .DML DELETE statements)
- MARK is default for INSERT, UPDATE, and UPDATE. IGNORE is the default for UPSERT.
- DUPLICATE – if a duplicate row is created as a result of an UPDATE or an INSERT (and the table doesn't support duplicate rows), the duplicate row is either ignored (IGNORE) or placed (MARK) in the Error_Table.
- MISSING – if a row is missing on an UPDATE or DELETE, the transaction is either ignored (IGNORE) or placed (MARK) in the Error_Table.
- EXTRA – lets you know if multiple rows are affected. If duplicate rows already exist and an UPDATE or a DELETE impacts multiple duplicate rows, then either the duplicates are ignored (IGNORE) or placed (MARK) in the Error_Table.

Sample TPump Script (1 of 2)

```
.LOGTABLE restartlog39_tpp;
.LOGON tdpid/username,password;
.BEGIN LOAD        SESSIONS 8                    SERIALIZE ON
                   PACK 20                       RATE 12000
                   ERRORTABLE  ACT_tpp_ET        ERRLIMIT 100 ;
.LAYOUT layout12;
  .FIELD table_code              1      CHAR(1);
  .FIELD A_accountno             2      INTEGER          KEY;
  .FIELD A_strnum                *      INTEGER;
  .FIELD A_street                *      CHAR(25);
  .FIELD A_city                  *      CHAR(20);
  .FIELD A_state                 *      CHAR(2);
  .FIELD A_zipcode               *      INTEGER;
  .FIELD A_balancefor            *      DECIMAL(10,2);
  .FIELD A_balancecur            *      DECIMAL (10,2);
  .FIELD C_customer_number       2      INTEGER          KEY;
  .FIELD C_last_name             *      CHAR(30);
  .FIELD C_first_name            *      CHAR(20);
  .FIELD C_social_security       *      INTEGER;
  .FIELD T_trans_number          2      INTEGER;
  .FIELD T_trans_date            *      CHAR(10);
  .FIELD T_accountno             *      INTEGER          KEY;
  .FIELD T_trans_id              *      CHAR(4);
  .FIELD T_trans_amount          *      DECIMAL(10,2);
```

Sample TPump Script (2 of 2)

The facing page shows the rest of the example TPump script. Note the two IMPORT clauses.

Note about the USE option with .DML LABEL. TPump uses all of the fields in the layout in 1) the macro definition, 2) the using clause of the macro definition, and 3) sends them in the data parcel even if they aren't referenced in the values clause.

- To minimize the amount of data placed into a data parcel, use the USE option to only specify the fields needed for the SQL statements that are part of the .DML LABEL.

- A problem can also occur when a field is redefined over another field with an incompatible data type. Use of the USE option helps avoid this problem.

Optional INMOD

An INMOD is a user exit routine used by TPump to supply or preprocess input records. The INMOD is specified as part of the IMPORT command.

Major functions performed by an INMOD include:

- Generating records to be passed to TPump.
- Validating a data record before passing it to TPump.
- Reading data directly from one or more database systems like IMS, Total.
- Converting fields in a data record before passing it to TPump.

Because of operational differences between TPump and the older utilities, some changes have been made to the INMOD utility interface for TPump. For compatibility with INMODs, the FDLINMOD parameter should be used. The use of this parameter provides support of existing INMODs, with the some restrictions, as noted in the TPump manual.

Sample TPump Script (2 of 2)

```
.DML LABEL Ins_Account
  USE (A_accountno, A_number, A_street, A_city, A_state, A_zipcode, A_balancefor,  A_balancecur);
  INSERT INTO Accounts VALUES
      (:A_accountno, :A_strnum, :A_street, :A_city, :A_state, :a_zipcode, :A_balancefor, :A_balancecur);

.DML LABEL Ins_Customer
  USE (C_customer_number, C_last_name, C_first_name, C_social_security);
  INSERT INTO Customer VALUES
      (:C_customer_number, :C_last_name, :C_first_name, :C_social_security);

.DML LABEL Ins_Trans
  USE  (T_trans_number, T_trans_date, T_accountno, T_trans_Id, T_trans_amount);
  INSERT INTO Trans VALUES
      (:T_trans_number, :T_trans_date, :T_accountno, :T_trans_Id, :T_trans_amount);

.IMPORT INFILE   datafile1  LAYOUT layout12
  APPLY Ins_Account            WHERE  table_code = 'A'
  APPLY Ins_Trans              WHERE  table_code = 'T'
  APPLY Ins_Customer           WHERE  table_code = 'C';

.IMPORT INFILE   datafile2   LAYOUT layout12
  APPLY Ins_Account            WHERE  table_code = 'A'
  APPLY Ins_Trans              WHERE  table_code = 'T'
  APPLY Ins_Customer           WHERE  table_code = 'C';

.END LOAD;
.LOGOFF;
```

TPump Compared with MultiLoad

If you have both MultiLoad and TPump utilities, you may want to know which to use when. In fact, TPump compliments MultiLoad.

Economy of Scale and Performance

MultiLoad performance improves as the volume of changes increases. In phase two of MultiLoad, changes are applied to the target table(s) in a single pass and all changes for any physical data block are effected using one read and one write of the block. The temporary table and the sorting process used by MultiLoad are additional overhead that must be "amortized" through the volume of changes. TPump, on the other hand, does better on relatively low volumes of changes because there is no temporary table overhead. TPump becomes expensive for large volumes of data because multiple updates to a physical data block will most likely result in multiple reads and writes of the block.

Multiple Statement Requests

The most important technique used by TPump to improve performance is the use of a multiple statement request. Placing more statements in a single request is beneficial for two reasons. First, because it reduces network overhead since large messages are more efficient than small ones. Second, (in ROBUST mode) it reduces TPump recovery overhead which amounts to one extra database row written for each request. TPump automatically packs multiple statements into a request based upon the PACK specification in the BEGIN LOAD command.

Macro Creation

For efficiency, TPump uses macros to modify tables, rather than the actual DML commands. The technique of changing statements into equivalent macros before beginning the job greatly improves performance.

- The size of network (and channel) messages sent to the Teradata Database by TPump is reduced.
- Teradata Database parsing engine overhead is reduced because the execution plans (or "steps") for macros are cached and re-used.

Locking and Transactional Logic

In contrast to MultiLoad, TPump uses row hash locking to allow for some amount of concurrent read and write access to its target tables. At any point TPump can be stopped and target tables are fully accessible. If TPump is stopped, however, depending on the nature of the update process, it may mean that the "relational" integrity of the data is impaired.

This differs from MultiLoad, which operates as a single logical update to one or more target tables. Once MultiLoad goes into phase two of its logic, the job is "essentially" irreversible and the entire set of table(s) is locked for write access until it completes. If TPump operates on rows that have associated "triggers," the triggers are invoked as necessary.

TPump Compared with MultiLoad

- MultiLoad performance improves as the volume of changes increases.
- TPump does better on relatively low volumes of changes.

- TPump improves performance via a multiple statement request.

- TPump uses macros to modify tables rather than the actual DML commands.

  Example of macro name – M20110826_105647_01136_001_001

- MultiLoad uses the DML statements.

- TPump uses row hash locking to allow for concurrent read and write access to target tables. It can be stopped with target tables fully accessible.
- In Phase 4, MultiLoad locks tables for write access until it completes.

Additional TPump Statements

The facing page identifies additional statements used by TPump.

DATABASE

The DATABASE statement changes the default database qualification for all unqualified DML and DDL statements. It only affects "native SQL" commands, and has no effect on the BEGIN LOAD command. The DATABASE command does affect INSERT, UPDATE, DELETE and EXEC statements issued as part of a load. (When TPump logs on sessions, it immediately issues a DATABASE statement on each session.)

The DATABASE command does not affect the placement of TPump macros.

EXEC(UTE)

The EXECUTE statement specifies a user-created macro for execution. The macro named in this statement is resident in the Teradata RDBMS and specifies the type of DML statement (INSERT, UPDATE, DELETE, or UPSERT) being handled by the macro.

The EXECUTE command immediately follows .DML LABEL;

Rules for user-created macros include:

- TPump expects the parameter list for any macro to exactly match the FIELD list specified by the LAYOUT in the script. FILLER fields are ignored. If the USE clause is used in the DML statement, TPump expects the parameter list for every macro in the DML statement to exactly match the field list specified by the USE clause.

- The macro should specify a single prime index operation: INSERT, UPDATE, DELETE, or UPSERT. TPump reports an error if the macro contains more than one supported statement. If the EXECUTE statement is replacing an INSERT, UPDATE, DELETE, or UPSERT statement in a job script, the EXECUTE statement must be placed at the same location as the INSERT, UPDATE, DELETE, or UPSERT statement that it replaces.

Additional TPump Statements

DATABASE Changes the default database qualification for all DML statements.

EXEC(UTE) Specifies a user-created macro for execution. The macro named is resident in the Teradata database.

```
DATABASE database ;
```

```
EXECUTE [database.]macro_name ( UPDATE/UPD
                                INSERT/INS
                                DELETE/DEL
                                UPSERT/UPS ) ;
```

Commands and statements in common with MultiLoad:

ACCEPT	IMPORT	RUN
DELETE	INSERT	SET
DISPLAY	LAYOUT	SYSTEM
DML	LOGON	TABLE
FIELD	LOGOFF	UPDATE
FILLER	LOGTABLE	
IF / ELSE / ENDIF	ROUTE	

Invoking TPump

The facing page displays the commands you can use to execute TPump.

Invoking TPump

Network Attached Systems: tpump [PARAMETERS] < *scriptname* >*outfilename*

Channel-Attached MVS Systems: // EXEC TDSTPUMP PARM= [PARAMETERS]

Channel-Attached VM Systems: EXEC TPUMP [PARAMETERS]

Channel Parameter	Network Parameter	Description
BRIEF	-b	Reduces print output runtime to the least information required to determine success or failure.
CHARSET=*charsetname*	-c *charsetname*	Specify a character set or its code. Examples are EBCDIC, ASCII, or Kanji sets.
ERRLOG=*filename*	-e *filename*	Alternate file specification for error messages; produces a duplicate record.
"*tpump command*"	-r '*tpump cmd*'	Signifies the start of a TPump job; usually a RUN FILE command that specifies the script file.
MACROS	-m	Keep macros that were created during the job run.
VERBOSE	-v	Additional statistical data in addition to the regular statistics.
	< *scriptname*	Name of file that contains TPump commands and SQL statements.
	> *outfilename*	Name of output file for TPump messages.

TPump Statistics

For each task, TPump accumulates statistical items and writes them to the customary output destination of the external system, SYSPRINT/stdout (or the redirected stdout), or the destination specified in the ROUTE command.

Candidate records considered. The number of records read.

Apply conditions satisfied. Represents the number of statements sent to the RDBMS. If there are no rejected or skipped records, this value is equal to the number of candidate records, multiplied by the number of APPLY statements referenced in the import.

Candidate records rejected. Represents the number of records that are rejected by the TPump client code because they are formatted incorrectly.

Candidate records with data errors (not applied). Represents the number of records resulting in errors on the Teradata Database. These records are found in the associated error table.

Statistics for Apply Label. This area breaks out the total activity count for each statement within each DML APPLY clause. The 'Type' column contains the values U for update, I for insert and D for delete. Note that unlike the other reported statistics, these values are NOT accumulated across multiple imports.

TPump Statistics

.	IMPORT 1	Total thus far
.	=========	==========
Candidate records considered:.....	200	200
Apply conditions satisfied:.....	200	200
Candidate records not applied:.....	0	0
Candidate records rejected:.....	0	0

**** Statistics for Apply Label : UPSERT_ACCOUNT
- Type: U
- Database: STUDENT130
- Table or Macro Name: Accounts
- Activity: 100
- Type: I
- Database: STUDENT130
- Table or Macro Name: Accounts
- Activity: 100

**** 10:31:48 UTY6677 Loading phase statistics
- Elapsed time: 00:00:00:01 (dd:hh:mm:ss)
- CPU time: 0.01 Seconds
- MB/sec: 0.015
- MB/cpusec: 1.5

Note: These statistics are not for the example TPump job shown earlier in this module.

**** 10:31:48 UTY0821 Error table STUDENT130.errtable_tpp is EMPTY, dropping table.

0018 .LOGOFF;

TPump Monitor

The TPump Monitor facility provides run-time monitoring of the TPump job. It allows users, via a command line interface, to track and alter the rate at which requests are issued to the Teradata Database.

To install the TPump tables, views, and macros for the TPump monitor facility, modify the following Linux script (with DBC password) and execute it with BTEQ.

/opt/teradata/client/14.00/sample/tpumpar.csql

With Windows XP, the script to execute with BTEQ is:

C:\Program Files\Teradata\Client\14.00\tpump\tpumpar.csql

The monitor interface is implemented by creating the table named SysAdmin.TPumpStatusTbl. When this table is present, TPump places rows into this table each minute a TPump job executes. This table is required to use the monitor functionality but is otherwise optional.

Checking Status of a Job

TPump users can find out the status of an import by querying against this table. TPump updates this table approximately once every minute.

Changing the Statement Rate as a Job Runs

TPump users can alter the statement rate of an import by updating this table. TPump examines this table approximately once every minute.

TPump Monitor

Tool to control and track TPump imports.

- The table SysAdmin.TPumpStatusTbl is updated once a minute.
- Alter the statement rate on an import by updating this table using macros.
- Use macros and views to access this table.

DBA Tools

View

- SysAdmin.TPumpStatus - view allows DBAs to view all of the TPump jobs.

Macro

- SysAdmin.TPumpUpdateSelect - allows DBAs to manage individual TPump jobs.

User Tools

View

- SysAdmin.TPumpStatusX - allows users to view their own TPump jobs.

Macro

- TPumpMacro.UserUpdateSelect - allows users to manage their own jobs.

Application Utility Checklist

The facing page adds the TPump capabilities to the checklist.

Automatic Restart – If the Teradata server restarts, the TPump utility will retry to connect to the Teradata database automatically and restart automatically.

Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions	ALL	LIMITED	Yes (SE)	Yes (SE)	Yes (SE)
DML Functions	ALL	INSERT	SELECT	INS/UPD/DEL	INS/UPD/DE
Multiple DML	Yes	No	Yes	Yes	Yes
Multiple Tables	Yes	No	Yes	Yes	Yes
Protocol Used	SQL	FASTLOAD	EXPORT	MULTILOAD	SQL
Conditional APPLY	No	No	No	Yes	Yes
Data Conversion	Yes	1 per column	Yes	Yes	Yes
Error Capture	No	Yes	N/A	Yes	Yes
Error Limits	No	Yes	N/A	Yes	Yes
User-written Routines	No	Yes	Yes	Yes	Yes
Automatic Restart	No	Yes*	Yes	Yes	Yes
Max Load Limit	No	Yes	Yes	Yes	No
Support Environment (SE)	No	No	Yes	Yes	Yes

Summary

The facing page summarizes some of the important concepts regarding the TPump utility.

Summary

- Allows near real-time updates from transactional systems into the warehouse.
- Performs INSERTs, UPDATEs, DELETEs, or UPSERTs.
- Alternative to MultiLoad for low-batch maintenance of large databases.
- Uses row-hash locks, allowing concurrent updates on the same table.
- Can always be stopped and locks dropped with no ill effect.
- User can specify how many updates occur minute by minute; can be changed as the job runs.

Module 39: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 39: Review Questions

Match the item in the first column to its corresponding statement in the second column.

_____ 1. TPump purpose	a. Query against TPump status table
_____ 2. MultiLoad purpose	b. Concurrent updates on same table
_____ 3. Row hash locking	c. Low-volume changes
_____ 4. PACK	d. Use to specify how many statements to put in a multi-statement request
_____ 5. MACRO	e. Large volume changes
_____ 6. Statement rate change	f. Used instead of DML

Lab Exercise 39-1

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

```
.LOGON      …;
.EXPORT DATA FILE = data39_1, CLOSE;
 EXEC AP.Lab39_1;
.EXPORT RESET
.LOGOFF;
```

The size of data39_1 should be 15,600 bytes.

The execution of the macro AP.Lab39_1 will cause 200 rows to be created in data39_1. 100 of these records will be used to update 100 existing rows in the Accounts table and 100 of the records will be used to add 100 new rows (accounts) to the Accounts table.

There is no code (A) since all of the records apply to the Accounts table.

Each record in the data39_1 file has the same data as the columns in the Accounts table.

Data row for Accounts table
Data row for Accounts table
:

A technique that can be used to create Linux scripts without using vi or vim is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where Linux is running and …

3. **cat > lab39_13.tpp** (or whatever filename you wish)

   Use the mouse to choose the **Edit ➔ Paste** function
   To exit the cat command, press either the DELETE key or CNTL C.

Lab Exercise 39-1

Lab Exercise 39-1

Purpose

In this lab, you will perform an operation similar to lab 38-2, using TPump instead of MultiLoad. For this exercise, use 4 SESSIONS with a PACK of 20 and a RATE of 4800.

What you need

Data file (*data39_1*) created from macro AP.Lab39_1.

Tasks

1. Delete all rows from the Accounts Table and use the following INSERT/SELECT to create 100 rows of test data:

   INSERT INTO Accounts SELECT * FROM AP.Accounts WHERE Account_Number < 20024101;

2. Export data to the file *data39_1* using the macro AP.Lab39_1.

3. Prepare a TPump script which performs an UPSERT operation (INSERT MISSING UPDATE) on your Accounts table as a single operation. Use the data from *data39_1* as input to the UPSERT script. If the row exists, UPDATE the Balance_Current with the appropriate incoming value. If not, INSERT a row into the Accounts table. In your script, be sure to set a statement rate.

4. Run the script.

5. Validate your results. TPump should have performed 100 UPDATES and 100 INSERTS with a final return code of zero.

Notes

Module 40

TERADATA
Raising Intelligence

Choosing a Utility

After completing this module, you will be able to:

- **Describe various solutions to applications.**
- **Compare how different utilities work for the same application.**

Notes

Table of Contents

Maximum Number of Load Jobs 40-4
Maximum Number of Load Jobs (cont.) 40-6
Solution 1: Update or Delete vs. Insert/Select 40-8
Solution 2: SQL Update vs. MultiLoad or TPump 40-10
Solution 3: SQL Update vs. FastLoad 40-12
Utility Considerations 40-14
Module 40: Various Ways of Performing an Update 40-16
Module 40: Choosing a Utility Exercise 40-18

Maximum Number of Load Jobs

With Teradata V2R6.0 (and previous releases), the DBS Control parameter MaxLoadTasks has a maximum limit of 15. One of the reasons that this limit is enforced is to prevent FastLoad, MultiLoad, and FastExport jobs from using up all available AMP worker tasks (AWTs).

In Teradata Database V2R6.1, the maximum number of concurrent load jobs is increased.

- For FastLoad and MultiLoad Jobs: up to 30 concurrent jobs
- For FastExport Jobs: Up to 60 jobs (minus the number of active FastLoad and MultiLoad jobs)

Users should be aware that running more load/unload utilities may impact other work and applications running concurrently (such as DSS queries or tactical queries) because of higher demand of the following resources: number of sessions, CPU, I/O, and memory.

This new feature is controlled by a new internal DBS Control parameter named MaxLoadAWT. (AWT – AMP Worker Tasks).

When MaxLoadAWT is zero (by default), this feature is disabled. Everything works as before. The DBS control flag MaxLoadTasks which specifies the concurrency limit for FastLoad, MultiLoad, and FastExport cannot be greater than 15.

When MaxLoadAWT is greater than zero, this feature is effectively enabled. This parameter specifies the maximum number of AWTs that can be used by FastLoad and MultiLoad jobs. The maximum allowable value is 60% of the total AWTs. Usually the maximum number of AWTs is 80; therefore, this maximum is 48.

Characteristics of this feature (when enabled) include:

- MaxLoadTasks only controls FastLoad and MultiLoad jobs. Its maximum limit is increased from 15 to 30.
- FastExport jobs are managed differently:
  - FastExport is no longer controlled by MaxLoadTasks flag.
  - FastExport limit is 60 minus number of active FastLoad and MultiLoad jobs.
  - A FastExport job is only rejected if the total number of active utility jobs is 60.
  - The minimum number of FastExport jobs that can run is 30. A FastExport job may be able to run even when FastLoad and MultiLoad jobs are rejected.

Answer to question 1 is 45. (60 – 15 FL/ML)
Answer to question 2 is 40. (60 – 20 FL/ML) 5 of the FL/ML jobs will be waiting.

Maximum Number of Load Jobs

There are two DBSControl parameters that control the maximum number of concurrent load jobs.

- MaxLoadTasks and MaxLoadAWT (AWT – AMP Worker Tasks)

If MaxLoadAWT = 0 (the default), then MaxLoadTasks has a range of 0 to 15.

- 15 is the maximum number of FastLoad, FastExport, and MultiLoad jobs

If MaxLoadAWT > 0, then MaxLoadTasks has a range of 0 to 30.

- 30 is the maximum number of FastLoad and MultiLoad jobs
- 60 is the maximum number of FastLoad, MultiLoad, and FastExport jobs.
  - Therefore, 60 – (# FL + # ML) = # possible FastExport jobs (i.e., 30 to 60)

Example situations: Assume MaxLoadTasks = 20 (and MaxLoadAWT > 0)

1. Assume 5 ML jobs and 10 FL jobs are running. How many FastExports can run?

2. Assume 15 ML jobs and 10 FL jobs have been started. How many FastExports can run?

Maximum Number of Load Jobs (cont.)

If the MaxLoadAWT parameter is greater than 0, then the maximum number of concurrent load utilities is controlled by both MaxLoadTasks and MaxLoadAWT parameters.

A new FastLoad or MultiLoad job is allowed to start only if BOTH MaxLoadTasks AND MaxLoadAWT limits are not reached.

FastLoad and MultiLoad jobs use a different number of AWTs depending on the phase the utility is in.

Utility		# of AWTs needed
FastLoad	Phase 1 (Acquisition)	3
	Phase 2 (Sort)	1
MultiLoad	Acquisition Phase	2
	Application Phase	1
FastExport	All phases	0*

* The count of AWTs only applies to FastLoad and MultiLoad jobs. FastExport jobs use AWTs, but the count doesn't apply. The SELECT phase of FastExport actually uses 2 AWTs, but these are executed as normal DML so no AWT is counted toward the AWT limit. The export phase is processed by the Load Control Task (LCT) so no AWT is required.

For example, with MaxLoadAWT = 48 and MaxLoadTasks = 30, possible job mixes include:

- 16 FastLoad jobs in Phase 1, or
- 9 FastLoad jobs in Phase 1 and 21 FastLoad jobs in Phase 2, or
- 24 MultiLoad jobs in Acquisition Phase, or
- 5 MultiLoad jobs in Acquisition Phase and 25 MultiLoad jobs in Application Phase

Answer to question 1 is No. (The limit of MaxLoadAWT is already reached.)
Answer to question 2 is No. (The limit of MaxLoadTasks is already reached.)
Answer to question 3 is Yes. (Neither limit has been reached.)

Maximum Number of Load Jobs (cont.)

If MaxLoadAWT > 0, then these rules are followed to start new load jobs:

- **A FastExport job is started if the total count of load jobs < 60.**
- **A FastLoad (FL) or MultiLoad (ML) is started if either limit is not exceeded.**
  - **MaxLoadTasks (0 – 30); has the limit of FL/ML load jobs been reached?**
  - **MaxLoadAWT (0 – 48*); has the limit of FL/ML AWTs been reached?**
    - **Maximum number is 60% of system AWTs (typically 80)**
  - **If the answer is NO to both limits, then the FL or ML job can be started.**
- **TASM utility throttles (if used) override the MaxLoadTasks and MaxLoadAWT values and uses 60% as the setting for the AWTs.**
- **The # of AWTs used by FastLoad and MultiLoad jobs depends on the utility phase.**

Utility	Phase	# of AWTs	Phase	# of AWTs
FastLoad	Phase 1 (Acquisition)	3	Phase 2 (Sort)	1
MultiLoad	Acquisition Phase	2	Application Phase	1

Example situations: MaxLoadTasks (MLT) = 20 and MaxLoadAWTs (MLA) = 40

Can a new FastLoad job be started given the following?

1. **Assume system is currently using 16 MLTs and 40 MLAs?**
2. **Assume system is currently using 20 MLTs and 32 MLAs?**
3. **Assume system is currently using 18 MLTs and 35 MLAs?**

Solution 1: Update or Delete vs. Insert/Select

As fast or as flexible as the application utilities might be, you sometimes need to employ a certain amount of ingenuity and imagination to make the best use of them.

Consider the simple global update on the facing page. It is a long-running table update that requires maintenance of a transient journal row for every row changed. In the event of failure, it will need to roll back the transaction as if it had never begun. The longer the transaction takes to complete, the greater the chance of failure.

Another way of performing this task might be to create a new table and populate it using the "fast path" INSERT/SELECT.

Another approach you might consider is to delete unneeded rows from a table and preserve the rows you wish to keep.

By looking at the problem in a different way, you can use the fast INSERT/SELECT performance to handle a series of updates.

Solution 1: Update or Delete vs. Insert/Select

Update all customers credit limit by 20%.

```
UPDATE Customer
   SET Credit_Limit = Credit_Limit * 1.20 ;
```

Delete all transactions prior to 2000.

```
DELETE FROM Trans
   WHERE Trans_Date  <  DATE '2000-01-01';
```

- SQL statements are treated as single transactions.
- This requires Transient journal space for every updated or deleted row in the target table until the transaction finishes.
- A failure will cause the system to back out all changes that have been completed (which could take hours if the table and the number of completions are very large).

```
CREATE TABLE Cust_N AS Customer WITH NO DATA ;
INSERT INTO Cust_N
   SELECT ... , Credit_Limit * 1.20 FROM  Customer ;
DROP TABLE Customer ;
RENAME TABLE Cust_N TO Customer ;
```

```
CREATE TABLE Trans_N  AS Trans WITH NO DATA;
INSERT INTO Trans_N  SELECT * FROM Trans
        WHERE Trans_Date  >  DATE '1999-12-31';
DROP TABLE Trans;
RENAME TABLE Trans_N TO Trans;
```

- This approach reduces the number of changes that would have to be backed out by the system in case of a failure.
- The "Fast path" INSERT / SELECT offers the fastest possible transfer of data that can be achieved in a single SQL statement.
- The same approach could also be used to delete rows from a table, simply by not selecting them for insert.

Solution 2: SQL Update vs. MultiLoad or TPump

The facing page displays another solution that uses the speed and sophisticated functionality of MultiLoad to attain good performance by writing updates to disk a block at a time, effectively removing the disk as a performance constraint.

If the percentage of updates is small as compared to the number of rows in the table, TPump may be a good choice.

Solution 2: SQL Update vs. MultiLoad or TPump

SQL Update

```
UPDATE   Customer
   SET   credit_limit        = credit_limit * 1.20
 WHERE   over_limit_count    > 0
   AND   late_payment_count  = 0 ;
```

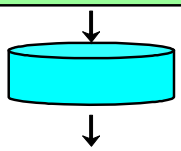
Full table Write lock

Full table scan

Potentially large Transient Journal

FastExport

```
SELECT   customer_number, phone, credit_limit * 1.20
  FROM   Customer
 WHERE   over_limit_count    > 0
   AND   late_payment_count  = 0 ;
```



MultiLoad *or* ***TPump***

```
UPDATE   Customer
   SET   credit_limit        = :credit_limit
 WHERE   phone               = :phone
   AND   customer_number     = :customer_number;
```

Full table Read lock

Full table scan

No Transient Journal

If the percentage of updates compared to number of table rows is large, use MultiLoad. If the percentage is small, use TPump.

MultiLoad advantages -

- Sorts updates by Primary Index.
- Each data block is accessed only once.
- Full automatic restart under all conditions.

Solution 3: SQL Update vs. FastLoad

FastLoad is limited in functionality in that, like the optimized INSERT/SELECT, it is used to insert rows into an initially empty table.

Even so, FastLoad is very fast. If you export the rows you wish to update to the host (updating the values in the process), and add the remainder of the rows, you have a complete host-resident file suitable for recovery. You can then take full advantage of FastLoad's excellent performance to insert all the rows into a new table:

- Create a new table.
- Use FastLoad to populate it at high speed.
- Drop the old table.
- Rename the new table to the old table name.

Again, by looking at the problem from a different perspective, you are able to employ the high speed of FastLoad to perform updates to a populated table.

Note: The examples shown for all solutions illustrate *updates* to the target table. Each of the solutions could equally be changed to *delete* rows from the target table.

Solution 3: SQL Update vs. FastLoad

SQL Update

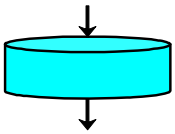
```
UPDATE   Customer
   SET   credit_limit        = credit_limit * 1.20
 WHERE   over_limit_count    > 0
   AND   arrears_count       = 0 ;
```

- Full table Write lock
- Full table scan
- Potentially large Transient Journal

FastExport

```
SELECT   . . . , credit_limit * 1.20, . . .
  FROM   Customer
 WHERE   over_limit_count      > 0
   AND   arrears_count         = 0
; SELECT  *
  FROM   Customer
 WHERE   NOT (over_limit_count  > 0 AND
              arrears_count = 0);
DELETE   FROM  Customer;
```

- Full table Read lock
- Full table scan
- No Transient Journal



FastLoad

An alternate solution when the external disk space is capable of housing the entire table.

Utility Considerations

While a selection of tools is available for batch-type processing, application success relies principally on choosing the right one.

In order to make a good choice, you have to ask the right questions. Some of these are obvious, such as “Is the utility supported on the host?” Other important concerns are less apparent: “What happens when things go wrong?” “Does this utility allow me to work within the window required by the user?”

To get a count of load jobs that are currently executing, you can use the following SQL.

```
SELECT    COUNT(DISTINCT LogonSequenceNo) AS Utility_Cnt
FROM      DBC.SessionInfo
WHERE     Partition IN ('Fastload', 'Export', 'MLoad');
```

Utility Considerations

- **Utility support**
  - Has the customer purchased the utility and does it run on your host?
- **Restart capability**
  - Is there a restart log?
  - What happens with a Teradata restart?
  - What happens if the host fails?
- **Multiple sessions**
  - Does the utility support multiple sessions?
  - How do you choose the optimum number?
- **Error handling**
  - Are errors captured in an error table?
  - Do you have control over error handling?
- **Are special processing routines needed?**
  - Does the utility support INMODs, OUTMODs, or AXSMODs?
- **Does the utility meet performance requirements?**
  - Does the job fit your batch window?
  - Do the tables require continuous (7 x 24) access by the user groups?

Module 40: Various Ways of Performing an Update

Consider this simple global update of a large table. We use a bank application since you are probably familiar with bank accounts, checks, deposits, etc.

At the end of each month, before it prints the bank statements, the bank (which maintains derived data for the account balance) is required to set the value of the Balance_Forward to Balance_Current. The data is already available in the Teradata database.

Based on what you have learned, try to evaluate the listed methods fastest to slowest.

Module 40: Various Ways of Performing an Update

Prior to producing a monthly statement, the Bank sets the Balance_Forward amount equal to the Balance_Current for 900,000 accounts:

```
UPDATE Accounts
    SET  Balance_Forward = Balance_Current ;
```

Which is the fastest method?

1. **Submit the statement above.**
2. **INSERT/SELECT revised values to a new table, drop the original table and rename the new table.**
3. **FastExport the data rows to the server and UPDATE using MultiLoad.**
4. **FastExport the data rows to the server and FastLoad these rows back to a new table. Drop the old table and rename the new.**

Order the above Fastest to Slowest _____ , _____ , _____

Module 40: Choosing a Utility Exercise

The facing page contains two scenarios. Make the best choice of which utilities to use for each of the two scenarios.

Module 40: Choosing a Utility Exercise

1. A sales table currently contains 24 months of transaction data. At the end of each month, 250 million rows are added for the current month and 250 million rows are removed for the oldest month. There is enough PERM space to hold 30 months worth of data.

   Which choice (from below) makes the most sense? _____

2. The customer decides to partition the table by month and maintain each month's data in a separate partition. At this time, only the most recent 24 months need to be maintained. At the end of each month, data is loaded into a new monthly partition and the oldest month is removed. The partition expression does not include the NO RANGE partition.

   Which choice (from below) makes the most sense? _____

Utility Choices:

a. Use FastLoad to add new data to existing table, and ALTER TABLE to remove old data.

b. Use MultiLoad to add new data to existing table, and ALTER TABLE to remove old data.

c. Use FastLoad to add new data to existing table, and MultiLoad to remove old data.

d. Use MultiLoad to add new data to existing table, and MultiLoad to remove old data.

e. Use TPump to add new data, and TPump to remove old data.

f. Use TPump to add new data, and ALTER TABLE to remove old data.

Notes

Module 41

Database Administration and Building the Database Environment

After completing this module, you should be able to:

- **Describe the purpose and function of an administrative user.**
- **Differentiate between creators, owners (parents), and children.**
- **Describe how to transfer ownership of databases and users.**

Notes

Table of Contents

Database Administration 41-4
Initial Teradata Database 41-6
Administrative User 41-8
Owners, Parents and Children 41-10
Creating New Users and Databases 41-12
Transfer of Ownership 41-14
DELETE/DROP Statements 41-16
Teradata Administrator – New System 41-18
Teradata Administrator – Hierarchy 41-20
Summary 41-22
Module 41: Review Questions 41-24

Database Administration

The facing page identifies some of the functions of a Teradata Database Administrator (DBA). These functions include:

- User Management – creation of databases, users, accounts, roles, and profiles
- Space Allocation and Usage – perm, spool, and temporary space
- Access of Objects (e.g., tables, views) – access rights, roles, use of views, etc.
- Access Control and Security – logon access, logging access, etc.
- System Maintenance – specification of system defaults, restarts, data integrity, etc.
- System Performance – use of Priority scheduler, job scheduling, etc.
- Resource Monitoring – use of ResUsage tables/views, query capture (DBQL), etc.
- Data Archives, Restores, and Recovery – ARC facility, Permanent Journals, etc.

Examples of tools available to the Teradata DBA include:

- Use of Data Dictionary/Directory tables and views to manage the system
- Teradata Administrator – graphical Windows tool to assist in administration
- Teradata Manager – suite of Windows tools for performance management, etc.
- Teradata Viewpoint – set of portlets for database monitoring and administration
- Teradata Analyst Toolset – Visual Explain, Index Wizard, Statistics Wizard, and Teradata SET
- Teradata Dynamic Workload Manager, Teradata Workload Analyzer, and Query Scheduler – job scheduling facility
- System utilities – e.g., dbscontrol, ferret, rebuild, etc.
- User scripts and 3rd party applications

To do these functions, it is important to understand key concepts such as the Teradata hierarchy and the concepts of ownership (parents and children). The hierarchy and the concept of ownership will be discussed in this module.

Acronyms:

Teradata DWM – Teradata Dynamic Workload Manager
DBQL – Database Query Log
DBW – Database Window
SET – System Emulation Tool

Database Administration

Some of the functions of a Teradata Database Administrator (DBA) include:

- User and Database Management
- Space Allocation and Usage
- Access of Objects (e.g., tables, views, macros, etc.)
- Access Control and Security
- System Maintenance
- System Performance and Resource Monitoring
- Data Archives, Restores, and Recovery

Examples of tools available to the Teradata DBA include:

- Use of Data Dictionary/Directory tables and views to manage the system
- Teradata Administrator – Windows administration utility
- Teradata Manager – suite of Windows tools – e.g., Teradata Performance Monitor
- Teradata Viewpoint – set of portlets for database monitoring and administration
- Teradata Analyst Toolset – Visual Explain, Index Wizard, Statistics Wizard, and SET
- Teradata Dynamic Workload Manager and Teradata Workload Analyzer
- Database Console Window and System utilities – e.g., dbscontrol, ferret, rebuild, etc.
- User scripts and 3rd party applications

To do these functions, it is important to understand key concepts such as …

- Teradata hierarchy and the concepts of ownership (parents and children)

Initial Teradata Database

The Teradata Database software includes the following users and databases:

DBC

With the few exceptions described below, a system user named DBC owns all usable disk space. DBC's space includes dictionary tables, views and macros discussed in the next module. The usable disk space of DBC initially reflects the entire system hardware capacity, less the following:

Sys_Calendar

This user is used to hold the system calendar table and views.

SysAdmin

SysAdmin is a system user with a minimum of space for table storage. SYSADMIN contains several supplied views and macros as well as a restart table for FastLoad jobs.

SystemFE User

This system user contains special macros used to generate diagnostic reports for Customer Engineers (field support personnel) logged on as this user. The default password is "service".

Crashdumps User

Crashdumps is a user provided for temporary storage of PDE dumps generated by the software. The default is 1 GB. You should enlarge the Crashdumps user based on the size of the configuration to accommodate at least three dumps.

PUBLIC and TDPUSER

PUBLIC and TDPUSER are "dummy" database names used by the database system software and appear in the system hierarchy. These users are defined with no permanent space. TDPUSER is used to support two-phase commit.

Default and All

Default and All are also "dummy" database names that are reserved by the database system software and don't appear in the hierarchy.

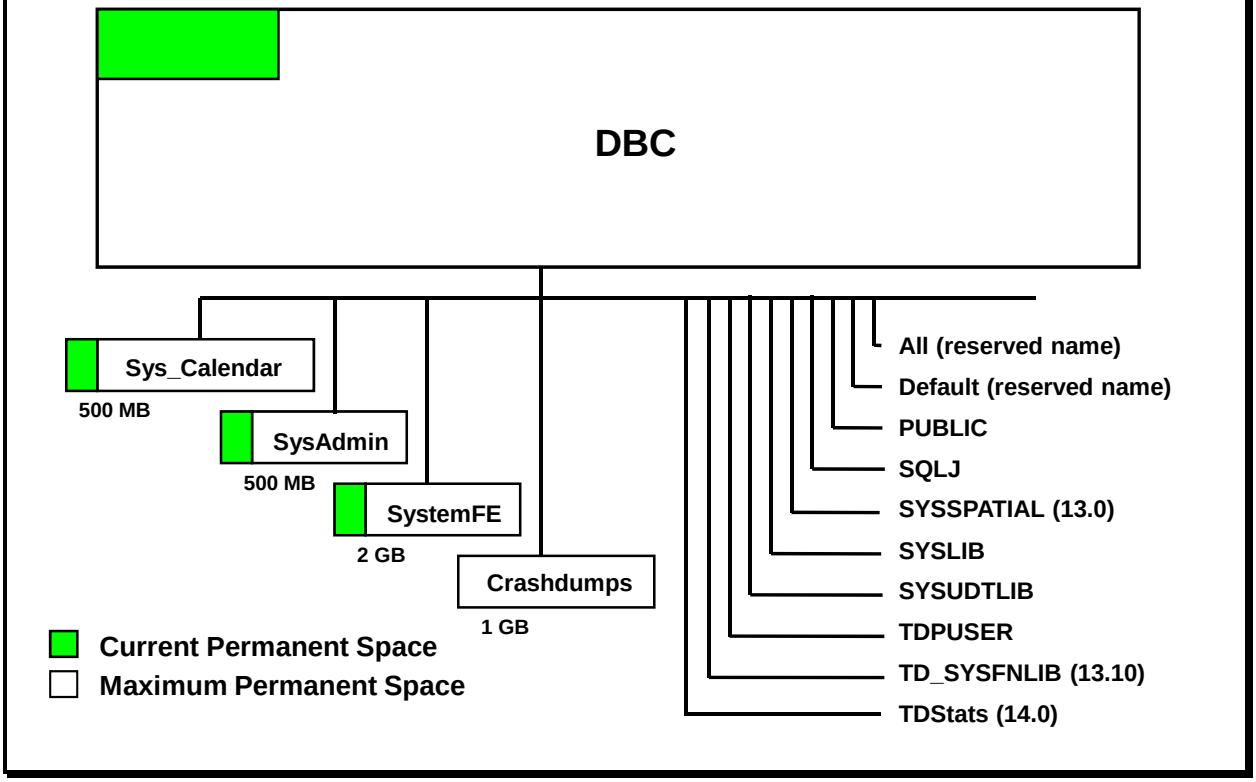
SQLJ, SYSLIB, SYSUDTLIB, and TD_SYSFNLIB Databases

The SYSLIB database can be used to store user-defined functions and the SYSUDTLIB database can be used to store user-defined data types. The SQLJ database (new with Teradata 12.0) contains a series of new views that reference the new dictionary tables to support Java external stored procedures. TD_SYSFNLIB database is new with Teradata 13.10 and supports domain-specific and temporal functions.

TDStats

This database contains collected statistics starting with Teradata 14.0).

Initial Teradata Database



Administrative User

System user DBC contains all Teradata Database software components and all system tables.

Before you define application users and databases, you should first use the CREATE USER statement to create a special administrative user to complete these tasks.

The amount of space for the administrative user is allocated from DBC's current PERM space. DBC becomes the owner of your administrative user and of all users and databases you subsequently create.

Be sure to leave enough space in DBC to accommodate the growth of system tables and logs.

You can name the user anything you would like. We have called the user SYSDBA.

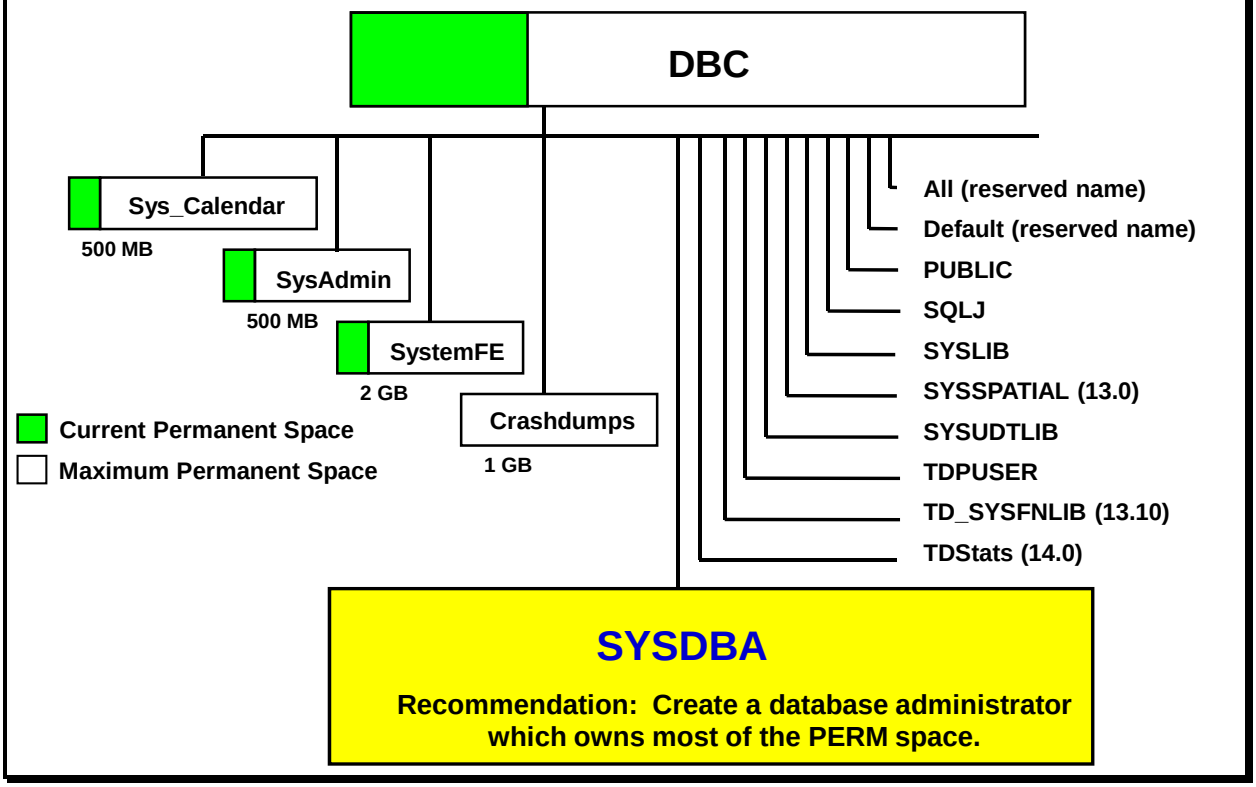
Create the administrative user, and then logon as that user to protect sensitive data in DBC. In addition, change and secure the DBC password.

To ensure perm space is from the administrative user, logon as that user to add other users and databases.

Notes:

- All space in the Teradata Database is owned. No disk space known to the system is unassigned or not owned.
- Think of a user as a database with a password. Both may contain (or "own") tables, views and macros.
- Both users and databases may hold privileges.
- Only users may logon, establish a session with the Teradata Database, and submit requests.

Administrative User



Owners, Parents and Children

As you define users and databases, a hierarchical relationship among them will evolve.

When you create new objects, you subtract permanent space from the assigned limit of an existing database or user. A database or user that subtracts space from its own permanent space to create a new object becomes the immediate owner of that new object.

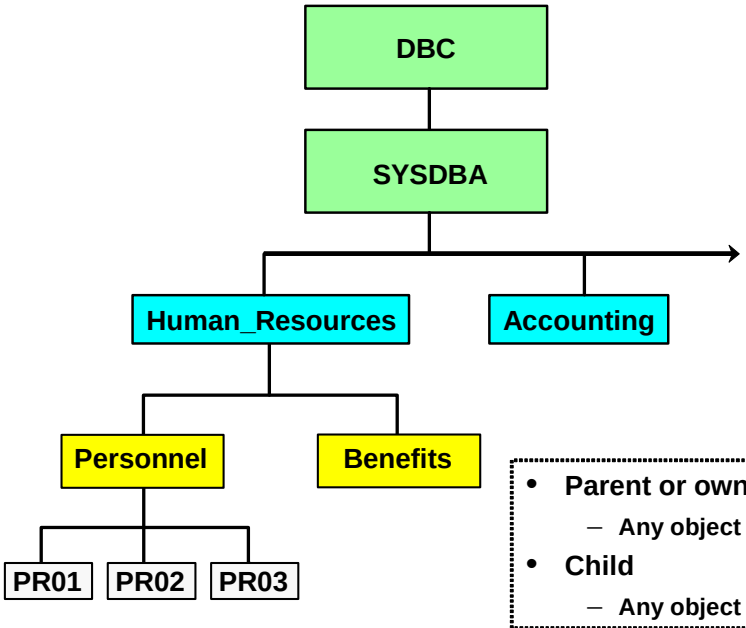
An “owner” or “parent” is any object above you in the hierarchy through which a direct line of ownership is established during the creation process. (Note that you can use the terms owner and parent interchangeably.) A “child” is any object below you in the hierarchy through which a direct line of ownership is established during the creation process.

The term “immediate parent” is sometimes used to describe a database or user just above you in the hierarchy.

Example

The diagram on the facing page illustrates Teradata system hierarchy. System user DBC is the owner, or parent, of all the objects in the hierarchy. The administrative user (SYSDBA) is the owner of all objects below it, such as Human Resources, Accounting, Personnel and Benefits. These objects are also children of DBC, since DBC owns SYSDBA.

Owners, Parents, and Children



- Parent or owner
  - Any object directly above you in the hierarchy
- Child
  - Any object directly below you in the hierarchy

Users may own databases and databases may own users.

Creating New Users and Databases

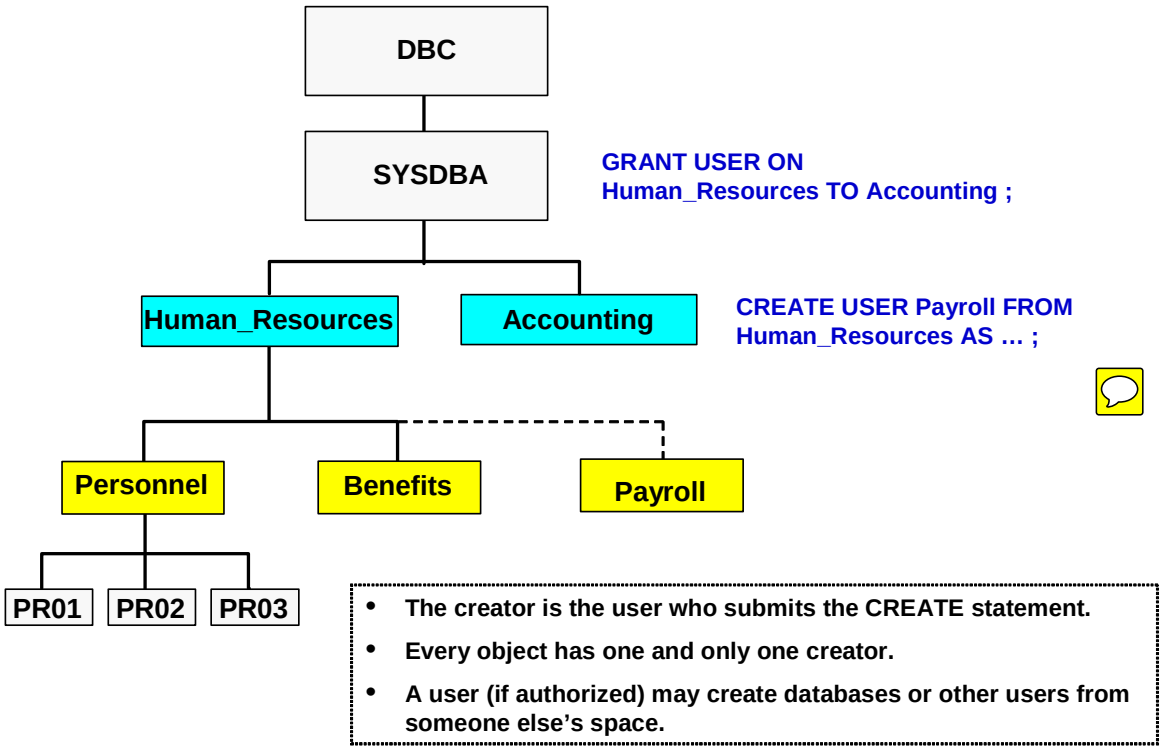
The “creator” of an object is the user who submitted the CREATE statement.

Every object has one and only one creator. If you are the creator of a new space, you automatically have access rights to that space and anything created in it.

Notes:

- While you may be the creator of an object, you are not necessarily the owner of the database or user that contains the object.
- You are the owner of an object if the new object is directly below you in the hierarchy.
- As a creator, you can submit a CREATE statement that adds a new object somewhere else in the hierarchy, assuming you have the appropriate privileges. In this instance, the creator (you) and the owner are two different users or databases.
- If authorized, you may create databases or users FROM someone else's space.
- You can transfer databases and users from one owner to another.

Creating New Users and Databases



- The creator is the user who submits the CREATE statement.
- Every object has one and only one creator.
- A user (if authorized) may create databases or other users from someone else's space.

Transfer of Ownership

The GIVE statement transfers a database or user space to a recipient you specify. The GIVE statement also transfers all child databases and users as well as the tables, views and macros owned by the transferred object.

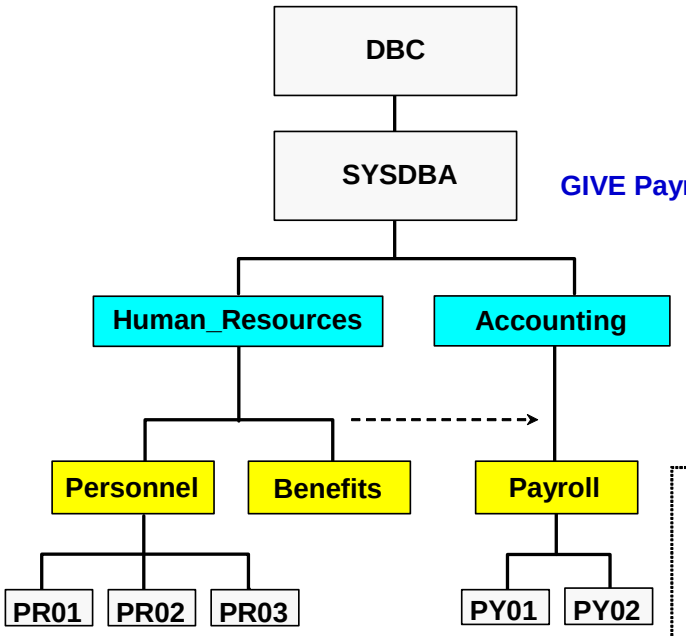
Rules affecting transfer of ownership:

- Use the GIVE statement to transfer databases and users only. (You cannot use the GIVE statement to transfer tables, views, and macros from one database to another.)
- To transfer an object, you must have DROP DATABASE privilege on the object to transfer and CREATE DATABASE privilege on the receiving object.
  - Even though you may be transferring a USER to another user, you need the CREATE DATABASE privilege on the user that is going to get the transferred user. A CREATE USER privilege will not work.
- You cannot give an object to one of its children.
- During a transfer, you transfer all objects the object owns.
- Transfer of ownership affects space ownership and access right privileges. When you transfer an object, the space the object owns is also transferred. The implications of how access rights are affected will be described in more detail later in this course.

Example

In the illustration on the facing page, the administrative user, SYSDBA, GIVES the user, Payroll, to Accounting. The original owner, Human_Resources, loses ownership of the perm space that belonged to Payroll. The new owner, Accounting, acquires ownership of the perm space that Payroll brings with it. All objects that belong to Payroll are transferred as well, including its tables, views, macros and children databases.

Transfer of Ownership



GIVE Payroll TO Accounting;

- The GIVE statement is valid to only transfer databases and users.
- To give an object, you must have:
  - DROP DATABASE privilege on the object to transfer.
  - CREATE DATABASE privilege on the receiving object.

DELETE/DROP Statements

DELETE DATABASE and DELETE USER statements delete all data tables, views, and macros from a database or user. The database or user remains in the Teradata Database as a named object and retains the available space. None of that space is any longer in use. All space used by the deleted objects becomes available as spool space until it is reused as perm space.

You must have DROP DATABASE or DROP USER privilege on the referenced database or user to delete objects from them. The database or user that you are dropping cannot own other databases or users.

DELETE USER Example

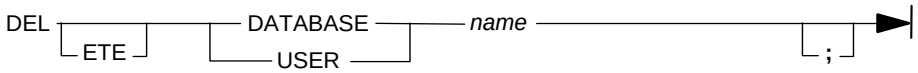
The diagram on the facing page illustrates a DELETE USER statement. Assume the user Personnel has three tables: TB01, TB02, and TB03. Human Resources logs on to the system and submits the DELETE USER statement on user Personnel. All tables are deleted from the user space owned by Personnel. The DELETE DATABASE/USER command does NOT delete a permanent journal, join indexes, or hash indexes.

DROP USER Example

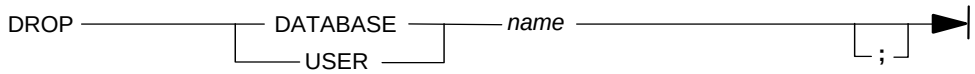
The DROP DATABASE or DROP USER statement drops empty databases or users only. You must delete all objects associated with the database or user before you can drop the DATABASE or USER. When you drop a database or user, its perm space is credited to the immediate owner.

The diagram on the facing page illustrates the DROP USER statement. Human Resources submits the DROP USER statement on user Personnel. The user Personnel is dropped from the hierarchy. The user space that belonged to user Personnel is returned to its parent, Human Resources.

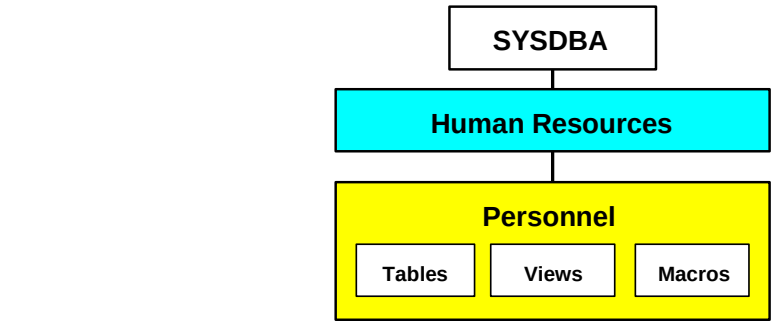
DELETE USER Syntax



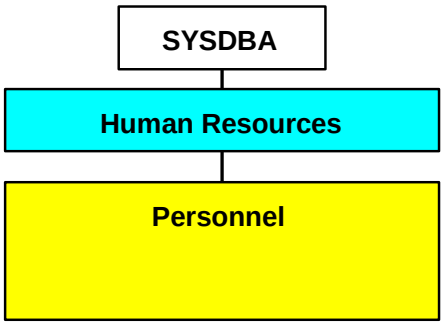
DROP USER Syntax



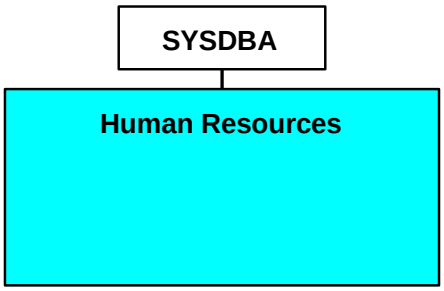
DELETE / DROP Statements



DELETE USER Personnel;



DROP USER Personnel;



Teradata Administrator – New System

Teradata Administrator (previously named WinDDI for Windows Data Dictionary Interface) is the Teradata Manager application that you can use to perform database administration tasks on the associated Teradata Database computer.

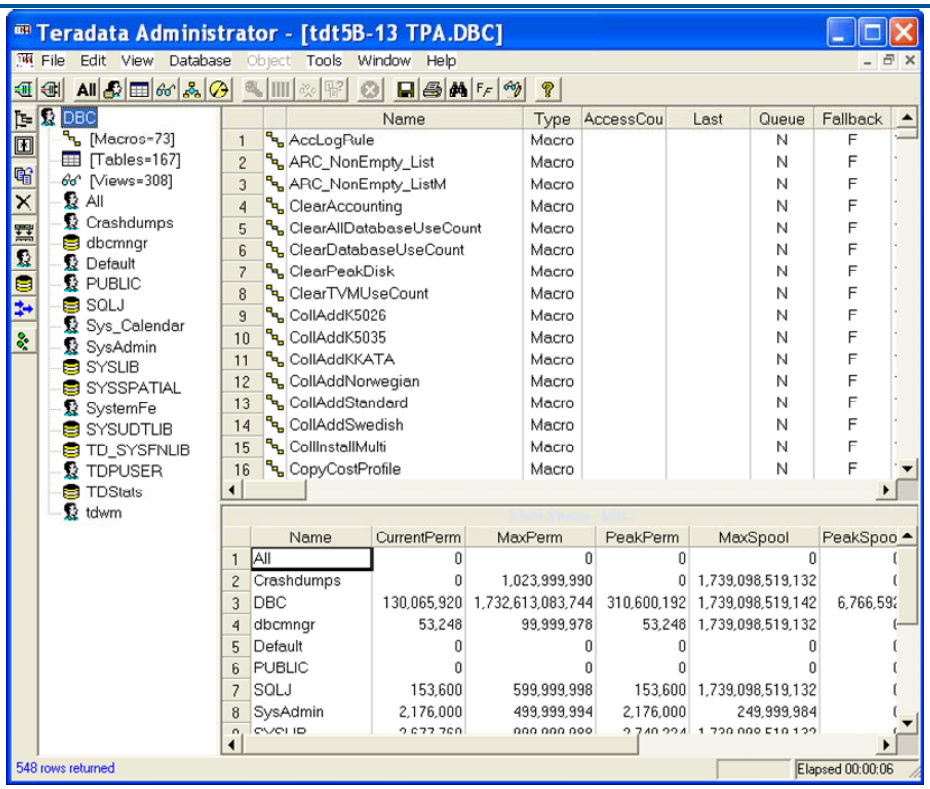
The facing page contains an example of the users and databases that exist in a newly initialized Teradata 14.0 system.

Teradata Administrator
New System

Teradata Administrator

- **GUI interface to Teradata hierarchy and objects.**
- **This example shows the default users and databases in a newly initialized Teradata 14.0 system.**

Which database has the majority of the Perm space?



Teradata Administrator – Hierarchy

The facing page contains an example of a screen display from a Teradata system and illustrates the hierarchy in the left pane.

You may use Teradata Administrator to perform the following functions:

- Create, modify and drop users or databases
- Create tables (using ANSI or Teradata syntax)
- Grant or revoke access/monitor rights
- Copy table, view or macro definitions to another database, or to another system
- Drop or rename tables, views or macros
- Move space from one database to another
- Run an SQL query
- Display information about a Database (list of tables, views, macros, child databases, rights)
- Display information about a Table (columns, journals, indexes, row counts, users, space summary), View (columns, info, rights, row count, users, show), or Macro (rights, users, info, show)

Teradata Administrator keeps a record of all the actions you take and can optionally save this record to a file. This record contains a time stamp together with the SQL that was executed, and other information such as the statement's success or failure.

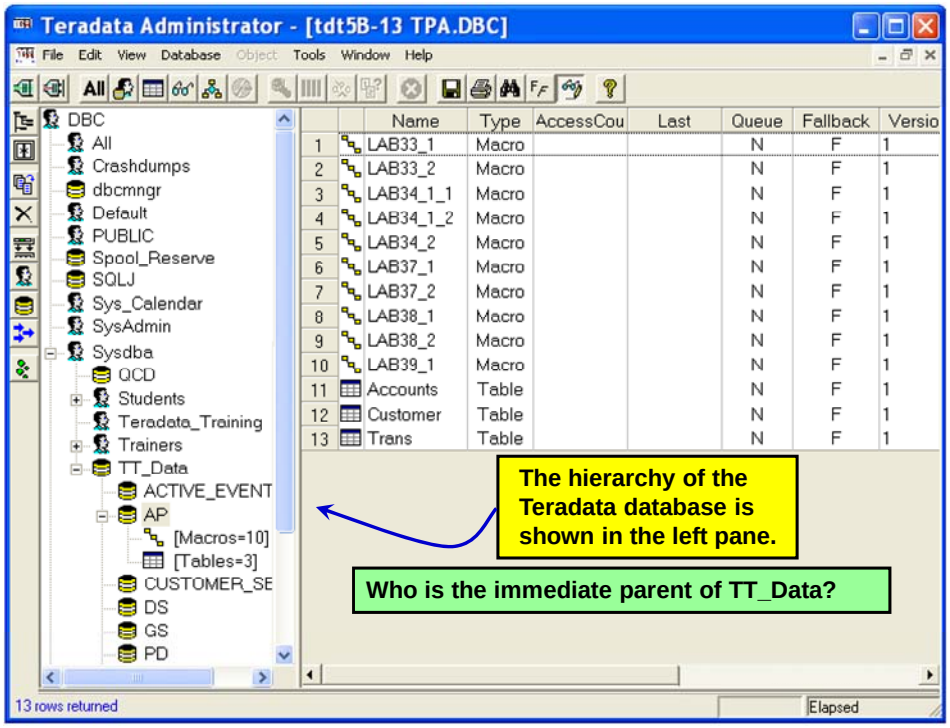
To use the viewing functions of Teradata Administrator, you must have Select access to the DBC views of the Teradata DBS. To use the Copy, Drop, Create or Grant tools you must have the corresponding privilege on the table or database that you are trying to modify or create. To use the Browse or Row Count features you must have select access to the Table or View.

Additional examples of Teradata Administrator displays and capabilities will be shown in various modules throughout this course.

Teradata Administrator – Hierarchy

Teradata Administrator

- **This example shows the hierarchy of a Teradata system and the objects in one of the databases.**
- **This utility also provides drag and drop capabilities.**



Summary

The facing page summarizes some important concepts regarding this module.

Summary

- Initially, system user DBC owns all space in the Teradata Database except that owned by system users and databases.
- The database administrator should create a special administrative user containing most of the space available which will become the owner of all administrator-defined application databases and users.
- Everyone directly higher in the hierarchy is a *parent* or *owner*. Everyone directly lower in the hierarchy is a *child*.
- Every object has one and only one creator. The creator is the user who executes the CREATE statement.
- The GIVE statement enables you to transfer a database or user. The following privileges are necessary:
  - DROP DATABASE on the given object.
  - CREATE DATABASE on the receiving object.
- You cannot DROP databases or users that own objects (tables, views, macros, journals or children databases/users).
- Teradata Administrator provides an easy-to-use Windows-based graphical interface to the Teradata Database Data Dictionary.

Module 41: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 41: Review Questions

1. True or False. You should use system user DBC to create application users and databases.

2. True or False. A database or user can have multiple Owners, but only one Creator.

3. True or False. An Owner and a Parent are two different terms that mean the same thing.

4. True or False. An Owner and a Creator are two different terms that mean the same thing.

5. True or False. An administrative user (e.g., Sysdba) will never have more permanent space than DBC.

6. True or False. The GIVE statement transfers a database or user space to a recipient you specify. It does not automatically transfer all child databases.

Notes

Module 42

The Data Dictionary

After completing this module, you will be able to:

- **Summarize information contained in the Data Dictionary tables.**
- **Differentiate between restricted and unrestricted views.**
- **Use the supplied Data Dictionary views to retrieve information about created objects.**

Notes

Table of Contents

Data Dictionary / Directory....42-4
Fallback Protected Data Dictionary Tables....42-6
Non-Hashed Data Dictionary Tables....42-8
Updating Data Dictionary Tables....42-10
Supplied Data Dictionary Views....42-12
Restricted Views....42-14
Suffix Options with Views....42-16
Selecting Information about Created Objects....42-18
Children View....42-20
Databases View....42-22
Users View....42-24
Tables View....42-26
Columns View....42-28
Indices View....42-30
Partitioning Constraints View....42-34
Show Table Checks View....42-36
Show Column Checks View....42-38
Triggers View....42-40
All Temporary Tables View....42-42
Referential Integrity Views....42-44
Using the DBC.Tables View....42-46
Referential Integrity States....42-48
DBC.All_RI_Children View....42-50
DBC.Databases2 View....42-52
Time Stamps in Data Dictionary....42-54
Teradata Administrator – List Columns of a View....42-56
Teradata Administrator – Object Options....42-58
Summary....42-60
Module 42: Review Questions....42-62
Lab Exercise 42-1....42-64
Lab Exercise 42-2 (optional)....42-70

Data Dictionary / Directory

The data dictionary/directory is a complete database composed of system tables, views, and macros that reside in system user DBC.

It is referred to as a Dictionary / Directory because it provides two functions:

- Dictionary – information you can view (e.g., you can view the columns and their attributes of a table)

- Directory – information to control the system (e.g., table names are converted to table IDs for the software to use)

The Teradata Data Dictionary / Directory is usually referred to as the Teradata Data Dictionary.

Data dictionary tables are present when you install the system.

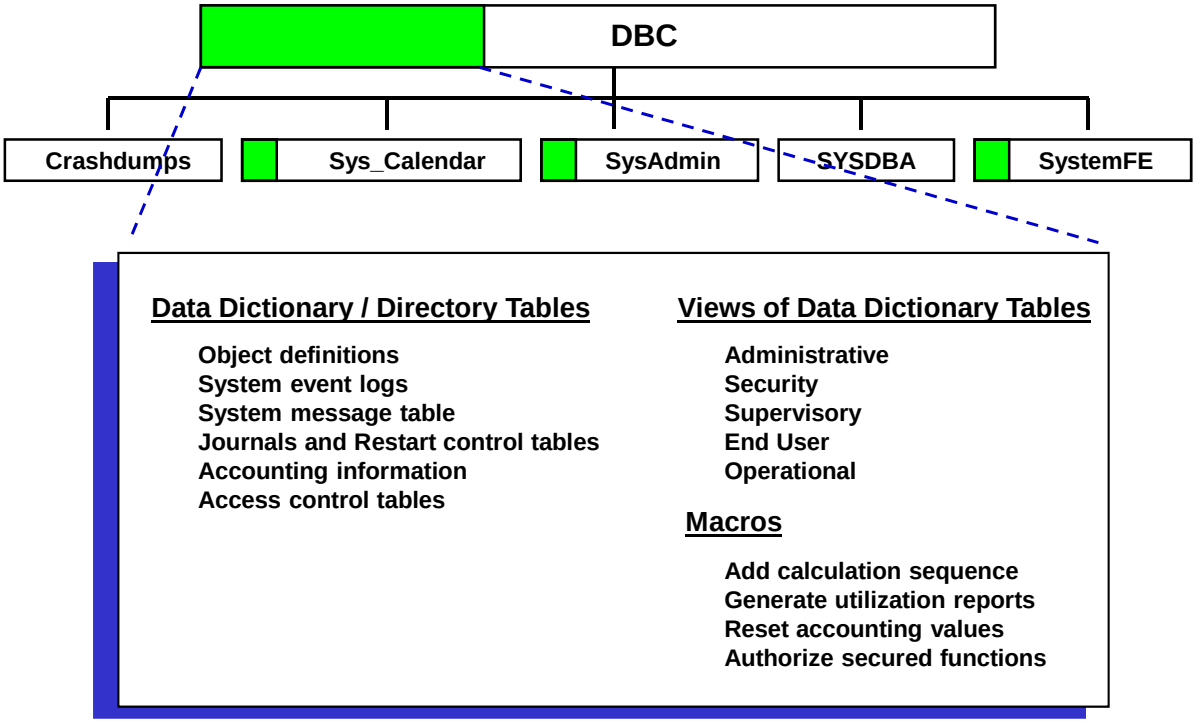
The system references some of these tables with SQL requests, while others are used for system or data recovery only.

Data dictionary views reference data dictionary tables. The system views and macros are created by running the Database Initialization Program (DIP) scripts. When a system is first installed, the **start dip** utility is executed by the installation person/team.

Data dictionary tables are used to:

- Store definitions of objects you create (e.g., databases, tables, indexes, etc.).
- Record system events (e.g., logon, console messages, etc.).
- Hold system message texts.
- Control system restarts.
- Accumulate accounting information.
- Control access to data.

Data Dictionary / Directory



Fallback Protected Data Dictionary Tables

Most data dictionary tables are fallback protected.

Fallback protection means that a copy of every table row is maintained on a different AMP vproc in the configuration. Fallback-protected tables are always fully accessible and are automatically recovered by the system.

Note: Every system database and user includes a dummy table named “ALL” (with an internal TableID of binary zeros). This table represents all the tables in a system database or user when, for example, privileges are granted or disk space is summarized at the database level.

Fallback Protected Data Dictionary Tables

AccessRights *Users Rights on objects*	**AccLogRuleTbl** *Specifies events to be logged*	**AccLogTbl** *Logged User-Object events*	**Accounts** *Account Codes by user*
ALL ***(Dummy) Represents all tables***	**ConstraintNames**	**DBase** *Database and User Profiles*	**DBCInfoTbl** *Software Release & Version*
ErrorMsgs *Message Codes and text*	**EventLog** *Session logon/logoff history*	**Hosts** *To override default char. sets*	**IdCol** *Maintains Identity column data*
Indexes *Defines indexes on tables*	**LogonRuleTbl** *Users Rights on objects*	**Next** *Internal ID for next create*	**OldPasswords** *Encoded password history*
Owners *Hierarchy (Downward)*	**Parents** *Hierarchy (Upward)*	**Profiles** *Users and logon attributes*	**RCConfiguration** *Archive/Recovery Config*
RCEvent *Archive/Recovery events*	**RepGroup** *Replication Groups for Tables*	**ResUsage** *Resource Usage tables*	**ReferencedTbls** *Referential Integrity (PK)*
ReferencingTbls *Referential Integrity (FK)*	**RoleGrants** *Users/Roles assigned to Roles*	**Roles** *Defined Roles*	**SecConstaints** *Security Constraint Objects*
SessionTbl *Current logon information*	**SW_Event_Log** *Database Console Log*	**SysSecDefaults** *Logon security options*	**TVFields** *Table/View column description*
TVM *Tables, Views and Macros*	**TableConstraints** *Table Constraints*	**TempTables** *Materialized Temporary Tables*	**TriggersTbl** *Stores trigger information*

(Partial list of Data Dictionary tables in Teradata 14.0)

Non-Hashed Data Dictionary Tables

The data dictionary tables on the following page contain rows that are *not* distributed using hash maps.

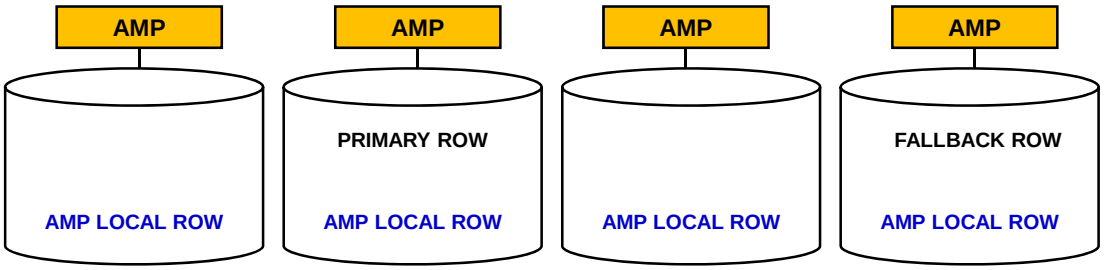
Rows in these tables are stored AMP-locally. For example, the DBC.Acctg table rows represent CPU time and I/O counts and are stored on the same AMP where the CPU time is used and I/Os are executed.

Note: User-defined table rows are *always* hash distributed ... either with or without a fallback copy.

Non-Hashed Data Dictionary Tables

Acctg *Resource usage by user/account*	**ChangedRowJournal** *Down-AMP Recovery Journal*	**DatabaseSpace** *Database and Table space accounting*
LocalSessionStatus *Last request status by AMP*	**LocalTransactionStatus** *Last TXN Consensus status*	**OrdSysChngTable** *Table-level recovery*
RecoveryLockTable *Recovery session locks*	**RecoveryPJTable** *Permanent Journal recovery*	**SavedTransactionStatus** *AMP recovery table*
SysRcvStatJournal *Recovery, reconfig, startup information*	**TransientJournal** *Actually implemented in WAL Log*	**UtilityLockJournalTable** *Host Utility Lock records*

AMP Cluster



Updating Data Dictionary Tables

Whenever you submit a data definition or data control statement, Teradata system software automatically updates data dictionary tables.

When you use the EXPLAIN modifier to describe a DDL statement, you can view updates to the data dictionary tables.

The EXPLAIN modifier is a helpful function that allows you to understand what happens when you execute an SQL statement.

- The statement is not executed.
- The type of locking used is described.
- At least five different tables are updated when you define a new table.

Updating Data Dictionary Tables

```
EXPLAIN     CREATE TABLE Orders
                  ( order_id        INTEGER NOT NULL,
                    order_date      DATE FORMAT 'yyyy-mm-dd',
                    cust_id         INTEGER )
                UNIQUE PRIMARY INDEX (order_id);
```

---

1) First, we lock TFACT.Orders for exclusive use.
2) Next, we lock a distinct DBC."pseudo table" for write on a RowHash for deadlock prevention, we lock a distinct DBC."pseudo table" for read on a RowHash for deadlock prevention, we lock a distinct DBC."pseudo table" for write on a RowHash for deadlock prevention, we lock a distinct DBC."pseudo table" for read on a RowHash for deadlock prevention, and we lock a distinct DBC."pseudo table" for write on a RowHash for deadlock prevention.
3) We lock DBC.Indexes for write on a RowHash, we lock DBC.DBase for read on a RowHash, we lock DBC.TVFields for write on a RowHash, we lock DBC.TVM for write on a RowHash, and we lock DBC.AccessRights for write on a RowHash.
4) We execute the following steps in parallel.
    1) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.
    2) We do a single-AMP ABORT test from DBC.TVM by way of the unique primary index.
    3) We do an INSERT into DBC.TVFields (no lock required).
    4) We do an INSERT into DBC.TVFields (no lock required).
    5) We do an INSERT into DBC.TVFields (no lock required).
    6) We do an INSERT into DBC.Indexes (no lock required).
    7) We do an INSERT into DBC.TVM (no lock required).
    8) We INSERT default rights to DBC.AccessRights for TFACT.Orders.
5) We create the table header.
6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> No rows are returned to the user as the result of statement 1.

Supplied Data Dictionary Views

System views are supplied from the data dictionary for frequently used data. The system views do not contain data. They are stored as entries in the data dictionary until you submit an SQL statement that uses them. Views of data dictionary tables are provided for the same reasons that views are defined for any database application.

Data dictionary table column names are re-titled and formatted. Derived values are computed from data dictionary tables. Most supplied views reference more than one table and have the join syntax included. Supplied views also allow you, as the database administrator, to limit access to data dictionary information and provide a consistent image of the data stored in the data dictionary. In practice, as the administrator you may grant permission to the appropriate members of your organization to use any supplied view.

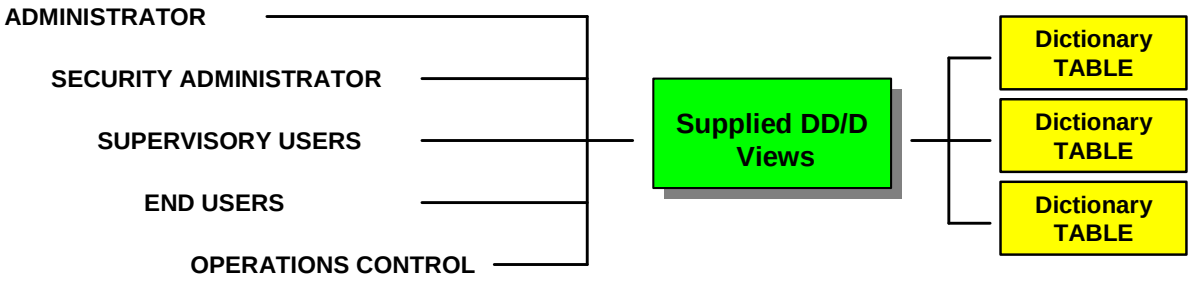
The installation script for the standard views is contained in the supplied Database Initialization Program (DIP) scripts which are normally executed by the installation teams.

The DIP installation screen lists each DIP script separately. You are given the option of choosing "All" to execute all the DIP scripts with one command, or you can choose and run each script separately. There are specific DIP scripts that can be executed to enable specific functions.

When DIP is executed, the following menu (e.g., Teradata 14.0) is provided:

1. DIPERR - Error Messages
2. DIPDEM - UDF/UDT/XSP/SPL Macros
3. DIPRSS - ResUsage Tables
4. DIPVIEWS - System Views
5. DIPOLH - Online Help
6. DIPSYSFE - System FE Macros
7. DIPACR - Access Rights
8. DIPCRASH - CrashDumps Database
9. DIPRUM - ResUsage Views/Macros
10. DIPCAL - Calendar Tables/Views
11. DIPCCS - Client Character Sets
12. DIPOCES - Cost Profiles
13. DIPUDT - UDT Macros
14. DIPSYSFNC - System Functions
15. DIPSQLJ - SQLJ Views/Procedures
16. DIPPWRSTNS - Password Restrictions
17. DIPRCO - Reconfig
18. DIPTDWM - TDWM Configuration
19. DIPSTATS - Automated Stats Mgmt
20. DIPALL - All of the above
21. DIPPDCR - PDCR Tables/Views
22. DIPACC - Access Logging
23. DIPPATCH - Stand-alone patches
24. DIPGLOP - GLOP Tables/Procedures

Supplied Data Dictionary Views



- Views are representations of data accessed/derived from DD/D tables.
  - Clarify tables – re-title tables and/or columns; reorder and format columns, etc.
  - Simplify operations – supply join operation syntax; select and project relevant rows and columns.
  - Limit access to data – exclude certain rows and/or columns from selection.
- View definitions are stored in DBC.TVM.
- View column information is stored in DBC.TVFields.
- DIP scripts install the dictionary views.
- Disclaimer: The view descriptions in this module may not include all the columns in the view. Appendix D includes the complete view and column definitions.

Restricted Views

There are two versions of the system views: restricted [x] and non-restricted [non-x]. The system administrator can load either or both versions.

Non-X views are named according to the contents of their underlying tables. DBC.DiskSpaceV, DBC.TableSizeV, and DBC.SessionInfoV are examples of Non-X views.

X Views are the same views with an appended WHERE clause. The WHERE clause limits the information returned by a view to only those rows associated with the requesting user.

Granted Rights

By default, the SELECT privilege is granted to PUBLIC User on most views in X and non-X versions. This privilege allows any user to retrieve view information via the SELECT statement. The system administrator can use GRANT or REVOKE statements to grant or revoke a privilege on any view to or from any user at any time.

Special Needs Views

Some views are applicable only to users who have special needs. For example, the administrator, a security administrator, or a Teradata field engineer may need to see information that other users do not need. Access to these views is granted only to the applicable user.

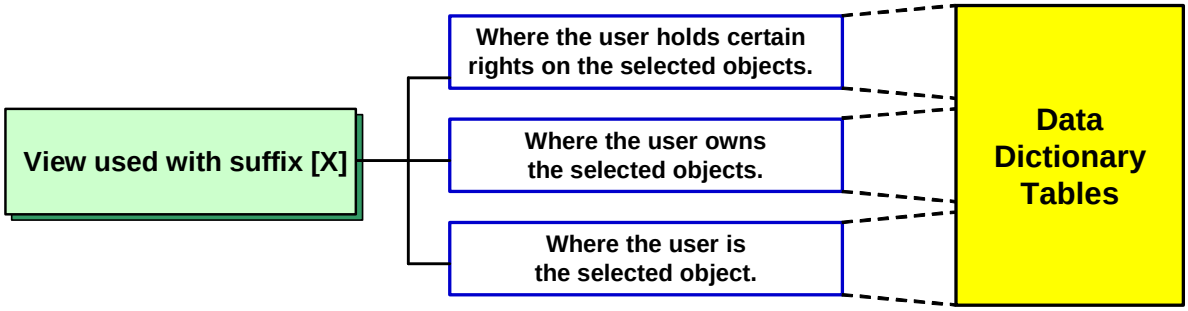
Access Tests

Limited views typically run three different tests before returning information from data dictionary tables to a user. Each test focuses on the user and his or her current privileges. It can take longer to receive a response when a user accesses a restricted view.

Restricted Views

These views limit the scope of what a user can access in the DD/D.

Views with an [X] suffix typically make the following three tests before returning information to the user:



For example,

- **The following query returns information about ALL parents and children.**

  SELECT Child, Parent FROM DBC.ChildrenV;

- **The restricted [x] version of this view selects only information on objects owned by the executing user or that the requesting user has access to:**

  SELECT Child, Parent FROM DBC.ChildrenVX;

Suffix Options with Views

The following are the view forms:

- Without the X (for example, DBC.AccountInfo and DBC.AccountInfoV), they display global information.
- With the X (for example, DBC.AccountInfoX and AccountInfoVX), they display information associated with the requesting user only.
- With the V (for example, AccessLogV), they display information associated with the Unicode version, where object name columns have a data type of VARCHAR(128).
- Without the V (for example, DBC.AccountInfo or DBC.AccountInfoX), they display information associated with the Compatibility version, where object name columns have a data type of CHAR(30).

Operations that use restricted views tend to take longer to run because these views access more data dictionary tables. By contrast, operations that use unrestricted views may run faster but return more rows.

To control access to data dictionary information, you can grant users permission to access only restricted views.

Suffix Options with Views

Views may optionally include a suffix of [V] and/or [X].

- **Views without the V are the "Compatibility" version of the views, where object name columns have a data type of CHAR(30).**
- **In general, the X version is for restricted views.**
- **Starting with Teradata 12.0, most views have a "V" version which is the newer version of the view. This version of a view has object name columns with a data type of VARCHAR(128). This version can display information associated with Unicode.**

For example, the recommended views to use for account information are:

- **DBC.AccountInfoV – displays global information using VARCHAR(128)**
- **DBC.AccountInfoVX – displays information associated with the requesting user only using VARCHAR(128).**

The older "compatibility" views (without the V) are:

- **DBC.AccountInfo – displays global information using CHAR(30)**
- **DBC.AccountInfoX – displays information associated with the requesting user only using CHAR(30).**

Selecting Information about Created Objects

The following views return information about created objects:

Note: The table “Indexes” is referenced by a view spelled “Indices.”

Object Definition System Views

View Name	Data Dictionary Table	Purpose
DBC.Children[V][X]	DBC.Owners	Provides information about hierarchical relationships.
DBC.Databases[V][X]	DBC.DBase	Provides information about databases, users and their immediate parents.
DBC.Users[V]	DBC.DBase	Similar to DataBases view, but includes columns specific to users.
DBC.Tables[V][X]	DBC.TVM	Provides data about tables, views, macros, triggers, and stored procedures.
DBC.ShowTblChecks[V][X]	DBC.TableConstraints	Database table constraint information.
DBC.ShowColChecks[V][X]	DBC.TVFields	Information about columns in tables and views, and parameters in macros.
DBC.Columns[V][X]	DBC.TVFields	Data about columns in tables and views as well as parameters in macros.
DBC.Indices[V][X]	DBC.Indexes	Data about indexes on tables.
DBC.IndexConstraints[V][X]	DBC.DBase DBC.TableConstraints	Provides information about index constraints implied by a partitioning expression
DBC.AllTempTables[V][X]	DBC.TempTables	Information about all global temporary tables materialized in the system.
DBC.Triggers[V][X]	DBC.TriggersTbl	Information about event-driven triggers attached to a single table and stored in the database.

Selecting Information about Created Objects

DBC.Children[V][X]	Hierarchical relationship information.
DBC.Databases[V][X]	Database, user and immediate parent information.
DBC.Users[V]	Similar to Databases view, but includes columns specific to users.
DBC.Tables[V][X]	Tables, views, macros, triggers, and stored procedures information.
DBC.ShowTblChecks[V][X]	Database table constraint information.
DBC.ShowColChecks[V][X]	Database column constraint information.
DBC.Columns[V][X]	Information about columns/parameters in tables, views, and macros.
DBC.Indices[V][X]	Table index information.
DBC.IndexConstraints[V][X]	Provides information about index constraints, e.g., PPI definition.
DBC.AllTempTables[V][X]	Information about global temporary tables materialized in the system.
DBC.Triggers[V][X]	Information about event-driven, specialized procedures attached to a single table and stored in the database.

Children View

The Children view lists the names of databases and users and their parents in the hierarchy.

Column	Definition
Child	Name of a child database or user
Parent	Name of a parent database or user

Example

The diagram on the facing page uses an SQL statement to list the parents of the current user. The SQL keyword USER causes the parser to substitute the "User Name" of the user who has logged on and submitted the statement. The results of the request show one child, student230, and four parents.

Children View

Provides the names of all databases, users and their owners.

DBC.Children[V][X]

Child	Parent

Example:
Using the unrestricted form of the view and a WHERE clause, list your parents.

```
SELECT  *
FROM    DBC.ChildrenV
WHERE   CHILD = USER;
```

Example Results:

Child	Parent
student230	DBC
student230	Students
student230	Sysdba
student230	TT_Class2

Databases View

The Databases view returns information about databases and users from the DBC.DBase table.

Notes:

- Only the *immediate* owner is identified in this view. Use the parent column of the Children view to select *all* owners.

- The data dictionary records the name of the creator of a system user or database, as well as the date and time the user created the object. This information is not used by the software, but is recorded in DBC.DBase for historical purposes.

Column definitions in this view include:

Column	Definition
OwnerName	The IMMEDIATE parent (owner)
ProtectionType	Default protection type for tables created within this database: F = Fallback N = No Fallback
JournalFlag	Two characters (before and after image) where: S = Single D = Dual N = None L = Local For Example: SD = Single before, dual after image NL = (Single) Local after image
CreatorName	Name of the user who created the object.
CreateTimeStamp	Date and time the user created the object.

Example

The SQL request on the facing page uses the Databases view to find the users with the names that start with TFACT and identify the creator, permanent disk space limit, and database type.

Databases View

Provides information about databases and users.

DBC.Databases[V][X]

DatabaseName	CreatorName	OwnerName	AccountName
ProtectionType	JournalFlag	PermSpace	SpoolSpace
TempSpace	CommentString	CreateTimeStamp	LastAlterName
LastAlterTimeStamp	DBKind	AccessCount **	LastAccessTimeStamp **

** Only updated if DBSControl ObjectUseCountCollectRate > 0

Example:

For databases/users with a name of "tfact%", find the creator name, when it was created, its max perm space, and the type (database or user).

```
SELECT    DatabaseName    (CHAR(10)) AS "Name"
         ,CreatorName     (CHAR(10)) AS "Creator"
         ,CreateTimeStamp
         ,PermSpace       (FORMAT 'zzz,zz9,999')
         ,DBKind
FROM      DBC.DatabasesV
WHERE     DatabaseName    LIKE 'TFACT%'
ORDER BY  1;
```

Example Results:

Name	Creator	CreateTimeStamp	PermSpace	DBKind
TFACT	DBC	2011-01-16 19:56:21	20,000,000	D
tfact01	Sysdba	2011-01-17 08:06:52	10,000,000	U
tfact02	Sysdba	2011-01-17 08:06:55	10,000,000	U
tfact03	Sysdba	2011-01-17 08:06:57	10,000,000	U

Users View

The Users view is a subset of the Databases view and:

- Limits rows returned from DBC.DBase to only USER rows (e.g., where there is a password).
- Restricts rows returned to:
- The current users' information.
- Information about owned users or databases (i.e., children).
- Information about users on which the current user has DROP USER or DROP DATABASE rights.
- Date and time a user is locked due to excessive erroneous passwords and the number of failed attempts since the last successful one.

The view features CreatorName and CreateTimeStamp columns that display the user name who created an object and the date and time he or she created it. The LastAlterName and LastAlterTimeStamp columns list the name of the last user to modify an object, as well as the date and time.

Note: The DBC.Users view is already a restricted view; there is no [X] version.

Column definitions in this view include:

Column	Definition
PermSpace	Maximum permanent space available for this user.
SpoolSpace	Maximum spool space available for this user.
DefaultCollation	A = ASCI E = EBCDIC M = Multinational H = Host (default) C = CharSet_Col J = JIS_Coll

Example:
The SQL statement on the facing page finds the user's default account code, name of his or her immediate owner, and spool space limit.

Users View

Provides information about the users that the requesting user owns or to which he or she has modify rights. This is a restricted view … there is no [x] version.

DBC.Users[V]

UserName	CreatorName	PasswordLastModDate	PasswordLastModTime
OwnerName	PermSpace	SpoolSpace	TempSpace
ProtectionType	JournalFlag	StartupString	DefaultAccount
DefaultDatabase	CommentString	DefaultCollation	PasswordChgDate
LockedDate	LockedTime	LockedCount	TimeZoneHour
TimeZoneMinute	DefaultDateForm	CreateTimeStamp	LastAlterName
LastAlterTimeStamp	DefaultCharType	RoleName	ProfileName
AccessCount **	LastAccessTimeStamp **		

** Only updated if DBSControl ObjectUseCountCollectRate > 0

Example:
Find your default account code, the name of your immediate owner, and max spool space.

```
SELECT   UserName
        ,DefaultAccount
        ,OwnerName
        ,SpoolSpace          (FORMAT 'zzz,zz9,999')
FROM     DBC.UsersV
WHERE    UserName = USER;
```

Example Results:

UserName	DefaultAccount	Owner	SpoolSpace
tfact02	$M_9038_&D&H	Teradata_Factory	50,000,000

Tables View

The Tables view accesses the data dictionary table, DBC.TVM, which contains descriptions of objects – tables, views, macros, journals, join indexes, triggers, stored procedures, etc..

The view features a TableKind column that allows you to specify the *kind* of object to reference. The view also features CreatorName and CreateTimeStamp columns that display the name of the user who created an object and the date and time he or she created it. The LastAlterName and LastAlterTimeStamp columns list the name of the last user to modify an object, as well as the date and time.

The PrimaryKeyIndexID column identifies the columns used as the primary index. As the administrator, use this view to find NO FALLBACK tables (where ProtectionType = 'N').

Additional column definitions for this view include:

Column	Definition
Version	A number incremented each time a user alters a table.
RequestText	Returns the text of the most recent DDL statement that was used to CREATE or MODIFY the table.
TableKind	T = Table M = Macro V = View G = Trigger P = Stored Procedure F = User-defined Function I = Join index N = Hash Index J = Journal table O = No Primary Index

The SQL statement on the facing page requests a list of all tables, views and macros that contain the letters “rights” in their name. The response displays the database name, table name, and a code for the type of object. Additional columns with Teradata V2R6 are:

RepStatus – identifies the replicated table status for the table. It is NULL if the table is not a member of any replication group.

UtilVersion – contains the utility version count. This column is modified to match the Version column when a significant change of the table definition occurs that would prohibit an incremental restore or copy of an archive.

QueueFlag – this field specifies the queue option as a single character whose value can be either Y if it is queue table, or N if it is not a queue table.

Tables View

Provides information about objects – tables, views, macros, journals, join indexes, hash indexes, triggers, stored procedures, etc.

DBC.Tables[V][X]

DataBaseName	TableName	Version	TableKind
ProtectionType	JournalFlag	CreatorName	RequestText
CommentString	ParentCount	ChildCount	NamedTblCheckCount
UnnamedTblCheckExist	PrimaryKeyIndexId	RepStatus	CreateTimeStamp
LastAlterName	LastAlterTimeStamp	RequestTxtOverFlow	AccessCount **
LastAccessTimeStamp **	UtilVersion	QueueFlag	CommitOpt

(this is not a compete list of columns – Appendix D has compete list of columns)

** Only updated if DBSControl ObjectUseCountCollectRate > 0

Example:
List the objects that contain the characters "rolerights" in their name.

```
SELECT    TRIM(DatabaseName) || '.' || TableName AS "Qualified Name"
          ,TableKind
FROM      DBC.TablesV
WHERE     TableName LIKE '%rolerights%'
ORDER BY  1, 2 ;
```

Example Results:

Qualified Name	TableKind
DBC.AllRoleRights	V
DBC.AllRoleRightsV	V
DBC.UserRoleRights	V
DBC.UserRoleRightsV	V

Columns View

The Columns view returns information from the DBC.TVFields table.

This data dictionary table includes information about:

- Table and view columns
- Macro and stored procedure parameters

Like several other views in this module, the Columns view features CreatorName and CreateTimeStamp columns that display the name of the user who created an object and the date and time he or she created it. The LastAlterName and LastAlterTimeStamp columns list the name of the last user to modify an object, as well as the date and time.

As an administrator, you may use this view to enforce domain constraints. The following SELECT statement provides an example:

```
SELECT    DatabaseName, TableName  FROM  DBC.TVFields
WHERE     ColumnTitle LIKE 'amount'
AND       ColumnType NE 'D'
OR        DecimalTotalDigits NE 7
OR        DecimalFractionalDigits NE 2
ORDER BY  1,2;
```

Some of the common column types for this view include:

I = INTEGER	F = FLOAT	BF = BYTE Fixed	CV = VARCHAR
I1 = BYTEINT	DA = DATE	BV = BYTE Variable	BO = BLOB
I2 = SMALLINT	D = DECIMAL	CF = CHARACTER Fixed	CO = CLOB
I8 = BYTEINTEGER	GF = GRAPHIC	GV = VARGRAPHIC	UT = UDT Type
TS = TIMESTAMP	TZ = TIME w ZONE	SZ = TIMESTAMP w ZONE	

Example

The SQL statement on the facing page displays selected parameters for the "ResCPUbyAMP" macro.

Miscellaneous Notes:

- The SPParameterType field specifies the type of the parameter in case of stored procedure object as I (in), O (out) and B (inout).
- The UpperCaseFlag field indicates whether the column is to be stored in uppercase and whether comparisons on the column are case specific. U = Uppercase and not specific, N = not uppercase and not specific, C = not uppercase and specific.
- The CompressValueList field contains the list of values that will be compressed from the column.
- ColumnUDTName – this field specifies the name of a UDT if that column data type is a UDT (User Defined Type).

Columns View

Provides information about columns in tables and views, and parameters in macros and stored procedures.

DBC.Columns[V][X]

DatabaseName	TableName	ColumnName	ColumnFormat
ColumnTitle	SPParameterType	ColumnType	UDTColumnName
ColumnLength	DefaultValue	Nullable	CommentString
DecimalTotalDigits	DecimalFractionalDigits	ColumnId	UpperCaseFlag
Compressible	CompressValue	ColumnConstraint	ConstraintCount
CreatorName	CreateTimeStamp	LastAlterName	LastAlterTimeStamp
CharType	IdColType	AccessCount **	LastAccessTimeStamp **
CompressValueList	TimeDimension *	VTCheckType *	TTCheckType *

* 13.10

** Only updated if DBSControl ObjectUseCountCollectRate > 0

Example:

Use this view to show the parameters of the DBC.ResCPUbyAMP macro.

```
SELECT   ColumnName, ColumnFormat, DefaultValue
FROM     DBC.ColumnsV
WHERE    DatabaseName = 'DBC' AND TableName = 'ResCPUbyAMP' ;
```

Example Results:

ColumnName	ColumnFormat	DefaultValue
FromDate	YYYY-MM-DD	Date
FromNode	X(6)	'000-00'
FromTime	99:99:99	0.00000000000000E000
ToTime	99:99:99	9.99999000000000E005
ToDate	YYYY-MM-DD	Date
ToNode	X(6)	'999-99'

Indices View

The Indices view returns information about each indexed column from the DBC.Indexes table. (A compound index returns multiple rows.) Use the view to list tables with non-unique primary indexes (NUPI). (These tables may be subject to skewed data distribution.)

The following SELECT statement shows an example of how to list NUPI tables:

```
SELECT      DatabaseName, TableName
FROM        DBC.Indices
WHERE       IndexType = 'P' AND UniqueFlag = 'N';
```

Column definitions for this view include:

Column	Definition
IndexType	P = Primary (non-partitioned table) Q= Primary (partitioned table) S = Secondary U = Unique constraint (USI with NOT NULL) K = Primary Key J = Join V = Value-ordered secondary H = Hash-ordered ALL (covering) secondary O = Value-ordered ALL (covering) secondary I = Ordering column of a composite secondary index M = Multi-Column Statistics D = Derived column partition statistics Q = Partitioned Primary Index
UniqueFlag	Y = Unique N = Non-unique
Column Position	Position of a column within an index. This value may be greater than one if the column is part of a composite index. 1 = Field 1 column of a join index 2 = Field 2 column of a join index

Example

The example on the facing page shows a data request about the Employee_Phone table indices in the current database. The result displays three rows. Notice that the secondary index is a composite index consisting of two columns.

Miscellaneous Notes:

IndexMode is H (secondary index rows are hash distributed to the AMPs), L (secondary index rows are on the same AMP as the referenced data row), or NULL (primary index). If the index type is J or N, index mode is L but has no meaning.

These 13.10 columns (UniqueOrPK, VTConstraintType, and TTConstraintType) only apply when the column is associated with a time dimension. The values in the column will be NULL for all tables NOT associated with a time dimension.

TT – Transaction Time Constraint; VT – Valid Time Constraint

Indices View

Provides information about each indexed column defined for each table.

DBC.Indices[V][X]

DatabaseName	TableName	IndexNumber	IndexType
UniqueFlag	IndexName	ColumnName	ColumnPosition
CreatorName	CreateTimeStamp	LastAlterName	LastAlterTimeStamp
IndexMode	AccessCount **	LastAccessTimeStamp **	UniqueOrPK*
VTConstraintType*	TTConstraintType*	SystemDefinedJI*	

* 13.10

** Only updated if DBSControl ObjectUseCountCollectRate > 0

Example:

Select information about the Employee Phone table indices in the current database.

```
SELECT      ColumnName   (CHAR(15))  AS  "Column Name"
           ,UniqueFlag                AS  "Unique"
           ,IndexType                 AS  "Type"
           ,IndexName                 AS  "Name"
           ,IndexNumber               AS  "IndNo"
           ,ColumnPosition            AS  "ColPos"
FROM       DBC.IndicesV
WHERE      TableName = 'Emp_Phone'
AND        DatabaseName = DATABASE
ORDER BY   IndNo, ColPos ;
```

Example Results:

Column Name	Unique	Type	Name	IndNo	ColPos
employee_number	N	P	?	1	1
area_code	N	S	ac_phone	4	1
phone_number	N	S	ac_phone	4	2

Indices View (Second Example)

Column definitions for this view include:

Column	Definition
IndexType	P = Primary S = Secondary U = USI (U is used if USI is created via UNIQUE constraint) J = Join V = Value-ordered secondary H = Hash-ordered ALL (covering) secondary O = Value-ordered ALL (covering) secondary I = Ordering column of a composite secondary index
UniqueFlag	Y = Unique N = Non-unique
Column Position	Position of a column within an index. This value may be greater than one if the column is part of a composite index. 1 = Field 1 column of a join index 2 = Field 2 column of a join index

An IndexType of U indicates a USI that is created via a UNIQUE constraint. However, a USI that is created as a UNIQUE INDEX in the table definition is still identified via the IndexType and UniqueFlag.

Example:

The example on the facing page shows a data request about the Department table indices in the current database. Notice that this example contains a join index.

Indices View (Second Example)

Example:
Select information about the Department table indices in the current database.

```
SELECT    ColumnName   (CHAR(15))  AS  "Column Name"
          ,UniqueFlag               AS  "Unique"
          ,IndexType                AS  "Type"
          ,IndexName                AS  "Name"
          ,IndexNumber              AS  "IndNo"
          ,ColumnPosition           AS  "ColPos"
FROM      DBC.IndicesV
WHERE     TableName = 'Department'
AND       DatabaseName = DATABASE
ORDER BY  IndNo, ColPos ;
```

Example Results:

Column Name	Unique	Type	Name	IndNo	ColPos
dept_number	Y	P	?	1	1
dept_name	Y	U	?	4	1
dept_number	N	J	Dept_Jnlx	8	1
dept_name	N	J	Dept_Jnlx	8	2
dept_mgr_number	N	J	Dept_Jnlx	8	3

SQL of how this Join Index was created:

```
CREATE JOIN INDEX Dept_Jnlx AS  SELECT          dept_number, dept_name, dept_mgr_number
                                FROM            Department
                                PRIMARY INDEX (dept_mgr_number);
```

Partitioning Constraints View

DBC.PartitioningConstraintsV provides information about index constraints, specifically a partitioning expression constraint. A partitioning expression is an implied index constraint.

A ConstraintType = Q indicates a partitioned primary index.

The ConstraintText indicates the partitioning constraint. The general format of the text will be:

CHECK ((<partitioning-expression>) BETWEEN 1 AND <max>)

<max> is 65535 or the number of partitions defined by the partitioning expression if the partitioning expression consists solely of a RANGE_N or CASE_N function.

The definition of the Sales_History table is:

```
CREATE SET TABLE DS.Sales_History, NO FALLBACK,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL
    (
     store_id INTEGER NOT NULL,
     item_id INTEGER NOT NULL,
     sales_date DATE FORMAT 'YYYY-MM-DD',
     total_revenue DECIMAL(9,2),
     total_sold INTEGER,
     note VARCHAR(256) CHARACTER SET LATIN NOT CASESPECIFIC)
UNIQUE PRIMARY INDEX ( store_id ,item_id ,sales_date )
PARTITION BY RANGE_N(sales_date  BETWEEN
DATE '2003-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' MONTH );
```

An example of constraint text for the Sales_History table is:

```
CHECK ((RANGE_N("sales_date"  BETWEEN
     DATE '2003-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' MONTH ))
     BETWEEN 1 and 65535)
```

The DBC.PartitioningConstraintsV is new starting with Teradata 14.0. The previous view was named DBC.IndexConstraints and is still available. However, it is recommended to use the new view as it contains additional information.

Partitioning Constraints View

This Teradata 14.0 view is the recommended view to use to display information about partitioning constraints for tables or join indexes.

- **The previous view was named DBC.IndexConstraintsV and is still available.**

DBC.PartitioningConstraintsV[X]

DatabaseName	TableName	IndexName	IndexNumber
ConstraintType	ConstraintText	ConstraintCollation	CollationName
CreatorName	CreateTimeStamp	CharSetID	SessionMode
ResolvedCurrent_Date	ResolvedCurrent_TimeStamp	DefinedMaxPartitions (14.0)	
MaxCombinedPartitions(14.0)	PartitioningLevels (14.0)	ColumnPartitioningLevel (14.0)	

Example:
List all of the partitioning expression constraints for all tables in the current database.

```
SELECT    TableName                AS "Table Name"
         ,ColumnPartitioningLevel  AS "CP"
         ,PartitioningLevels       AS "# PLevels"
         ,ConstraintText           AS "Constraint Text"
FROM      DBC.PartitioningConstraintsV
WHERE     DatabaseName = USER;
```

Results:

Table Name	CP	# PLevels	Constraint Text
Orders_PPI	0	1	CHECK ((RANGE_N("orderdate" BETWEEN DATE '200
Orders_PPI_ML	0	2	CHECK (/*02*/ RANGE_N(orderdate BETWEEN DATE
Orders_CP	1	1	CHECK (/*01 02 01*/ PARTITION#L1 /*1 12+65522*/ =1)
Orders_CP_TP	1	2	CHECK (/*02 02 01*/ PARTITION#L1 /*1 12+10*/ =1 AND

Show Table Checks View

The ShowTblChecks view displays database table constraint information. The view features CreatorName and CreateTimeStamp columns that display the name of the user who created an object, and the date and time he or she created it. The LastAlterName and LastAlterTimeStamp columns list the name of the last user to modify an object, as well as the date and time.

Column definitions for this view include:

Column	Definition
DatabaseName	Names of databases that contain tables with table-level checks.
TableName	Table names that have table-level constraints.
CheckName	Table-level check name.
TblCheck	Returns the text for the table level or named column level check constraint.
CreatorName	Name of the user who created the object.
CreateTimeStamp	The time the object was created.

The SQL to create these table level constraints follows:

```
ALTER TABLE Employee ADD CONSTRAINT Emp_Chk1
  CHECK (Employee_number >= 100000);

ALTER TABLE Department ADD CONSTRAINT Dept_Chk1
  CHECK (Dept_number >= 1000);

ALTER TABLE Job
  ADD CHECK (Job_code >= 3000);
```

Note: If a check constraint is created at the column level and it is a named constraint, then it will appear in this view.

```
CREATE SET TABLE TFACT.Dept2 ,
    (   dept_number INTEGER NOT NULL
        CONSTRAINT Dept_chk2 CHECK (dept_number >= 1000),
        dept_name CHAR(20)  NOT NULL UNIQUE,
        dept_mgr_number INTEGER,
        budget_amount DECIMAL(10,2) )
UNIQUE PRIMARY INDEX ( dept_number );
```

Show Table Checks View

Provides information about check constraints at the table level and "named" column constraints.

DBC.ShowTblChecks[V][X]

DatabaseName	TableName	CheckName	TblCheck
CreatorName	CreateTimeStamp		

Example:
Display table constraint information.

```
SELECT      TableName (CHAR(10))
           ,CheckName (CHAR(10))
           ,TblCheck
FROM        DBC.ShowTblChecksV
WHERE       DatabaseName = 'TFACT';
```

Example Results:

TableName	CheckName	TblCheck
DEPARTMENT	Dept_Chk1	CONSTRAINT "Dept_Chk1" CHECK ("Dept_nu
EMPLOYEE	Emp_Chk1	CONSTRAINT "Emp_Chk1" CHECK ("Employe
JOB	?	CHECK ("Job_code" >= 3000)

Note: The first two are named constraints and the third is an unnamed constraint. All three of these constraints were created at the table level.

Show Column Checks View

The ShowColChecks view displays database column constraint information for unnamed column level constraints. The view features CreatorName and CreateTimeStamp columns that display the name of the user who created an object and the date and time he or she created it.

Column definitions for this view include:

Column	Definition
DatabaseName	Names of databases that contain tables with column-level checks.
TableName	Table names that have column-level constraints.
ColumnName	Names of columns that contain column-level check constraints.
ColCheck	Returns text for the column-level check condition.
CreatorName	Name of the user who created the object.
CreateTimeStamp	The time the object was created.

The SQL to create these column level constraints follows:

```
CREATE SET TABLE TFACT.Emp_2
  (employee_number INTEGER NOT NULL  CHECK (employee_number >= 100000),
     dept_number       INTEGER,
           :
  salary_amount DECIMAL(10,2))
UNIQUE PRIMARY INDEX ( employee_number );

CREATE SET TABLE TFACT.Dept_2
    (dept_number INTEGER NOT NULL       CHECK (dept_number >= 1000),
        :
     budget_amount DECIMAL(10,2))
UNIQUE PRIMARY INDEX ( dept_number );

CREATE SET TABLE TFACT.Job_2
    (job_code INTEGER NOT NULL          CHECK (Job_code >= 3000),
     job_desc CHAR(20) NOT NULL UNIQUE)
UNIQUE PRIMARY INDEX ( job_code );
```

Show Column Checks View

Provides information about unnamed column check constraints.

DBC.ShowColChecks[V][X]

DatabaseName	TableName	ColumnName	ColCheck
CreatorName	CreateTimeStamp		

Example:
Show information about column constraints for a database.

```
SELECT    TableName      (CHAR(10))
         ,ColumnName     (CHAR(10))
         ,ColCheck
FROM      DBC.ShowColChecksV
WHERE     DatabaseName = 'TFACT';
```

Example Results:

TableName	ColumnName	ColCheck
EMP_2	employee_number	CHECK ("employee_number" >= 100000)
DEPT_2	dept_number	CHECK ("dept_number" >= 1000)
JOB_2	job_code	CHECK ("job_code" >= 3000)

Note: A second set of Employee, Department, and Job tables were created with unnamed CHECK constraints at the column level.

Triggers View

The Triggers view provides information about event-driven, specialized procedures attached to a single table and stored in the Teradata database.

Using Triggers

Characteristics of triggers include:

- To define a trigger, use the CREATE TRIGGER statement.
- To cause the database to execute a trigger, use the INSERT, UPDATE or DELETE statements on the specified table or view.
- There are two kinds of triggers:
- Row triggers (R) evaluate each row changed by the trigger action.
- Statement triggers (S) evaluate the entire statement.
- When a triggered statement fires a trigger, cascading ensues that, in some instances, can fire other triggers and become triggering statements.
- Use the REFERENCING clause when you reference subject tables that are qualified with old or new table values. In addition, all subject table columns must use new or old correlation names.

Note: A positioned (updateable cursor) UPDATE or DELETE is not allowed to fire a trigger and generates an error. In addition, the FastLoad and MultiLoad utilities return an error if you have any triggers enabled on the target table.

Column definitions for this view include:

Column	Definition
ActionTime	Indicates when the triggered action fires. B = Before trigger statement changes the table. A = After trigger statement changes the table.
Event	Indicate which of the following SQL statements fires the trigger: U = UPDATE I = INSERT D = DELETE
Column Position	Position of a column within an index. This value may be greater than one if the column is part of a composite index.

Triggers View

Provides information about event-driven triggers attached to a single table and stored in the database.

DBC.Triggers[V][X]

DatabaseName	SubjectTableDataBaseName	TableName
TriggerName	EnabledFlag	ActionTime
Event	Kind	OrderNumber
TriggerComment	RequestText	CreatorName
CreateTimeStamp	LastAlterName	LastAlterTimeStamp
AccessCount **	LastAccessTimeStamp **	CreateTxtOverflow
VTEventType *	TTEventType *	

* 13.10 column

** Optionally updated

Example:

Show if the trigger is enabled, when it fires, the type of statement that fires it, and the kind.

```
SELECT    TableName              (CHAR(10))  AS TName
         ,TriggerName            (CHAR(12))
         ,EnabledFlag            AS  "Enabled"
         ,ActionTime             AS  Action
         ,Event
         ,Kind
FROM     DBC.TriggersV
WHERE    DatabaseName = DATABASE;
```

Example Results:

TName	TriggerName	Enabled	Action	Event	Kind
Employee	Raise_Trig	Y	A	U	R

All Temporary Tables View

This view provides information about all global temporary tables materialized in the system.

Global Temporary Tables

Use global temporary tables to store temporary, immediate results from multiple queries into working tables. To create a global temporary table, you must state the keywords GLOBAL TEMPORARY in the CREATE TABLE statement. The temporary table defined during the CREATE TABLE statement is referred to as the base temporary table.

When referenced in an SQL session, a local temporary table is materialized with the exact same definition as the base table. Once the temporary table is materialized, subsequent DML statements referring to that table are mapped to the materialized instance.

Note: After you create a global temporary table definition, use the INSERT statement to create a local instance of the global temporary table to use during the session.

Temporary versus Permanent Tables

Temporary tables are different than permanent tables in the following ways:

- They are always empty at the start of a session.
- Their contents cannot be shared by other sessions.
- You can empty them at the end of each transaction.
- The system automatically drops them at the end of each session.

All Temporary Tables View

Provides information about all global temporary tables materialized in the system.

DBC.AllTempTables[V][X]

HostNo	SessionNo	UserName	B_DatabaseName
B_TableName	E_TableID		

Example:
Show all temporary tables materialized in the system.

```
SELECT      HostNo
           ,SessionNo
           ,UserName   (CHAR(10))
           ,B_DatabaseName
                 AS "DataBase"
           ,B_TableName      AS "Table Name"
FROM       DBC.AllTempTablesV;
```

Example Results:

HostNo	SessionNo	UserName	Database	Table Name
01	20887	TFACT02	PD	GT_DEPTSALARY
01	20908	TFACT01	PD	GT_DEPTSALARY

Referential Integrity Views

The facing page identifies a number of views that can be used to list tables with referential integrity and the state of the referential integrity on the tables.

Additional views that specifically provide referential integrity information are listed below. The effects of referential integrity on the database can be seen in the series of views identified with the letters “RI”. The X views were implemented in Teradata V2R6.0.

VIEW NAME	DESCRIPTION
DBC.All_RI_Children[V][X]	This view shows the Referential Integrity Constraints defined in the database, from the Child-Parent point of view.
DBC.All_RI_Parents[V][X]	This view shows the same information as the above view, but from the Parent-Child perspective.
DBC.RI_Distinct_Children[V][X]	Provides information about tables in child-parent order without the duplication that could result from multi-column foreign keys.
DBC.RI_Distinct_Parents[V][X]	Provides information about tables in parent-child order without the duplication that could result from multi-column foreign keys.
DBC.RI_Child_Tables[V][X]	Provides information about tables in child-parent order. It is similar to the All_RI_Children view but returns the internal Ids of databases, tables, and columns.
DBC.RI_Parent_Tables[V][X]	Provides information about all tables in parent-child order. It is similar to the All_RI_Parents view but returns the internal IDs of databases, tables, and columns instead of names.

Referential Integrity Views

Views that can be used to provide information about Referential Integrity are:

View	Description
DBC.Tables[V][X]	Can be used to list parent and child counts for tables.
DBC.All_RI_Children[V][X]	Information about tables in child-parent order. Also identifies if RI constraint is consistent or inconsistent.
DBC.All_RI_Parents[V][X]	Similar to above view but provides Information in parent-child order.
DBC.Databases2[V][X]	Can be used to identify child tables with unresolved reference constraints.

Note: Additional RI views are listed on the facing page.

Using the DBC.Tables View

The DBC.Tables[V][X] views (described previously) provide parent and child counts for tables within a database.

The examples on the following pages are based on the following CREATE and ALTER TABLE statements.

```
CREATE SET TABLE      Employee
     (      employee_number      INTEGER       NOT NULL      PRIMARY KEY
           ,dept_number          INTEGER
           ,emp_mgr_number       INTEGER
           ,job_code             INTEGER
           ,last_name            CHAR(20)
           ,first_name           VARCHAR(20)
           ,salary_amount        DECIMAL(10,2) ) ;

CREATE SET TABLE      Department
     (      dept_number          INTEGER       NOT NULL      PRIMARY KEY
           ,dept_name            CHAR(20)      NOT NULL      UNIQUE
           ,dept_mgr_number      INTEGER
           ,budget_amount        DECIMAL (10,2) ) ;

CREATE SET TABLE      Job
     (      job_code             INTEGER       NOT NULL      PRIMARY KEY
           ,job_desc             CHAR(20)      NOT NULL      UNIQUE ) ;

CREATE SET TABLE      Emp_Phone
     (      employee_number      INTEGER       NOT NULL
           ,area_code            SMALLINT      NOT NULL
           ,phone_number         INTEGER       NOT NULL
           ,extension            INTEGER
           ,PRIMARY KEY  (employee_number, area_code, phone_number) )
   PRIMARY INDEX  (employee_number);

ALTER TABLE Employee          ADD CONSTRAINT emp_dept_ref
     FOREIGN KEY (dept_number) REFERENCES
               Department (dept_number);

ALTER TABLE Employee          ADD CONSTRAINT emp_job_ref
     FOREIGN KEY (job_code) REFERENCES
               Job (job_code);

ALTER TABLE Employee ADD CONSTRAINT emp_mgr_ref
     FOREIGN KEY (emp_mgr_number) REFERENCES
               Employee (employee_number);

ALTER TABLE Department        ADD CONSTRAINT dept_mgr_ref
     FOREIGN KEY (dept_mgr_number) REFERENCES
               Employee (employee_number);

ALTER TABLE Emp_Phone         ADD CONSTRAINT phone_emp_ref
     FOREIGN KEY (employee_number) REFERENCES
               Employee (employee_number);
```

Using the DBC.Tables View

The DBC.Tables[V][X] views can be used to list parent and child counts for tables.

- ParentCount – how many foreign keys does a table have or how many parents does the table reference?
- ChildCount – how many foreign keys reference this table or how many children does the table have?

Example:
List the tables objects in the database TFACT and identify parent and child counts.

```
SELECT     TableName
          ,TableKind
          ,ParentCount
          ,ChildCount
FROM       DBC.TablesV
WHERE      DatabaseName = 'TFACT'
AND        TableKind = 'T'
ORDER BY   2, 1 ;
```

Example Results:

TableName	TableKind	ParentCount	ChildCount
Department	T	1	1
Employee	T	3	3
Emp_Phone	T	1	0
Job	T	0	1
Salary_Log	T	0	0

Referential Integrity States

The facing page identifies three states that are associated with a Referential Integrity constraint.

Referential Integrity States

The status of a Referential integrity constraint is classified as follows:

- **Unresolved reference constraint – the FK exists, but the PK does not.**
  - **Creating a table with a FK before creating the table with Parent Key (PK).**
  - **Restoring a table with a FK and the table with the PK does not exist or hasn't been restored.**
- **Inconsistent reference constraint – both the FK and the PK exist, but the constraint is marked as inconsistent.**
  - **When either the child or parent table is restored, the reference constraint for the child table is marked as inconsistent.**
    - **no inserts, updates, deletes or table changes are allowed**
- **Consistent reference constraint – both the FK and the PK exist and are considered consistent, but FK values without a corresponding PK value are identified as "invalid rows".**
  - **Typically occurs when the reference constraints are created on already populated tables or …**
  - **Following a REVALIDATE REFERENCES command after a restore of either the child or parent table.**

DBC.All_RI_Children View

The DBC.All_RI_Children[X] views provide information about all tables in child-parent order.

A table can have many referential constraints defined. When either the child or parent table is restored, these constraints are marked ***inconsistent***. The DBC.All_RI_Children view provides information about reference constraints.

The REVALIDATE REFERENCES FOR statement validates these inconsistent constraints against the target table.

If inconsistent constraints remain after a REVALIDATE REFERENCES FOR statement has been executed, the SQL statement ALTER TABLE DROP INCONSISTENT REFERENCES must be used to remove them.

REVALIDATE REFERENCES FOR creates error tables containing information about data rows that failed referential constraint checks.

DBC.All_RI_Children View

This view is used to identify tables with RI in child-parent order and can also be used to show if the RI constraint is consistent or inconsistent.

DBC.All_RI_Children [V][X]

IndexID	IndexName	ChildDB	ChildTable
ChildKeyColumn	ParentDB	ParentTable	ParentKeyColumn
InconsistencyFlag	CreatorName	CreateTimeStamp	

Example:

```
SELECT    IndexID          (FORMAT 'z9')  AS ID
         ,IndexName        ,ChildTable          ,ChildKeyColumn
         ,ParentTable      ,ParentKeyColumn     ,InconsistencyFlag AS ICF
FROM      DBC.ALL_RI_ChildrenV
WHERE     ChildDB = 'PD'    ORDER BY  3, 4 ;
```

Results:

ID	IndexName	ChildTable	ChildKeyColumn	ParentTable	ParentKeyColumn	ICF
0	dept_mgr_ref	Department	dept_mgr_number	Employee	employee_number	N
0	emp_dept_ref	Employee	dept_number	Department	dept_number	Y
8	emp_mgr_ref	Employee	emp_mgr_number	Employee	employee_number	Y
4	emp_job_ref	Employee	job_code	Job	job_code	Y
0	phone_emp_ref	Emp_Phone	employee_number	Employee	employee_number	N

Options to handle inconsistent references:

- **ALTER TABLE tname DROP INCONSISTENT REFERENCES; – FK constraints are dropped – use the ALTER TABLE command to create new references constraints.**
- **Use the REVALIDATE REFERENCES command (ARC facility).**

DBC.Databases2 View

The DBC.Databases2[V][X] views provide ID definition information about databases and provide a count of unresolved reference constraints for any tables within the database.

It is similar to the Databases view but returns the ID of the database and Referential Integrity (RI) information instead of the other information (creator name, owner name, etc.) provided by the Databases view.

You can control who has access to internal ID numbers by limiting the access to the Databases2 view while allowing more users to access the names via the Databases view.

Example

The SQL request on the facing page uses the Databases2 view to find databases that have tables with unresolved references.

The columns selected are:

DatabaseName	Returns the name of a database with the indicated count of unresolved references.
DatabaseId	Returns the ID of the database with the indicated count of unresolved references.
UnresolvedRICount	Returns the total number of unresolved Referential Integrity (RI) constraints in the database. If one table has 2 unresolved reference constraints, then the count is 2.

DBC.Databases2 View

The DBC.Databases2[V][X] views provide information about unresolved reference constraints.

Unresolved reference constraints are caused by:

- Creating a table with a Foreign Key before creating the table with Parent Key (Primary Key).
- Restoring a table with a Foreign Key and the Parent Key (Primary Key) table does not exist or hasn't been restored.

DBC.Databases2

DatabaseName	DatabaseID	UnresolvedRICount

Example: List all databases with unresolved references.

```
SELECT    DatabaseName
          ,DatabaseId
          ,UnresolvedRICount
FROM      DBC.Databases2V
WHERE     UnresolvedRICount > 0;
```

Results:

DatabaseName	DatabaseID	UnresolvedRICount
PD	0000FE03	2

Restore the Parent Table to "resolve" the references constraint, but the references constraint is marked as inconsistent.

Time Stamps in Data Dictionary

The Teradata Database includes a time stamp in most of the data dictionary tables.

The time stamp feature is meant to facilitate and enhance your system administration tasks by providing a means to identify obsolete objects, and clean up and recapture space. Time stamps also help you determine when a change to an object occurred for system maintenance activities and problem investigations.

The facing page shows the time stamp fields in dictionary tables, system views and dictionary views. A description of these fields follows.

Time Stamp Field Definitions:

Create Time Stamp	The time the object was created in ANSI TimeStamp Format.
CreateUID	User ID of the user who created the object.
CreatorName	Name of the user who created the database, table, or the name of the user's creator.
LastAlterName	Name of the user who last updated the object.
LastAlterTimeStamp	The time the object was last updated in ANSI TimeStamp format.
LastAlterUID	User ID of the user who last updated the object.
AccessCount	The number of times the object was accessed. (Only used if requested.)
LastAccessTimeStamp	The time the object was last accessed in ANSI TimeStamp format.

Time Stamps in Data Dictionary

The Teradata Database features a time stamp in the Data Dictionary tables.

- **CreateTimeStamp, CreatorName, LastAlterTimeStamp, LastAlterName**

This feature can help system administration tasks by providing a means to identify objects recently updated, obsolete objects, etc. and who altered the objects.

Example:
List all tables that have been altered in August of 2011.

```
SELECT    TRIM(DatabaseName) || '.' || TableName
                             AS "Qualified Name"
          ,LastAlterName     AS "User Name"
          ,LastAlterTimeStamp AS "Last Alter Date & Time"
FROM      DBC.TablesV
WHERE     EXTRACT (YEAR FROM LastAlterTimeStamp) = 2011
AND       EXTRACT (MONTH FROM LastAlterTimeStamp) = 8
ORDER BY  1, 2 ;
```

Example Results:

Qualified Name	User Name	Last Alter Date & Time
HR_Tab.Job	tfact03	2011-08-23 08:10:14
HR_Tab.Location	tfact03	2011-08-23 17:19:07
HR_Tab.Raise_Trig	Sysdba	2011-08-20 04:58:14
HR_Tab.Salary_Log	Sysdba	2011-08-20 04:58:14
TFACT.New_Sales	tfact03	2011-08-24 16:25:27

Teradata Administrator – List Columns of a View

Teradata Administrator can be used to perform many of the functions described in this module.

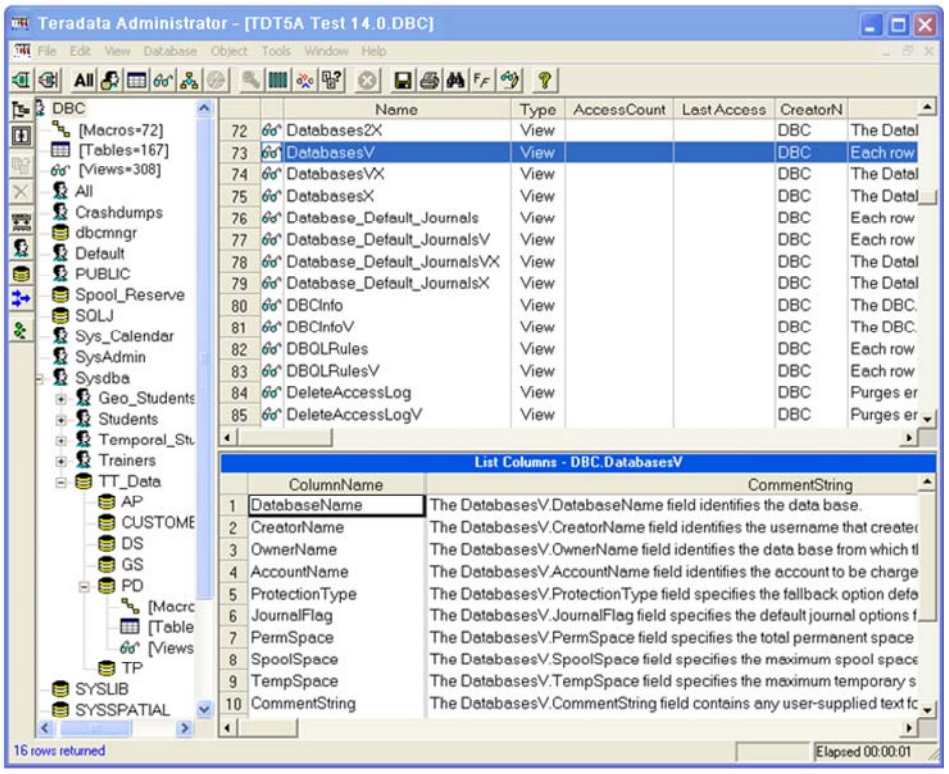
The facing page shows a simple example of using Teradata Administrator to view the columns in a system view.

Teradata Administrator

List Columns of a View

Teradata Administrator can be used to list the columns of DD/D views (and tables).

Appendix D of this manual contains a listing of all the DD/D views and columns.



Teradata Administrator – Object Options

Teradata Administrator can be used to perform a wide range of functions. The facing page shows an example of the options available at a Table object level.

To use the viewing functions of Teradata Administrator, you must have Select access to the DBC views of the Teradata Database. To use the Copy, Drop, Create or Grant options, you must have the corresponding privilege on the table or database that you are trying to modify or create. To use the Browse or Row Count features you must have select access to the Table or View.

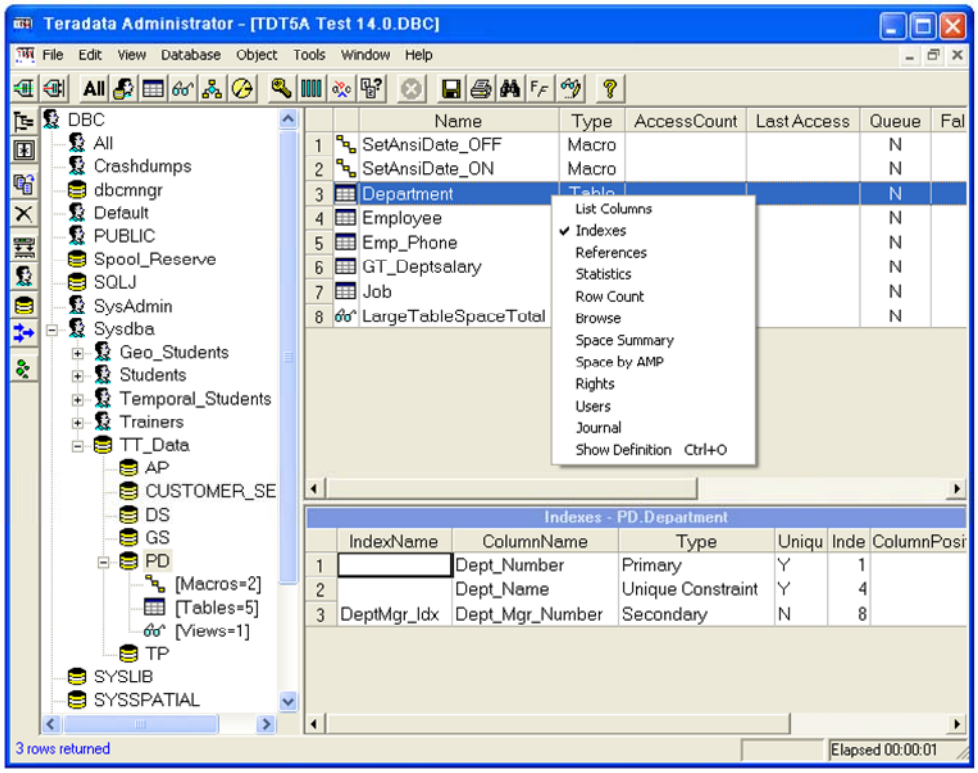
Teradata Administrator

Object Options

Teradata Administrator can also be used to display object details.

For example, right-click on the object (e.g., Department table) and a menu of options is displayed.

In this example, the Indexes option was selected.



Summary

The facing page summarizes some important concepts regarding this module.

Summary

- The data dictionary consists of tables, views, and macros stored in system user DBC.
- The Teradata Database software automatically updates data dictionary/directory tables as you create or drop objects.
- You can access data dictionary tables with supplied views.
- Data dictionary tables keep track of all created objects:
  - Database and users
  - Tables, views, macros, triggers, stored procedures, and user-defined functions
  - Columns and indexes
  - Hierarchies
- Note: To access information about individual objects stored in the data dictionary, use the HELP and SHOW commands.

Module 42: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 42: Review Questions

1. True or False. The DBC.Databases view only contains information about databases; users are not included in this view.

2. True or False. The DBC.Users view only contains information about users; databases are not included in this view.

3. True or False. Queries that use restricted views usually take less time to execute than queries that use unrestricted views.

4. True or False. All of the data dictionary tables are Fallback protected.

5. If a child table exists and the parent table doesn't, the reference constraint is marked as _____.
   a. Inconsistent
   b. Unresolved
   c. Missing
   d. Invalid

6. After executing the ALTER TABLE … ADD FOREIGN KEY … statement, Foreign Key values that are missing in the parent table are marked in an error table and are known as _____ rows.
   a. Inconsistent
   b. Unresolved
   c. Missing
   d. Invalid

Lab Exercise 42-1

Notes about DBC.DBCInfoV columns:

- The Release column provides the PDE release number.
- The Version column provides the Teradata software version number.

Lab Exercise 42-1

Lab Exercise 42-1

Purpose

In this lab, you will use Teradata SQL Assistant to view information in the data dictionary using various Data Dictionary views – Appendix D has details on Data Dictionary views.

What you need

SELECT Access to the Data Dictionary views.

Tasks

1. Using the DBC.DBCInfoV view, find the release and version of the system you are logged on:

   Release (PDE) _____ Version (Teradata) _____

2. Using the DBC.ChildrenV view, list your parents' user names.

   _____ _____ _____ _____

3. Using the DBC.DatabasesV view, find your:

   Immediate parent's name _____
   Creator's name _____
   Default account code _____
   Perm space limit _____
   Spool space limit _____
   Temp space limit _____

Lab Exercise 42-1 (cont.)

The following pages describe the tasks for this lab exercise.

Lab Exercise 42-1 (cont.)

4. Using the DBC.UsersV view, find your:

   Default database name _____
   Default collation sequence _____
   Default date format _____
   Create time stamp _____
   Last password modification date _____

   OPTIONAL: SHOW this view. Note the WHERE conditions. (Remember, this is a restricted view, even though it does not have an [X] suffix.)

5. Using the DBC.TablesV view, find the number of tables in the DD/D (user DBC) that are:

   Fallback protected _____

   Not Fallback protected _____

   Modify the query to find the number of tables OTHER THAN DD/D (not DBC tables) that are:

   Fallback protected _____

   Not Fallback protected _____

Lab Exercise 42-1 (cont.)

The following pages describe the tasks for this lab exercise.

Lab Exercise 42-1 (cont.)

6. **Using the DBC.PartitioningConstraintsV view, answer the following questions; hint, use the ColumnPartitioningLevel column to determine if a table is column partitioned or not.**

   For your user, how many tables have a PPI? ____
   For your user, how many tables have column partitioning? ____

   For the system, how many tables have a PPI? ____
   For the system, how many tables have column partitioning? ____

   What is the constraint type for PPI tables? ____

7. **Using the DBC.IndicesV view, find the number of tables OTHER THAN Dictionary tables that have non-unique primary indexes (NUPI):**

   Number of NUPI tables _____

8. **Using the DBC.ColumnsV view, find the number of columns in the entire system defined with *default values:***

   Number of columns _____

   Optional: Modify the query to find the number of objects that have columns defined with *default values:*

   Number of tables _____

Lab Exercise 42-2 (optional)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Use the following SQL for this exercise.

To populate a table:

```
INSERT INTO empty_tablename
SELECT  * FROM  non_empty_tablename;
```

To grant an Access Right on a table:

```
GRANT REFERENCES ON tablename TO username;
```

To grant an Access Right on a specific column:

```
GRANT REFERENCES (column_name) ON tablename TO username;
```

Lab Exercise 42-2 (optional)

Lab Exercise 42-2 (optional)

Purpose

In this lab, you will use Teradata SQL Assistant to establish References constraints between 4 populated tables and view the associated data dictionary entries.

What you need

Populated PD tables and empty tables in your database

Tasks

1. Use INSERT/SELECT to place all rows from the populated PD tables into your empty tables. Verify the number of rows in your tables.

PD.Employee	to populate	Employee	Count = _____
PD.Department	to populate	Department	Count = _____
PD.Job	to populate	Job	Count = _____
PD.Emp_Phone	to populate	Emp_Phone	Count = _____

2. Use the GRANT statement to GRANT yourself the REFERENCES access rights on the tables.

   GRANT REFERENCES ON *tablename* TO *username*;

Lab Exercise 42-2 (optional – cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Use the following SQL for this exercise.

To create a References constraint:

```
ALTER TABLE          child_tablename
  ADD CONSTRAINT     constraint_name
        FOREIGN KEY  (child_name)
        REFERENCES   parent_tablename (parent_column);
```

To select from the DBC.ALL_RI_ChildrenV view:

```
SELECT      Indexid          (FORMAT 'z9') AS ID
           ,IndexName
           ,ChildTable
           ,ChildKeyColumn
           ,ParentTable
           ,ParentKeyColumn
FROM       DBC.ALL_RI_ChildrenV
WHERE      ChildDB = USER
ORDER BY   1;
```

Lab Exercise 42-2 (optional – cont.)

3. **Create a References constraint between the Employee.Dept_Number column and the Department.Dept_Number column.**

   What is the name of the Employee RI error table? _____

   How many rows are in this table? _____

   Which department is not represented in the department table? _____

4. **Use the DBC.All_RI_ChildrenV view (qualify the ChildDB to your database) and verify this References constraint.**

   What is the IndexID of this constraint? _____

Notes

Module 43

Space Allocation and Usage

After completing this module, you will be able to:

- **Define permanent space, spool space and operating system space requirements.**
- **Estimate system capacity.**
- **Use the AllSpace, DiskSpace, and TableSize views to monitor disk space utilization.**
- **Use Teradata Administrator to view database and table space utilization.**

Notes

Table of Contents

Permanent Space Terminology 43-4
Spool Space Terminology 43-6
Temporary Space Terminology 43-8
Resetting Peak Values 43-10
Assigning Perm and Spool Limits 43-12
Giving One User to Another 43-14
Teradata Administrator – Move Space 43-16
Reserving Space for Spool 43-18
Views for Space Allocation Reporting 43-20
DiskSpace View 43-22
TableSize View 43-24
AllSpace View 43-26
DataBaseSpace Table 43-28
Different Views — Different Results 43-30
Additional Utilities to View Space Utilization 43-32
Teradata Administrator – Database Menu Options 43-34
Teradata Administrator – Object Menu Options 43-36
Transient Journal Space 43-38
Ferret Utility 43-40
Ferret SHOWSPACE Command 43-42
Ferret SHOWBLOCKS 43-44
  Ferret SHOWBLOCKS – Subtable Detail 43-46
Module 43: Review Questions 43-48

Permanent Space Terminology

MaxPerm

MaxPerm is the maximum number of bytes available for table, index, and permanent journal storage in a system database or user.

The number of bytes specified is divided by the number of AMP vprocs in the system. The result is recorded on each AMP vproc and may not be exceeded on that vproc. *

Perm space limits are deducted from the limit set for the immediate parent of the object defined.

Perm space is acquired when data is added to a table. The space is released when you delete or drop objects.

CurrentPerm

CurrentPerm is the total number of bytes (including table headers) in use on the database to store the tables, subtables and permanent journals contained in a User or Database. This value is maintained on each AMP vproc.

PeakPerm

PeakPerm is the largest number of bytes ever actually used to store data in a user or database. This value is maintained on each AMP vproc.

Reset the PeakPerm value to zero by using the ClearPeakDisk Macro supplied in User DBC.

Note: Space limits are enforced at the database level. A database or user may own several small tables or a few large tables as long as they are within the MaxPerm limit set on each AMP.

***** Minor exceptions to this rule may occur occasionally. For example, utilities that write data to disk a block at a time (such as FastLoad) check space limits *after* a block is written.

Permanent Space Terminology

MaxPerm

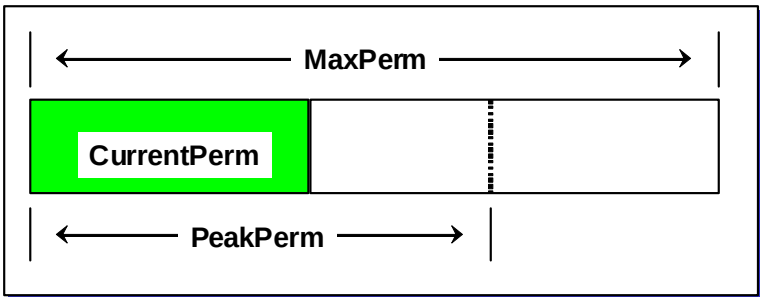
The maximum number of bytes *available* for table, index and permanent journal storage in a database or user.

CurrentPerm

The total number of bytes *in use* to store the tables, subtables, and permanent journals contained in the database or user.

PeakPerm

The *largest* number of bytes actually used to store data in this user since the value was last reset.



Spool Space Terminology

MaxSpool

MaxSpool is a value used to limit the number of bytes the system will allocate to create spool files for a user.

The value you specify may not exceed that of a user's immediate parent (database or user) at the time you create the user. If you do not specify a value, MaxSpool defaults to the parent's MaxSpool value.

Limit the spool space you allocate to users to reduce the impact of "runaway" transactions, such as accidental product joins.

Spool space marked (last use) is recovered by a worker task that is initiated every five minutes.

CurrentSpool

CurrentSpool is the number of bytes in use for running transactions. This value is maintained on each AMP vproc for each user.

PeakSpool

PeakSpool is the maximum number of bytes used by a transaction run for a user since the value was last reset by the ClearPeakDisk Macro (supplied in system user DBC).

Spool Space Terminology

MaxSpool

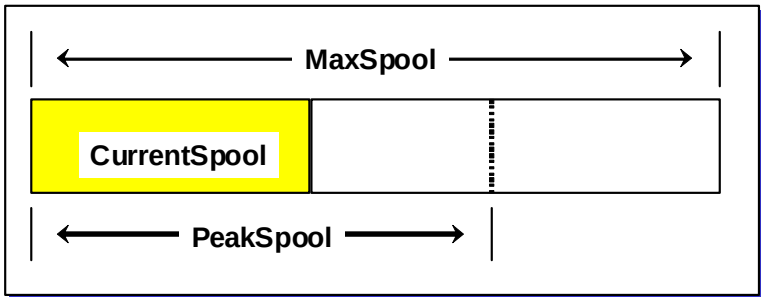
A value used to limit the number of bytes the system will consume to create spool files for a user.

CurrentSpool

The number of bytes currently in use for running transactions.

PeakSpool

The maximum number of bytes used by a transaction run for this user since the value was last reset.



Temporary Space Terminology

MaxTemp

MaxTemp is a value used to limit the number of bytes the system will use to store data for global temporary tables for a user.

The value you specify may not exceed that of a user's immediate parent (database or user) at the time you create the user. If you do not specify a value, MaxTemp defaults to the parent's MaxTemp value.

CurrentTemp

CurrentTemp is the number of bytes in use for global temporary tables. This value is maintained on each AMP vproc for each user.

PeakTemp

PeakTemp is the maximum number of bytes used by global temporary tables for a user since the value was last reset by the ClearPeakDisk Macro (supplied in system user DBC).

Temporary Space Terminology

MaxTemp

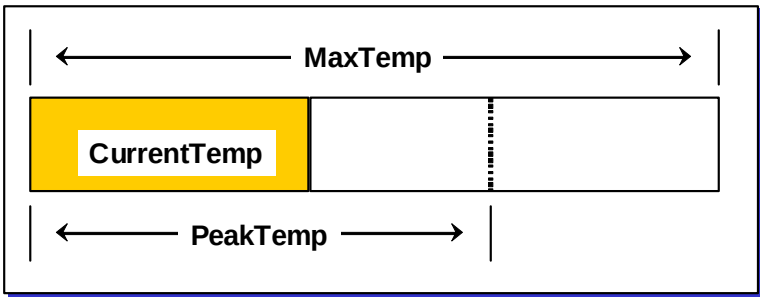
A value used to limit the number of bytes the system will use to store data for global temporary tables for a user.

CurrentTemp

The number of bytes in use for global temporary tables.

PeakTemp

The maximum number of bytes used by global temporary tables for a user since the value was last reset.



Temporary space is released when the user terminates the session or when the user frees the temporary space (e.g., Deleting the rows in a global temporary table.

Resetting Peak Values

From time to time, the administrator needs to clear out the values accumulated in the DBC.DataBaseSpace table. These values must be reset to restart the data collection process.

DBC.ClearPeakDisk

The Teradata software provides a macro to reset the PeakPerm, PeakSpool, and PeakTemp values in the DBC.DataBaseSpace table. It may be used to reset the peak values for the next collection period.

Resetting Peak Values

The ClearPeakDisk macro resets PeakPerm, PeakSpool, and PeakTemp values in the DatabaseSpace table.

```
SHOW MACRO DBC.ClearPeakDisk;
```

```
REPLACE MACRO DBC.ClearPeakDisk AS
(
UPDATE DatabaseSpace
SET  PeakPermSpace  = CurrentPermSpace,
     PeakSpoolSpace = 0,
     PeakTempSpace = 0 ALL ;
) ;
```

This macro may be used to reset peak values for the next data collection period.

To clear accounting values:

```
EXEC DBC.ClearPeakDisk;
```

```
*** Update completed.  3911 rows changed.
*** Time was 4 seconds.
```

Assigning Perm and Spool Limits

You define permanent and spool space limits at the database or user level, not at the table level.

When you create databases or users, perm space limits are deducted from the available (unused) space of the immediate owner.

The spool space limit may not exceed that of the immediate owner at the time you create an object. If you do not specify a spool space limit, the new object "inherits" its limit from the immediate owner (user or database).

Example

The diagram on the facing page illustrates how Teradata manages permanent and spool space.

A user, Payroll, has a 25 GB permanent space limit and a 50 GB spool space limit.

Payroll creates two new users, PY01 and PY02. After Payroll creates the new objects, its maximum Perm space drops to 15 GB. PY01 has 6 GB of maximum Perm and PY02 has 4 GB.

Later, Payroll drops user PY02. Payroll's maximum Perm space increases to 19 GB since it regains the permanent space that used to belong to PY02.

Payroll has a limit of 50 GB of maximum Spool. When it creates PY01, it assigns 35 GB of maximum Spool to the new user. Since there is no statement of spool space for PY02, its maximum Spool defaults to the limit of its immediate parent: 50 GB.

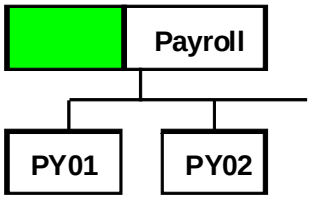
The amount of maximum Perm increases and decreases as the owner creates and drops new users. The spool space figure remains constant even when the owner adds and drops users.

Assigning Perm and Spool Limits

MaxPerm = 25E9 MaxSpool = 50E9

```
CREATE USER PY01 AS PASSWORD = abc, PERM = 6E9, SPOOL = 35E9;

CREATE USER PY02 AS PASSWORD = xyz, PERM = 4E9;
```

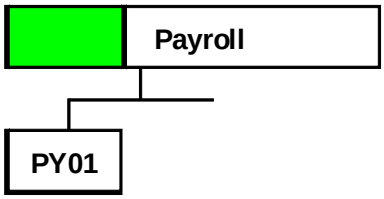


MaxPerm = 15E9 MaxSpool = 50E9

The maximum Spool value may not exceed that of the immediate owner at the time you create the new user.

What is the maximum Spool limit of PY02?

```
DROP USER PY02;
```



MaxPerm = 19E9 MaxSpool = 50E9

The Perm space from PY02 is returned back to the immediate owner.

Giving One User to Another

When you give an object to another object in the hierarchy, all space allocated to that object goes with it. If you drop the object, its space is credited to its immediate owner.

When you give databases or users, all descendants of the given object remain descendants of the given object.

When you give an object to new parents, the ownership of space is transferred; however the limits remain the same.

If you drop an object, its space is credited to its immediate owner.

The facing page illustrates giving a database/user from one database/user to another such as giving Payroll from Human_Resources to Accounting.

Adjusting Perm Space Limits

You can easily adjust perm space limits. Using the illustration on the facing page as an example, you could transfer 10 GB from Human_Resources to Accounting using the following technique:

1. CREATE DATABASE Temp FROM Human_Resources AS PERM = 10E9;
2. GIVE Temp TO Accounting;
3. DROP DATABASE Temp;

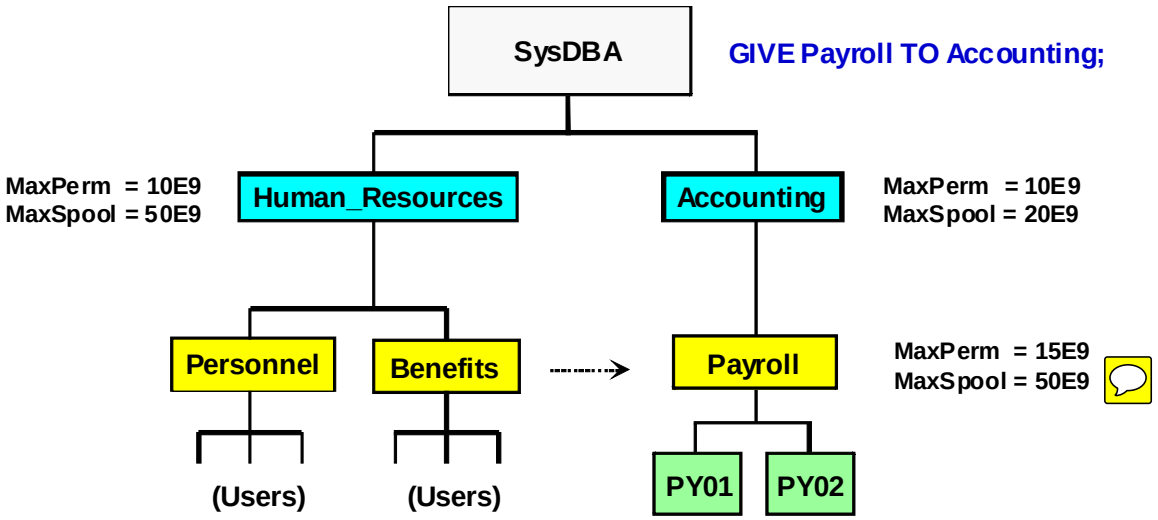
The database TEMP is NOT shown on the facing page, but is used as an example of how space could be transferred from one database to another.

Notes:

You enforce limits when you create an object.

Objects you give may have spool limits that exceed that of their current owner.

Giving One User to Another



- Payroll ownership is transferred (GIVE) to Accounting.
- All descendants (child users/databases, tables, views, etc.) of a "given" object remain with the "given" object.
- The GIVE command may also be used to move Permanent space from one database/user to another database/user.

Teradata Administrator – Move Space

The Move Space function of Teradata Administrator makes it easy to move space from one database/user to another database/user. The Tools > Move Space menu choice displays the Move Space dialog box that you use to reallocate permanent disk space from one database to another:

The example on the facing page illustrates moving 20 MB of Permanent space from the HR_Tab database to the Payroll_Tab database.

The user who has logged onto Teradata Administrator requires the DROP DATABASE access right on HR_Tab and the CREATE DATABASE access right on Payroll_Tab in order to do this move space operation.

As discussed previously, the following commands would accomplish the same thing.

1. CREATE DATABASE Temp FROM HR_Tab AS PERM = 20E6;
2. GIVE Temp TO Payroll_Tab;
3. DROP DATABASE Temp;

Teradata Administrator – Move Space

Techniques to move Perm space.

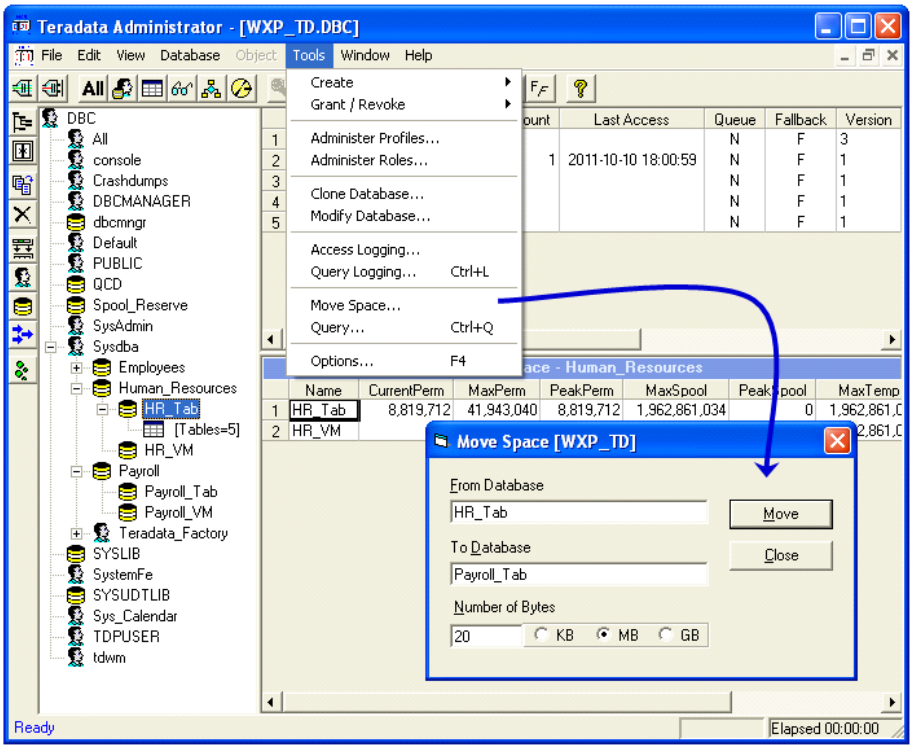
- GIVE command
- Teradata Administrator

GIVE command

1) CREATE temp database with PERM space.
2) GIVE temp database to another database.
3) DROP temp database.

Teradata Administrator

- Tools -> Move Space – moves Perm space from one database to another.



Reserving Space for Spool

Spool space serves as temporary storage for returned rows during transactions that users submit. To ensure that space is always available, you may want to set aside about 20 to 25% of total available space as spool space. To do this, you can create a special database called Spool_Reserve. This database will not be used to load tables.

Decision support applications should reserve more of the total disk space as reserved spool space since their SQL statements generate larger spool files. OLTP applications can use less as reserved spool space because their statements generate smaller spool files.

The above actions guarantee that data tables will never occupy more than 60% to 75% of the total disk space. Since there is no data stored in Spool_Reserve, the system will use its permanent space as spool space when necessary.

Orphan or Phantom Spool Issues

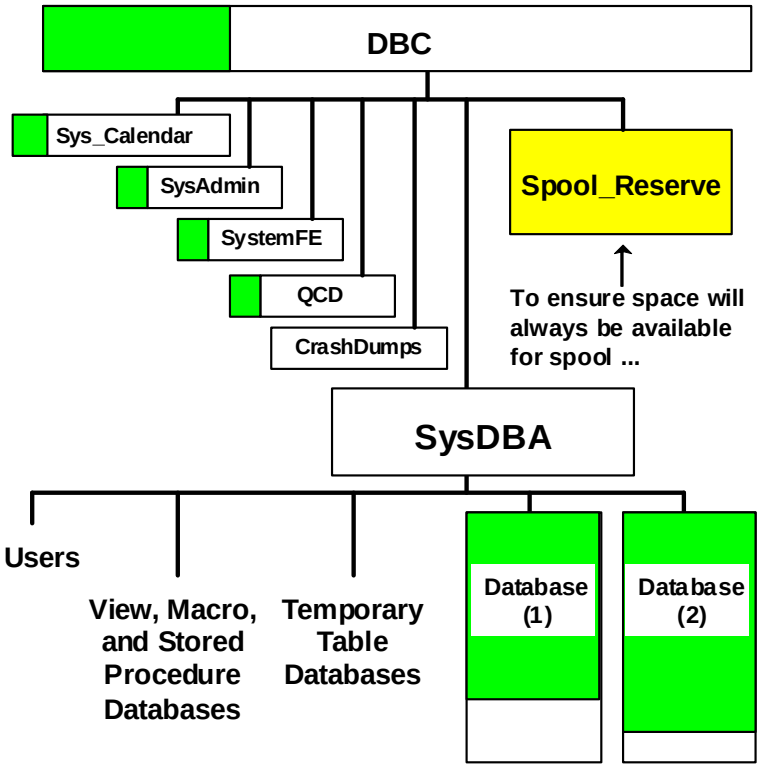
Spool tables are temporary work tables which are created and dropped as queries are executed. When a query is complete, all of the spool tables it used should be dropped automatically.

Like all tables, spool tables require a Table ID. There is a range of tableids exclusively reserved for spool tables (C000 0001 thru FFFF FFFF) and the system cycles through them. If a spool table is not dropped correctly, it remains in existence. Eventually, the system will cycle through all the table ids and reassign the tableid which is in use by our left over spool table. Usually, the presence of this table is detected, the query which was going to use the tableid is aborted – even though it is innocent of any wrongdoing – and the following message is returned to the user and put in the error log:

*** FAILURE 2667 Left-over spool table found: transaction aborted.

This case is very unusual, but can happen. The unusual cases of leftover spool are covered in another module later in this course.

Reserving Space for Spool



CREATE DATABASE Spool_Reserve AS PERM = *spool_amount* ;

(The space used by Spool_Reserve will reduce the total available permanent space in the system by 25-40% or whatever is appropriate.)

Do not use the Spool_Reserve database to store tables. Data tables will occupy up to 60-75% of the total disk space.

Views for Space Allocation Reporting

Use the following system views to report current space allocation:

DBC.DiskSpace[V][X]

This view gives AMP vproc information about disk space usage for any database or account. It gets this information from the ALL table.

DBC.TableSize[V][X]

This view gives AMP vproc information about disk space usage (excluding spool) for any table or account.

DBC.AllSpace[V][X]

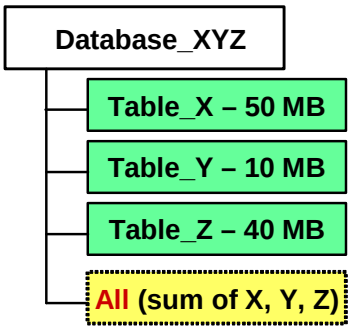
This view gives AMP vproc information about disk space usage (including spool) for any database, table, or account.

Each of these views references the non-hashed DBC.DataBaseSpace table.

Views for Space Allocation Reporting

View Name	Description
DBC.DiskSpace[V][X]	AMP vproc information about disk space usage (including spool) for any database or account.
DBC.TableSize[V][X]	AMP vproc information about disk space usage (excluding spool) for any database, table or account.
DBC.AllSpace[V][X]	AMP vproc information about disk space usage (including spool) for any database, table, or account.

Example:



SUM (CurrentPerm) using the 3 views

DBC.DiskSpaceV

Table All – 100 MB

DBC.TableSizeV

Table_X –	50 MB
Table_Y –	10 MB
Table_Z –	40 MB
	100 MB

DBC.AllSpaceV

Table_X –	50 MB
Table_Y –	10 MB
Table_Z –	40 MB
Table All –	100 MB
	200 MB

DiskSpace View

The DiskSpace[V][X] view provides AMP vproc information about disk space usage at the database level. This view includes spool space usage.

The PeakSpool column can be used to determine the maximum amount of spool space that a user has used (via DatabaseName and AccountName columns) .

Example

The SELECT statement on the facing page calculates the percentage of disk space used in the owner's database. The result displays a partial report with five rows. The DS database has the highest percentage of utilized space at 88.41%. SystemFE has the lowest at 12.39%.

Note: In the statement, use NULLIFZERO to avoid a divide exception.

DiskSpace View

Provides AMP Vproc disk space usage at the database level, including spool space.

DBC.DiskSpace[V][X]

Vproc	DatabaseName	AccountName	MaxPerm	MaxSpool
MaxTemp	CurrentPerm	CurrentSpool	CurrentTemp	PeakPerm
PeakSpool	PeakTemp	MaxProfileSpool	MaxProfileTemp	

This view selects values for TableName "ALL" (TableID = '000000000000' XB).

Example:
Calculate the percentage of permanent space used in databases and users.

```
SELECT    DatabaseName
          ,CAST (SUM (MaxPerm) AS FORMAT 'zzz,zzz,zz9')
          ,CAST (SUM (CurrentPerm) AS FORMAT 'zzz,zzz,zz9')
          ,CAST (((SUM (CurrentPerm))/
                NULLIFZERO (SUM(MaxPerm)) * 100)
                   AS FORMAT 'zz9.99%') AS "% Used"
FROM      DBC.DiskSpaceV
GROUP BY  1
ORDER BY  4 DESC ;
```

Example Results:

DatabaseName	Sum(MaxPerm)	Sum(CurrentPerm)	% Used
DS	209,715,200	185,413,632	88.41%
TFACT	104,857,600	45,396,480	43.29%
AP	20,000,000	3,978,240	19.89%
Sys_Calendar	15,000,000	2,647,040	17.65%
SystemFe	1,000,000	123,904	12.39%

TableSize View

The TableSize view is a Data Dictionary view that provides AMP Vproc information about disk space usage at a table level, optionally for tables the current User owns or has SELECT privileges on.

Example

The SELECT statement on the facing page looks for poorly distributed tables by displaying the CurrentPerm figures for a single table on all AMP vprocs.

The result displays one table, **Table2**, which is evenly distributed across all AMP vprocs in the system. The CurrentPerm figure is nearly identical across all vprocs. The other table, **Table2_nupi**, is poorly distributed. The CurrentPerm figures range from 95,232 bytes to 145,920 bytes on different AMP vprocs.

TableSize View

Provides AMP Vproc disk space usage at table level.

DBC.TableSize[V][X]

Vproc	DatabaseName	AccountName	TableName
CurrentPerm	PeakPerm		

Excludes TableName "All" (TableID = '000000000000' XB)

Example:
Look at table distribution across AMPs.

```
SELECT    Vproc
          ,CAST (TableName
                AS FORMAT 'X(20)')
          ,CurrentPerm
          ,PeakPerm
FROM      DBC.TableSizeV
WHERE     DatabaseName = USER
ORDER BY  TableName, Vproc ;
```

Vproc	TableName	CurrentPerm	PeakPerm
0	Table2	127,488	253,440
1	Table2	127,488	253,440
2	Table2	127,488	253,952
3	Table2	127,488	253,952
4	Table2	128,000	255,488
5	Table2	128,000	255,488
6	Table2	126,976	251,904
7	Table2	126,976	251,904
0	Table2_nupi	95,232	95,232
1	Table2_nupi	95,232	95,232
2	Table2_nupi	145,920	145,920
3	Table2_nupi	145,920	145,920
4	Table2_nupi	123,904	123,904
5	Table2_nupi	123,904	123,904
6	Table2_nupi	145,408	145,408
7	Table2_nupi	145,408	145,408

AllSpace View

The AllSpace[V][X] view provides AMP vproc information about disk space usage at the database and table level. This information includes the “All” table.

Example

The SELECT statement on the facing page lists the MaxPerm and CurrentPerm figures for each table in the user's space. The result displays three table names: **All**, **Table2**, and **Table2_nupi**.

The “All” table represents all tables that reside in the user's space. The MaxPerm figure for “All” is the amount of permanent space defined for that user. There are only two tables in this user's defined space.

Note: The “Table2 and Table2_nupi” tables display zero bytes in the MaxPerm column. This is because tables do not have MaxPerm space, only databases and users do, as represented by the “All” table.

AllSpace View

AMP vproc disk space usage at the database AND table level.

DBC.AllSpace[V][X]

Vproc	DatabaseName	AccountName	TableName
MaxPerm	MaxSpool	MaxTemp	CurrentPerm
CurrentSpool	CurrentTemp	PeakPerm	PeakSpool
PeakTemp	MaxProfileSpool (13.0)	MaxProfileTemp (13.0)	

Includes TableName "All" (TableID = '000000000000'XB)

Example:

List all tables (by Vproc) contained in the user's space.

```
SELECT    Vproc
          ,CAST (TableName AS
                FORMAT 'X(20)')
          ,MaxPerm
          ,CurrentPerm
FROM      DBC.AllSpaceV
WHERE     DatabaseName = USER
ORDER BY  TableName, Vproc ;
```

Vproc	TableName	MaxPerm	CurrentPerm
0	All	1,250,000	222,720
1	All	1,250,000	222,720
2	All	1,250,000	273,408
3	All	1,250,000	273,408
:	:	:	:
0	Table2	0	127,488
1	Table2	0	127,488
2	Table2	0	127,488
3	Table2	0	127,488
:	:	:	:
0	Table2_nupi	0	95,232
1	Table2_nupi	0	95,232
2	Table2_nupi	0	145,920
3	Table2_nupi	0	145,920
:	:	:	:

DataBaseSpace Table

The DataBaseSpace table tracks and stores information about disk space usage for objects in the Teradata system. The information is updated as users create new databases and add tables to them. The facing page illustrates four columns from DataBaseSpace. The SQL to generate this report follows:

```
SELECT    DatabaseID, TableID, Vproc, CurrentPermSpace
FROM      DBC.DataBaseSpace
WHERE     DatabaseID='00001404'XB
ORDER BY  2;
```

DataBaseSpace Columns

The four columns described below are used by the AllSpace, DiskSpace and TableSize views to produce disk space utilization reports:

DatabaseID

A DatabaseID is a unique identification number assigned to a database when the CREATE DATABASE statement is issued. The SQL statement adds a new row to the DataBaseSpace table and automatically assigns an internal database ID that corresponds with the database name assigned by the user.

TableID

TableID is a unique identification number assigned to a table when the CREATE TABLE statement is issued. The SQL statement adds a new row to the DataBaseSpace table and automatically assigns an internal table ID that corresponds with the table name assigned by the user.

Each database has a table ID 000000000000. This table has a special purpose. It displays the total amount of PermSpace used by the entire database, not just a single table. This table name is referenced as “All”.

Vproc

Vproc is the **logical vproc number** where a table is stored. Since tables are evenly distributed across all AMP vprocs, a single table is stored on several different vprocs.

CurrentPermSpace

CurrentPermSpace is the number of bytes of permanent space taken up on a specific vproc by that table. Table ALL (ID of 000000000000) displays the total amount of permanent space used by the tables stored in that database.

DataBaseSpace Table

Four columns from DBC.DataBaseSpace:

Notes:
1,019,904
+ 1,020,928
2,040,832

2,040,832
1,019,904
+ 1,020,928
4,081,664

DatabaseID	TableID (UPI)	Vproc	CurrentPermSpace	
00001404	00000000000	0	222,720	
00001404	00000000000	1	222,720	
00001404	000000000000	2	273,408	Table "All" Sum is 2,040,832
00001404	000000000000	3	273,408	
00001404	000000000000	4	251,904	
00001404	000000000000	5	251,904	
00001404	000000000000	6	272,384	
00001404	000000000000	7	272,384	
00001404	0000260C0000	0	127,488	
00001404	0000260C0000	1	127,488	
00001404	0000260C0000	2	127,488	
00001404	0000260C0000	3	127,488	Table2 Sum is 1,019,904
00001404	0000260C0000	4	128,000	
00001404	0000260C0000	5	128,000	
00001404	0000260C0000	6	126,976	
00001404	0000260C0000	7	126,976	
00001404	0000270C0000	0	95,232	
00001404	0000270C0000	1	95,232	
00001404	0000270C0000	2	145,920	
00001404	0000270C0000	3	145,920	Table2_nupi Sum is 1,020,928
00001404	0000270C0000	4	123,904	
00001404	0000270C0000	5	123,904	
00001404	0000270C0000	6	145,408	
00001404	0000270C0000	7	145,408	

Different Views — Different Results

Conflicting Results

It would seem logical that query results would be the same for any of the preceding views, since they all use the same underlying table. However, query results can differ depending upon which view you select.

The SQL statements and results on the facing page illustrate how a single SQL statement can produce a different result for each view.

For example, when we select the MAX (CurrentPerm) and SUM (CurrentPerm) from each of the AllSpace, DiskSpace, and TableSize views, our results will differ. The SUM (CurrentPerm) value from the DBC.AllSpace view represents the Sum of "All" tables (i.e., the database total) *plus* the sum of *each* table in the database. The results are misleading. We suggest that you do not use this query.

We recommend that you use the DBC.DiskSpace view for queries at the *database* level and use the DBC.TableSize view for queries at the *table* level.

Sum(CurrentPerm)

The DiskSpace view displays 2,040,832 bytes of total permanent space used for database ID 00001404. This figure reflects the total number of bytes stored on each processor in table ID 000000000000 or table ALL. Remember, the DiskSpace view reports on database space usage.

The TableSize view also displays 2,040,832 bytes of total CurrentPerm. This figure comes from the individual tables stored within the same database. The total comes from adding all of the bytes in tables 0000260C0000 and 0000270C0000 together. Since DiskSpace reports on the database and TableSize reports on the individual tables in the database, both result in the same figure.

The AllSpace view displays 4,081,664 bytes which is double what the other two views reported. This view displays the total of all tables including table ID 000000000000 or table ALL. Since table name ALL already contains the totals from all of the other tables, the resulting total is double what it should be.

Maximum(CurrentPerm)

Both DiskSpace and AllSpace display 273,408 bytes as the largest number of permanent space. Both views read the result from table 000000000000. TableSize, on the other hand, displays 145,920. TableSize looks at individual tables. It excludes figures stored in table ID 000000000000 or table ALL.

Different Views — Different Results

```
SELECT    MAX(CurrentPerm)
          ,SUM(CurrentPerm)
FROM      DBC.DiskSpaceV
WHERE     DatabaseName = USER ;
```

Maximum (CurrentPerm)	Sum (CurrentPerm)
273,408	2,040,832

Values only represent table ALL.

```
SELECT    MAX(CurrentPerm)
          ,SUM(CurrentPerm)
FROM      DBC.TableSizeV
WHERE     DatabaseName = USER ;
```

Maximum (CurrentPerm)	Sum (CurrentPerm)
145,920	2,040,832

Values represent all of the actual tables except table ALL.

```
SELECT    MAX(CurrentPerm)
          ,SUM(CurrentPerm)
FROM      DBC.AllSpaceV
WHERE     DatabaseName = USER ;
```

Maximum (CurrentPerm)	Sum (CurrentPerm)
273,408	4,081,664

Values represent all of the actual tables and table ALL.

Additional Utilities to View Space Utilization

Examples of additional tools that may be used to view database and table space utilization are provided in this module.

Additional Utilities to View Space Utilization

Teradata Administrator – graphical tool to easily view space usage

- **Database menu**
  - **Child Space – space usage for all child databases of the selected database**
  - **Table space – space usage for all tables of the selected database**
- **Object menu**
  - **Space Summary – current and peak perm usage of the specified table**
  - **Space by AMP – current and peak perm usage of the specified table by AMP**

Ferret – system utility – started via Database Console or Viewpoint Remote Console

- **ShowSpace – space usage (perm, spool, and temporary) at the system level**
- **ShowBlocks – allocation of permanent space at the table and subtable level**

Question – Why use ShowBlocks to determine space usage at a table level?

- **"How much perm space is currently being used by a secondary index?"**
  - **This level of detail is not available with DBC.TableSizeV and Teradata Administrator – only provide current perm space usage at the table level.**
  - **ShowBlocks provide subtable space information – multiply the typical block size times the number of blocks to determine subtable space usage.**

Teradata Administrator – Database Menu Options

Use the command selections on the Database pull-down menu to indicate the type of information you want displayed.

A check mark indicates the current setting of your database Default View option (i.e., the information displayed when you double click on a database.)

Note: The Database menu does not appear on the Teradata Administrator menu bar until you establish a connection with a database server.

Select a database from the database tree windowpane and make a selection from the Database pull-down menu. Selections and corresponding information displayed are identified in the table below.

Selection	Display Information
List Tables	Each table in the selected database
List Views	Each view in the selected database
List Macros	Each macro in the selected database
Database Info	The selected database itself
Database Rights	Access rights for each table, view, and macro in the selected database
Table Space	Space usage for each table in the selected database
Child Space	Space usage for each database that is owned directly by the selected database
List Databases	All databases and users created under the selected database
List All DB/TVM	Each table, view, macro, trigger and join index in the selected database and show all Child Database/Users in the tree
Open / Close DB	Expansion or contraction of the database tree

Teradata Administrator
Database Menu Options

Teradata Administrator can be used to easily view database/user space usage.

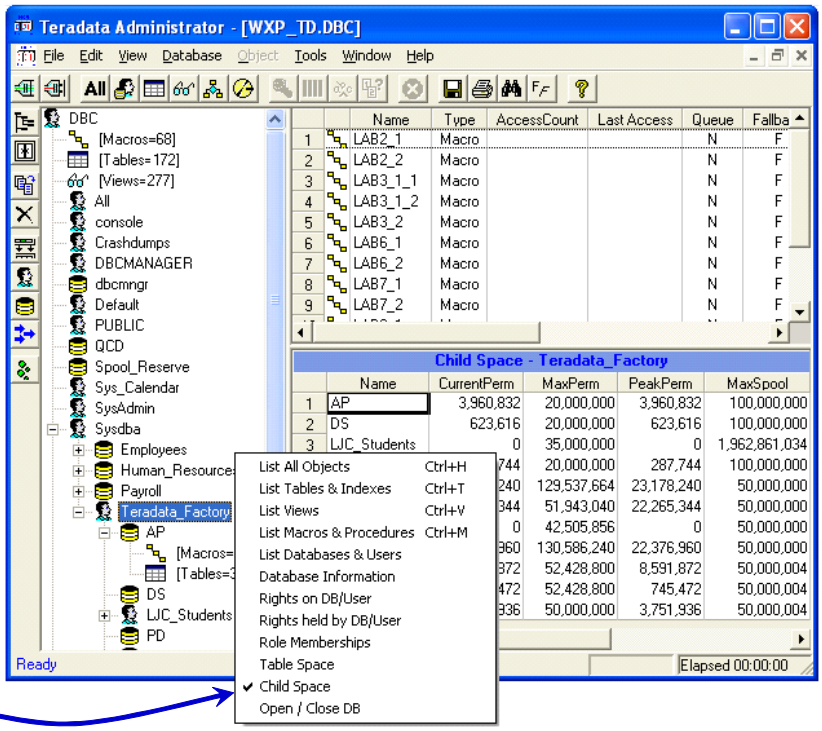
Database menu

- **List databases, tables, views and macros.**
- **Display database information and access rights.**
- **View table and child space usage.**
- **Information appears in grid area.**

Database > Child Space displays a space usage report for each database that is owned directly by selected database.

OR

Right-click on a database/user and select Child Space.



Teradata Administrator – Object Menu Options

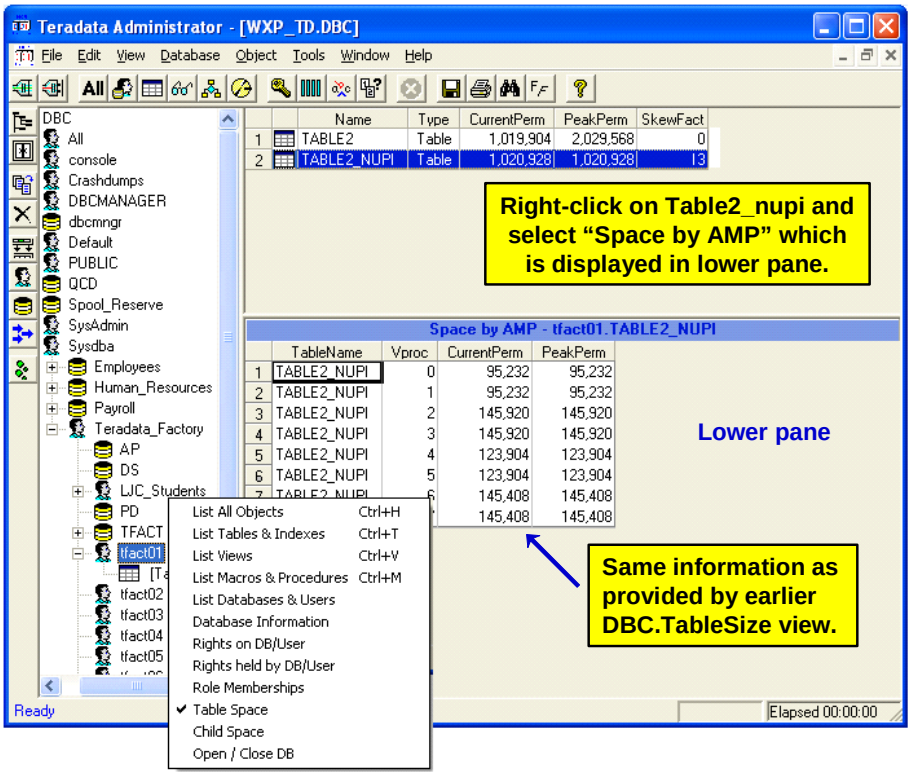
Select an item in the upper grid area, and use the submenu selections on the Object menu to display detail information, described below, about the selected table, macro, or view.

Object Type	Selection	Display Information
Table, View	List Columns	Information about the columns of the selected table or view
Table	Index	The indexes for the selected table
Table	Statistics	Statistics information for the selected table
Table, View	Row Count	The number of rows in the selected table or view
Table, View	Browse	Information from the data rows of the selected table or view
Table, View, Macro	Info	General information about the selected object
Table	Space Summary	Space usage information for the selected table
Table	Space by AMP	Space usage by AMP information for the selected table
Table, View, Macro	Rights	Access rights for the selected object
Table, View, Macro	Users	The users who have access rights to the selected object
Table	Journal	The journal table for the selected table
Table, View, Macro	Show Definition	The text that was used to create the selected object

Teradata Administrator
Object Menu Options

Object menu

- **Select a table, view or macro from the upper grid area and the desired submenu selection.**
- **This menu can also be seen by right-clicking on the object.**
- **Information appears in lower pane area.**



Transient Journal Space

Starting with Teradata V2R6.2, the Transient Journal images are maintained within the WAL Log. The WAL Log includes the following:

- Redo Records for updating disk blocks and insuring file system consistency during restarts, based on operations performed in cache during normal operation.
- Transient Journal (TJ) records used for transaction rollback.

The WAL Log is conceptually similar to a table, but the log has a simpler structure than a table. Log data is a sequence of WAL records, different from normal row structure and not accessible via SQL.

The system maintains the before-update copies of rows updated within a transaction in the Transient Journal (TJ). Prior to V2R6.2, the TJ records were stored in the system table DBC.TransientJournal. These are now stored in the WAL log.

For historical and reporting purposes, the Transient Journal still appears as a table within DBC. Its space may be greater than the maximum PERM space of DBC, but it is not getting its space from DBC. This table entry effectively indicates the size of the WAL log which is outside of DBC's perm space.

After-update images (REDO images), as well as all of the following items, are also contained within the file system Write Ahead Log, or WAL:

- Images of updates made to data blocks
- Images of updates made to cylinder indexes
- Images of updates made to File Information Blocks (FIBs)
- Instructions for where and how to use all these change images.

These WAL images are called *redo records*. After the system applies the appropriate set of WAL log redo records to the data on disk, then the data blocks, cylinder indexes, and FIB images appear as if the updated copies of those blocks that had really only been in memory, had actually been written to disk. In other words, the redo records apply their updates to older versions of those blocks.

The TJ records in the WAL log are *un*do records. After the system finishes processing the redo records, the data is in a consistent state, which permits the processing of the undo records.

During file system startup, and before the AMPs begin to come up, the file system handles any redo records in the WAL log that need to be processed. After that, the file system finishes its part of the startup process and the database software goes into normal recovery mode, where it processes any applicable TJ records in the same way they have always been processed.

Teradata Administrator
Transient Journal Space

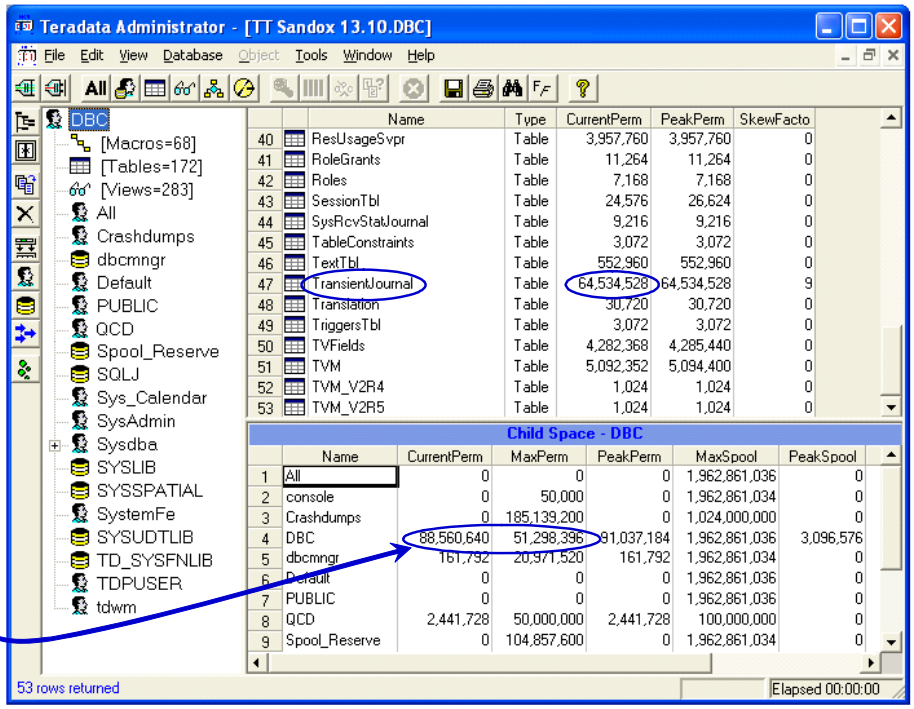
Transient Journal (TJ) images are maintained within the WAL Log.

For historical purposes, the TransientJournal table still appears within DBC.

TJ space is actually allocated in the WAL log and may have a value greater than DBC's maximum perm space.

This entry actually represents WAL Log space that is allocated.

Note that the MaxPerm of DBC is 51 MB, but its CurrentPerm is 88 MB.



Ferret Utility

To maintain data integrity, the Ferret utility (File Reconfiguration Tool) enables you to display and set various disk space utilization attributes associated with the Teradata database.

When you select the Ferret utility attributes and functions, it dynamically reconfigures the data on the disks to correspond with the selections.

Depending on the functions, Ferret can operate at the vproc, table, subtable, disk, or cylinder level.

Start Ferret from the DBW connected to the Teradata database. Note that the Teradata database must be in the Logons Enabled state.

The commands within the Ferret utility that we will discuss in this module include:

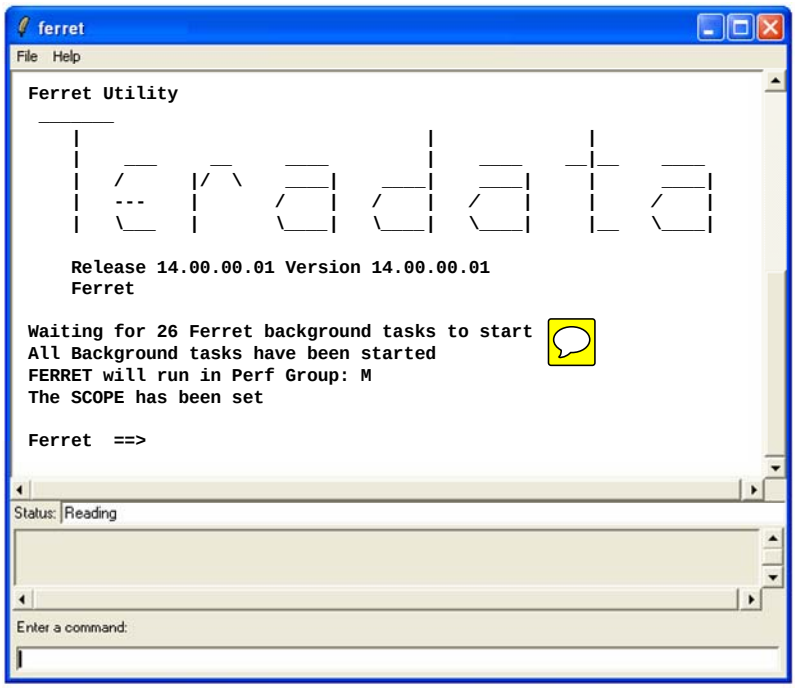
- SCOPE
- SHOWSPACE
- SHOWBLOCKS

To start the Ferret utility, enter the following command in the Supervisor screen of the DBW:

START FERRET

You will be placed in the interactive partition where the Ferret utility was started.

Ferret Utility



Functions of FERRET covered in this module:

- **SHOWSPACE**
- **SHOWBLOCKS**

26 background tasks were started – one for each AMP.

From Supervisor (of Database Window), enter: START FERRET

Ferret SHOWSPACE Command

The SHOWSPACE command reports the amount of disk cylinder space currently in use and the amount of cylinder space that remains available. Use SHOWSPACE to determine if disk compaction or system expansion is required.

SHOWSPACE is a command you execute from within the Ferret utility. To start the utility, enter ***START FERRET*** in the Supervisor window. Within the Ferret application window, enter ***SHOWSPACE*** (upper or lowercase). The Showspace command reports on physical disk utilization, reported as:

- Permanent space
- Spool space
- Global Temporary space
- Journal space
- Lost disk space from disk flaws
- Free disk space

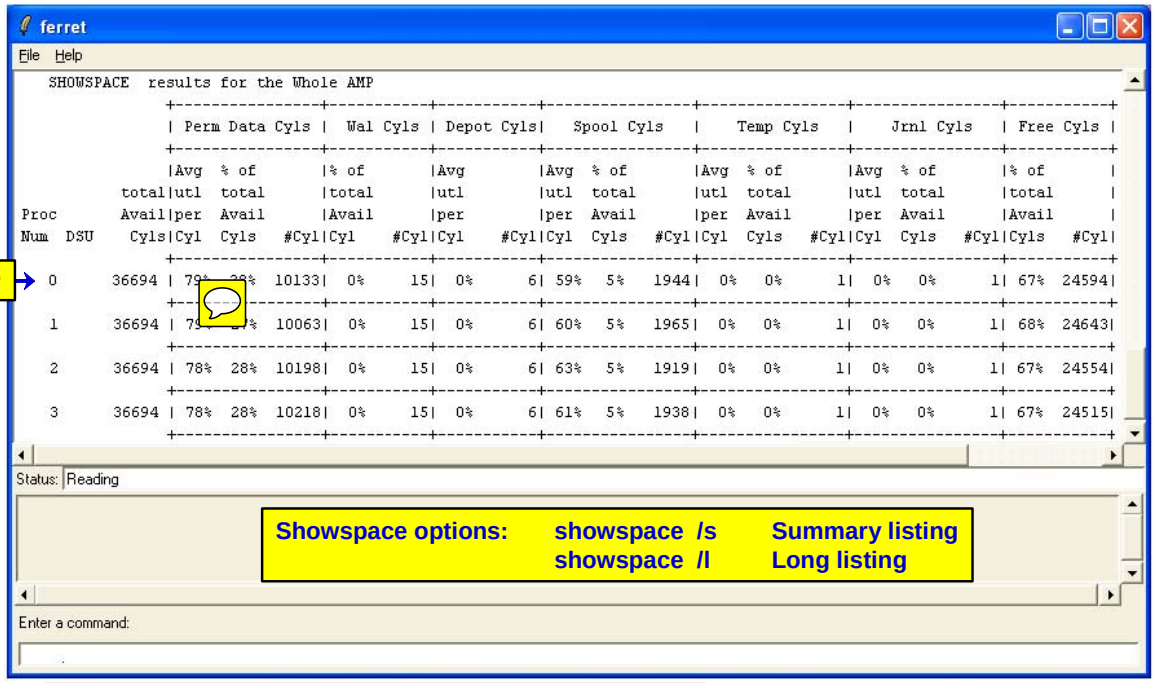
The facing page shows the results of a SHOWSPACE command. Notice the command displays the average utilization per cylinder for permanent and spool space. It displays the percentage of total available cylinders as well as the number of cylinders for all types of space.

Enter an **S** for a summary report that displays only subtotals for all AMP vprocs in the system. The facing page shows an example of a Showspace summary report.

The typical percentage of cylinders used for Permanent data is 28%.

Enter an **L** for a full report that displays Pdisk and Vdisk information for all AMP vprocs in the system. The full report format displays information separately for each of the Pdisks used by an AMP vproc, as well as total space utilization for the vproc.

Ferret SHOWSPACE Command




```
                +---------------+----------+----------+---------------+---------------+---------------+----------+
                | Perm Data Cyls |  Wal Cyls | Depot Cyls|   Spool Cyls   |    Temp Cyls   |    Jrnl Cyls   | Free Cyls |
                +---------------+----------+----------+---------------+---------------+---------------+----------+
                |Avg  % of       |% of      |Avg       |Avg  % of       |Avg  % of       |Avg  % of       |% of      |
          total|utl  total      |total     |utl       |utl  total      |utl  total      |utl  total      |total     |
Proc       Avail|per  Avail      |Avail     |per       |per  Avail      |per  Avail      |per  Avail      |Avail     |
Num  DSU    Cyls|Cyl  Cyls   #Cyl|Cyl   #Cyl|Cyl   #Cyl|Cyl  Cyls   #Cyl|Cyl  Cyls   #Cyl|Cyl  Cyls   #Cyl|Cyls   #Cyl|
                +---------------+----------+----------+---------------+---------------+---------------+----------+
   0      36694 | 79%  28%  10133|  0%     15|  0%      6| 59%   5%   1944|  0%   0%      1|  0%   0%      1| 67%  24594|
                +---------------+----------+----------+---------------+---------------+---------------+----------+
   1      36694 | 79%  27%  10063|  0%     15|  0%      6| 60%   5%   1965|  0%   0%      1|  0%   0%      1| 68%  24643|
                +---------------+----------+----------+---------------+---------------+---------------+----------+
   2      36694 | 78%  28%  10198|  0%     15|  0%      6| 63%   5%   1919|  0%   0%      1|  0%   0%      1| 67%  24554|
                +---------------+----------+----------+---------------+---------------+---------------+----------+
   3      36694 | 78%  28%  10218|  0%     15|  0%      6| 61%   5%   1938|  0%   0%      1|  0%   0%      1| 67%  24515|
                +---------------+----------+----------+---------------+---------------+---------------+----------+
```

AMP

Showspace options: **showspace /s** **Summary listing**
showspace /l **Long listing**

Approximately, how large (in GB) is each AMP's Vdisk?

What is the typical percentage of cylinders (per AMP) that is used for Permanent data?

Ferret SHOWBLOCKS

The Ferret utility includes a SHOWBLOCKS command that displays the data block size and/or the number of rows per data block for a defined scope.

The SHOWBLOCKS command displays the following disk space information for a defined range of data blocks and cylinders.

SHOWBLOCKS /M – displays subtable information.

SHOWBLOCKS /L – displays minimum, average, and maximum number of rows per block.

SHOWBLOCKS /S – displays table level information – doesn't display empty tables (tables with no rows).

Another option to set the scope for the table (example on facing page) is to use the following command:

- SCOPE TABLE ("PD.Employee" 0)

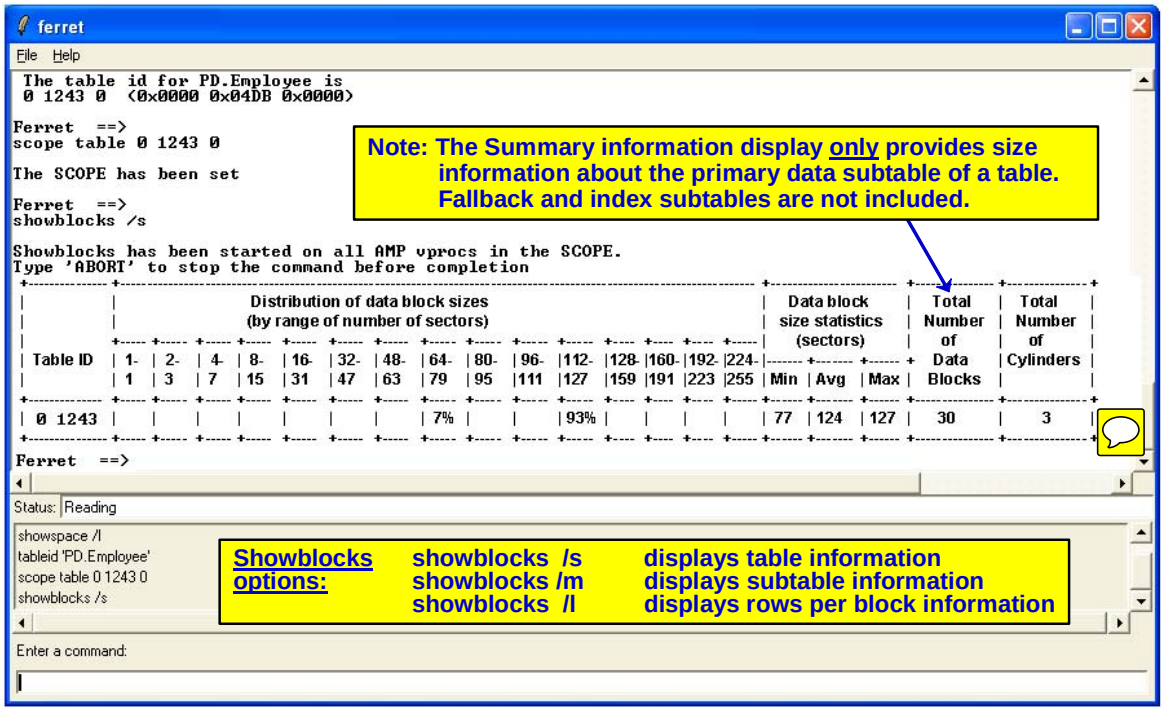
The facing page poses this question – "How large (in MB) is primary data subtable?"

The solution is (typical block size) x (total number of blocks).

124 sectors x 512 bytes = 62 KB

62 KB x 30 blocks = 1920 KB or 2 MB.

Ferret SHOWBLOCKS Command



How large (in MB) is this primary data subtable?

Ferret SHOWBLOCKS – Subtable Detail

The Ferret ShowBlocks utility also allows you to view block sizes down to the subtable level. The **/m** option provides this level of detail

Notes about Subtable IDs (shown in decimal in ShowBlocks report):

0 – Header
1024 – Primary data subtable
2048 – Fallback subtable

1028 – 1st Secondary Index subtable
2052 – 1st Secondary Index Fallback subtable

1032 – 2nd Secondary Index subtable
2056 – 2nd Secondary Index Fallback subtable

1536 – 1st Reference Index subtable
2560 – 1st Reference Index Fallback subtable

1792 – 1st BLOB or CLOB subtable
2816 – 1st BLOB or CLOB subtable

1794 – 2nd BLOB or CLOB subtable
2818 – 2nd BLOB or CLOB subtable

The facing page poses this question – "How large (in MB) is first secondary index?"

The 1st secondary index subtables have subtable IDs of 1028 and 2052.

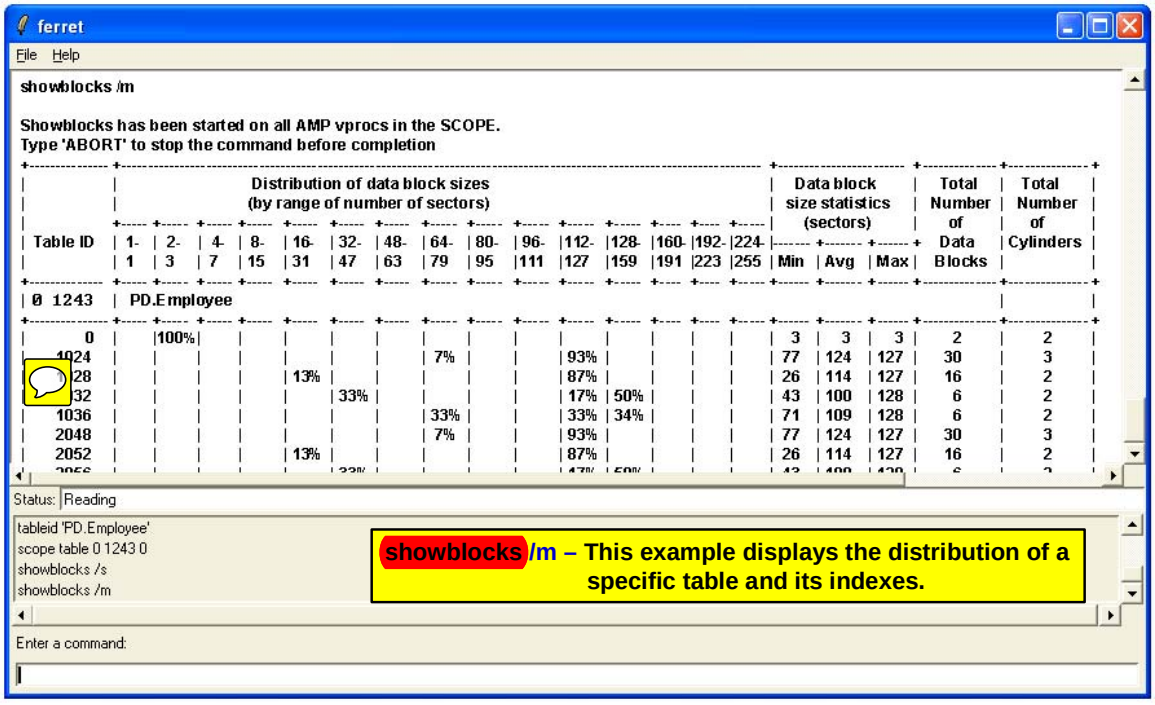
The solution is (typical block size) x (total number of blocks).

114 sectors x 512 bytes = 57 KB

57 KB x 16 blocks x 2 = 1824 KB or 1.8 MB.

Note: Subtable ID of 1028 is 16 blocks and the fallback (2052) is also 16 blocks. The 1st secondary index uses a total of 32 blocks.

Ferret SHOWBLOCKS – Subtable Detail



How large (in KB) is the 1st secondary index?

Module 43: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 43: Review Questions

1. True or False. Space limits are enforced at the table level.
2. True or False. When you use the GIVE statement to transfer a database/user, only the tables allocated to the original database/user are transferred to the new database/user.
3. True or False. You should reserve at least 25% of total available space for spool.
4. The DBC. _____ view provides disk space usage at the table level and excludes table ALL.
5. The DBC. _____ view only provides disk space usage at the database level.

Notes

Module 44

Users, Accounts, and Accounting

After completing this module, you will be able to:

- **Use Teradata accounting features to determine resource usage by user or account.**
- **Explain how the database administrator uses system accounting to support administrative functions.**
- **Use system views to access system accounting information.**

Notes

Table of Contents

Creating New Users & Databases 44-4
CREATE DATABASE Statement 44-6
CREATE USER Statement 44-8
CREATE USER and the Data Dictionary 44-10
CREATE USER and the Data Dictionary (cont.) 44-12
MODIFY USER Statement 44-14
Teradata Administrator – Tools Menu > Create Options 44-16
Creating and Using Account IDs 44-18
    Using Account IDs with Logon 44-18
Dynamically Changing an Account ID 44-20
    Syntax 44-20
    Examples 44-20
Account Priorities 44-22
Account String Expansion 44-24
    ASE Variables: 44-24
ASE Accounting Example 44-26
    Background Information 44-26
    Tasks 44-26
System Accounting Views 44-28
    AccountInfo Views 44-28
    AMPUsage Views 44-28
AccountInfo View 44-30
    Example 44-30
AMPUsage View 44-32
AMPUsage View – Example 44-34
    Example 44-34
Users, Accounts & Accounting Summary 44-36
Module 44: Review Questions 44-38
Lab Exercise 44-1 44-40
  Lab Exercise 44-1 (cont.) 44-42
  Lab Exercise 44-1 (cont.) 44-44

Creating New Users & Databases

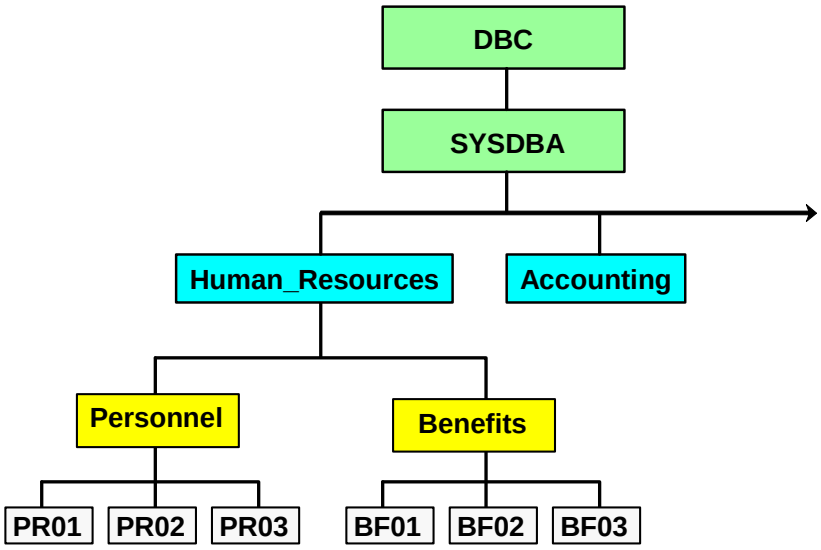
As the database administrator, you create system databases and tables and assign user privileges and access rights to tables.

To perform the above tasks, you must:

- Determine database information content and create macros to ensure the referential integrity of the database.
- Define authorization checks and validation procedures.
- Perform audit checks on the database for LOGON, GRANT, REVOKE and other privilege statements.

You can give the authority to use the CREATE DATABASE or CREATE USER statements to any application user. He or she may then create other system users or databases from his or her own space, or if specifically authorized, from the space of another system database or user.

Creating New Users and Databases



You can grant CREATE DATABASE and/or CREATE USER authority to any user.

The user may then create other users and databases from:

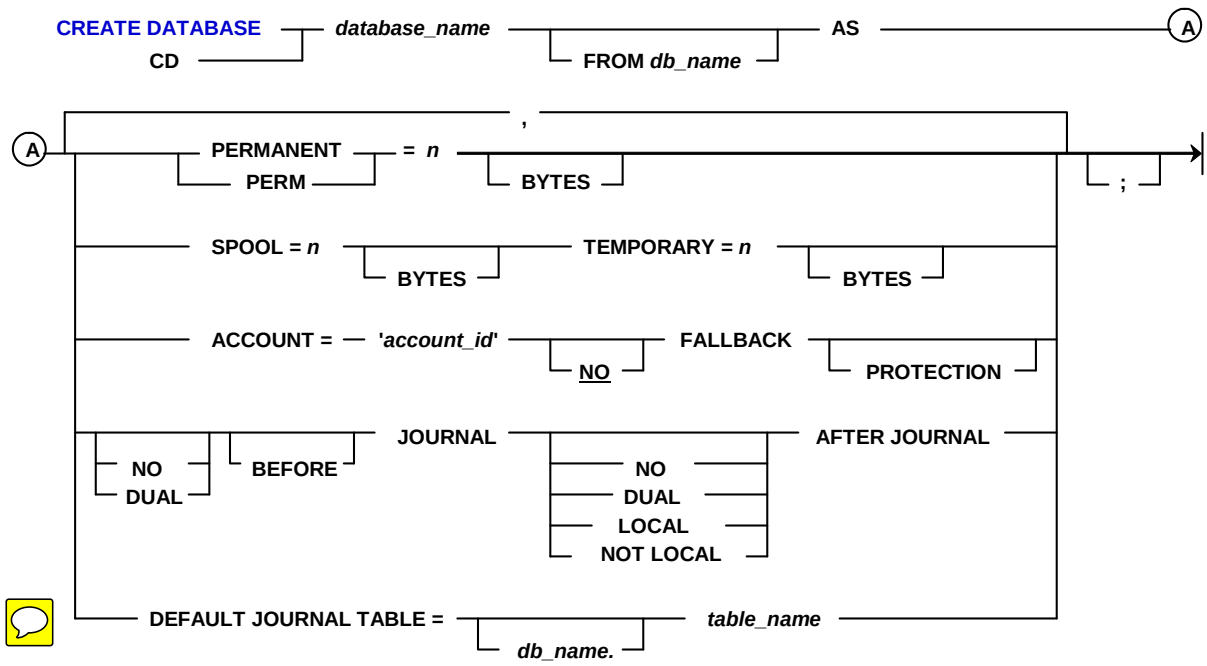
- The user's own space, or
- The space of another user or database (if authorized).

CREATE DATABASE Statement

As the database administrator, you use the CREATE DATABASE statement to add new databases to the existing system. The permanent space for new databases you create comes from the immediate parent database or user. A database becomes a uniquely named collection of tables, views, macros, triggers, stored procedures, and access rights.

The spool and temporary definitions are not relevant to a database. However, the spool and temporary definitions establish the maximum and default value for databases/users that are created as children under this database.

CREATE DATABASE Statement



CREATE USER Statement

The CREATE USER statement enables you to add new users to the system. The permanent space for these new users comes from the immediate parent database or user.

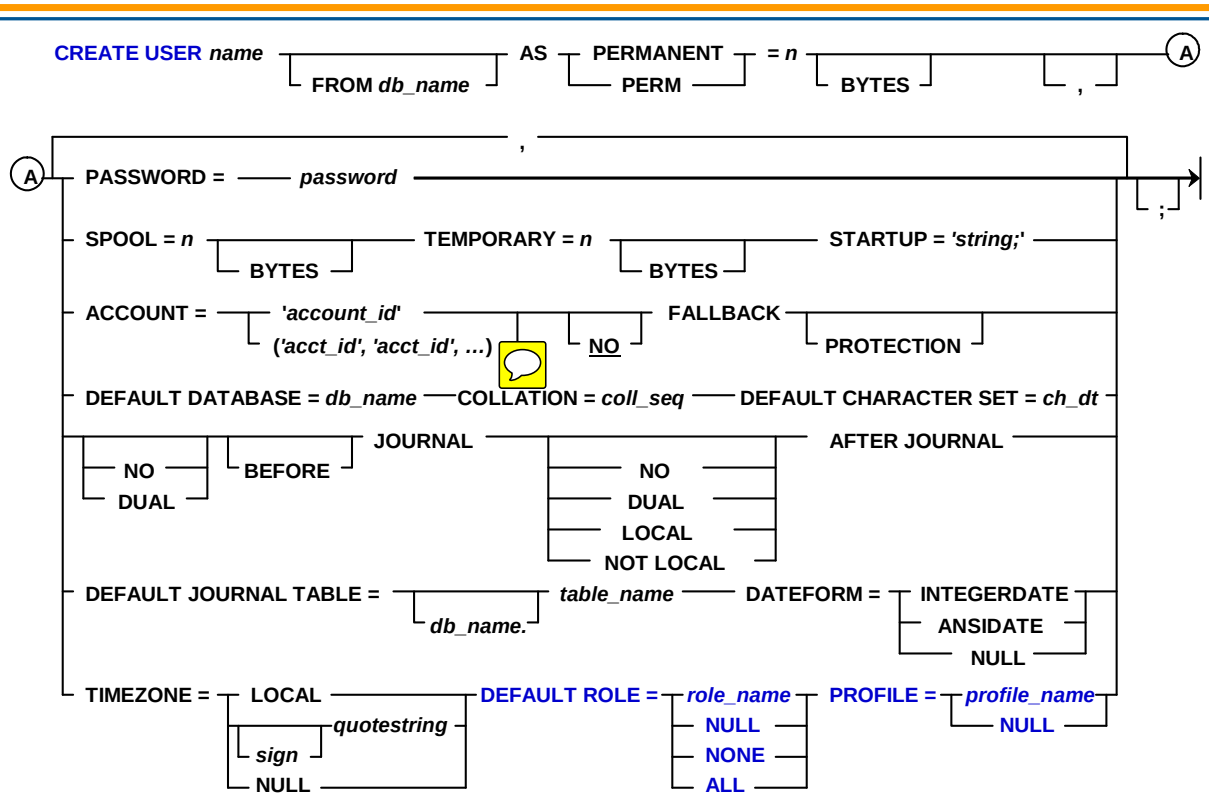
Users have passwords while databases do not. User passwords allow users to log on to the Teradata database and establish sessions.

When you create a new user, you also create a temporary password for the user. When the user logs on for the first time, he or she is prompted to change the password. Note: This assumes the password expiration time period is not set to 0.

If a user forgets the password, you can assign a new temporary password. (As another option, you can set user passwords not to expire.)

Acronym: ch_dt – Character Data Type

CREATE USER Statement



CREATE USER and the Data Dictionary

In addition to creating a new system user, the CREATE USER statement also defines space.

A user is associated with a password and an account, and can log on, establish a session, and execute SQL statements. A user, not a database, performs these actions.

Notice the entries the CREATE USER statement makes in the data dictionary.
Default values associated with the CREATE USER statement are:

Entry	**Defaults to the value of**
FROM database	Current CREATOR
SPOOL	Same value as the OWNER
TEMPORARY	Same value as the OWNER
STARTUP	Null (no startup string)
ACCOUNT	Immediate OWNER'S first account ID
DEFAULT DATABASE	Username

There are also two types of rights granted automatically when you use the CREATE USER statement:

- The rights granted to a newly created user or database on itself.
- The rights granted on a newly created user, database, or object to the creating user.

By issuing a CREATE USER statement, the creator gains certain automatic rights over the created object.

As shown on the facing page, the database administrator logged on as Sysdba and creates tfact06:

```
CREATE USER tfact06 AS PERM = 100E6, SPOOL = 1E9,
    PASSWORD = secure1time;
```

CREATE USER and the Data Dictionary

EXPLAIN

CREATE USER tfact06 AS PERM = 100E6, SPOOL = 1E9, PASSWORD = secure1time;

Explanation

1) First, we lock data base tfact06 for exclusive use.
2) Next, we lock a distinct DBC."pseudo table" for write on a RowHash to prevent global deadlock for DBC.DataBaseSpace.
3) We lock a distinct DBC."pseudo table" for write on a RowHash to prevent global deadlock for DBC.AccessRights.
4) We lock a distinct DBC."pseudo table" for write on a RowHash to prevent global deadlock for DBC.Parents.
5) We lock a distinct DBC."pseudo table" for write on a RowHash to prevent global deadlock for DBC.Owners.
6) We lock DBC.DataBaseSpace for write, we lock DBC.AccessRights for write, we lock DBC.Parents for write, we lock DBC.Owners for write, we lock DBC.Accounts for write on a RowHash, we lock DBC.DBase for write on a RowHash, and we lock DBC.DBase for write on a RowHash.
7) We execute the following steps in parallel.
    1) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index with no residual conditions.
    2) We do a single-AMP ABORT test from DBC.Roles by way of the unique primary index with no residual conditions.
    3) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.
    4) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.
    5) We do an INSERT into DBC.DBase.
    6) We do a single-AMP UPDATE from DBC.DBase by way of the unique primary index with no residual conditions.
    7) We do a single-AMP RETRIEVE step from DBC.Parents by way of the primary index with no residual conditions into Spool 1 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 1 by row hash.
8) We do an all-AMPs MERGE into DBC.Owners from Spool 1 (Last Use).

CREATE USER and the Data Dictionary (cont.)

Several steps are performed in parallel during the CREATE USER statement.

CREATE USER and the Data Dictionary (cont.)

9) We execute the following steps in parallel.
    1) We do an INSERT into DBC.Owners.
    2) We do a single-AMP RETRIEVE step from DBC.Parents by way of the primary index with no residual conditions into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash.
10) We do an all-AMPs MERGE into DBC.Parents from Spool 2 (Last Use).
11) We execute the following steps in parallel.
    1) We do an INSERT into DBC.Parents.
    2) We do an INSERT into DBC.Accounts.
    3) We do a single-AMP RETRIEVE step from DBC.AccessRights by way of the primary index into Spool 3 (all_amps), which is redistributed by hash code to all AMPs.
12) We execute the following steps in parallel.
    1) We do a single-AMP RETRIEVE step from DBC.AccessRights by way of the primary index into Spool 3 (all_amps), which is redistributed by hash code to all AMPs.
    2) We do an all-AMPs RETRIEVE step from DBC.AccessRights by way of an all-rows scan into Spool 4 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 4 by row hash.
13) We do an all-AMPs JOIN step from DBC.Owners by way of a RowHash match scan, which is joined to Spool 4 (Last Use). DBC.Owners and Spool 4 are joined using a merge join. The result goes into Spool 3 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 3 by row hash.
14) We do an all-AMPs MERGE into DBC.AccessRights from Spool 3 (Last Use).
15) We flush the DISKSPACE and AMPUSAGE caches.
16) We do an all-AMPs ABORT test from DBC.DataBaseSpace by way of the unique primary index.
17) We do an INSERT into DBC.DataBaseSpace.
18) We do an all-AMPs UPDATE from DBC.DataBaseSpace by way of the unique primary index with no residual conditions.
19) We flush the DISKSPACE and AMPUSAGE caches.
20) We spoil the parser's dictionary cache for the database.
21) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> No rows are returned to the user as the result of statement 1.

MODIFY USER Statement

The MODIFY USER statement enables you to change the options of an existing user.

Options you can change without the DROP DATABASE privilege include:

- Password
- Startup string
- Default database
- Collation
- Fallback Protection default
- Default Dateform
- Default Character Set data type
- Timezone
- Permanent journal default options

Options requiring the DROP DATABASE privilege are:

- PERMANENT space limit
- SPOOL space limit
- TEMPORARY space limit
- Account codes
- Release password lock
- DROP DEFAULT JOURNAL TABLE
- Role
- Profile

The FOR USER option effectively established a temporary password that can be used to logon one time by the user. This option is only effective if the ExpirePassword attribute (set in DBC.SysSecDefaults or a profile) is set to a value greater than 0.

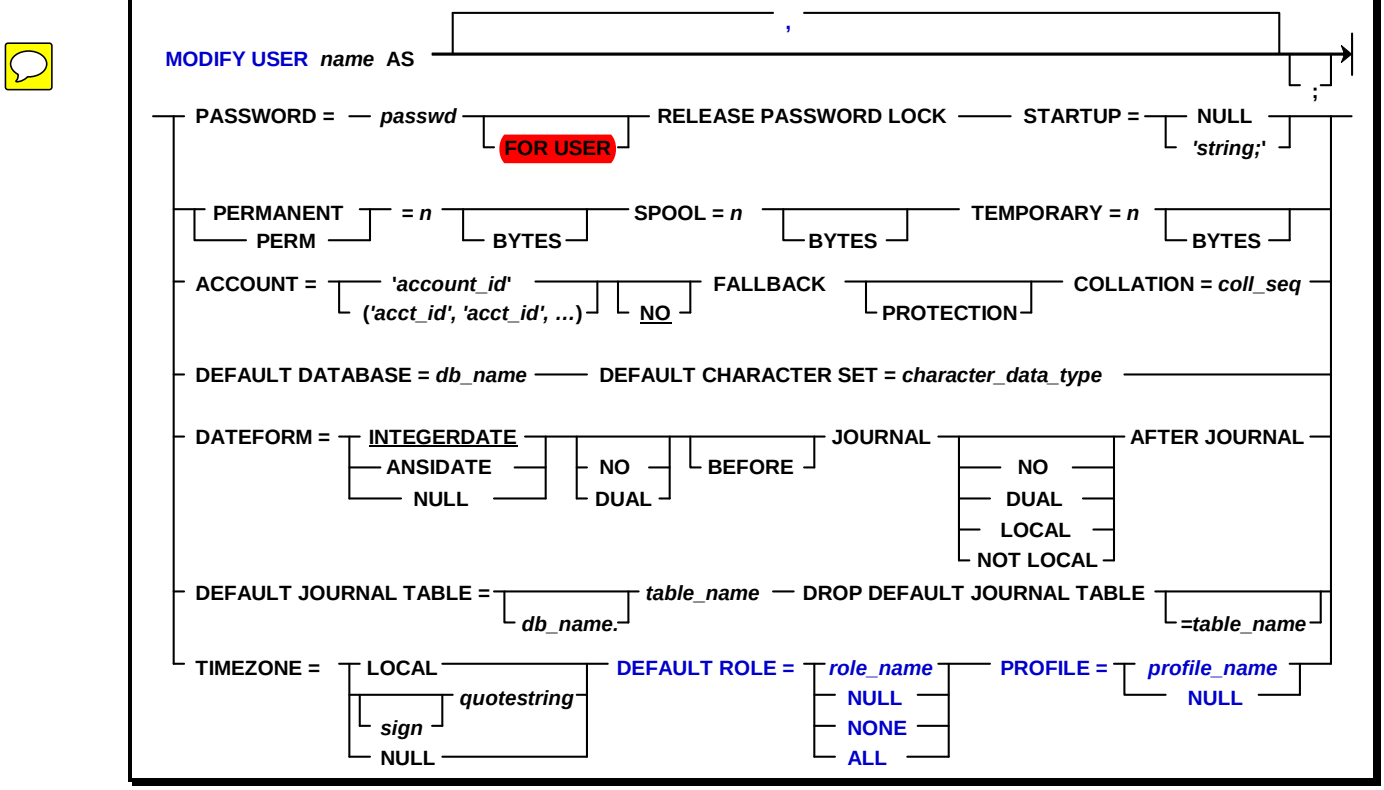
```
MODIFY USER RobertSmith AS PASSWORD = secret FOR USER;
```

The existing password immediately expires and is replaced by “secret”. In this example, “secret” is effectively a temporary password that allows a one-time logon. The value for PasswordChgDate is reset to 0.

Note: The PasswordChgDate column is also set to 0 when a new user is created – assuming that ExpirePassword is set to a value greater than 0.

The temporary password expires immediately when the user logs on for the first time and the user needs to select a new, permanent password at that time. Another option is to use the MODIFY USER command without the FOR USER option.

MODIFY USER Statement



Teradata Administrator – Tools Menu > Create Options

The Tools menu provides the following options.

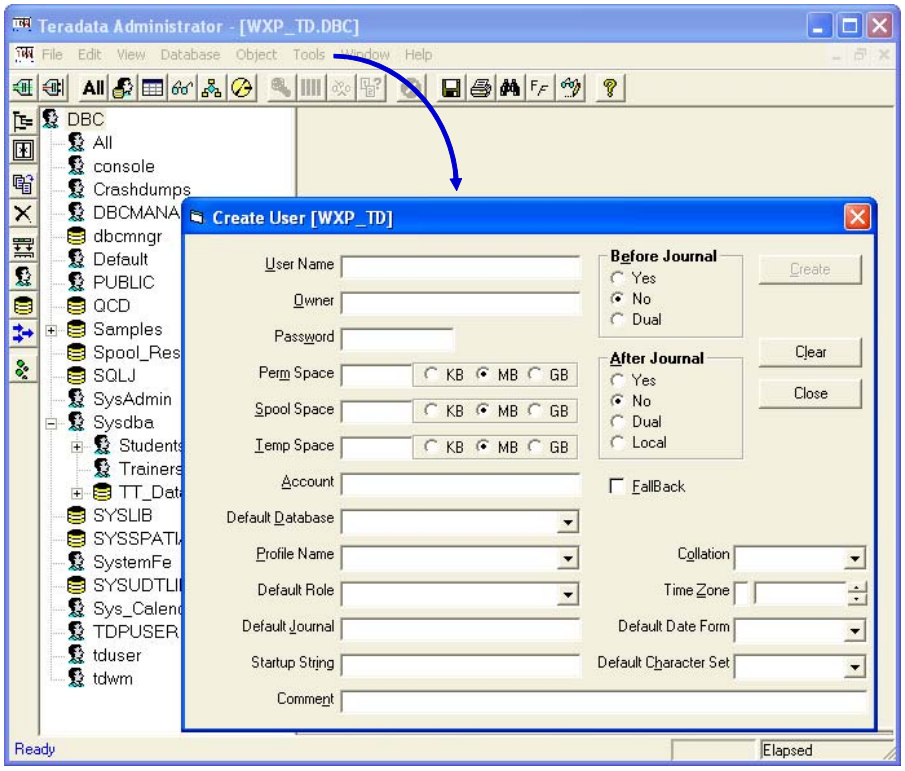
Menu Selection	Function / Options
Create	**Create an entirely new object – Database, Table, User, Profile, or Role.**
Grant/Revoke	Grant or revoke general access privileges to users. Options include Object Rights, System Rights, Logon Rights, or Column Rights.
Administer Profiles	Create and manage Profiles for users.
Administer Roles	Create and manage Roles.
Clone User	Create a new user either identical or closely related to an existing user.
Modify User	Change the specifications of an existing user.
Access Logging	Create and manage Access Log rules.
Query Logging	Create and manager Query Log rules.
Move Space	Reallocate permanent disk space from one database to another (efficient if not a direct descendant or parent).
Query	Create, modify, test, or run SQL query scripts.
Options	Configure the operational preferences for Teradata Administrator.

Teradata Administrator
Tools Menu > Create Options

Teradata Administrator can be used to create and manage users and databases.

Tools menu

- **Selections to create and modify databases and users, grant/revoke access rights, and send ad hoc query requests to Teradata.**
- **Options include the ability to clone a user, move space, and set preferences.**
- **This example illustrates how to create a new user by completing the entries.**



Creating and Using Account IDs

When you create a user, you can specify one or more account IDs that a new user can specify. Account codes may be used to track system CPU, I/O usage, or space usage. When the user logs on, the user can specify a valid account ID, or let the first account ID in the user row (from CREATE or MODIFY USER) become the default.

You should determine an account ID scheme for ease of accounting and priorities.

Account IDs may begin with the characters $L, $M, $H, or $R to identify the priorities low, medium, high, and rush, respectively. The relative level of CPU service is 1, 2, 4, and 8, respectively. These priority levels will be discussed on the following pages.

Using Account IDs with Logon

All logons require an account ID. A user can submit an explicit account ID by including it in the logon string. It must be a valid ID specified in the last CREATE or MODIFY USER statement. If no ID is specified in the CREATE or MODIFY statements, it defaults to the ID of the immediate owner's database.

Note:

batch logon syntax:

.LOGON tdpid/user_name, password, 'account_ID';

BTEQ Interactive logon syntax:

.LOGON tdpid/user_name,, 'account_ID' **(note the two commas)**
Enter password when prompted

Creating and Using Account IDs

```
CREATE USER tfact07                                  Names user
FROM   Sysdba                                        Name of immediate owner in hierarchy
AS     PERM          = 1E9                           Amount of Permanent space
      ,SPOOL         = 20E9                          Maximum amount of Spool space
      ,PASSWORD      = Secure12                      Initial password
      ,FALLBACK                                      Specifies Fallback as the default protection type
      ,ACCOUNT       = (  '$M_9038_&S&D&H',          Default account – medium priority
                          '$H0+EDUC&S&D&H',          Opt. Account – high priority – recommended format
                          '$M1$LOAD&S&D&H',          Opt. Account – performance group M1
                          '$M_9038' );               Opt. account – medium priority – no ASE
```

A logon can optionally include an account ID; the first account ID is the default account ID.

batch logon syntax: .LOGON tdpid/username,password,'account_id'

Example: .LOGON tdt6-1/tfact07,Secure12,'$H0+EDUC&S&D&H

BTEQ Interactive logon syntax: .LOGON tdpid/username,,'account_id'

Example: .LOGON tdt6-1/tfact07,,'$H0+EDUC&S&D&H'

Enter password when prompted
****

SQL Assistant Notes:

- With an ODBC connection, single quotes for the account id are optional.
- With a .NET connection, the single quotes for the account id are not used.

Dynamically Changing an Account ID

You can dynamically change your Account ID without logging off and logging back on. One reason you may want to do this is to change your session's priority. This is also called "nicing a query". "Nicing" is a UNIX term that means manipulating the scheduling priority of a "running" task. You typically "nice" a query to re-prioritize jobs. For instance, you could nice a query to a higher priority to run a business-critical job sooner than under its originally defined priority.

Self-nicing refers to a user specifying changes on his/her own request or session. Asynchronous nicing refers to a super user or system administrator manipulating another user's account.

Use the SET SESSION ACCOUNT statement to change your performance group (account priority) for the next SQL query you run, or for all jobs for the remainder of the current session.

Syntax

For the next SQL statement:

SET SESSION ACCOUNT = '*Account_ID*' FOR REQUEST;

For the remainder of the current session:

SET SESSION ACCOUNT = '*Account_ID*' FOR SESSION;

Examples

You cannot change a priority to exceed the priority originally defined by the performance group for an account or to a forbidden priority level. The following chart shows three accounts and the defined, permitted and forbidden priorities.

Account	*Defined Priority*	*Priority Definition*	*Permitted Priority Changes*	*Forbidden Priority Changes*
Sales	$H	High	<=$H	$R
Marketing	$M	Medium	<$H	$H
Development	$L	Low	None	>$L

You can see that you cannot change marketing's account priority to high, because it exceeds the group's original priority definition and is a forbidden priority for the account.

As another example, you can change sale's priority group to low or medium, because they do not exceed the groups' original priority definition and are not forbidden for the account.

Lastly, you cannot change development's priority. The chart shows that no priority changes are permitted and that the account cannot have any priority that exceeds low.

Dynamically Changing an Account ID

- You can change your Account ID without logging off. This may be done to re-prioritize a query. This is also referred to as "nicing a query".
- You can change Account IDs for the next SQL statement you run, or for all jobs for the remainder of the current session.
- To change Account IDs, use the SET SESSION ACCOUNT statement:

Syntax:

For the next SQL statement : `SET SESSION ACCOUNT = 'Account_ID' FOR REQUEST;`

For the rest of the current session: `SET SESSION ACCOUNT = 'Account_ID' FOR SESSION;`

Example:

For the rest of the session: `SET SESSION ACCOUNT = '$H0+EDUC&S&D&H' FOR SESSION;`

- Note: You can only use valid account IDs. Therefore, you cannot exceed the priority defined by the performance groups in your account ID.

Account Priorities

Account IDs may begin with the characters $L, $M, $H, or $R to identify the priorities low, medium, high, and rush, respectively. The relative level of CPU service is 1, 2, 4, and 8, respectively.

The Priority Scheduler facility lets you use codes to assign users to performance groups using these levels and user-defined levels of CPU usage. The Priority Scheduler facility will be described in detail later in the course.

You can design billing algorithms to reflect the usage of higher or lower account priorities. That way, a user with $H account priority is charged more for using system resources than a user with $L account priority.

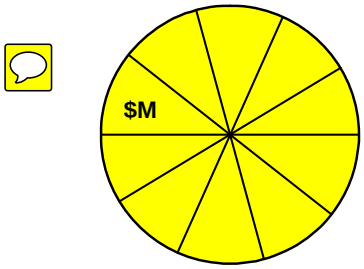
Character	Priority	**CPU Service**	Comments
$L	Low	1	Consider using for queries not needed immediately, such as batch jobs.
$M	Medium	2	Consider using for complex ad hoc queries.
$H	High	4	Consider using for tactical or OLTP queries.
$R	Rush	8	Use for very critical queries

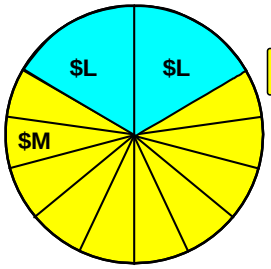
Account Priorities

Performance Groups	Default Weights	Possible Uses
$L	5	Low – consider using for low priority queries (e.g., batch jobs)
$M	10	Medium – consider using for complex ad hoc queries
$H	20	High – consider using for index access queries (e.g., tactical)
$R	40	Rush – consider for critical index access user queries

Performance Group	Active Sessions	Relative Weight Calc.	Percent Allocated
$L	N	0	0
$M	Y (10)	10/10	100
$H	N	0	0
$R	N	0	0

Performance Group	Active Sessions	Relative Weight Calc.	Percent Allocated
$L	Y (2)	5/15	33.3
$M	Y (10)	10/15	66.7
$H	N	0	0
$R	N	0	0





Account String Expansion

Account String Expansion (ASE) is an optional feature that enables you to use substitution variables in the account ID portion of the user's logon string. These variables enable you to include date and time information in the string. You must explicitly modify a user's logon to order to use ASE. The variables are resolved at logon time or at actual SQL execution time.

Account strings cannot exceed 30 characters. If, as a result of string expansion, you generate a string longer than 30 characters, the system truncates all characters to the right of position 30. Separation characters, such as colons in time fields and slashes in dates, are included in the character count.

ASE Variables:

&L The **logon time stamp** variable causes the logon time stamp to be inserted into the account string. The full logon time stamp consists of 15 characters and becomes truncated if &L is placed in position 17 or higher. The value inserted into AMPUsage is established at logon time. It does not change unless the user logs off then logs on again.

&D The **date** variable causes the date to be inserted into the account string. The value becomes truncated if you place &D at, or to the right of, position 26 or higher. You can use truncation to monitor resources on a yearly or monthly basis.

&T The **time** variable inserts the time of day into the account string. The value becomes truncated if you place &T at, or to the right of, position 26 or higher. You can use truncation to monitor resources hourly or by the minute. This variable allows for one-second granularity, thus causing a row to be written for virtually every individual SQL request.

&H The **hour** variable inserts the hour of the day into the account string. The inserted value consists of two characters and becomes truncated if you place &H to the right of position 29.

&I The **logon host ID/session number/request number** variable inserts the logon host ID, the session number and the request number into the account string.

&S The **session number** variable inserts the current session number into the account string.

Account String Expansion

- **ASE is used to provide more detailed utilization reports and user accounting data.**
  - **ASE increases the granularity of information returned with the DBC.AMPUsageV.**
  - **For queries that span multiple hours, the time will be accumulated in its entirety to the query's start hour.**
- **You may add the following substitution variables to a user's account string. The system resolves the variables at logon or at SQL statement execution time.**

&S	**Session number**	**(SSSSSSSSS)**
&D	**Date**	**(YYMMDD)**
&H	**Hour**	**(HH)**
&I	**Logon hostid, session number, request number**	**(LLLL SSSSSSSSS RRR RRR RRR)**
&L	**Logon timestamp**	**(YYMMDDHHMISS.hh)**
&T	**Time**	**(HHMISS)**

- **$xxxWORK&S&D&H is the recommended account format where WORK is a 4-character work load type starting in the 5th position of an account id.**
  - **For $L, $M, $H, and $R in Resource Partition 0, then examples are:**

    $M0+EDUC&S&D&H or $M0_EDUC&S&D&H (+ and _ are simply placeholders)
  - **If a two-character performance group name like M1 (or MD) is used, then examples are:**

    $M1$TACT&S&D&H or MDLOAD&S&D&H

ASE Accounting Example

Background Information

Two existing users, TFACT01 and TFACT02, logged onto the system using an account string defined as &S&D&H. The DBC.Acctg table contains a number of rows generated by the ASE feature.

Logon example**: .LOGON DBC/tfact01, password, '$M_9038_&S&D&H';**

Tasks

You need to create a table, view, and a number of reports that provide billing and resource usage information based on the statistics collected by the AMPUsage view.

Step 1. Create AmpUsageSum table.
Create a table to hold the collected statistics from the AMPUsage view. This table will serve as the basis for all of the other objects that you create. This is a history table since it contains stored historical data.

Step 2. Populate AMPUsageSum table.
After you build the AmpUsageSum table, use the INSERT command to populate it with row information from the DBC.AMPUsage view.

Step 3. Create Usage view
Use the CREATE statement to combine columns from the DBC.AMPUsage view and DBC.LogOnOff view into the Usage view.

Step 4. Create billing and resource usage reports.
Once the view is completed, construct SQL statements to SELECT information from the Usage view to create billing and resource usage reports.

ASE Accounting Example

- Users TFACT01 and TFACT02 each log on with $M0+9038&S&D&H account string.
  - Each hour a new row is placed in Acctg.
- Impact of using ASE variables with Acctg.

ASE Variable	Performance Impact	Data Capacity Impact
none	Negligible	1 row per account per AMP.
&D	Negligible	1 row per account per day per AMP.
&H	Negligible	1 row per account per hour per AMP (all days go into 1 hour).
&D&H	Negligible	1 row per account per hour per day per AMP.
&S&D&H	Negligible	1 row per account per session per hour per day per AMP.
&L	Negligible	1 row per logon (LAN) or session pool.
&T	Potentially Non-negligible	1 row per query per AMP.

- Perform the following tasks to extract accounting information:

  Step 1. Create AMPUsageSum table.
  Step 2. Populate AMPUsageSum table.
  Step 3. Create Usage view.
  Step 4. Create billing and resource usage reports.

System Accounting Views

The Teradata Database provides two system-supplied views to support accounting functions.

AccountInfo views provide information about valid accounts, and AMPUsage views provide information about the usage of each AMP vproc by user and account.

AccountInfo Views

The DBC.AccountInfo[V][X] views provide information about valid accounts for a specific user. The information provided is based on data from the DBC.Accounts table in the data dictionary. Each time a CREATE or MODIFY statement indicates an account ID, a row is either inserted or updated in the DBC.Accounts table.

(When you use restricted views, you must be the requester or have modify rights turned on.)

AMPUsage Views

The DBC.AMPUsage[V][X] views provide information about the usage of each AMP vproc for each user and account. It is based on information in the DBC.Acctg table in the data dictionary and supplies information about AMP CPU time consumed, and the number of AMP to DSU read and write operations generated by a given user or account. It also tracks the activities of any console utilities.

Each time a user logs on or submits an SQL request; a row is either inserted or updated in the DBC.Acctg table. If the user_name/account_name does not exist, then a new row is inserted. If the row already exists in the DBC.Acctg table, then it is updated. The rows in this table track how much AMP usage the specific user_name/account_name generates. This information may be used to bill an account for system resource use.

System Accounting Views

View	Description
DBC.AccountInfo[V][X]	Returns each Account Name (Account ID) associated with a user (for users the requestor owns).
DBC.AMPUsage[V][X]	Provides information about I/O and AMP CPU usage by user and account.

AccountInfo View

The DBC.AccountInfo[V][X] views shown on the facing page provide information about each user and the valid account codes associated with each user. When the requesting user indicates the [X] view, they can only see information about users that they own or have modify rights on.

The UserOrProfile column is new with V2R5 and indicates whether the user is an actual user or a profile.

Example

The SQL statement on the facing page requests a list of all users with a valid HIGH priority account code.

AccountInfo View

Provides information about each user and the valid account codes associated with each.

(X views – lists users and accounts the requestor owns or has modify rights to).

DBC.AccountInfo[V][X]

UserName	AccountName	UserOrProfile

Example:
Identify all users with a valid HIGH priority code.

```
SELECT    *
FROM      DBC.AccountInfoV
WHERE     AccountName LIKE '$H%'
ORDER BY  1 ;
```

Example Results:

UserName	AccountName	UserOrProfile
AP	$H_9038	User
Cust_Service_Gold	$H_&S_&D&H	Profile
Employee	$H_&S_&D&H	Profile
Students	$H_9038	User
Sysdba	$H_9038	User
TDPUSER	$H	User
tfact01	$H_9038_&S_&D&H	User
tfact07	$H_9038	User

AMPUsage View

The DBC.AmpUsage[V][X] views use the underlying DBC.Acctg table to provide accounting information by username and account. You can update this view. This view provides CPU activity and logical I/O counts explicitly requested by the following two sources:

- AMP database software
- File system that is running in the context of an AMP worker task

This view can be used to determine which user or users are consuming CPU and I/O resources on a system.

The system requests I/Os to execute a step in the user's query. The DBC.AmpUsage views do not include I/Os the operating system performs for swapping or I/Os caused by parsing the user's query. The system charges a logical I/O even if the segment you request is cached and no physical I/O is done.

Column definitions in this view include:

Column	Definition
Vproc	The virtual processor ID
VprocType	AMP
Model	System model (e.g., 5650, etc.)

AMPUsage View

AMPUsage views are updateable views that use the DBC.Acctg table to provide accounting information by username and account.

These views can be used to determine which users are consuming CPU and/or I/O resources.

AMPUsage will accumulate CPU and I/O usage for every unique account.

DBC.AMPUsage[V][X]

AccountName	UserName	CPUTime	DiskIO
Vproc	VprocType	Model	

CPUTime: Total number of AMP CPU seconds used (increments of 1/100 second).
DiskIO: Total number of logical disk I/O operations.
Vproc: AMP Vproc number
VprocType: AMP
Model: Model number (e.g., 5650)

AMPUsage View – Example

Example

The SQL statement on the facing page requests totals for CPU time and I/O for user TFACT03. The totals are aggregates of all resources used across all AMP vprocs. The result returns six rows, one for each unique account ID that has been expanded.

AMPUsage View – Example

Example:
Show CPU time and I/O totals for a single user.

```
SELECT      UserName          (CHAR(10))
           ,AccountName       (CHAR(25))
           ,SUM (CPUTime)     (FORMAT 'zzzz.99')
           ,SUM (DiskIO)      (FORMAT 'zzz,zzz,999')
FROM        DBC.AMPUsageV
WHERE       UserName = 'tfact03'
GROUP BY    1, 2
ORDER BY    3 DESC ;
```

Example Results:

UserName	AccountName	Sum(CPUTime)	Sum(DiskIO)
TFACT03	$M_9038	37,259.45	462,339,216
TFACT03	$M_9038_000051018_11091614	1,924.32	41,821,581
TFACT03	$M_9038_000051018_11091615	989.25	18,710,619
TFACT03	$M_9038_000051019_11091614	184.63	1,091,912
TFACT03	$L_9038_11091908	113.28	819,457
TFACT03	$L_9038_11091909	42.56	105,115

Note:
The Account IDs for TFACT03 are:
- $M_9038
- $M_9038_&S_&D&H
- $L_9038_&D&H

To reset counters for ALL rows or selected rows, you can use the DBC.ClearAccounting macro.

```
SHOW MACRO DBC.ClearAccounting;

REPLACE MACRO DBC.ClearAccounting
AS  ( UPDATE Acctg SET CPU = 0, IO = 0 ALL; );
```

Users, Accounts & Accounting Summary

The facing page summarizes some important concepts regarding this module.

Users, Accounts & Accounting Summary

- To establish execution time priorities, use the first two characters in the account code and the performance groups.
- Your position in the hierarchy does not affect your priority.
- You can define accounting mechanisms:
  - Charge-back billing
  - System usage reporting
  - Capacity planning
  - Performance analysis
- To reset data dictionary tables used to collect accounting information, use:
  - DBC.ClearAccounting macro
  - DBC.ClearPeakDisk macro

Module 44: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 44: Review Questions

1. True or False. You can only give the authority to use the CREATE DATABASE and CREATE USER statements to certain types of users.

2. True or False. An individual user with a $L priority will always receive less CPU time than a user with a $M priority.

3. True or False. A user can use the MODIFY USER statement to change their password, default database, and date format.

4. When creating a new user, which option defaults to the immediate owner's value. ___

   a. SPOOL
   b. FALLBACK PROTECTION
   d. All of the Account_IDs
   c. DEFAULT DATABASE

5. When creating a new user, which options are required with the CREATE USER command. ___

   a. SPOOL
   b. PERMANENT
   c. User name
   d. PASSWORD

Lab Exercise 44-1

The following pages describe the tasks for this lab exercise.

Lab Exercise 44-1

Lab Exercise 44-1

Purpose

In this lab, you will use Teradata SQL Assistant or Teradata Administrator to view information in the data dictionary regarding space usage and accounting information (use Appendix D).

Tasks

1. Using the DBC.DiskSpaceV view, find the total disk storage capacity of the system on which you are logged on:

   Total capacity _____

2. Using the same view, find how much of the space is currently in use:

   Current space utilization _____

   Write a query to show what percentage of system capacity is currently in use. _____%

   OPTIONAL: Write a query to show which databases/users are currently using (current perm) the largest percentage of their max perm space limit (group by database/user).

Lab Exercise 44-1 (cont.)

The following pages describe the tasks for this lab exercise.

Lab Exercise 44-1 (cont.)

3. **Using the DBC.DatabasesV view, find the total number of databases and users defined in the system.**

   Total row count (databases and users) _____

   Using this view, how many users are there? _____

   Using this view, how many databases are there ? _____

   Who is the creator of AP? _____

   Who is the owner of AP? _____

4. **Using the TableSizeV view, find the name and size of each table in the DBC user. List the Data Dictionary tables in DESCending order by size.**

   List the six largest tables:

   1. _____ **2.** _____ **3.** _____

   4. _____ **5.** _____ **6.** _____

Lab Exercise 44-1 (cont.)

The following pages describe the tasks for this lab exercise.

Lab Exercise 44-1 (cont.)

5. **Using the DBC.AMPUsageV view, find the number of AMP vprocs defined on your system. (HINT: Use a WHERE condition to reduce the number of DD/D table rows considered.)**

   Number of AMPS _____

6. **Using the DBC.AccountInfoV view, list all of your valid account codes.**

   _____ _____

7. **Using the DBC.AMPUsageV view, write a query to show the number of AMP CPU seconds and logical disk I/Os that have been charged to your:**

   User ID _____ **Seconds** _____ **I/Os** _____

Notes

Module 45

Profiles

After completing this module, you should be able to:

- **List two advantages of utilizing profiles.**
- **Use profiles when creating new users.**
- **Use system views to display profile information.**

Notes

Table of Contents

Profiles 45-4
Example of Simplifying User Management 45-6
Implementing Profiles 45-8
Impact of Profiles on Users 45-10
CREATE/MODIFY PROFILE Statement 45-12
Password Attributes (CREATE/MODIFY PROFILE) 45-14
Teradata Password Control 45-16
Teradata Password Control (cont.) 45-18
Teradata Password Control Options 45-20
CREATE PROFILE Example 45-22
Teradata Administrator CREATE PROFILE Example 45-24
CREATE PROFILE Example (cont.) 45-26
DROP PROFILE Statement 45-28
ProfileInfo View 45-30
Miscellaneous SQL Functions 45-32
Summary 45-34
Module 45: Review Questions 45-36
Lab Exercise 45-1 45-38

Profiles

Profiles define system attributes. By assigning a profile to a group of users, you can ensure that all group members operate with a common set of attributes.

To manage system attributes for groups, a database administrator can:

- Create a different profile for each user group, based on system attributes that group members share.

  You can define values for all or a subset of the parameters in a profile. If you do not set the value of a parameter, the system uses the setting defined for the user in a CREATE USER or MODIFY USER statement.

- Assign profiles to users.

  The parameter settings in a user profile override the settings for the user in a CREATE USER or MODIFY USER statement.

Like roles, the concept of ownership and ownership hierarchy is not be applicable to profiles.

Profiles

What is a “profile”?

- A profile is a set of common user attributes that can be applied to a group of users.
- Profile parameters include:
  - Account id(s)
  - Default database
  - Spool space allocation
  - Temporary space allocation
  - Password attributes (expiration, etc.)

What are advantages of using “profiles”?

- Profiles simplify user management.
  - A change of a common attribute requires an update of a profile instead of each individual user affected by the change.
  - Specify password security controls for groups of users.

How are “profiles” managed?

- New DDL commands, tables, view, command options, and access rights.
  - CREATE PROFILE, MODIFY PROFILE, DROP PROFILE, and SELECT PROFILE
  - New system table - DBC.Profiles
  - New system views - DBC.ProfileInfo[V][X]

Example of Simplifying User Management

The profile concept provides a solution to the following problem.

A customer has a group of 10,000 users that are assigned the same amount of spool space, the same default database, and the same account ID. Changing any of these parameters for 10,000 users is a very time-consuming task for the database administrators.

The database administrators' task will be simplified if they can create a profile that contains one or more system parameters such as accounts ids, default database, spool space and temporary space. This profile is assigned to the group of users.

This would simplify system administration because a parameter change requires updating only the profile instead of each individual user.

In summary, a set of parameters may be assigned certain values in a profile and this profile may be assigned to a group of users and thereby have them share the same settings. This makes changing parameters for a group of users a single step instead of a multi-step (one for each user in the group) process.

Example of Simplifying User Management

Example:

- **The problem:**
  - **A customer has a group of 10,000 users that are assigned the same spool space, the same default database, and the same account ID.**
  - **Changing any of these parameters for 10,000 users can be a time-consuming task.**

- **A solution using profiles:**
  - **Create a profile that contains these parameters and assign that profile to the users.**
  - **This would simplify system administration because a parameter change requires updating only the profile instead of each individual user.**

Implementing Profiles

The CREATE PROFILE and DROP PROFILE access rights are system rights. These rights are not on a specific database object. Note that the PROFILE privileges can only be granted to a user and not to a role or database.

Profiles enable you to manage the following common parameters:

- Account strings, including ASE codes and Performance Groups
- Default database
- Spool space
- Temporary space
- Password attributes, including:

  – Expiration
  – Composition (length, digits, and special characters)
  – Allowable logon attempts
  – Duration of user lockout (indefinite or elapsed time)
  – Reuse of passwords

Note: In the example on the facing page, another technique of granting CREATE PROFILE and DROP PROFILE to Sysdba is to use the following SQL.

GRANT PROFILE TO SYSDBA WITH GRANT OPTION;

The key word PROFILE will give both the CREATE PROFILE and DROP PROFILE access rights.

Implementing Profiles

What access rights are used to support profiles?

- CREATE PROFILE – needed to create new profiles
- DROP PROFILE – needed to modify and drop profiles

Who is allowed to create and modify profiles?

- Initially, only DBC has the CREATE PROFILE and DROP PROFILE access rights.
- As DBC, give the "profile" access rights to the database administrators (e.g, Sysdba).

```
GRANT CREATE PROFILE, DROP PROFILE TO Sysdba WITH GRANT OPTION;
```

How are users associated with a profile?

- The CREATE PROFILE command is used to create a profile of desired attributes.

```
CREATE PROFILE Employee_P AS … ;
```

- The PROFILE option is used with CREATE USER and MODIFY USER commands to assign a user to a specific profile.

```
CREATE USER Emp01 AS …, PROFILE = Employee_P;
MODIFY USER Emp02 AS PROFILE = Employee_P;
```

Impact of Profiles on Users

The assignment of a profile to a group of users is a way of ensuring that all members of a group operate with a common set of parameters. Therefore, the values in a profile always take precedence over values defined for a user via the CREATE and MODIFY USER statements.

All members inherit changed profile parameters. The impact is immediate, or in response to a SET SESSION statement, or upon next logon, depending on the parameter:

- SPOOL and TEMP space allocations are imposed immediately. This will affect the current session of any member who is logged on at the time his or her user definition is modified.
- Password attributes take effect upon next logon.
- Account IDs and a default database are considered at next logon unless the member submits a SET SESSION ACCOUNT statement, in which case the account ID must agree with the assigned profile definition.

Order of Precedence

With profiles, there are 3 ways of setting accounts and default database. The order of precedence (from high to low) is as follows:

1. The DATABASE statement is used to set the current default database or the SET SESSION ACCOUNT is used to set the account ID. However, a user can only specify a valid account ID.
2. Specify them in a profile and assign the profile to a user.
3. Specify accounts or default database for a user through the CREATE USER/MODIFY USER statements.

Impact of Profiles on Users

The assignment of a profile to a group of users is a way of ensuring that all members of a group operate with a common set of parameters.

Profile definitions apply to every assigned user, overriding specifications at the system or user level.

- **However, any profile definition can be NULL or NONE.**

All members inherit changed profile parameters. The impact on current users is as follows:

- **SPOOL and TEMPORARY space allocations are imposed immediately.**
- **Password attributes take effect upon next logon.**
- **Database and Account IDs are considered at next logon unless the member submits a SET SESSION ACCOUNT statement.**

Order of Precedence for parameters:

1. **Specify database or account ID at session level**
2. **Specified parameters in a Profile**
3. **CREATE USER or MODIFY USER statements**

CREATE/MODIFY PROFILE Statement

The CREATE PROFILE statement enables you to add new profiles to the system. The CREATE PROFILE access right is required in order to execute this command. The syntax is shown on the facing page.

Profile names come from their own name space. Like roles, the concept of ownership and ownership hierarchy is not applicable to profiles.

A parameter not set in a profile will have a value of NULL. Resetting a parameter to NULL will cause the system to apply the user's setting instead. In a profile, the SPOOL and TEMPORARY limits may not exceed the current space limits of the user submitting the CREATE/MODIFY PROFILE statement.

The default database specified in a profile need not refer to an existing database. This is consistent with current CREATE USER and MODIFY USER statements where a non-existent default database may be specified. An error will be returned when the user tries to create an object within the non-existent database.

It is not necessary to define all of the parameters in a profile, a subset will also do. The parameter values in a user profile take precedence over the values set for the user. For example, if a user is assigned a profile containing Default Database and Spool Space, the profile settings will override the individual settings previously made via a CREATE USER or MODIFY USER statement.

Accounts in a profile will also override, not supplement, any other accounts the user may have. The assignment of a profile to a group of users is a way of ensuring that all group members operate with a common set of parameters. The first account in a list will be the default account.

If a parameter in a profile is not set, then the user's setting will be applied.

Note when using the CREATE USER command:

- When creating a new user, if the PROFILE option specifies a Profile that does not exist, you will get the following error.

  Error 5653: Profile 'profile_name' does not exist.

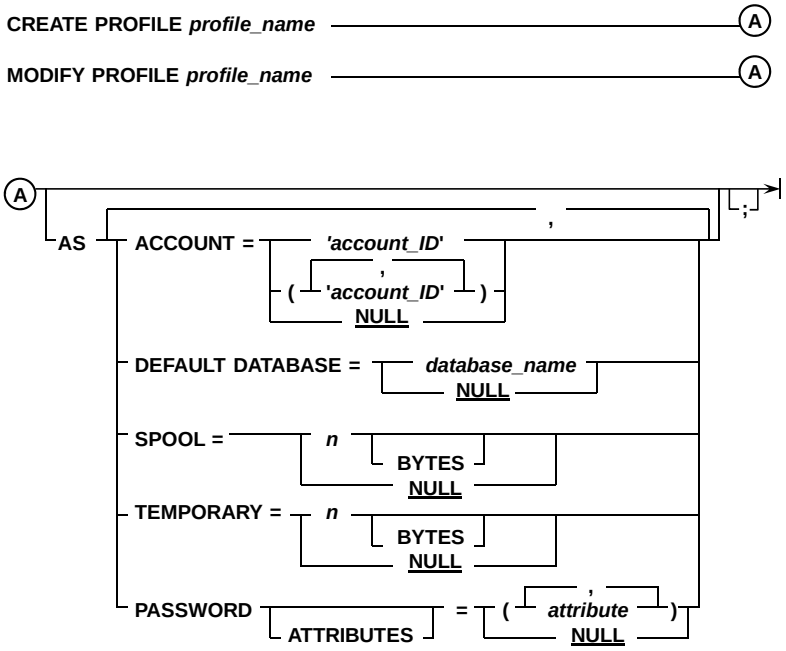
Modify Profile Statement

The MODIFY PROFILE statement enables you to change the options of an existing profile. The DROP PROFILE access right is required in order to execute this command. The syntax is similar to CREATE PROFILE and is also shown on the facing page.

To remove a profile from a user,

MODIFY USER *username* AS PROFILE = NULL;

CREATE/MODIFY PROFILE Statement



Password Attributes (CREATE/MODIFY PROFILE)

The facing page describes the Password Attributes associated with the CREATE PROFILE and MODIFY PROFILE commands.

If a parameter is not specified in a profile and is not specified with the CREATE USER or MODIFY USER statement, the following list of defaults applies.

Parameter	Default Value
Account ID	The default account ID (first account ID of the immediate owner of the user).
Performance Group	$M
DEFAULT DATABASE	Username
SPOOL	The same SPOOL value as the owner of the space in which the user is being created.
TEMPORARY	The same TEMPORARY value as the owner of the space in which the user is being created.

If the password attributes option is not specified in a profile, then the password attributes specified in the DBC.SysSecDefaults table are used for the users assigned to this profile.

Password Attributes
(CREATE/MODIFY PROFILE)

Password Attributes

EXPIRE	=	*n* / NULL	(# of days; 0 doesn't expire)
MINCHAR	=	*n* / NULL	(range is 1 - 30)
MAXCHAR	=	*n* / NULL	(range is 1 - 30)
DIGITS	=	*c* / NULL	(options are Y, y, N, n, R, r)
RESTRICTWORDS	=	*c* / NULL	(options are Y, y, N, n)
SPECCHAR	=	*c* / NULL	(options are Y, y, N, n, ...)
MAXLOGONATTEMPTS	=	*n* / NULL	(# of attempts; 0 = never locked)
LOCKEDUSEREXPIRE	=	*n* / NULL	(# of minutes; 0 = not locked; -1 = locked indefinitely)
REUSE	=	*n* / NULL	(# of days; 0 - reuse immediately)

Teradata Password Control

Forcing users to create passwords with one or more of the special character options enhances password security. It also may make the password harder for the user to remember and to type in at logon. Consider these two factors when deciding how elaborate the password special character requirements should be for your system

Many passwords would be relatively easy for an intruder to guess, especially if some of the letters are known. Forcing users to create passwords with one or more digits enhances password security.

When specifying the maximum password length, keep in mind that some users may try to create a password of maximum length. Because it is more difficult to remember a long password, the user is more likely to write it down rather than memorize it – and it is strongly recommended that users do not write passwords down.

Adding and Removing Restricted Words

The PasswordRestrictWords (DBC.SysSecDefaults) or RestrictWords (profiles) parameter determines whether or not a password is subject to the content restrictions defined in the DBC.PasswordRestrictions list. A default set of Restricted Words is automatically installed when a system is upgraded to Teradata Database 12.0 or greater.

You can add words to the Restricted Words list, using the following form:

```
INSERT INTO DBC.PasswordRestrictions
          VALUES ('newrestrictedword');
```

Note: Although the default Restricted Words list is composed of English words, the words you add can be in any supported character set.

You can also remove words from the list using the following form:

```
DELETE FROM DBC.PasswordRestrictions WHERE
          (RestrictedWords = 'wordtobedeleted');
```

The DBC.RestrictedWords (or DBC.RestrictedWordsV) views can be used to view restricted words.

Teradata Password Control

Description

- At the system level, the DBC.SysSecDefaults table has password parameters to control system-wide password attributes. Similar parameters are available at the profile level to establish password attributes for a group of users.
- The DBS Password Control feature provides additional requirements on valid Teradata user passwords.
  - These requirements, which mainly consist of enforcing character variation within a password string, can be enabled or disabled by the DBA/Security Administrator on a system/user basis.
- Starting with Teradata 12.0, you can specify whether or not a password is subject to the content restrictions defined in the table DBC.PasswordRestrictions.

Customer Benefit

- These new features allow for the requirement and enforcement of stronger passwords.
- Many passwords would be relatively easy for an intruder to guess, especially if they contain common words or names. Forcing users to create passwords that do not use common words or names enhances password security.

Teradata Password Control (cont.)

The PasswordDigits or Digits parameter determines if digits may be used in a password.

The default value for the PasswordDigits or the Digits parameter is **Y**: Digits are allowed in a password.

The acceptable values for the PasswordDigits (DBC.SysSecDefaults – table or DBC.SecurityDefaults - view) or Digits (Profile) parameter are:

- **Y** = Digits are allowed
- **N** = Digits are not allowed
- **R** = At least one digit is required

Note: The values are not case sensitive.

Password Special Characters

One of the key password parameters is "PasswordSpecChar" or "SpecChar". This parameter determines how ASCII special characters can be used in a password. It includes the following options:

- special characters are allowed/not allowed/required
- passwords must contain at least one alpha character
- no password can contain the database username
- passwords must contain a mixture of upper/lower case letters

The default value of this parameter is **Y**:

- special characters are allowed in a password
- username is allowed in the password string
- alpha characters are allowed but not required
- mixed upper and lower case characters are allowed but not required

Teradata Password Control (cont.)

Options

- The PasswordSpecChar (or SpecChar) parameter is used to establish the following password control rules.
  - Do not allow a password to contain the user name
  - Allow or require a mixture of upper/lower case characters
  - Allow or require at least one alpha character
  - Allow, not allow, or require at least one special character
- The PasswordDigits (or Digits) parameter is used to allow, not allow, or require a numeric digit in the password.
- The PasswordRestrictWords (or RestrictWords) parameter is used to indicate that a password cannot contain one of the "restricted words".
- Note: The DBC.SecurityDefaultsV (view) can be used to view/update DBC.SysSecDefaults.

Teradata Password Control Options

The PasswordSpecChar parameter (DBC.SysSecDefaults) or SPECCHAR (Profile) determines how special characters can be used in a password. It includes the following options:

- special characters are allowed/not allowed /required
- passwords must contain at least one alpha character
- no password can contain the database username
- passwords must contain a mixture of upper/lower case letters

The default value of the PasswordSpecChar parameter is **Y**:

- special characters are allowed in a password
- username is allowed in the password string
- alpha characters are allowed but not required
- mixed upper and lower case characters are allowed but not required

A Password *can* contain ...

- 1 to 30 characters (UTF-8 *or* UTF-16 characters)
- Letters **A** through **Z** and/or **a** through **z**
- Digits **0** through **9** in single-byte or multi-byte form
- **Note:** A password can be all-numeric only if it is enclosed in quotes as shown in the following example: password = “12341234”
- The following special characters, in either single-byte or multi-byte form:
- **$** (dollar sign)
- **_** (underscore)
- **#** (pound sign)
- Other special characters may be used (if they are not specifically prohibited by the rules in the reference manuals) *and* if the password is enclosed in quotes (“ ”).

Teradata Password Control Options

Options Table for PasswordSpecChar and SpecChar

Option PasswordSpecChar or SpecChar	N	Y	A	B	C	D	E	F	G	H	I	J	K	L	M	O	P	R
Rule Username	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N
Rule Upper/Lower	Y	Y	Y	Y	Y	Y	R	R	R	Y	Y	Y	Y	Y	Y	R	R	R
Rule One Alpha	Y	Y	Y	R	R	R	R	R	R	Y	Y	Y	R	R	R	R	R	R
Rule Special Chars.	N	Y	R	N	Y	R	N	Y	R	N	Y	R	N	Y	R	N	Y	R

N – Not Allowed, Y – Allowed, but not required, R – Required
Note: These options are not case sensitive. You can use "A" or "a".

CREATE PROFILE Example

The facing page contains a simple example of creating a profile, assigning it to a user, and then removing it from the user with the MODIFY USER command.

As mentioned previously, profile definitions apply to every assigned user, overriding specifications at the system or user level.

Answers to first set of two questions on facing page:

```
1E9
$M0_EDUC&S&D&H
```

Answers to second set of three questions on facing page:

```
2E9
$M0_EDUC&S&D&H
$M_&S&D&H
```

CREATE PROFILE Example

Create a profile.

```
CREATE PROFILE Employee_P AS
    ACCOUNT                 = ('$M0_EDUC&S&D&H', '$L0_EDUC&S&D&H'),
    DEFAULT DATABASE        = HR_VM,
    SPOOL                   = 1E9,
    TEMPORARY               = 500E6,
    PASSWORD                = (EXPIRE = 90,  MINCHAR = 8, MAXLOGONATTEMPTS = 3,
                               LOCKEDUSEREXPIRE = 60, REUSE = 180,
                               DIGITS = 'R', RESTRICTWORDS = 'Y', SPECCHAR = 'P');
```

Assign the profile to a user.

```
CREATE USER Emp01 AS        PERM = 0,
                            PASSWORD = emp01pass,
                            PROFILE = Employee_P,
                            SPOOL = 2E9,
                            ACCOUNT = '$M_&S&D&DH';
```

What is the spool space limit for Emp01?
What is the default account code for Emp01?

Assume this command is executed: `MODIFY USER Emp01 AS PROFILE = NULL;`

What is the spool space limit for Emp01?
What is the default account code for Emp01 for the current session?
What is the default account code for Emp01 for a new session?

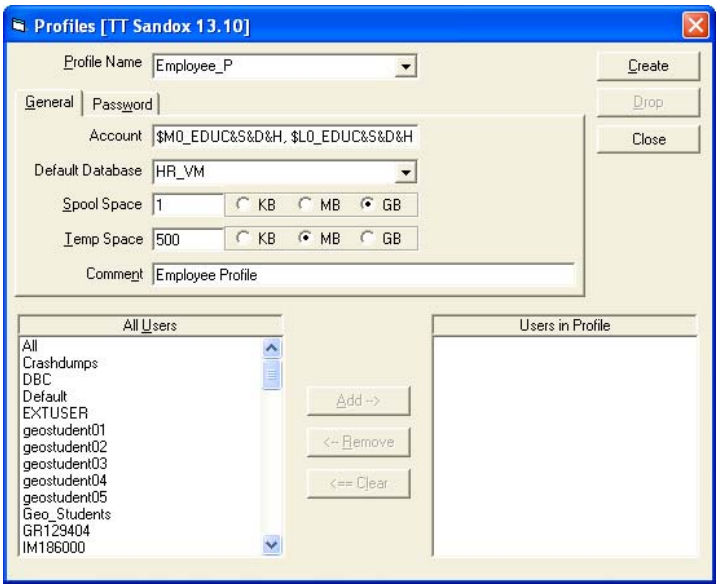
Teradata Administrator CREATE PROFILE Example

The facing page contains an example of creating a profile using Teradata Administrator.

Teradata Administrator

CREATE PROFILE Example

From Teradata Administrator, use the Tools > Administer Profiles menus.



These 2 options are equivalent to:

SPOOL=1E9,
TEMPORARY=500E6

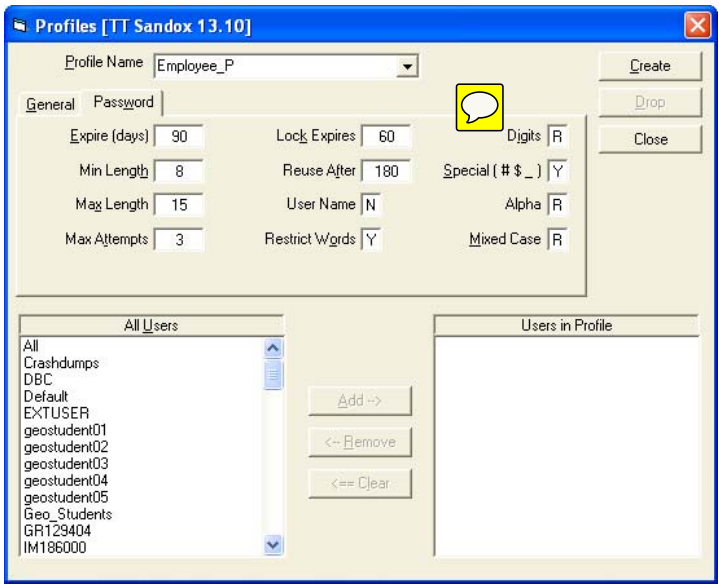
```
CREATE PROFILE Employee_P AS
        ACCOUNT = ('$M0_EDUC&S&D&H', '$L0_EDUC&S&D&H')
        DEFAULT DATABASE = HR_VM, SPOOL = 1E9, TEMPORARY = 500E6, …
```

CREATE PROFILE Example (cont.)

The facing page continues the example of creating a profile using Teradata Administrator.

CREATE PROFILE Example (cont.)

From Teradata Administrator, use the Tools > Administer Profiles menus.



These options are equivalent to:

DIGITS='R', RESTRICTWORDS='Y', SPECCHAR='P'

```
CREATE PROFILE Employee_P ...
PASSWORD = (EXPIRE=90, MINCHAR=8, MAXCHAR=15, MAXLOGONATTEMPTS=3,
      LOCKEDUSEREXPIRE=60, REUSE=180, DIGITS='R', RESTRICTWORDS='Y', SPECCHAR='P');
```

DROP PROFILE Statement

The DROP PROFILE statement drops the named profile. The DROP PROFILE access right is required in order to execute this command.

The syntax is simply:

DROP PROFILE *profile_name*;

When a profile is dropped, users who have the profile assigned to them continue to have that profile assigned to them; the system does *not* reset the profile for the affected users to NULL. Affected users receive no warnings or errors the next time they log on.

The effects of re-creating a profile with the same name as the dropped profile are not immediate. The parameter settings in the re-created profile take effect the next time that users (who are assigned the profile) log on.

DROP PROFILE has the following effects on users (sessions) logged on with the profile:

- Spool and temporary space settings immediately change to the settings defined for the affected users.
- Account and database settings change to the settings defined for the affected users the next time the users log on or explicitly change the settings.

However, changes to the list of valid account IDs take effect immediately. Users may only explicitly change to an account ID in the list of account IDs available to them.

DROP PROFILE Statement

Syntax:

DROP PROFILE *profile_name*;

Notes:

- **If a profile is dropped, users who have the profile assigned to them continue to have that profile assigned to them.**
- **Effects on sessions logged on with the profile that has been dropped:**
  - **Spool and temporary space settings immediately change user's settings.**
  - **Account and database settings change to user's settings on next log on or if user explicitly changes the settings.**
  - **Users may only explicitly change to an account in the list of account IDs available to them.**
- **The effects of re-creating a profile with the same name as the dropped profile are not immediate.**
  - **The parameter settings in the re-created profile take effect the next time the users log on.**

ProfileInfo View

The DBC.ProfileInfo view will list all profiles and their parameter settings. This information is taken from the DBC.Profiles table.

The DBC.ProfileInfoX view will list the profile, if any, and its parameter settings for the current user.

Extension to COMMENT command:

COMMENT [ON] PROFILE <profile name> [[AS] <comment string>]

– inserts or retrieves comments in CommentString column of the DBC.Profiles table for the named profile.

Implementation Note:

Like accounts for the DBC.Dbase table, only the default account is stored in the DBC.Profiles table. All other accounts associated with the profile will be stored in DBC.Accounts table. The ProfileInfoV view provides the first or default account ID.

ProfileInfo View

Provides information about profiles that exist in the system.

DBC.ProfileInfo[V][X]

ProfileName	DefaultAccount	DefaultDB
SpoolSpace	TempSpace	ExpirePassword
PasswordMinChar	PasswordMaxChar	PasswordDigits
PasswordSpecChar	PasswordRestrictWords	MaxLogonAttempts
LockedUserExpire	PasswordReuse	CommentString
CreatorName	CreateTimeStamp	LastAlterName
LastAlterTimeStamp		

Example:

List profiles that exist in the system.

```
SELECT    ProfileName
         ,DefaultAccount AS "Def Acct"
         ,DefaultDB
         ,SpoolSpace
         ,TempSpace
FROM      DBC.ProfileInfoV
ORDER BY  1;
```

Example Results:

ProfileName	Def Acct	DefaultDB	SpoolSpace	TempSpace
Cust_Service_P	$M_&S&D&H	CS_VM	200000000	100000000
Cust_Gold_P	$H_&S&D&H	CS_VM	200000000	100000000
Employee_P	$M0_EDUC&S&D&H	HR_VM	1000000000	500000000
Payroll_P	$M_&S&D&H	Payroll_VM	200000000	100000000

Miscellaneous SQL Functions

As you may notice, the DBC.AccountInfoV view has a column named UserOrProfile. This corresponds to the column named RowType in the DBC.Accounts table.

The RowType column is necessary because profile and user names come from separate name spaces. Since profile accounts are also stored in DBC.Accounts, they will be confused with accounts of a user with the same name. Hence, the RowType column is necessary to distinguish a user account from a profile account. This column will have a value of P (for Profile) or U (for User).

The facing page contains 3 simple examples of SQL functions that a user can execute to identify their user, profile, role, or current database information.

Miscellaneous notes:

- If you are accessing the Teradata Database through a proxy connection, CURRENT_USER returns the proxy user name. Otherwise, it functions exactly like the USER built-in function and returns the session user name.

- If you are accessing the Teradata Database through a proxy connection, then CURRENT_ROLE returns the current role of the proxy user. If you are not accessing the Teradata Database through a proxy connection, CURRENT_ROLE functions exactly like the ROLE built-in function and returns the session current role, which is the current role of the session user.

- CURRENT_ROLE is not supported in the FastLoad and MultiLoad utilities.

Miscellaneous SQL Functions

Example 1: As Emp01, identify the current user, role, profile, and database information.

```
SELECT USER, ROLE, PROFILE, DATABASE;
```

Result 1:

User	Role	Profile	Database
EMP01	HR_R	EMPLOYEE_P	HR_VM

Example 2: As Emp01, list the profile attributes.

```
SELECT * FROM DBC.ProfileInfoVX;
```

Result 2:

ProfileName	DefaultAccount	DefaultDB	SpoolSpace	TempSpace
Employee_P	$M0_EDUC&S&D&H	HR_VM	1000000000	500000000

Example 3: As Emp01, list account information.

```
SELECT * FROM DBC.AccountInfoVX;
```

Result 3:

Name	AccountName	UserOrProfile
Employee_P	$M0_EDUC&S&D&H	Profile
Employee_P	$L0_EDUC&S&D&H	Profile
Emp01	$M_&S&D&H	User

Summary

The facing page summarizes some important concepts regarding this module.

Summary

- A profile is a set of common user parameters that can be applied to a group of users.
- The CREATE PROFILE command is used to create a profile of desired attributes.
  - CREATE PROFILE *profile_name* AS … ;
- The PROFILE option (new) is used with CREATE USER and MODIFY USER commands to assign a user to a specific profile.
  - CREATE USER *user1* AS …, PROFILE = *prof_name*;
  - MODIFY USER *user2* AS PROFILE = *prof_name*;

Module 45: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 45: Review Questions

Answer the following questions:

1. List 2 advantages of utilizing profiles.

   _____

   _____

2. If a user's profile is set to NULL, which two are immediately affected in the current session.

   a. SPOOL value
   b. Session priority
   c. Default database
   d. TEMPORARY value

Match each term to the definition.

___	1. CREATE PROFILE	a.	Lists all words that can not be included in passwords
___	2. DBC.ProfileInfoV	b.	System access right needed to create a profile
___	3. DBC.PasswordRestrictions	c.	Lists the profiles currently in the system

Lab Exercise 45-1

The following page continues this lab exercise.

Lab Exercise 45-1

Lab Exercise 45-1

Purpose

In this lab, you will use Teradata SQL Assistant or Teradata Administrator to create a profile and multiple users. This lab will also provide an opportunity to use the DBC.ProfileInfoV view.

What you need

Tables from PPI exercise and a user account with system profile privileges.

Tasks

1. Create a user profile with a profile name that is the same as your user name (e.g., "studentxxx_P" where xxx is your student number). The attributes of your profile are:

```
ACCOUNT            = '$M0+FACT&S&D&H',
DEFAULT DATABASE   = studentxxx (i.e., your database),
SPOOL              = 50E6,
TEMPORARY          = 50E6,
PASSWORD           = (EXPIRE = 91,  MINCHAR = 6, MAXLOGONATTEMPTS = 3,
                      LOCKEDUSEREXPIRE = 5, REUSE = 365,
                      DIGITS='R', RESTRICTWORDS='Y', SPECCHAR='P');
```

2. Use the DBC.ProfileInfoV view to display information about profiles in the system.

   How many profiles are defined in the system? _____

Lab Exercise 45-1 (cont.)

The following page continues this lab exercise.

Lab Exercise 45-1 (cont.)

3. Create two new users in the system with the following attributes.

   User name: studentxxx_A (where xxx is your student number)
   Perm space: 0
   Password: studentxxxA
   Profile: studentxxx_P

   User name: studentxxx_B (where xxx is your student number)
   Perm space: 0
   Password: studentxxxB
   Profile: studentxxx_P

4. Logon to Teradata as "studentxxx_A".

   Were you prompted to enter a new password? If so, set the password to a new value.

   Why were you prompted to enter a new password? _____

Notes

TERADATA
Raising Intelligence

Access Rights

After completing this module, you will be able to:

- **Use the DBC.AllRights, DBC.UserRights and DBC.UserGrantedRights views to obtain information about current users.**
- **Use views and macros to access information about privileges.**
- **Use the GRANT and REVOKE statements to assign and remove access rights.**
- **Understand the impact of the GIVE statement with access rights.**

Notes

Table of Contents

Privileges/Access Rights 46-4
Access Rights Mechanisms 46-6
    Automatic Rights 46-6
    Explicit Rights 46-6
    Ownership Rights 46-6
    Access Rights Views 46-6
CREATE TABLE – Automatic Rights 46-8
CREATE USER – Automatic Rights 46-10
    Example 46-10
Implicit, Automatic, and Explicit Rights 46-12
    Example 46-12
GRANT Command 46-14
  GRANT PUBLIC Implementation Change 46-14
Granting Rights at Database Level 46-16
GRANT Rights at the Table or Column Level 46-18
  Access Rights and Triggers 46-18
REVOKE Command 46-20
    REVOKE Recipients 46-20
Revoking Non-Existent Rights 46-22
    Example 46-22
Removing a Level in the Hierarchy 46-24
    Transfer Ownership 46-24
    Delete User or Database 46-24
    Drop User 46-24
    Access Rights 46-24
Inheriting Access Rights 46-26
    Example 46-26
The GIVE Statement and Access Rights 46-28
    Example 46-28
Access Rights and Views 46-30
Access Rights and Nested Views 46-32
System Views for Access Rights 46-34
    DBC.AllRights[V][X] 46-34
    DBC.UserRights[V] 46-34
    DBC.UserGrantedRights[V] 46-34
AllRights and UserRights Views 46-36
UserGrantedRights View 46-38
Teradata Administrator – Grant/Revoke Rights 46-40
Teradata Administrator – Rights on DB/User 46-42
Access Rights Summary 46-44
Module 46: Review Questions 46-46

Privileges/Access Rights

Your privileges (access rights) define the types of activities you can perform during a session. The following operations are examples that require you to have specific privileges:

- CREATE
- DROP
- REFERENCES
- INDEX

} DDL

- SELECT
- UPDATE
- INSERT
- DELETE
- EXECUTE
- EXECUTE PROCEDURE

} DML

- CHECKPOINT
- DUMP
- RESTORE

} Archive/Recovery

Examples of objects that privileges (access rights) are associated with include:

Users	Macros	Columns of tables
Databases	Triggers	Columns of views
Tables	Stored Procedures	
Views	User-defined Functions	

Notes:

- To use UPDATE or DELETE commands, you must also have the SELECT right on the object.

- Additional rights you need to control access to performance monitoring functions are discussed in another module.

- A column can only be specified with the SELECT (Teradata 13.0), INSERT (Teradata 13.0), UPDATE or REFERENCES access right.

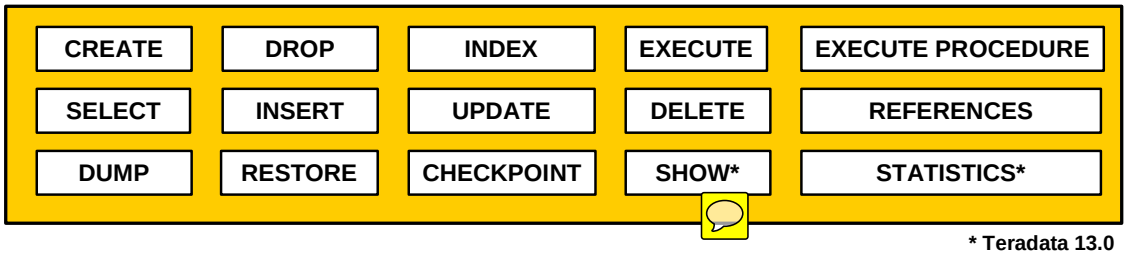
- SHOW privilege (Teradata 13.0) – this privilege enables you to have access to database object definitions and create text without having access to the data contained by the objects on which the privilege is granted. For example, SHOW permits a user to execute HELP and SHOW requests against an object while at the same time not being able to SELECT from it.

- STATISTICS privilege (Teradata 13.0) – allows a user to collect or drop statistics on an object (e.g., table). INDEX and DROP TABLE can still be used, but STATISTICS does not grant users the wider capabilities associated with those privileges. STATISTICS can be granted at both the table and database levels.

Privileges/Access Rights

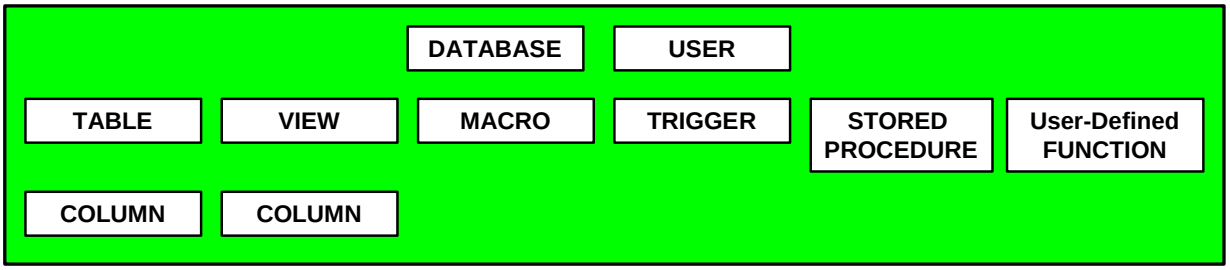
A privilege (or access right) is the right of a specific user to perform a specified operation.

Note: Some access rights don't directly correspond to an SQL statement.



*** Teradata 13.0**

On a specified <u>Object</u>



Access Rights Mechanisms

The data dictionary includes a system table called DBC.AccessRights that contains information about the access rights assigned to existing users.

The DBC.AccessRights table internally has the following indexes:

PI – NUPI (UserId, DatabaseId)
SI – NUSI (TVMId)

Access rights may be categorized in one three ways:

- Automatic (or Default) Access Rights
- Explicit Access Rights
- Implicit (or Ownership) Access Rights

Automatic Rights

Automatic rights are privileges given to creators and, in the case of users and databases, their created objects. When a user submits a CREATE statement, new rows are inserted in the DBC.AccessRights table. All rights are automatically removed for an object when it is dropped.

Explicit Rights

Explicit rights are privileges conferred by using a GRANT statement. This statement inserts new rows into the DBC.AccessRights table. Explicit rights can be removed using the REVOKE statement.

Ownership Rights

Owners (Parents) have the implicit right to grant rights on any or all of their owned objects (Children), either to themselves or to any other user or database. If an owner grants him or herself rights over any owned object, the parser will validate that GRANT statement even though the owner holds no other privileges.

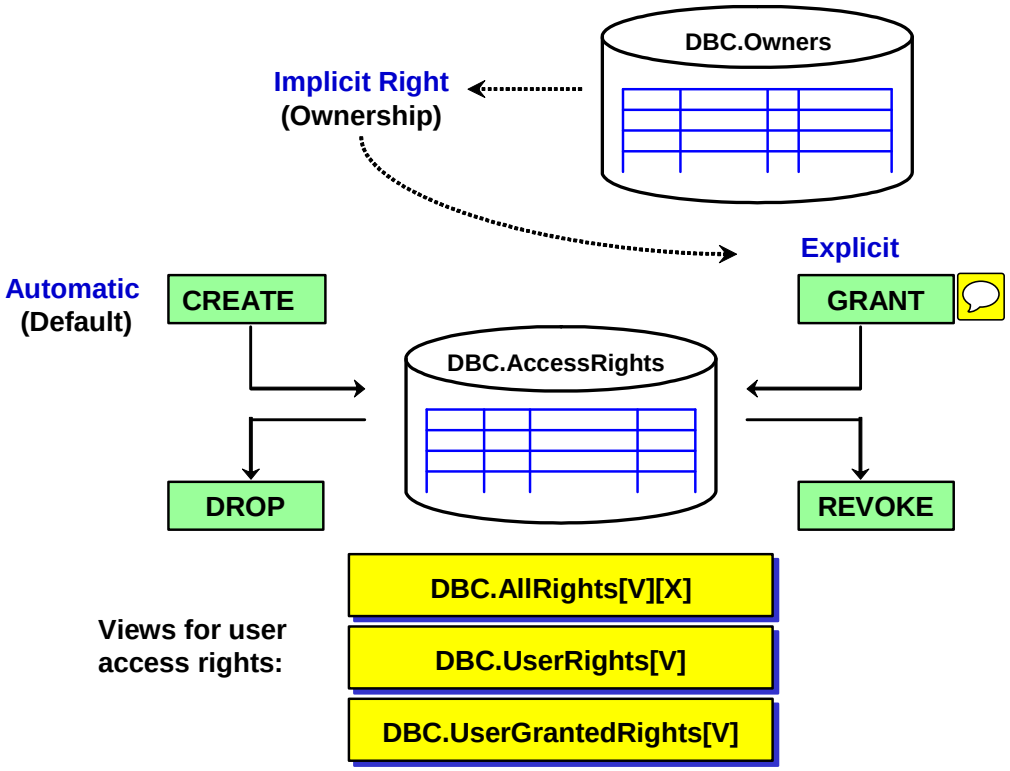
Ownership rights cannot be taken away unless ownership is transferred.

Access Rights Views

The data dictionary contains views that return information about access rights:

- DBC.AllRights[V][X] and DBC.UserRights[V]
- DBC.AllRoleRights[V][X] and DBC.UserRoleRights[V][X]
- DBC.UserGrantedRights[V][X]

Access Rights Mechanisms



CREATE TABLE – Automatic Rights

The SQL request is preceded by the modifier EXPLAIN. As a result, the parser prints out the AMP steps (in simple English) that the CREATE statement generates.

In step 4 (parallel step 11 on the facing page), you can see how the system adds each access right to the AccessRights table.

The following access rights are inserted; each with the grant authority:

- SELECT (R)
- INSERT (I)
- UPDATE (U)
- DELETE (D)
- DROP TABLE (DT)
- INDEX (IX)
- REFERENCES (RF)
- CREATE TRIGGER (CG)
- DROP TRIGGER (DG)
- DUMP (DP)
- RESTORE (RS)
- STATISTICS (ST) - new with Teradata 13.0

If a view is created, 5 access rights are added.

Creation of a macro causes 2 access rights to be added.

CREATE TABLE – Automatic Rights

```
EXPLAIN      CREATE TABLE TFACT.Customer
                  (Customer_Number   INTEGER,     Last_Name        CHAR(30),
                   First_Name        CHAR(20),    Social_Security  INTEGER)
             UNIQUE PRIMARY INDEX (Customer_Number)
             UNIQUE INDEX (Social_Security);
```

1) First, we lock TFACT.Customer for exclusive use.
2) Next, we lock a distinct DBC."pseudo table" for write on a RowHash for deadlock prevention, we lock a distinct DBC."pseudo table" for write on a RowHash for deadlock prevention, we lock a distinct DBC."pseudo table" for write on a RowHash for deadlock prevention, and we lock a distinct DBC."pseudo table" for read on a RowHash for deadlock prevention.
3) We lock DBC.ArchiveLoggingObjsTbl for read on a RowHash, we lock DBC.TVM for write on a RowHash, we lock DBC.TVFields for write on a RowHash, we lock DBC.Indexes for write on a RowHash, we lock DBC.DBase for read on a RowHash, and we lock DBC.AccessRights forwrite on a RowHash.
4) We execute the following steps in parallel.
    1) We do a single-AMP ABORT test from DBC.ArchiveLoggingObjsTbl by way of the primary index.
    2) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.
    3) We do a single-AMP ABORT test from DBC.TVM by way of the unique primary index.
    4) We do an INSERT into DBC.TVFields (no lock required).
    : :
    8) We do an INSERT into DBC.Indexes (no lock required).
    9) We do an INSERT into DBC.Indexes (no lock required).
    10) We do an INSERT into DBC.TVM (no lock required).
    11) We INSERT default rights to DBC.AccessRights for TFACT.Customer.
5) We create the table header.
6) We create the index subtable on TFACT.Customer.
7) We modify the table header TFACT.Customer.
8) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> No rows are returned to the user as the result of statement 1.

CREATE USER – Automatic Rights

When you create a new user or database, the system automatically generates access rights for the created object and the creator of the object. The system inserts this rights information into the DBC.AccessRights table when you submit a CREATE request. You can remove these rights from the DBC.AccessRights table with the REVOKE statement.

Example

In the example on the facing page, user SYSDBA logs on to the system and creates a new user called Accounting. Both SYSDBA and Accounting have the following privileges written into the DBC.AccessRights table:

CREATE TABLE	DROP TABLE
CREATE VIEW	DROP VIEW
CREATE MACRO	DROP MACRO
CREATE TRIGGER	DROP TRIGGER
SELECT	INSERT
UPDATE	DELETE
EXECUTE	DROP PROCEDURE
CHECKPOINT	RESTORE
DUMP	DROP FUNCTION
CREATE AUTHORIZATION	DROP AUTHORIZATION
STATISTICS	

Note: CREATE and DROP AUTHORIZATION access rights are new with Teradata V2R6.1

In addition, user SYSDBA has the following rights over Accounting as its creator:

- CREATE Database/User
- DROP Database/User

CREATE USER – Automatic Rights

By issuing a CREATE USER statement, the CREATOR causes Automatic rights to be generated for both the created user and the creator.

SYSDBA → Accounting SYSDBA creates a new user named Accounting.

Both SYSDBA and Accounting are given the following rights over Accounting:

CREATE Table	DROP Table	CREATE View	DROP View
CREATE Macro	DROP Macro	CREATE Trigger	DROP Trigger
SELECT	INSERT	UPDATE	DELETE
EXECUTE	DROP Procedure	DROP Function	DUMP
RESTORE	CHECKPOINT	CREATE Authorization	DROP Authorization
STATISTICS*	* 13.0		

SYSDBA is given the following additional rights over Accounting:

CREATE Database	DROP Database	CREATE User	DROP User

Implicit, Automatic, and Explicit Rights

Implicit rights belong to the owners of objects. Owners do not require rows in the DBC.AccessRights table to grant privileges on owned objects. Ownership rights cannot be "revoked." An owner has the implicit right to GRANT privileges over any owned object.

When you submit a CREATE statement, the system automatically adds new rows to the DBC.AccessRights table. You can remove automatic rights with the REVOKE or DROP statements.

GRANT and REVOKE statements control explicit rights. The GRANT statement adds new rows to the DBC.AccessRights table. The REVOKE statement removes them.

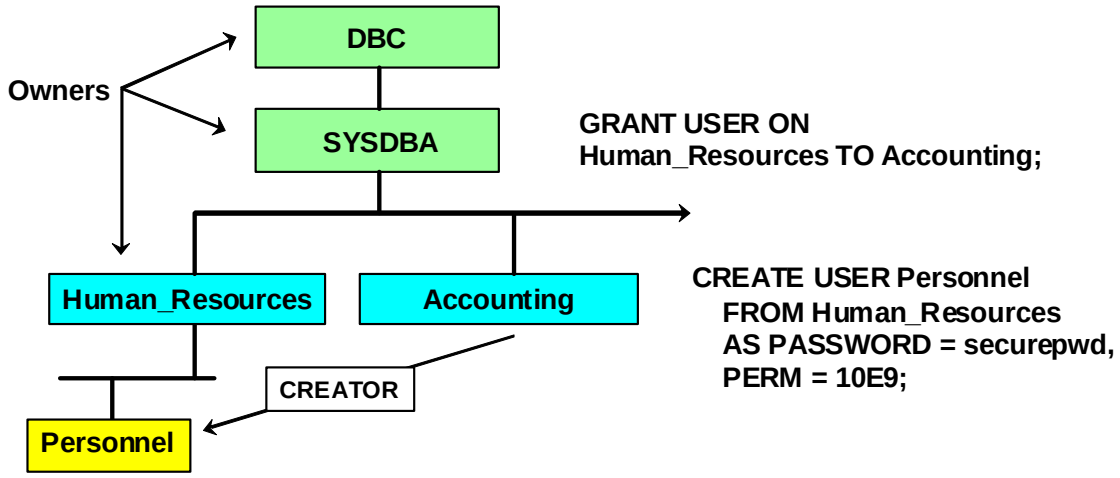
Example

In the example, Accounting is the creator. The system automatically inserts rows for access rights in DBC.AccessRights for the creator (Accounting) and for the created user (Personnel). These rights can be revoked.

The user named Personnel is the created object. The database Personnel automatically receives all but four access rights on itself. These rights are inserted *automatically* in DBC.AccessRights. These rights can be revoked.

The user named Human_Resources is the immediate owner. The system does not insert any rows in the Data Dictionary for Human Resources. However, Human_Resources has the owner's implicit right to grant itself rights over Personnel. You cannot revoke the right to GRANT (or re-GRANT) rights over owned objects.

Implicit, Automatic, and Explicit Rights



How many automatic access rights are created for Personnel?

How many automatic access rights are created for Human_Resources?

How many automatic access rights are created for Accounting?

GRANT Command

You can use the GRANT statement to give to users, databases, or roles one or more privileges on a database, user, table, view, macro, trigger, stored procedure, or user-defined function.

To grant a privilege, you must:

- Have the privilege itself and have GRANT authority
  OR
- Be an owner

The recipient of an explicitly granted privilege may be:

• Username	The specific user(s) or database(s) named
• PUBLIC	Every user in the DBC system (same as ALL DBC)
• ALL username	The named user and ALL descendants
• Role	Specified role or roles

Access rights that a new user inherits because the ALL or PUBLIC option is used are referred to as “inherited rights”.

The **WITH GRANT OPTION** confers on the recipient “Grant Authority”. The recipient (or “Grantee”), holding this authority, may then grant the access right to other users, databases, or roles.

Syntax for REFERENCES or UPDATE access right for a column:

```
GRANT REFERENCES [(columnname list or
                        ALL BUT column_name_list)] …

GRANT UPDATE [(columnname list or
                        ALL BUT column_name_list)] …
```

GRANT PUBLIC Implementation Change

The PUBLIC option of the GRANT command allows privileges to be granted to all existing and future users.

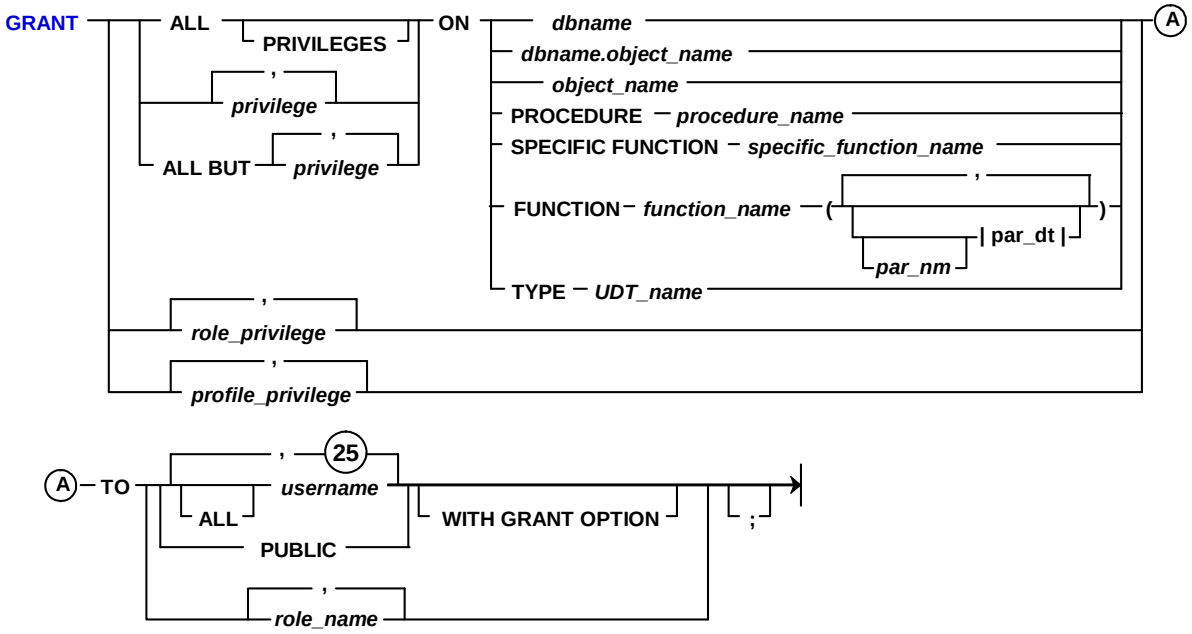
Starting with V2R5, the PUBLIC implementation (also works with the ALL DBC syntax) was changed from one dictionary row per PUBLIC right per user to one row per right. That is, a single row per access right is placed in the DBC.AccessRights table when the PUBLIC option is used.

The use of ALL DBC effectively works the same as PUBLIC.

GRANT Command
(SQL Form)

To GRANT a privilege, the user (grantor) must have one of the following:

- **Have the privilege granted, and hold GRANT authority on the privilege**
- **Be an owner of the object.**



Granting Rights at Database Level

The facing page illustrates privileges granted at the database level.

A system structure for the Teradata database is shown on the facing page and this hierarchy will be used in numerous examples.

Keys to the hierarchy on the facing page are:

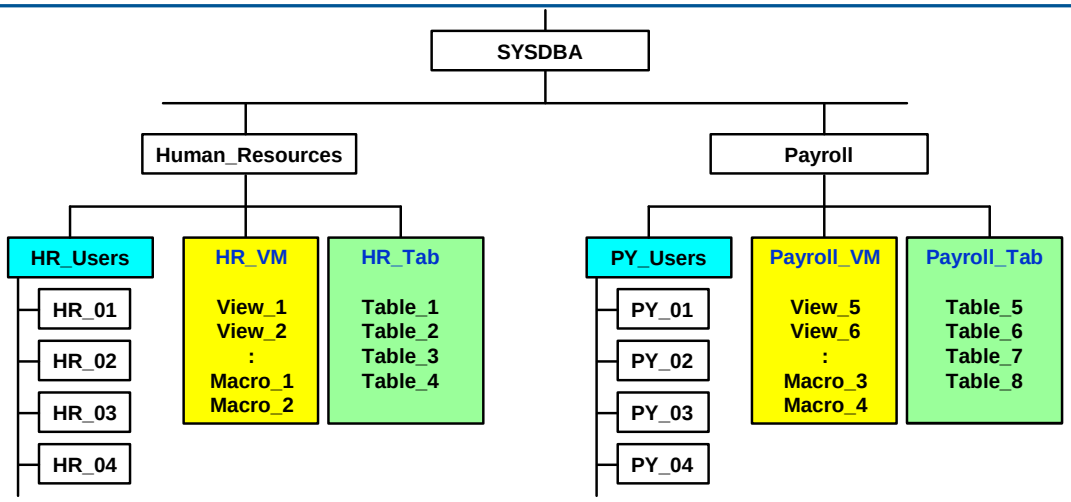
- HR_Users – users that require SELECT and EXECUTE access rights on the views and macros in the HR_VM database.

- PY_Users – users that require SELECT and EXECUTE access rights on the views and macros in the Payroll_VM database.

The database HR_VM will have the SELECT WITH GRANT OPTION access right on the database named HR_Tab.

The database Payroll_VM will have the SELECT WITH GRANT OPTION access right on the database named Payroll_Tab.

Granting Rights at Database Level




```
GRANT SELECT ON HR_Tab TO HR_VM WITH GRANT OPTION;
GRANT SELECT, EXECUTE ON  HR_VM TO ALL HR_Users;

GRANT SELECT ON Payroll_Tab TO Payroll_VM WITH GRANT OPTION;
GRANT SELECT, EXECUTE ON  Payroll_VM TO ALL PY_Users;
```

The ALL option grants the SELECT and EXECUTE privileges to HR_Users and all of its current and future descendants on the database HR_VM.

GRANT Rights at the Table or Column Level

Prior the Teradata 13.0, only the UPDATE and REFERENCES privileges can be granted at the table level or at the column or columns level.

Starting with Teradata 13.0, the SELECT, INSERT, UPDATE, and REFERENCES privileges can be granted at the table or the column or columns level.

The **INDEX** privilege must be granted at the table level, to permit the creating of secondary indexes.

Access Rights and Triggers

To create or replace a trigger, specific access rights are required.

Access Rights to Create Triggers:

- CREATE TRIGGER privilege on the subject table or the database.
- SELECT privilege on any column referenced in a WHEN clause or a triggered SQL statement subquery.
- INSERT, UPDATE, or DELETE privileges on the triggered SQL statement target table, depending on the triggered SQL statement.

Access Rights to Replace Triggers:

- DROP TRIGGER privilege on the subject table or the database. The exception is when you use the REPLACE TRIGGER statement when no target trigger exists and you instead create a new trigger.
- SELECT privilege on any column referenced in a WHEN clause or a triggered SQL statement subquery.
- INSERT, UPDATE, or DELETE privileges on the triggered SQL statement target table, depending on the triggered SQL statement.

Example:
```
CREATE TRIGGER trigger1
    AFTER UPDATE OF (col1) ON table1 FOR EACH ROW
    WHEN NEW col1 > 100
    INSERT INTO log_table VALUES …
```

GRANT Rights at the Table or Column Level

Prior to Teradata 13.0, only the UPDATE and REFERENCES privileges can be granted at the table level or the column(s) level.

Starting with Teradata 13.0, the SELECT, INSERT, UPDATE, and REFERENCES privileges can be granted at the table or the column(s) level.

Examples assigning the UPDATE privilege to a table or columns of a table:

```
GRANT UPDATE                          ON Employee TO tfact01;
GRANT UPDATE (salary_amount)          ON Employee TO tfact01;
GRANT UPDATE (ALL BUT salary_amount)  ON Employee TO tfact01;
```

To CREATE or ALTER a table with foreign key references:

```
GRANT REFERENCES                          ON Employee TO tfact01;
GRANT REFERENCES (employee_number)        ON Employee TO tfact01;
GRANT REFERENCES (ALL BUT employee_number) ON Employee TO tfact01;
```

The INDEX privilege is granted at the table level to allow a user to CREATE or DROP indexes on a table or to allow a user to collect statistics on a table.

```
GRANT INDEX ON Employee TO tfact01;
```

REVOKE Command

REVOKE is passive in that it:

- Does *not* add rows to DBC.AccessRights.
- Removes rows from the DBC.AccessRights table *only* if the privileges specified exist.
- Does *not* cascade through the hierarchy unless you specify the “ALL username” option.
- Is not automatically issued for privileges granted by a grantor dropped from the system.

The REVOKE statement removes rights inserted in the DBC.AccessRights table by a CREATE statement. It can also remove explicit rights inserted in the DBC.AccessRights table by the GRANT statement.

REVOKE Recipients

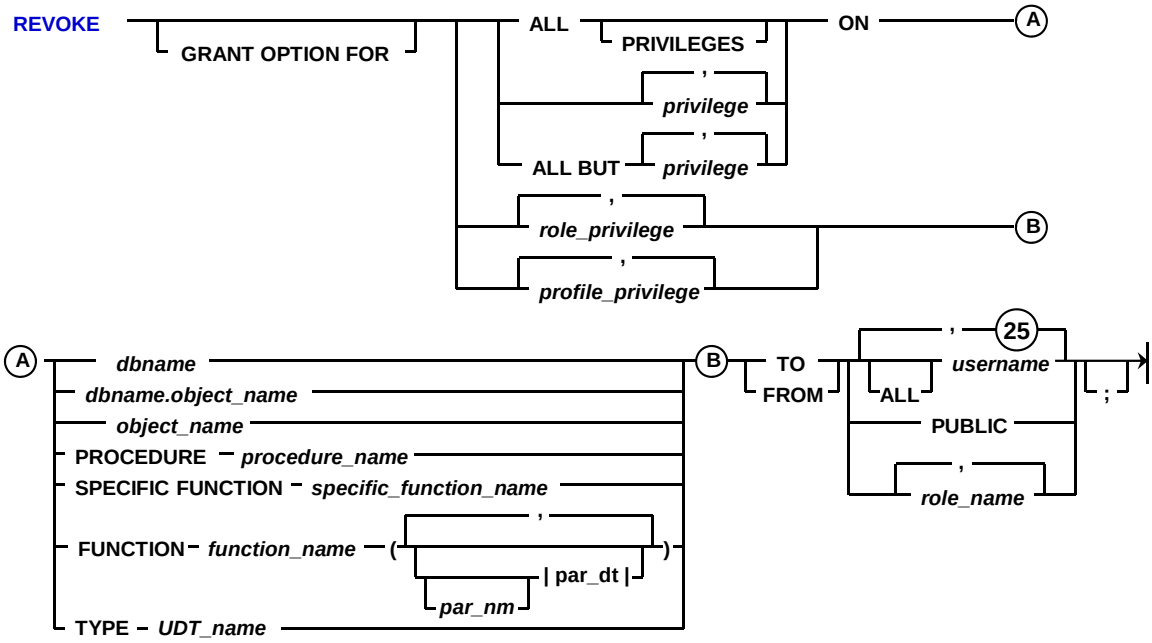
The REVOKE statement can remove privileges from one of the following:

- username — A specific named user(s)
- PUBLIC — Every user in the DBC system
- ALL username — The named user and all of his descendants
- Role — Specified role or roles

REVOKE Command
(SQL Form)

To REVOKE a privilege, the user must have one of the following:

- **Have the privilege granted, and hold GRANT authority on the privilege**
- **Be an owner of the object.**



Revoking Non-Existent Rights

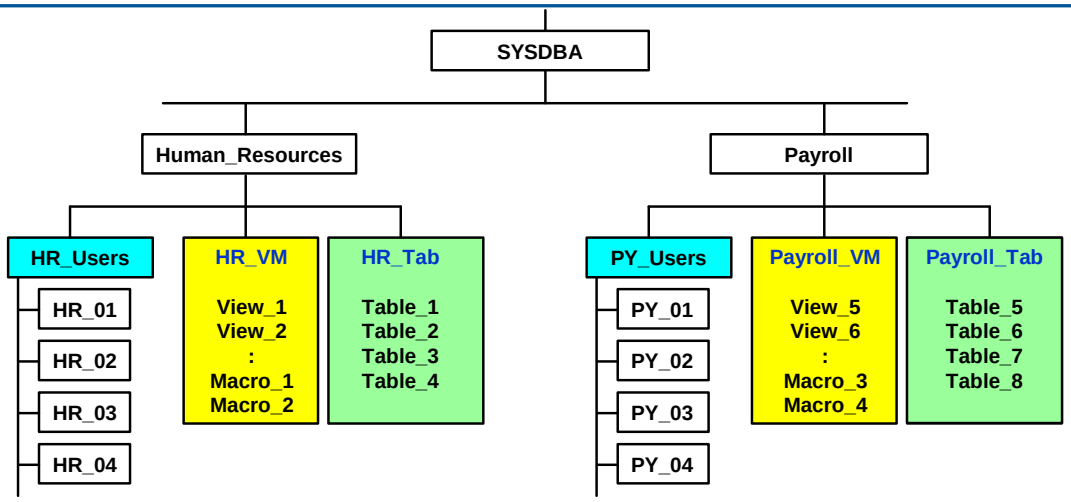
A REVOKE statement at the object level cannot remove privileges from that object that were granted at the database or user level because there is no correlating row in the DBC.AccessRights table for the individual object.

Example

The diagram on the facing page illustrates privileges granted at the database level. User Payroll logs on to the system, and grants the SELECT privilege to user PY_Users and ALL of its descendants on the database Payroll_VM.

Later, Payroll REVOKES the SELECT privilege from ALL PY_Users only on View_6 that resides in Payroll_VM. Although the system returns the message “Revoke Accepted,” nothing actually happened. The user PY_Users and its descendants still have the SELECT privilege on all views residing in database Payroll_VM because the DBC.AccessRights table does not have a row correlating to View_6. Since the row granting select at the database level is still intact, all access rights remain in effect.

Revoking Non-Existent Rights




```
GRANT SELECT ON Payroll_VM TO ALL PY_Users;
Grant Accepted.

REVOKE SELECT ON Payroll_VM.View_6 FROM ALL PY_Users;
Revoke Accepted.
```

REVOKE is passive. It does not add rows to DBC.AccessRights, but removes rows if they exist.

Removing a Level in the Hierarchy

The example on the facing page demonstrates how to remove a level from an existing hierarchy. In the first diagram, user A is the owner of users B, C, and D. User A no longer needs user B. He wants to keep users C and D.

Transfer Ownership

The first thing user A needs to do is transfer ownership of user C to A. When user A submits the GIVE statement, both user C and user D will be transferred. That is because the GIVE statement transfers the named object and all of its children. Since user D is a child of user C, both objects are transferred under user A.

Delete User or Database

In order to DROP user B, user A must first delete all objects from user B. The DELETE USER command will delete all data tables, views, triggers, stored procedures, and macros from a database or user. This command will not remove a Permanent Journal, Hash Indexes, or Join Indexes from a user or database.

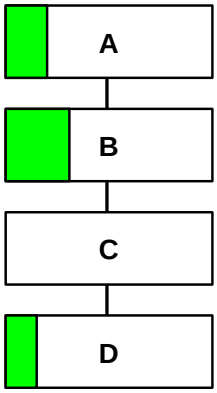
Drop User

After user A removes all objects from user B, user A can submit the DROP statement.

Access Rights

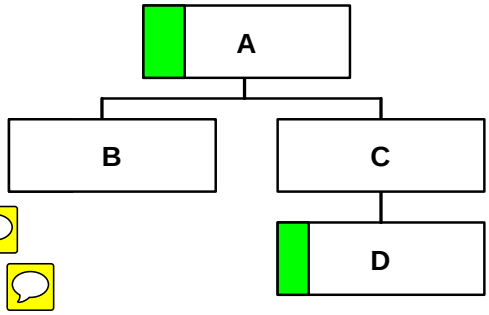
The privileges for user C and user D remain intact. Although user B, their original creator, no longer exists, the privileges granted or caused to be granted are not automatically revoked. Note that user A has recovered the perm space held by user B.

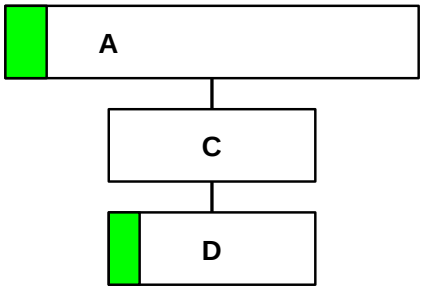
Removing a Level in the Hierarchy



LOGON with the required privileges, and

1) GIVE C TO A;
2) DELETE USER B;
3) DROP USER B;





Although B no longer exists as a user, the privileges granted or caused to be granted are not automatically revoked.

Inheriting Access Rights

You may inherit access rights by the placement of your user in the hierarchy. As an administrator, you can set up access rights so that any new object added to an existing user or database inherits specific access rights. Doing so saves time since you do not need to submit a GRANT statement each time you add a new user.

The immediate owner (user or database) of a view or table that is referenced by another must have the right on the referenced object that is specified (SELECT, EXECUTE, etc.) and must have that right with the GRANT option.

Example

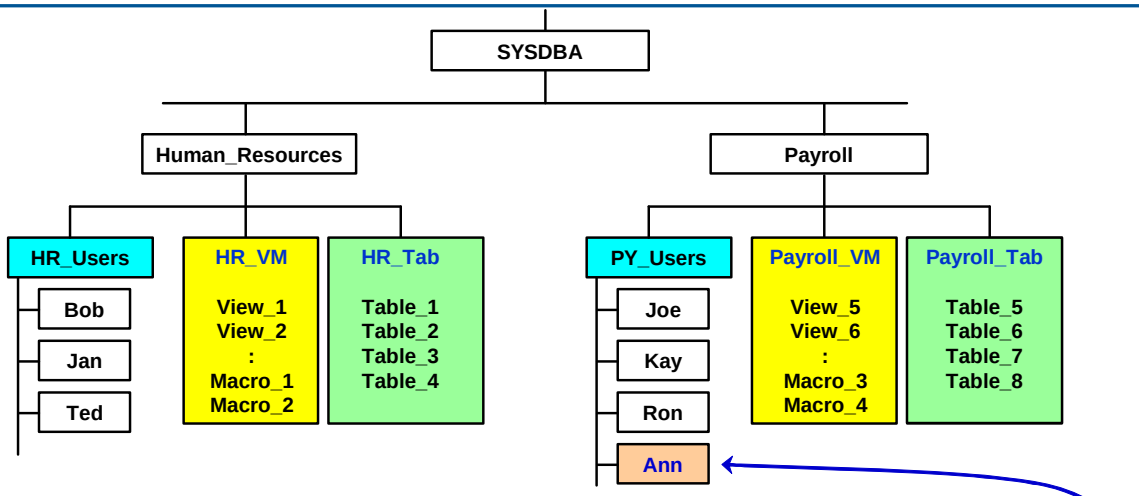
The example on the facing page illustrates a user inheriting access rights.

The user Human_Resources logs on the system and grants the SELECT and EXECUTE privileges to user HR_Users and all of its current and future descendants on the database HR_VM.

The user Payroll also logs on the system and grants the SELECT and EXECUTE privileges to user PY_Users and all of its current and future descendants on the database Payroll_VM.

Later, Payroll creates a new user called Ann from the space owned by user PY_Users. Ann inherits the SELECT and EXECUTE privileges on database Payroll_VM database.

Inheriting Access Rights




```
GRANT SELECT ON Payroll_Tab TO Payroll_VM WITH GRANT OPTION;
GRANT SELECT, EXECUTE ON Payroll_VM TO ALL PY_Users;

CREATE USER Ann FROM PY_Users AS PERM = 0, PASSWORD = temp;
```

Ann "inherits" the SELECT and EXECUTE access rights for the database Payroll_VM.

The GIVE Statement and Access Rights

When you give a user to another owner, privileges are not altered. **The GIVE statement does not alter DBC.AccessRights.** No rights on the given database or user are granted to the new ownership hierarchy as a result of the GIVE statement. The database or user that you GIVE does not receive any access rights from its new owner. The new owner gains *implicit* access rights over the transferred object and the old owner loses them.

Example

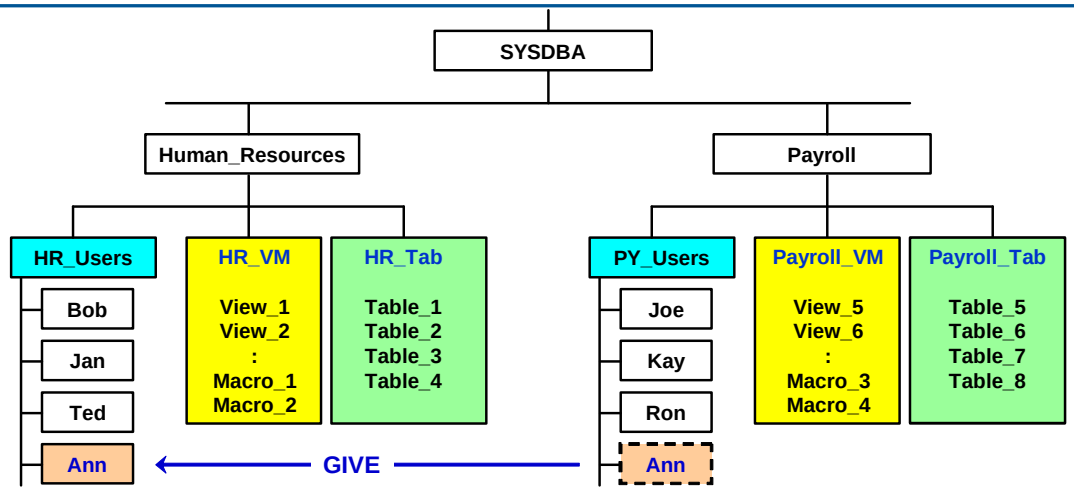
In the example on the facing page, Sysdba logs on to the system and gives user Ann to HR_Users. Ann retains the privileges that she inherited from PY_Users when she was created. Ann does not inherit any access privileges from the new owner, HR_Users, or from Human_Resources

HR_Users is Ann's new owner. It has ownership rights over Ann. PY_Users loses ownership rights over Ann when she is transferred.

The syntax of the GIVE statement is as follows:

GIVE ***database_name*** **TO** ***recipient_name*****;**

The GIVE Statement and Access Rights



NOT Recommended

```
.LOGON sysdba, password;

GIVE Ann TO HR_users;
```

The GIVE command transfers ownership, but does not change any access rights.

Recommended

```
.LOGON sysdba, password;

DROP USER Ann;
CREATE USER Ann FROM HR_Users ...;
```

The DROP will cause Ann's access rights to be removed for Payroll_VM. The CREATE will allow Ann to inherit access rights for HR_VM.

Access Rights and Views

Views may be nested up to 64 levels.

View names are fully expanded (resolved) at creation time.

The system checks access rights at creation time, and validates them again at execution time. Any database referenced by the view requires access rights on all objects accessed by the view.

The facing page shows an example of a nested view.

You can create a view with the intention of read access only, or for controlled UPDATES use. For read access, the SELECT right is needed. For updates, the UPDATE right is needed.

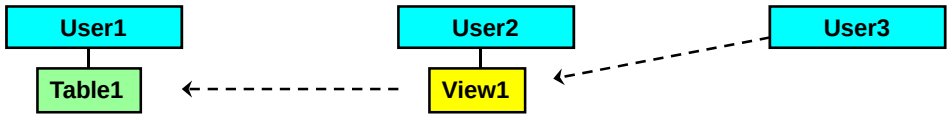
For other users to access a view, the owner must grant the appropriate rights on the view and must have the appropriate rights WITH GRANT OPTION.

The system verifies that the creator has the appropriate right on the objects being referenced when a view is created. It also verifies that the creator has the rights needed to execute the statements defined in a macro. To grant to another user any privilege on a view or macro that references objects owned by a third user, the owner of the view or macro must have the appropriate rights with GRANT OPTION.

Teradata also verifies that the appropriate privileges exist on the objects being referenced for any user who attempts to access a view or execute a macro. This ensures that a change to a referenced object does not result in a violation of access rights when the view or macro referencing that object is invoked.

Access Rights and Views

- View names are fully expanded (resolved) at creation time.
- The system checks access rights at creation time, and validates them again at execution time.




```
GRANT SELECT
   ON Table1 TO User2;
```

```
CREATE VIEW View1
   AS SELECT ...
   FROM User1.Table1;
```

```
SELECT *  FROM User1.Table1;
Fails - Error 3523
```

```
SELECT * FROM View1;
```

```
SELECT * FROM User2.View1;
Fails - Error 3523
```

```
GRANT SELECT
   ON View1 TO User3;
```

```
SELECT * FROM User2.View1;
Fails - Error 5315
```

```
GRANT SELECT
   ON Table1 TO User2
   WITH GRANT OPTION;
```

```
GRANT SELECT
   ON View1 TO User3;
```

```
SELECT * FROM User2.View1;
Success
```

```
SELECT * FROM User1.Table1;
Fails - Error 3523
```

3523 : The user does not have SELECT access to [owner.object].
5315: The owner does not have SELECT WITH GRANT OPTION ...

Access Rights and Nested Views

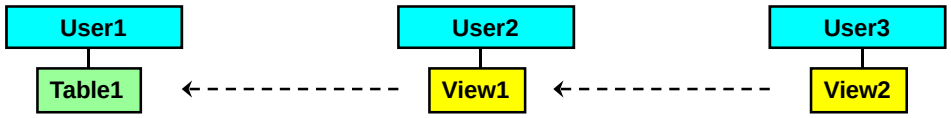
Views that reference other views are sometimes called nested views. Views may be nested up to 64 levels. View names are fully expanded (resolved) at creation time.

The system checks access rights at creation time, and validates them again at execution time. Any database referenced by the view requires access rights on all objects accessed by the view.

The previous example is continued on the facing page.

Access Rights and Nested Views

- Views that reference other views are sometimes called nested views. Views may be nested up to 64 levels.
- The system validates access rights at execution time.




```
GRANT SELECT
   ON Table1 TO User2
   WITH GRANT OPTION;
```

```
GRANT SELECT
   ON View1 TO User3
   WITH GRANT OPTION;
```

User2 can select from Table1 and can create and use views that access Table1.

User3 is given SELECT access on View1 and can create View 2. User3 can access Table1 via View1 or View2.

User3 can GRANT SELECT access to View2 to other users.

```
REVOKE GRANT OPTION
   FOR SELECT
   ON Table1 FROM User2;
```

```
SELECT * FROM View1;
Success
```

```
SELECT * FROM View2;
Fails - Error 5315

SELECT * FROM User2.View1;
Fails - Error 5315
```

If you REVOKE access rights from any user in the chain, the system issues the following message:

5315: The owner does not have SELECT WITH GRANT OPTION …

System Views for Access Rights

There are three system views you can use to obtain information about access rights. (These views access the DBC.AccessRights table to obtain needed information.) They are:

- DBC.AllRights[V][X]
- DBC.UserRights[V]
- DBC.UserGrantedRights[V]

DBC.AllRights[V][X]

The DBC.AllRights[X] views provide information about all rights that have been automatically or explicitly granted.

DBC.UserRights[V]

This view provides information about all rights that the user has acquired, either automatically or explicitly.

DBC.UserGrantedRights[V]

This view provides information about rights that the current user has explicitly granted to other users.

System Views for Access Rights

View	Description
DBC.AllRights[V][X]	Provides information about all rights that have been automatically or explicitly granted.
DBC.UserRights[V]	Provides information about all rights the user has acquired, either automatically or explicitly.
DBC.UserGrantedRights[V]	Provides information about rights which the current user explicitly has granted to other users.

AllRights and UserRights Views

Examples of access rights and their abbreviations include:

DATABASE	=	**CREATE (CD)**	+	**DROP (DD)**
USER	=	**CREATE (CU)**	+	**DROP (DU)**
TABLE	=	**CREATE (CT)**	+	**DROP (DT)**
VIEW	=	**CREATE (CV)**	+	**DROP (DV)**
MACRO	=	**CREATE (CM)**	+	**DROP (DM)**
TRIGGER	=	**CREATE (CG)**	+	**DROP (DG)**
PROCEDURE	=	**CREATE (PC)**	+	**DROP (PD)**
FUNCTION	=	**CREATE (CF)**	+	**DROP (DF)**

Examples of Access Rights and their codes include:

AE = ALTER EXTERNAL PROCEDURE
AF = ALTER FUNCTION
AP = ALTER PROCEDURE
AS = ABORT SESSION
CA = CREATE AUTHORIZATION
CD = CREATE DATABASE
CE = CREATE EXTERNAL PROCEDURE
CF = CREATE FUNCTION
CG = CREATE TRIGGER
CM = CREATE MACRO
CO = CREATE PROFILE
CP = CHECKPOINT
CR = CREATE ROLE
CT = CREATE TABLE
CU = CREATE USER
CV = CREATE VIEW
D = DELETE
DA = DROP AUTHORIZATION
DD = DROP DATABASE
DF = DROP FUNCTION
DG = DROP TRIGGER
DM = DROP MACRO
DO = DROP PROFILE
DP = DUMP
DR = DROP ROLE
DT = DROP TABLE
DU = DROP USER
DV = DROP VIEW
E = EXECUTE
EF = EXECUTE FUNCTION
* GC = CREATE GLOP
* GD = DROP GLOP
* GM = GLOP MEMBER
I = INSERT
IX = INDEX
MR = MONITOR RESOURCE
MS = MONITOR SESSION
NT = NONTEMPORAL
* OD = OVERRIDE DELETE POLICY
* OI = OVERRIDE INSERT POLICY
* OP = CREATE OWNER PROCEDURE
* OS = OVERRIDE SELECT POLICY
* OU = OVERRIDE UPDATE POLICY
PC = CREATE PROCEDURE
PD = DROP PROCEDURE
PE = EXECUTE PROCEDURE
RO = REPLICATION OVERRIDE
R = RETRIEVE/SELECT
RF = REFERENCE
RS = RESTORE
* SA = SECURITY CONSTRAINT ASSIGNMENT
* SD = SECURITY CONSTRAINT DEFINITION
SS = SET SESSION RATE
SR = SET RESOURCE RATE
* ST = STATISTICS
* TH = CTCONTROL
U = UPDATE
UU = UDT Usage
UT = UDT Type
UM = UDT Method

* Reserved for future use (or associated with Teradata 13.0)

The RESTORE statement also allows the recipient to execute ROLLBACK, ROLLFORWARD, and DELETE JOURNAL commands in the ARC facility. The DROP allows COMMENT ON and COLLECT STATISTICS on the object.

AllRights and UserRights Views

Provides information about the objects on which all users (DBC.AllRights), or the current user (DBC.UserRights), have automatically or explicitly been granted privileges.

DBC.AllRights[V][X]

UserName	DatabaseName
TableName	ColumnName
AccessRight	GrantAuthority
GrantorName	AllnessFlag
CreatorName	CreateTimeStamp

DBC.UserRights[V]

DatabaseName	TableName
ColumnName	AccessRight
GrantAuthority	GrantorName
CreatorName	CreateTimeStamp

Example:

All rights held by the user at the database level (for user tfact07).

```
SELECT    DatabaseName          (FORMAT 'X(16)')
          ,AccessRight
          ,GrantorName          (FORMAT 'X(16)')
FROM      DBC.UserRights
WHERE     Tablename = 'ALL'
ORDER BY  1, 2;
```

Example Results:

DatabaseName	AccessRight	GrantorName
AP	R	DBC
PD	D	SYSDBA
PD	I	SYSDBA
PD	R	SYSDBA
PD	U	SYSDBA
tfact07	CG	SYSDBA

UserGrantedRights View

The DBC.UserGrantedRights[V] view provides information about objects on which the current user has explicitly granted privileges. When you submit the GRANT statement, the system stores explicit privileges as rows in the DBC.AccessRights table.

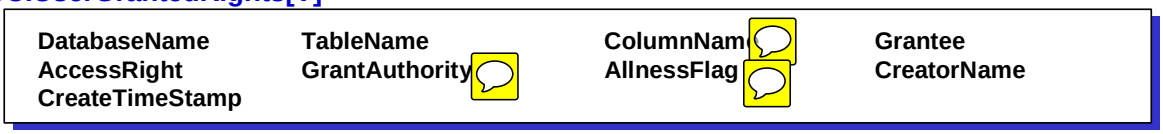
Column definitions in this view include:

Column	Definition
Grantee	The recipient of the access right.
AllnessFlag	Y (Yes) indicates the privilege was granted to all. N (No) indicates the privilege was not granted to all.

UserGrantedRights View

Provides information about objects on which the current user has explicitly granted privileges to other users.

DBC.UserGrantedRights[V]



Example:

List the rights explicitly granted by the current user.

```
SELECT    DatabaseName          (FORMAT 'X(12)')
          ,TableName            (FORMAT 'X(15)')
          ,Grantee              (FORMAT 'X(10)')
          ,AccessRight
          ,AllnessFlag
FROM      DBC.UserGrantedRights
ORDER     BY 1, 2, 3, 4;
```

Example Results:

DatabaseName	TableName	Grantee	AccessRight	AllnessFlag
AP	All	tfact07	R	N
DS	Daily_Sales	tfact03	R	N
DS	Daily_Sales	tfact03	RF	N
DS	Order_Item_JI	tfact03	IX	N
PD	All	Students	R	Y

Teradata Administrator – Grant/Revoke Rights

The facing page contains an example of the Grant/Revoke dialog box that is provided when using the menus of Teradata Administrator.

Tools ➔ Grant/Revoke ➔ Object Rights

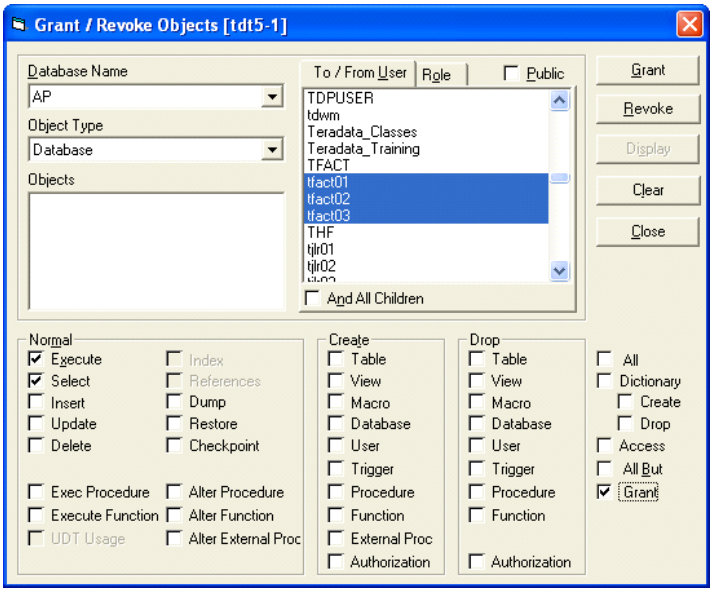
The help facility of Teradata Administrator also lists all of the Access Right Codes.

Teradata Administrator
GRANT/REVOKE Rights

Teradata Administrator can be used to easily grant or revoke access rights.

Tools ➔ Grant/Revoke ➔ Object Rights

- **Select the object name and object type.**
- **Select who is going to get the right.**
- **Select the rights.**

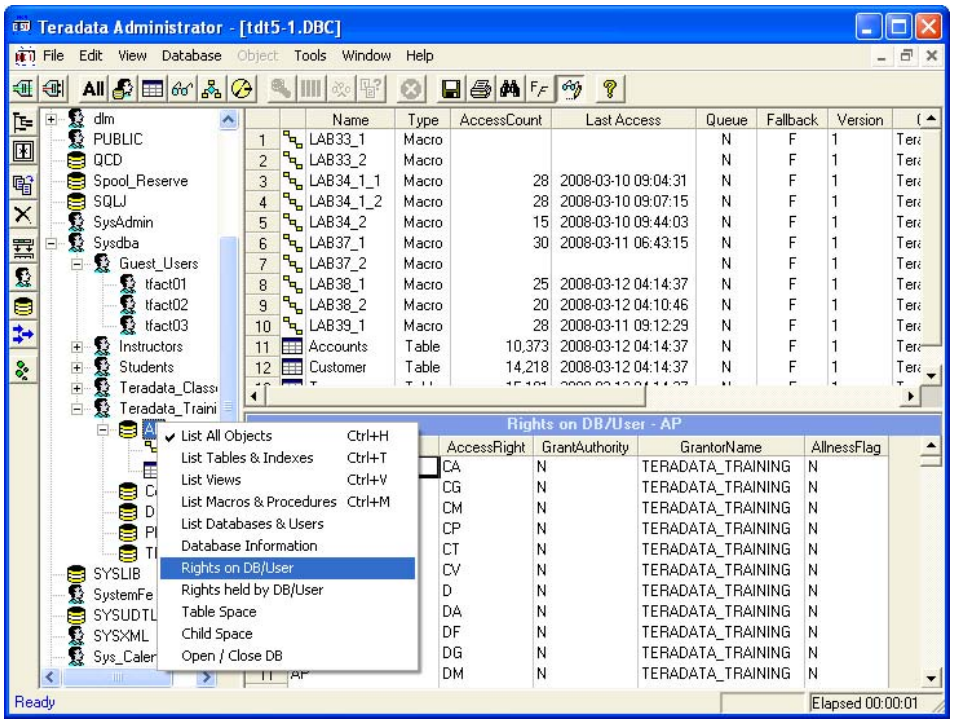


Teradata Administrator – Rights on DB/User

The facing page contains an example of using Teradata Administrator to view the access rights that are on a specific database or user.

Teradata Administrator
Rights on DB/User

Teradata Administrator can also be used to easily view existing access rights



Right-click on the database AP and select the option.

In this example, Rights on DB/User was selected.

Access Rights Summary

The facing page summarizes some important concepts regarding this module.

Access Rights Summary

- Access Rights (Privileges) are maintained in the data dictionary.
- Rows are inserted into or removed from DBC.AccessRights by:
  - CREATE or DROP statements
  - GRANT or REVOKE statements
- Creators are given automatic rights on created objects.
- A newly created user or database is given all rights on themselves except:
  - CREATE Database/User
  - DROP Database/User
- Owners have the right to grant privileges on their owned objects.
- The GIVE command affects ownership, but not information in the DBC.AccessRights table.

Module 46: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 46: Review Questions

1. True or False There are only two types of access rights or privileges: explicit and implicit.

2. True or False The primary statements you use to manage access rights are GRANT, REVOKE, and GIVE.

3. The _____ option on the GRANT command grants privileges to a database or user and all of its current and future descendants.

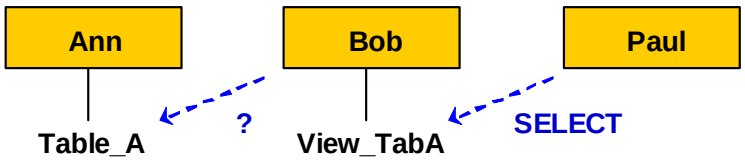
4. The _____, _____, _____, and _____ access rights can be granted at the column level.

5. The _____ user is used to grant an access right to every user in the system.

6. Given the following: Ann owns Table_A, Bob creates View_TabA and grants SELECT on View_TabA to Paul.

   What access right does Ann give Bob on Table_A so Paul can use View_TabA to access Table_A?

   _____



Notes

Module 47

TERADATA
Raising Intelligence

Roles

After completing this module, you should be able to:

- **List 3 advantages of utilizing roles.**
- **Identify the access rights needed to create roles.**
- **Number the steps of access right validation.**
- **Specify the order of precedence of user/session parameters.**
- **Specify the default role when creating new users.**
- **Use system views to display role information.**

Notes

Table of Contents

What are Roles? 47-4
Advantages of Roles 47-4
Access Rights without Roles 47-6
Access Rights Issues (prior to Roles) 47-6
Access Rights Using a Role 47-8
Implementing Roles 47-10
Current or Active Roles 47-12
Nesting of Roles 47-14
Example of Using “Nested Roles” 47-16
Access Rights Validation and Roles 47-18
SQL Statements to Support Roles 47-20
GRANT Command (SQL Form) 47-22
REVOKE Command (SQL Form) 47-24
GRANT and REVOKE Commands (Role Form) 47-26
System Hierarchy (used in following examples) 47-28
Example of Using Roles 47-30
Example of Using Roles (cont.) 47-32
Example of Using Roles (cont.) 47-34
Roles for Directory-Based Users 47-36
Roles for Proxy Users 47-38
RoleInfo View 47-40
RoleMembers View 47-42
DBC.AccessRights and “Rights” Views 47-44
AllRoleRights and UserRoleRights Views 47-46
Steps to Implementing Roles 47-48
Summary 47-50
Module 47: Review Questions 47-52
Lab Exercise 47-1 47-54
Lab Exercise 47-2 47-58

What are Roles?

An additional database administration and security concept called **roles** can be used to simplify database administration.

A role can be viewed as a pseudo-user with privileges on a number of database objects. Any user granted a role can take on the identity of the pseudo-user and access all of the objects it has rights to.

A database administrator can create different roles for different job functions and responsibilities, grant specific privileges on database objects to these roles, and then grant these roles to users.

Advantages of Roles

Advantages of roles include:

- Simplify access rights administration

  A database administrator can grant rights on database objects to a role and have these rights automatically applied to all users assigned to that role. When a user's function within his organization changes, it is easier to change his/her role than deleting old rights and granting new rights that go along with the new function.

- Reduce disk space usage

  Maintaining rights on a role level rather than on an individual level makes the size of the DBC.AccessRights table much smaller. Instead of inserting one row per user per right on a database object, one row per role per right is placed in the DBC.AccessRights table.

- Better performance – roles can improve performance and reduces dictionary contention for DDL

  If roles are fully utilized on a system, roles will reduce the size of the AccessRights table and improve the performance of DDL commands that do full-file scans of this table.

  – Faster DROP/DELETE USER/DATABASE, DROP TABLE/VIEW/MACRO due to shorter scans of the DBC.AccessRights table.

  – Faster CREATE USER, DATABASE - remove copy of hierarchical inherited rights.

- Less dictionary contention during DDL operations because the commands use less time.

What is a Role?

A Role is an administration/security features which can help simplify the management of access rights.

What is a "role"?

- A role is simply a collection of access rights.
  - Rights are first granted to a role and the role is then granted to users.
- A DBA can create different roles for different job functions and responsibilities.

What are the advantages of using "roles"?

- Simplify access rights management by allowing grants and revokes of multiple rights with one request.
  - useful when an employee changes job function (role) within the company.
  - if a job function needs a new access right, grant it to the role and it is effective immediately.
- The number of access rights in the DBC.AccessRights table is reduced.
- Improves performance and reduces dictionary contention for DDL, especially CREATE USER.
  - Removal of hierarchical inherited rights improves DDL performance and reduces dictionary contention.

Access Rights without Roles

The facing page illustrates the following:

- If 10 users have the SELECT access right on each of 10 views, there would be 100 rows in the DBC.AccessRights table for these 10 users.

- What if there were 50,000 users in the system and there were 500 views needed by each user? The DBC.AccessRights table would have 25 million rows.

When a new user is added in this simple example, 10 rows have to be added to the DBC.AccessRights table.

Access Rights Issues (prior to Roles)

The role concept provides a solution to the following problem.

Prior to Teradata V2R5 and the concept of roles, there are typically 2 ways of granting rights to a large user base:

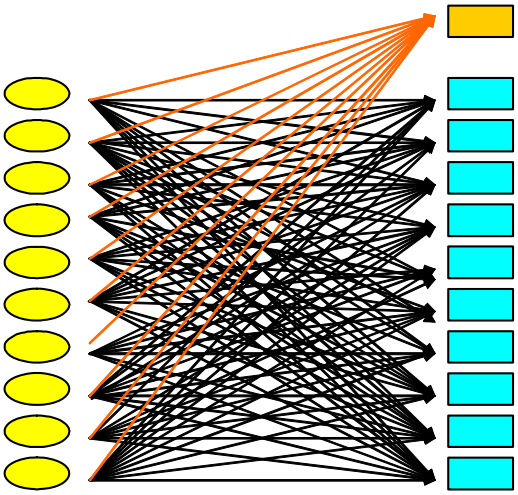
1. Use the ALL option of the GRANT statement to grant rights on the shared object(s) to a parent database. Sometimes this is referred to as a "profile" database or a "group" database in V2R4.1. Do not confuse the logical profile database with the Profile capability in V2R5.

   GRANT SELECT ON *database_object* TO ALL *profile_database*;

   Then, create users under the profile database. The system will automatically grant all rights held by the profile database to each user created under the profile database. This is frequently referred to as "inherited rights".

2. Grant the rights to users individually – an administrative nightmare.

Access Rights Without Roles



GRANT SELECT ON View1 TO New_User; GRANT SELECT ON View2 TO New_User; …

When a new user is given the SELECT access right to these 10 views, 10 new access right rows are added to the DBC.AccessRights table.

In this simple example, these 10 views and 11 users would place 110 access right rows in the DBC.AccessRights table.

Access Rights Using a Role

When creating a new user, only one right to use a role needs to be granted, as opposed to a right for every table/view/macro/stored procedure that the user needs to access.

As mentioned earlier, a role can be viewed as a pseudo-user with privileges on a number of database objects. Any user granted a role can take on the identity of the pseudo-user and access all of the objects it has rights to.

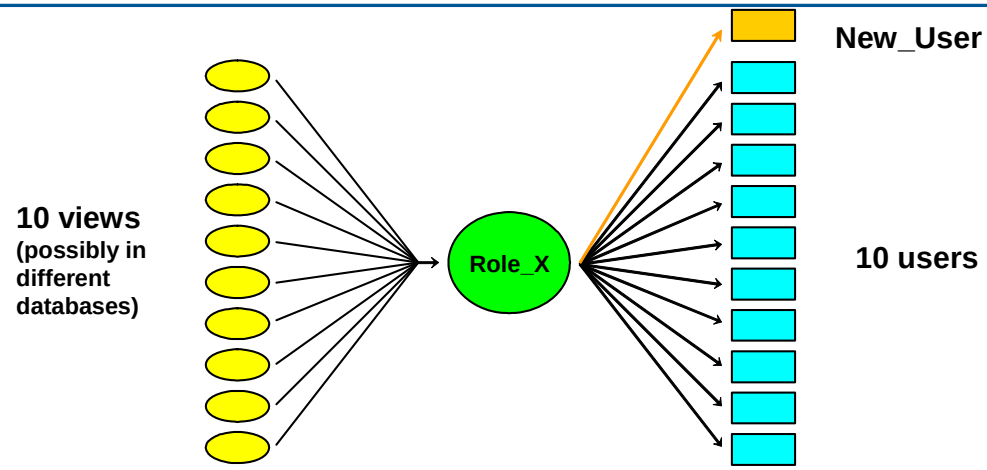
A database administrator can create different roles for different job functions and responsibilities, grant specific privileges on database objects to these roles, and then grant these roles to users.

In the example on the facing page, the GRANT Role_X to New_User places a row in the DBC.RoleGrants table, not the DBC.AccessRights table.

Note:

When an access right is granted to a role, a row in placed in the DBC.AccessRights table. The DBC.AllRights system view only shows access rights associated with users, not roles. The DBC.UserRoleRights system view shows access right rows associated with roles.

Access Rights Using a Role



First, create a role and grant privileges to the role.

```
CREATE ROLE Role_X;
GRANT SELECT ON View1 TO Role_X; GRANT SELECT ON View2 TO Role_X;  …
```

When creating a new user, only one right to use a role needs to be granted.

```
GRANT Role_X TO New_User;
```

This command places a row in the DBC.RoleGrants table, not DBC.AccessRights.

With 10 views, 1 role, and 11 users, there would be 10 rows in the DBC.AccessRights table and 11 rows in the DBC.RoleGrants table.

Implementing Roles

Roles define access privileges on database objects. When you assign a default role to a user, you give the user access to all the objects that the role has been granted privileges to. A default role that has a role as a member gives the user additional access to all the objects that the nested role has privileges to.

A newly created role does not have any associated privileges until grants are made to it. To manage user access privileges, you can:

- Create different roles for different job functions and responsibilities.
- Grant specific privileges on database objects to the roles.
- Assign default roles to users.
- Add members to the role.
- Members of a role can be users or other roles.
- Roles can only be nested one level. Thus, a role that has a role member cannot also be a member of another role.

The CREATE ROLE and DROP ROLE access rights are system rights. These rights are not on a specific database object. Note that the ROLE privileges can only be granted to a user and not to a role or database.

The example on the facing page explicitly identifies the CREATE ROLE and DROP ROLE rights for Sysdba. Another technique of granting both the CREATE ROLE and DROP ROLE access rights to Sysdba is to use the following SQL.

GRANT ROLE TO SYSDBA WITH GRANT OPTION;

The key word ROLE will give both the CREATE ROLE and DROP ROLE access rights.

Note:

If Sysdba is only given the CREATE ROLE access right, Sysdba can create new roles and Sysdba can drop roles that he/she has created. Sysdba would not be able to drop roles created by other users (such as DBC).

The syntax to create a new role is simply:

CREATE ROLE *role_name;*

When a role is first created, it does not have any associated rights until grants are made to it.

Implementing Roles

What access rights are used to create new roles?

- CREATE ROLE – needed to create new roles
- DROP ROLE – needed to drop roles

Who is allowed to create and modify roles?

- Initially, only DBC has the CREATE ROLE and DROP ROLE access rights.
- As DBC, give the "role" access rights to the database administrators (e.g., Sysdba).

```
GRANT CREATE ROLE, DROP ROLE TO Sysdba WITH GRANT OPTION;
```

How are access rights associated with a role?

- First, create a role.

```
CREATE ROLE HR_Role;
```

  A newly created role does not have any associated rights until grants are made to it.

- Use the GRANT (or REVOKE) command to assign (or take away) access rights to (or from) the role.

```
GRANT SELECT, EXECUTE ON HR_VM TO HR_Role;
```

Current or Active Roles

With Teradata V2R5.0, at any time, only one role may be the session's current role. Enabled roles are the session's current role plus any nested roles. At logon time, the current role will be the user's default role.

Starting with Teradata V2R5.1, the SET ROLE ALL option is available and this allows a user to have all valid roles (for that user) to be active or available.

Create User or Modify User

The user executing the CREATE USER command with the DEFAULT ROLE option must have ADMIN privileges on a specified role. The default role is automatically granted to the newly created user.

The user executing the MODIFY USER command with the DEFAULT ROLE option must also have ADMIN privileges on a specified role. The new default role must have first been directly granted to the user before modifying the DEFAULT ROLE with the MODIFY USER command.

Current or Active Roles

How are users associated with a role?

- The role needs to be granted to the user.

  GRANT HR_Role TO user1; 

The current or active role can be set to a specific role or to the key word ALL.

- A specific role can established as the current role.
  - A user has the access rights of the current role plus any nested roles.
- SET ROLE ALL allows a user to have all valid roles (for that user) to be active.
- At logon, the current role is determined by the DEFAULT ROLE value for the user.

  CREATE/MODIFY USER user1 AS … , DEFAULT ROLE = HR_Role;

  or CREATE/MODIFY USER user1 AS … , DEFAULT ROLE = ALL;
- A user may change roles by executing the following command within their session.

  SET ROLE *role_name*;

  or SET ROLE ALL;
- ANSI Note: A session that has only one current role complies with the ANSI standard.

Nesting of Roles

Roles define access privileges on database objects. When you assign a default role to a user, you give the user access to all the objects that the role has been granted privileges to. It is possible to grant a role to another role. This is referred to as “nesting”. Teradata supports one level of nesting.

If a role has another role as a member (a role has been granted to a role) and the role is the active role for a user, then a user gets additional access to all the objects that the nested role has privileges to.

For example:

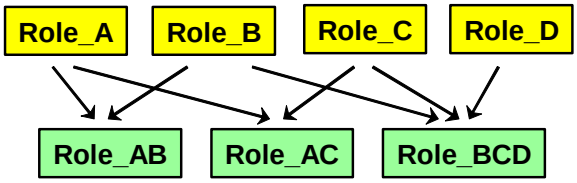
- Assume Role_A and Role_B is granted to Role_AB and assume that Role_AB is the current role of a user. The user then has the following access rights:

  – Access rights directly assigned to the user
  – Access rights assigned to Role_A
  – Access rights assigned to Role_B
  – Access rights assigned to Role_AB

Nesting of Roles

You can GRANT a role to another role – referred to as "nesting of roles".

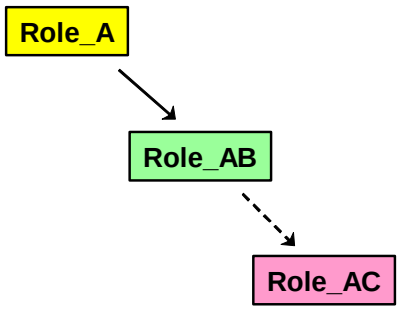
Allowed: 1 level of nesting




```
GRANT Role_A TO Role_AB;
GRANT Role_B TO Role_AB;
GRANT Role_A TO Role_AC;
GRANT Role_C TO Role_AC;
GRANT Role_B TO Role_BCD;
GRANT Role_C TO Role_BCD;
GRANT Role_D TO Role_BCD;
```

A user that is granted access to Role_AB also has all of the access rights associated with Role_A, Role_B, and Role_AB.

Not Allowed: 2nd level of nesting



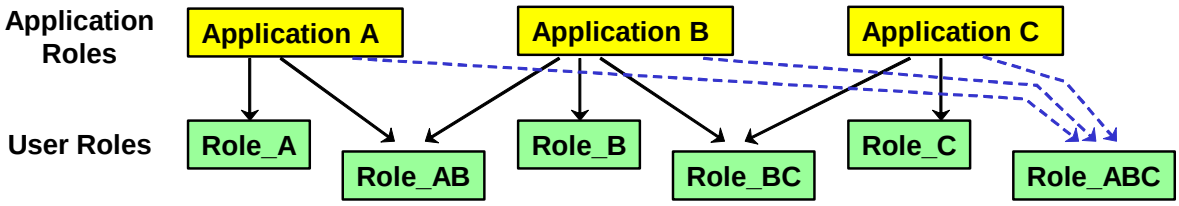

```
GRANT Role_A TO Role_AB;   /*accepted*/
GRANT Role_AB TO Role_AC;  /*fails*/
```

Example of Using “Nested Roles”

The facing page contains an example of using nested roles.

Example of Using "Nested Roles"

Access rights required by an Application are assigned to "application" roles.



Users are assigned to a role at this level based on job requirements.

Characteristics include:

- Users are only assigned to one "role" – ANSI standard.
- Provides a logical separation between application access rights and user access rights.
  - Access rights for an application are only assigned to a single "application" role.
  - For example, if a user needs to use Applications A, B, and C, then the user is granted access to Role_ABC.

Access Rights Validation and Roles

The validation of access rights for accessing a given database object is carried out in one or more steps. The first step verifies if a right has been granted on an individual level. If no such right exists and there is a current role for the session, then the second and third steps verify if a right has been granted to a role. The actual search goes like this:

1) Search the AccessRights table for a UserId-ObjectId pair entry for the required right. In this step, the system will check for rights at the database/user level and at the object (e.g., table, view) level.

2) If the access right is not yet found and the user has a current role, search the AccessRights table for RoleId-ObjectId pair entry for the required right.

3) If not yet found, retrieve all roles nested within the current role from the RoleGrants table. For each nested role, search the AccessRights table for RoleId-ObjectId pair entry for the required right.

4) If not yet found, check if the right is a Public right.

Performance note: If numerous roles are nested within the current role, there may have noticeable performance impact on “short requests”. A few more access right checks won't be noticed on a 1-hour query.

Notes: The following indexes are placed on the DBC.AccessRights, DBC.RoleGrants, and DBC.Roles tables and are used by Teradata software.

DBC.AccessRights
- PI – (NUPI) – (UserId, DatabaseId)
- SI – (NUSI) – (TVMId)

DBC.RoleGrants
- PI – (NUPI) – (GranteeId)
- SI – (USI) – (GranteeId, RoleId)
- SI – (NUSI) – (RoleId)

DBC.Roles
- PI – (UPI) – (RoleNameI)
- SI – (USI) – (RoleId)

Access Rights Validation and Roles

Validation of access rights for accessing a given database object will be carried out in the following steps.

Order of access right validation is:

1) **Check the DBC.AccessRights table for the required right at the individual level.**
2) **If the user has a current role, check the DBC.AccessRights table for the required right at the role level.**
3) **Retrieve all roles nested within the current role from the DBC.RoleGrants table. For each nested role, check the DBC.AccessRights table for the required right.**
4) **Check if required right is a PUBLIC right.**

SQL Statements to Support Roles

Some miscellaneous rules concerning roles include:

- Roles may only be granted to users and other roles.
- There is no limit on the number of roles that can be granted to a grantee.
- The default role for a user will automatically be made the current role for the session when he first logs on. The default role can be established with the CREATE USER or MODIFY USER commands.
- A role grantor can only be a user, but a role grantee can be a user or another role. A role may share the same name as a profile, table, column, view, macro, trigger, or stored procedure. However, a role name must be unique amongst users, databases and roles.
- The role creator is automatically granted membership to the newly created role WITH ADMIN OPTION, which makes the role creator a member of the role who can grant membership to the role to other users and roles.

Dropping a Role

The following users can drop a role:

1. DBC
2. Any user given the system right DROP ROLE
3. Any user granted the role WITH ADMIN OPTION
4. A user whose current role has the specified role as a nested role, and the nested role was granted to the current role WITH ADMIN OPTION

The creator does not have the implicit right to drop a role. If WITH ADMIN OPTION and DROP ROLE rights are revoked from him/her, he/she will not be able to drop the role.

Default role settings for all users with the dropped role as their default role do *not* reset to NULL. Affected users receive no warnings or errors the next time they log on. The system does not use the obsolete default role for privileges validation.

If a dropped default role is subsequently recreated, it reassumes its status as the default role, but it has a different role ID number than it had before being dropped.

SQL Statements to Support Roles

Command Syntax:

CREATE ROLE *role_name*; 

GRANT *access_rights* TO *role_name*;

GRANT *role_name* TO *user_name* [WITH ADMIN OPTION];

- **ADMIN OPTION allows grantee the right to grant or drop the role.**

SET ROLE *role_name* / NONE / NULL / ALL;

- **Assigns/changes current role for session.**
- **Role must be granted to user before statement is valid.**
- **SET ROLE ALL; All valid roles for user are available to user.**

CREATE/MODIFY USER user1 AS …, DEFAULT ROLE = *role_name*;

- **When the user logs on, the default role will become the session's initial current role.**

Other commands:

REVOKE ... *role_name* … ;

DROP ROLE *role_name ;*

SELECT ROLE ;

GRANT Command (SQL Form)

Once a new role is created, access rights can be added to or withdrawn from the role with GRANT/REVOKE statements.

Roles may be granted privileges on the following database objects.

- Database
- Table
- View
- Macro
- Column
- Triggers
- Stored procedures
- Join and Hash indexes

Roles may not be granted on the following access rights (or functions).

- CREATE ROLE and DROP ROLE
- CREATE PROFILE and DROP PROFILE
- CREATE USER and DROP USER
- CREATE DATABASE and DROP DATABASE

- CTCONTROL – Grants the privilege to connect as a proxy permanent or proxy application user through the specified trusted user, storing the information in *DBC.ConnectRulesTbl*. Authorizes a user to grant or revoke the CONNECT THROUGH privilege using the GRANT CONNECT or REVOKE CONNECT statements. You can only grant CTCONTROL to specific users

.

- REPLCONTROL (controls two separate functions
  - The privilege to define and manage replication groups.
  - The ability to run SQL statements that change columnar data values for a table when that table is in a state that would not otherwise allow changes to be made.

Exceptions

A role cannot have descendants, i.e., the ALL option of a GRANT/REVOKE statement cannot be applied to a role. The following statement is not allowed.

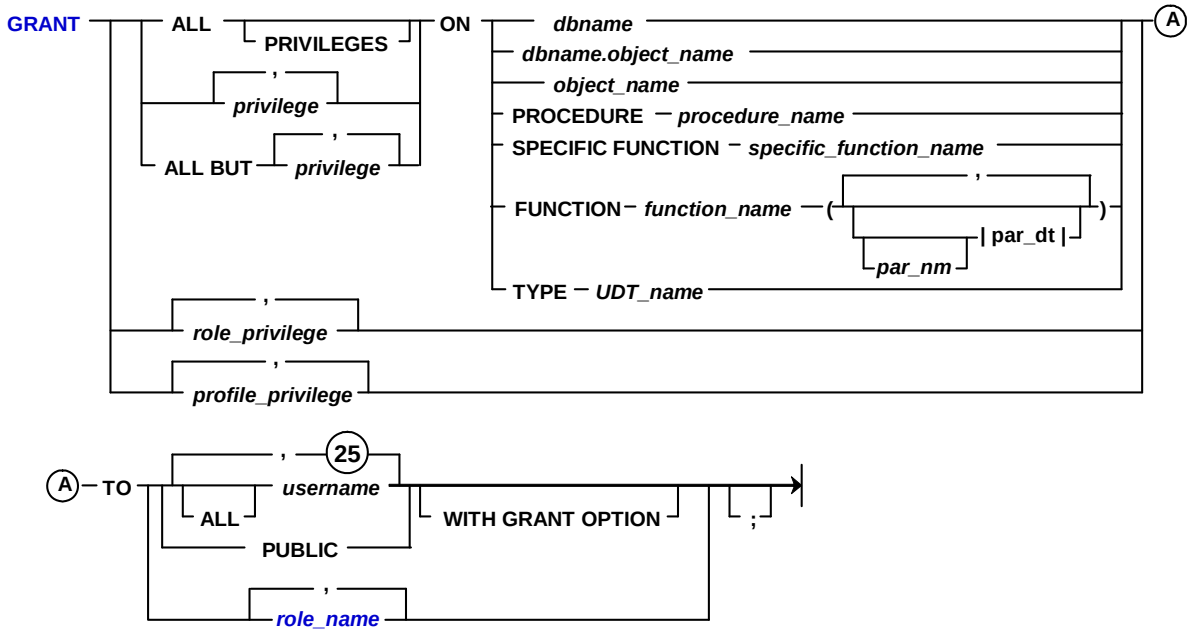
```
GRANT <right> ON <database object> TO ALL <role name>;
```

ANSI also disallows a right to be granted to a role with the GRANT option. The following statement is also illegal.

```
GRANT <right> ON <db object> TO <role name> WITH GRANT OPTION;
```

GRANT Command
(SQL Form)

The GRANT command may be used to grant access rights to roles.

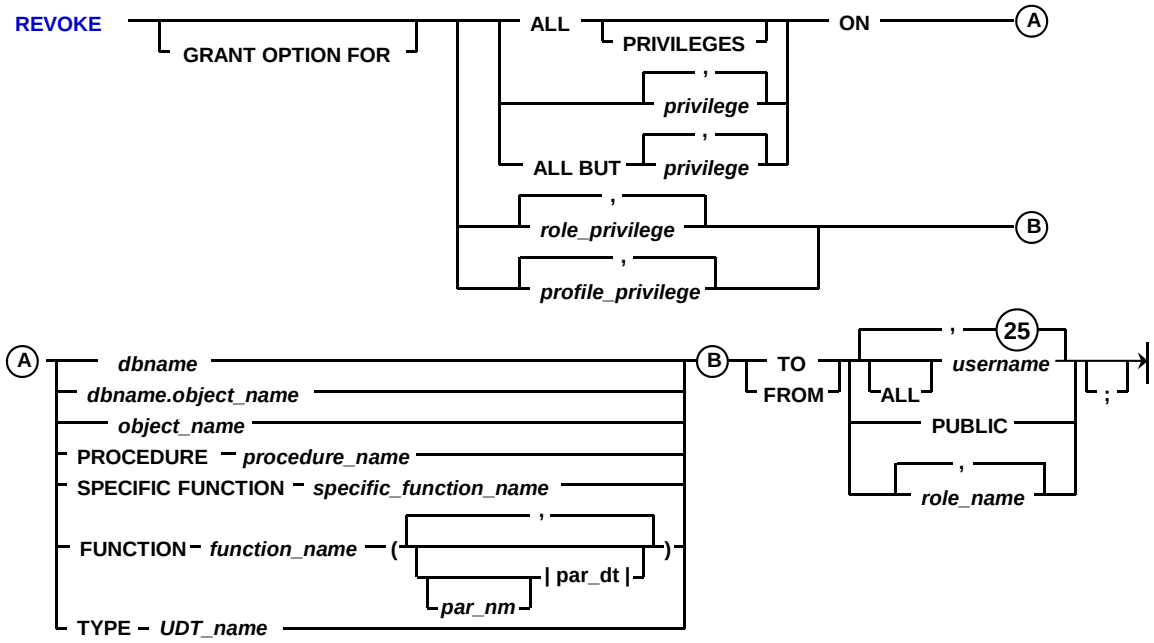


REVOKE Command (SQL Form)

The facing page shows the syntax for the REVOKE Command.

REVOKE Command
(SQL Form)

The REVOKE command may be used to revoke access rights from roles.



GRANT and REVOKE Commands (Role Form)

GRANT (Role Format) is used to grant role membership to users or other roles.

role_name

This is a list of one or more comma-separated names of roles to which membership or administrative ability is being granted

TO *user_name* or *role_name*

This is a list of names of role grantees. Grantees can be users or roles; however, a role cannot be granted membership to itself.

WITH ADMIN OPTION

The role grantees have the right to use DROP ROLE, GRANT, and REVOKE statements to administer the roles to which they are becoming members.

A GRANT statement that does not include WITH ADMIN OPTION does not revoke a previously granted WITH ADMIN OPTION privilege from grantee.

REVOKE (Role Format) is used to revoke role membership to users or other roles.

ADMIN OPTION FOR

The role members maintain membership status, but lose the right to use GRANT, REVOKE, and DROP ROLE statements to administer the roles to which they are members.

If ADMIN OPTION FOR does not appear in the REVOKE statement, the system removes the specified roles or users as role members.

role_name

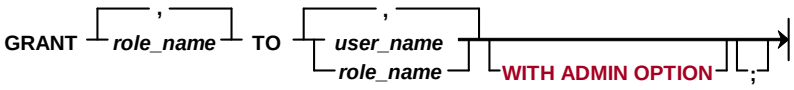
This is a list of one or more comma-separated names of roles from which membership or administrative ability is being revoked. The system ignores duplicate role names.

TO/FROM *user_name* or *role_name*

This identifies the names of role members that are losing membership or administrative ability. Members can be users or roles.

GRANT and REVOKE Commands (Role Form)

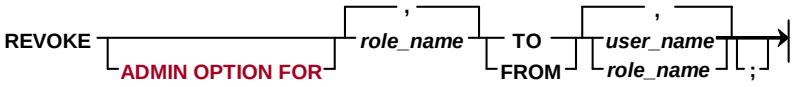
The syntax to grant a role to a user (or role) is:



WITH ADMIN OPTION

Gives the role grantee(s) the right to use DROP ROLE, GRANT, and REVOKE statements to administer the roles to which they are becoming members.

The syntax to revoke a role from a user (or role) is:



ADMIN OPTION FOR

The role members maintain membership status, but lose the right to administer the roles to which they are members.

If this option is not used, the system removes the specified roles or users as role members.

System Hierarchy (used in following examples)

A system structure for the Teradata database is shown on the facing page and this hierarchy will be used in numerous examples.

Keys to the hierarchy on the facing page are:

- Roles will have a _R at the end of the role_name. For example, HR_R represents the Human Resources Role.

- Inquiry Users – users that require SELECT and EXECUTE access rights on the views and macros in the VM databases. These users will be assigned either to the HR_R, Pay_R, or the HR_PAY_R.

- Update Users – users that require SELECT, EXECUTE, INSERT, UPDATE, and DELETE access rights on the views and macros in the VM databases. These users will be assigned either to the HR_Upd_R, Pay_Upd_R, or the HR_PAY_Upd_R.

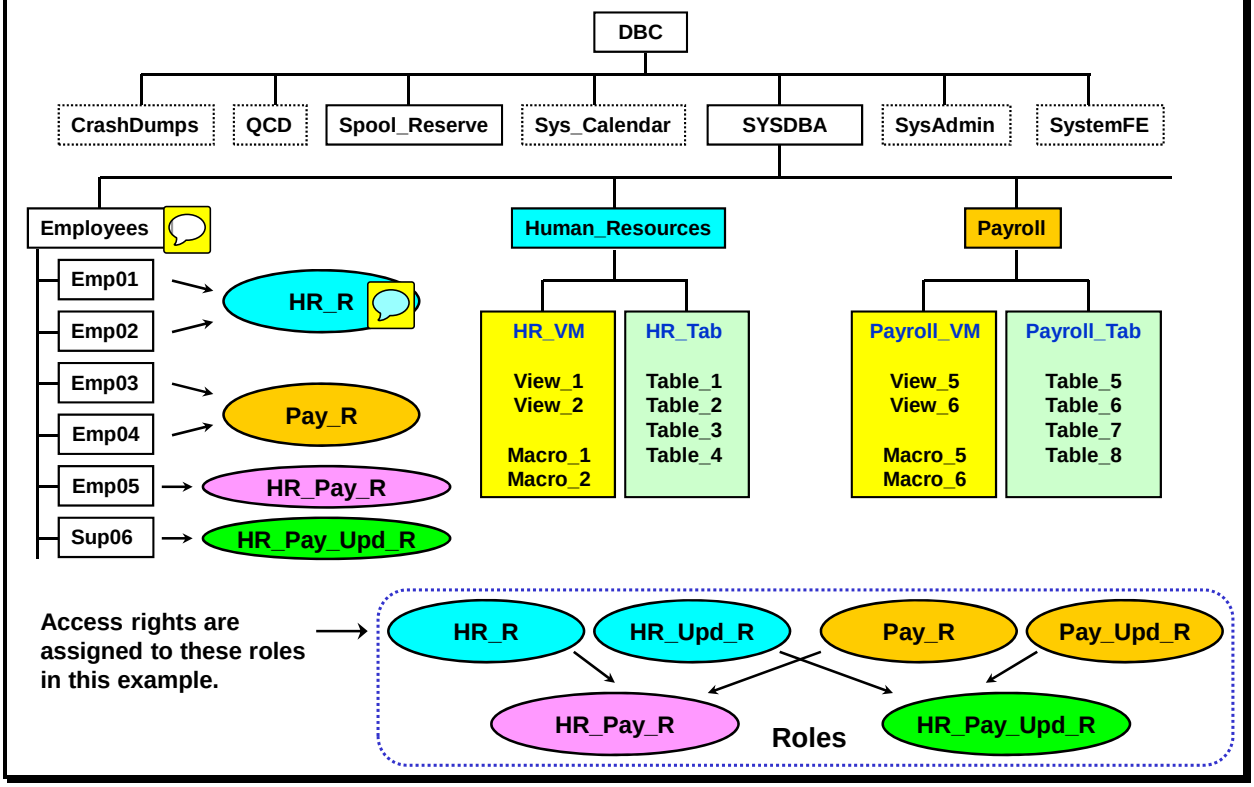
The database HR_VM will have SELECT, EXECUTE, INSERT, UPDATE, and DELETE privileges WITH GRANT OPTION on the database named HR_Tab. Likewise for Payroll_VM and Payroll_Tab.

Dropping a User

When a DROP USER command is issued, both individual rights and role rights granted to the user being dropped will be deleted from the DBC.AccessRights and the DBC.RoleGrants tables. Deletions of database objects within the user space prior to the DROP USER command will cause corresponding deletions of DBC.AccessRights rows for rights granted on these objects to roles and other users/databases.

However, rights granted by the dropped user that are not on objects within its space will remain in the system. This would include role rights. Roles and profiles created by the dropped user will remain in the system.

System Hierarchy
(used in following examples)



Example of Using Roles

The facing page contains a simple example of creating a role, assigning access rights to it, and granting the role to users.

The default role for a user will automatically be made the current role for the session when the user first logs on. **The role must be currently granted to the user (otherwise, it is ignored).**

Only a partial listing of the access rights that would be assigned to roles is shown on the facing page. Additionally, these commands would also be executed to complete the example.

- **GRANT SELECT, EXECUTE ON Payroll_VM TO Pay_R;**
- **GRANT SELECT, EXECUTE, INSERT, UPDATE, DELETE ON Payroll_VM TO Pay_Upd_R;**
- **GRANT HR_Upd_R TO HR_Pay_Upd_R;** /* nested role */
- **GRANT Pay_Upd_R TO HR_Pay_Upd_R;** /* nested role */

Example of Using Roles

Create roles.

```
CREATE ROLE HR_R;                    CREATE ROLE HR_Upd_R;
CREATE ROLE Pay_R;                   CREATE ROLE Pay_Upd_R;
CREATE ROLE HR_Pay_R;                CREATE ROLE HR_Pay_Upd_R;
```

Assign access rights to the roles (partial listing).

```
GRANT SELECT, EXECUTE                                  ON HR_VM TO HR_R;
GRANT SELECT, EXECUTE, INSERT, UPDATE, DELETE          ON HR_VM TO HR_Upd_R;
```

Grant users permission to use the roles (partial listing).

```
GRANT HR_R              TO Emp01, Emp02;
GRANT Pay_R             TO Emp03, Emp04;
GRANT HR_R              TO HR_Pay_R;        /*nested role*/
GRANT Pay_R             TO HR_Pay_R;        /*nested role*/
GRANT HR_Pay_R          TO Emp05;
GRANT HR_Pay_Upd_R      TO Sup06  WITH ADMIN OPTION;
```

Modify the user to set the default role.

```
MODIFY USER Emp01 AS DEFAULT ROLE = HR_R;
MODIFY USER Emp02 AS DEFAULT ROLE = HR_R;
MODIFY USER Emp03 AS DEFAULT ROLE = Pay_R;
MODIFY USER Emp04 AS DEFAULT ROLE = Pay_R;
MODIFY USER Emp05 AS DEFAULT ROLE = HR_Pay_R;
MODIFY USER Sup06  AS DEFAULT ROLE = HR_Pay_Upd_R;
```

Example of Using Roles (cont.)

The facing page continues the example.

Answer to first question on facing page:

Emp01 does not have UPDATE permission to update the Employee table via the Employee_v view. The error returned is:

5315: The user does not have UPDATE access to HR_VM.Employee_v.Dept_Number.

Answer to second question on facing page:

Both SQL statements work for Emp05 because the access rights for HR_R and Pay_R are nested within HR_Pay_R.

Example of Using Roles (cont.)

Emp01 – is granted to HR_R role and this is the current role.

```
SELECT      *
FROM        HR_VM.Employee_v
WHERE       Employee_Number = 100001;          (success)

UPDATE      HR_VM.Employee_v
SET         Dept_Number=1001
WHERE       Employee_Number=100001;            (success or fail?)
```

Does this statement succeed or fail for Emp01?

Emp05 – is granted to HR_Pay_R role and this is the current role.

```
SELECT      *
FROM        HR_VM.Employee_v
ORDER BY    1;                                 (success or fail)

SELECT      *
FROM        Payroll_VM.Salary_v
ORDER BY    1;                                 (success or fail)
```

Do both of these statements succeed for Emp05?

Example of Using Roles (cont.)

The facing page continues the example.

If a user tries to use the SET ROLE command to specify a role they have not been granted access, the user will get the following error:

> 5621: User has not been granted a specified role.

Answer to first question: The statement fails because Emp05's current role is only provides Select access and this role does not have update permission on Employee_v.

Answer to second question: The statement succeeds because Emp05's current role is now HR_Pay_Upd_R and this role does have update permission on Employee_v.

Answer to third question: Assuming that the default role for Emp05 is HR_Pay_R, the statement will fail until Emp05 uses the SET ROLE command or uses a MODIFY USER command to change the DEFAULT ROLE.

For example:

```
MODIFY USER Emp05 as DEFAULT ROLE = HR_Pay_Upd_R;
```

Example of Using Roles (cont.)

Sup06 – is granted to HR_Pay_Upd_R role WITH ADMIN OPTION.

```
GRANT HR_Pay_Upd_R TO Emp05;                    (success)
```

Emp05 – HR_Pay_R is current role.

```
SELECT      *
FROM        HR_VM.Employee_v
WHERE       Employee_Number=100001;             (success)

UPDATE      HR_VM.Employee_v
SET         Dept_Number=1001
WHERE       Employee_Number=100001;             (success or fail?)
```

Does this statement fail for Emp05?

Emp05 – executes the following SET ROLE command

```
SET ROLE    HR_Pay_Upd_R;

UPDATE      HR_VM.Employee_v
SET         Dept_Number=1001
WHERE       Employee_Number=100001;
```

Will this UPDATE statement succeed this time?

Will this UPDATE statement succeed the next time Emp05 logs on?

Roles for Directory-Based Users

There are a couple of options in providing access rights to directory-based users.

- Map each directory user to one or more database users that already have database privileges.
- You can optionally create external roles and grant privileges to them. Then map each directory user to one or more of the external roles.

The use of roles by directory users depends on the setting of the AuthorizationSupported property:

- When the AuthorizationSupported property is set to no, directory users can log on using a username that matches a database username. They have access to roles in which the matching database username is a member.
- When the AuthorizationSupported property is set to yes, directory users are authorized privileges according to the roles (and users) they are mapped to in the directory.

Implementing Roles for Directory Authorization of Database Privileges

1. Create external roles.
2. Review directory user management options and select a user provisioning strategy.
3. Create one or more directory role objects with names that match Teradata Database external roles and map the roles to directory group objects.

   Since roles are assigned by mapping instead of role grants, assignments cannot include WITH ADMIN OPTION.

The system records external roles in the data dictionary, along with database roles, but when you map an external role to a directory user, the system does not insert a row in the DBC.RoleGrants table.

Roles for Directory-Based Users

What is the difference between an internal role and an external role?

- External roles are mapped to users on the directory server (e.g., LDAP).
- Internal roles are granted to users with the GRANT command.

How do roles work for directory users?

- An assigned external role overrides the default role of the permanent user on Teradata.
- To associate rights with an external role, first create the role.

```
CREATE EXTERNAL ROLE HR_Ext_Role;
```

- Use the GRANT (or REVOKE) command to assign (or take away) access rights to (or from) the role, as with internal roles.

```
GRANT SELECT, EXECUTE ON HR_VM TO HR_Ext_Role;
```

- A user may change roles to any external or internal role available to them by executing the SQL command:

```
SET ROLE role_name;
```

- External roles are identified in the DBC.Roles table. However, when you map an external role to a directory user, the system does not insert a row in DBC.RoleGrants.

Roles for Proxy Users

For proxy users that are either permanent database users or users unknown to the database, you can specify one or more roles in the GRANT CONNECT THROUGH statement that defines the proxy.

For proxy users that are also permanent database users:

- You can specify WITHOUT ROLE to use the privileges granted to the permanent user.
- You can assign row level security constraints to the permanent user or the user profile. Proxy user sessions use the profile constraints, if assigned. If no constraints are assigned in the profile, the session uses the user constraints. The user can also use the SET SESSION CONSTRAINT command to access any assigned security constraints.

The following chart lists characteristics of roles for Proxy Users.

CONNECT THROUGH option	**SET QUERY_BAND option**	**Proxy connection role**
With roles	PROXYROLE = *rolename*	Rolename, if it has been specified in GRANT CONNECT
	PROXYROLE = ALL	All roles in the GRANT privilege
	PROXYROLE = NONE or PROXYROLE = NULL	Not permitted
	PROXYROLE is not used	All roles in the GRANT privilege
Without roles	PROXYROLE = *rolename*	Rolename, if it is granted to the permanent user
	PROXYROLE = ALL	All permanent user's roles
	PROXYROLE = NONE or PROXYROLE = NULL	NULL
	PROXYROLE is not used	Permanent user's default role

Roles for Proxy Users

Proxy users use standard internal roles, if these are specified in the WITH ROLE clause of the GRANT CONNECT THROUGH statement or associated with the permanent proxy user.

- A proxy user is an end user that logs on to the database through a trusted user application. The system identifies and authorizes the user an individual.
- Proxy users can be either permanent database users or other end users unknown to the database.

How are roles associated with a proxy user?

- Use the GRANT CONNECT THROUGH statement to grant the application the privilege to connect as the application users:

```
GRANT CONNECT THROUGH Application1_User TO App_End_User1 [WITH ROLE HR_Role];
```

- Each time the application performs a request for an application user, it must issue the SET QUERY_BAND statement before its SQL queries:

```
SET QUERY_BAND = 'PROXYUSER=App_End_User1;[PROXYROLE=HR_Role;]' FOR SESSION;

SELECT FirstName, LastName FROM HR_VM.Employee;
...

SET QUERY_BAND = NONE FOR SESSION;
```

RoleInfo View

The DBC.RoleInfo (or DBC.RoleInfoV) views list all of roles, their creators, and the creation timestamp in the system. This information is taken from the DBC.Roles and the DBC.Dbase tables.

The DBC.RoleInfoX (or DBC.RoleInfoVX) views only return roles that a user has created. Users that can create roles need the system access right – CREATE ROLE.

Extension to COMMENT command:

COMMENT [ON] ROLE <role_name> [[AS] <comment string>]

- Inserts or retrieves comments in CommentString column of the DBC.Roles table for the named role.

Example:

COMMENT ON ROLE HR_R AS 'SEL and EXE rights for HR_VM';

RoleInfo View

Provides information about roles.

DBC.RoleInfo[V][X]

RoleName	CommentString	CreatorName
CreateTimeStamp	ExtRole	

Example: List role names that exist in the system.

```
SELECT      RoleName, CreatorName, CreateTimeStamp
FROM        DBC.RoleInfoV
ORDER BY    1;
```

Results:

RoleName	CreatorName	CreateTimeStamp
HR_Pay_R	Sysdba	2011-01-24 17:25:41
HR_Pay_Upd_R	Sysdba	2011-01-24 17:25:44
HR_R	Sysdba	2011-01-24 17:25:02
HR_Upd_R	Sysdba	2011-01-24 17:25:19
Pay_R	Sysdba	2011-01-24 17:25:34
Pay_Upd_R	Sysdba	2011-01-24 17:25:37

RoleMembers View

The DBC.RoleMembers (or DBC.RoleMembersV) views list each role and all of its members.

The DBC.RoleMembersX (or DBC.ROleMemebersVX) views list all roles, if any, directly granted to the user.

For example, Emp05 executes the following statement:

```
SELECT      RoleName, Grantor, WhenGranted, DefaultRole, WithAdmin
FROM        DBC.RoleMembersX
ORDER BY  1;
```

The result is:

RoleName	Grantor	WhenGranted	DefaultRole	WithAdmin
HR_Pay_R	Sysdba	2010-01-24 17:32:51	Y	N

RoleMembers View

Provides information about roles and its members.

DBC.RoleMembers[V][X]

RoleName	Grantee	GranteeKind	Grantor
WhenGranted	DefaultRole	WithAdmin	

Example: List roles and the members that have access to the HR database.

```
SELECT    RoleName, Grantee, GranteeKind, DefaultRole, WithAdmin
FROM      DBC.RoleMembersV
WHERE     RoleName IN ('HR_Pay_R', 'HR_Pay_Upd_R', 'HR_R' ,'HR_Upd_R')
ORDER BY  1, 2;
```

Results:

RoleName	Grantee	GranteeKind	DefaultRole	WithAdmin
HR_Pay_R	DBC	User	N	Y
HR_Pay_R	Emp05	User	Y	N
HR_Pay_R	Sysdba	User	N	Y
HR_Pay_Upd_R	DBC	User	N	Y
HR_Pay_Upd_R	Sup06	User	Y	Y
HR_Pay_Upd_R	Sysdba	User	N	Y
HR_R	DBC	User	N	Y
HR_R	Emp01	User	Y	N
HR_R	Emp02	User	Y	N
HR_R	HR_Pay_R	Role	N	N
HR_R	Sysdba	User	N	Y
HR_Upd_R	DBC	User	N	Y
HR_Upd_R	HR_Pay_Upd_R	Role	N	N
HR_Upd_R	Sysdba	User	N	Y

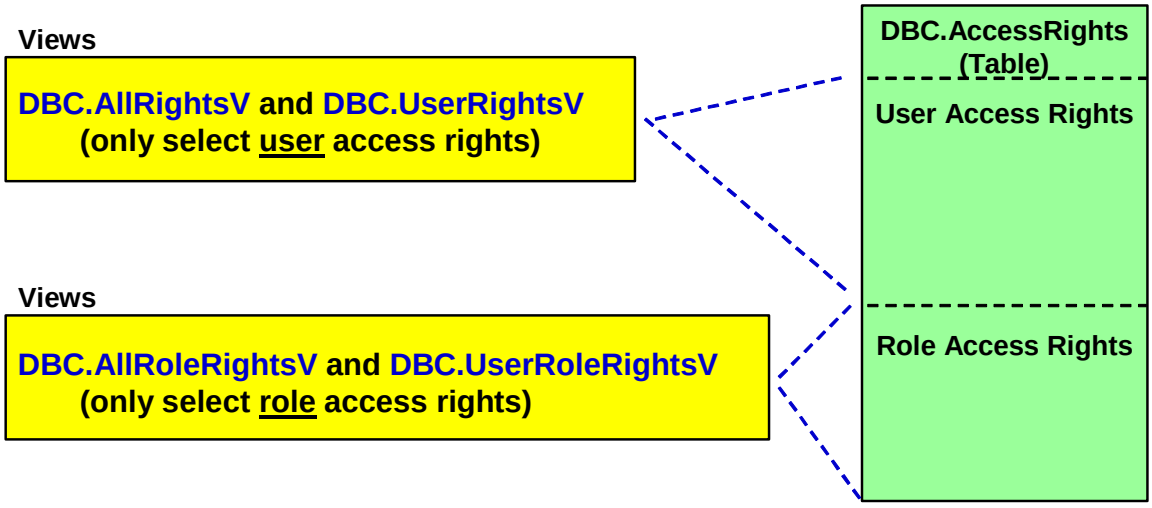
DBC.AccessRights and “Rights” Views

The facing page illustrates the difference between the various “Rights” views of the DBC.AccessRights table.

DBC.AccessRights and "Rights" Views

AllRights[V][X] – lists all rights granted to users in the system.
UserRights[V] – lists all rights granted to the current user.

AllRoleRights[V] – lists all rights granted to roles in the system.
UserRoleRights[V] – lists all rights granted to the enabled roles of the user.



AllRoleRights and UserRoleRights Views

The DBC.AllRoleRights[V] and DBC.UserRoleRights[V] views provide information about role and access rights granted to roles in the system.

DBC.UserRoleRights[V] view lists all rights granted to the current role of the user and its nested roles.

AllRoleRights and UserRoleRights Views

AllRoleRights[V] – lists all rights granted to roles in the system.
UserRoleRights[V] – lists all rights granted to the enabled roles of the user.

DBC.AllRoleRights and DBC.UserRoleRights

RoleName	DatabaseName	TableName	ColumnName
AccessRight	GrantorName	CreateTimeStamp	

Example: List current role rights.

```
SELECT    RoleName, DatabaseName, TableName, ColumnName, AccessRight
FROM      DBC.UserRoleRightsV
ORDER BY  1;
```

Example Results for Emp01:

RoleName	DatabaseName	TableName	ColumnName	AccessRight
HR_R	HR_VM	All	All	R
HR_R	HR_VM	All	All	E

Example Results for Emp05:

The default role of Emp05 is HR_Pay_R which has 2 nested roles.

HR_R and Pay_R

RoleName	DatabaseName	TableName	ColumnName	AccessRight
HR_R	HR_VM	All	All	R
HR_R	HR_VM	All	All	E
Pay_R	Payroll_VM	All	All	R
Pay_R	Payroll_VM	All	All	E

Steps to Implementing Roles

The facing page identifies a sequence of steps to consider when implementing roles. A sample query and results are also provided.

Steps to Implementing Roles

1. Identify individual rights to be converted into role rights.
2. Create roles and grant appropriate rights to each role.
3. Grant roles to users and assign users their default roles.
4. Revoke from users individual rights that have been replaced by role rights.

Sample query to identify individual rights that may be good candidates for conversion to roles.

```
SELECT    DatabaseName,
          TVMName,
          COUNT(*)          AS RightsCount
FROM      DBC.AccessRights  AR,
          DBC.TVM           TVM,
          DBC.DBase         DBASE
WHERE     AR.tvmid = TVM.tvmid
AND       AR.databaseid = DBASE.databaseid
AND       AR.fieldid = 0
GROUP BY  DatabaseName, TVMName
ORDER BY  3 DESC;
```

Results:

DatabaseName	TVMName	RightsCount
DS	All	110
QCD	All	86
HR_Tab	All	72
HR_VM	All	68
Payroll_VM	All	67
Payroll_Tab	All	67

Summary

The facing page summarizes some important concepts regarding this module.

A summary of the rules for using roles are as follows:

- You can grant one or more roles to one or more users and/or roles; thus:
  - A role can have many members
  - A user or role can be a member of more than one role
- Only single-level nesting is allowed; that is, a role that has a member role cannot also be a member of another role.
- An access privilege granted to an existing role immediately affects any user and/or role that is specified as a recipient in the GRANT statement and currently active within a session.
- The privileges of a role granted to another role are inherited by every user member of the grantee role.
- When a user logs on, the assigned default role is the initial current role for the session and is used to authorize access after all checks against individually granted rights have failed.
- Once the session is active, the user can submit a SET ROLE statement to change or nullify the current role.

Summary

- A role is simply a collection of access rights.
- Rights are first granted to a role and the role is then granted to users.
  - CREATE ROLE *role_name*;
  - GRANT *access_right* ON *object* TO *role_name*;
  - GRANT *role_name* TO *user_name*;

Module 47: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 47: Review Questions

Answer the following questions:

1. List 3 advantages of utilizing roles and profiles.

   _____
   _____
   _____

2. How many levels of role nesting are currently allowed? ____

3. True or False. A user can use the SET ROLE command to set their current role to any defined role in the system.

4. True or False. Roles may only be granted to users and other roles.

Match each term to the definition.

___	1. WITH ADMIN OPTION	a.	Established by the SET ROLE command
___	2. CREATE ROLE	b.	Lists the roles currently in the system
___	3. DBC.RoleInfo	c.	System access right needed to create a role
___	4. DBC.UserRoleRights	d.	Lists all of a user's role rights – including nested roles
___	5. DEFAULT ROLE	e.	Allows the user to assign other users to the role
___	6. Current role	f.	Option with the MODIFY USER statement

Lab Exercise 47-1

The following page continues this lab exercise.

Lab Exercise 47-1

Lab Exercise 47-1

Purpose

In this lab, you will use Teradata SQL Assistant or Teradata Administrator to work with access rights. This lab will also provide an opportunity to use some system views.

Tasks

1. Using the DBC.AllRightsV view, find the total number of rows in the DBC.AccessRights table assigned to users.

   Total number of user rights (AllRightsV) _____

   Using the DBC.AllRoleRightsV view, find the total number of rows in the DBC.AccessRights table assigned to roles.

   Total number of role rights (AllRoleRightsV) _____

2. Using the DBC.UserRightsV view, how many access rights do you currently have?

   Total number of your user rights (UserRightsV) _____

   How do you think most of these access rights were granted? _____

   Execute the following SQL command and then recheck the number of Access Rights you have.

   ```
   CREATE TABLE Emp_Phone2 AS PD.Emp_Phone  WITH NO DATA;
   ```

   What is the total number of your user rights? _____

   How many new access rights were created? _____

Lab Exercise 47-1 (cont.)

The following page continues this lab exercise.

Lab Exercise 47-1 (cont.)

3. For your Emp_Phone2 table, use the GRANT command to give the SELECT access right to the database AP.

   Use the GRANT command to give the SELECT WITH GRANT access right to the database PD.

   Check the total number of user rights returned _____

   Did this count change? _____

   If not, why not? _____

   Use the DBC.UserGrantedRightsV view to show any user rights that you may have explicitly granted.

   How many rows are returned with this view? ____

Lab Exercise 47-2

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

DBC.RoleInfo[V][X]

Lab Exercise 47-2

Lab Exercise 47-2

Purpose

In this lab, you will use Teradata SQL Assistant or Teradata Administrator to use roles. This lab will also provide an opportunity to use the RoleInfoV, RoleMembersV, and UserRoleRightsV views.

What you need

Tables from PPI exercise and a user account with system role and profile privileges.

Tasks

1. Three roles are available for your use. The role names have your user name incorporated into them. Your role names will be unique in the system. For example, if your user name is "student102", then your role names are:

Role1_102	Role1_xxx (note - this role may be referred to as "Role1" throughout this lab).
Role2_102	Role2_xxx
Role3_102	Role3_xxx

Using the DBC.RoleInfoV view, what is the total number of roles defined in the system? _____

Using the DBC.RoleInfoVX view, what is the number of roles that you have created? _____

Using the DBC.RoleMembersVXview, what is your (studentxxx) default role? _____

Using the DBC.RoleMembersVXview, which roles do you have the "With Admin" option?

_____ _____ _____

Lab Exercise 47-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

DBC.ProfileInfo[V][X]

ProfileName	**DefaultAccount**	**DefaultDB**
SpoolSpace	**TempSpace**	**ExpirePassword**
PasswordMinChar	**PasswordMaxChar**	**PasswordDigits**
PasswordSpecChar	**PasswordRestrictWords**	**MaxLogonAttempts**
LockedUserExpire	**PasswordReuse**	**CommentString**
CreatorName	**CreateTimeStamp**	**LastAlterName**
LastAlterTimeStamp		

Lab Exercise 47-2 (cont.)

2. Grant the following access rights to the specified roles as follows:

Access Rights	Tables	Role Name
SELECT	Orders, Orders_2012	Role1_xxx
SELECT	Orders_PPI	Role2_xxx
INSERT, UPDATE, DELETE	Orders_PPI	Role3_xxx

3. Grant these roles as following:

```
Grant Role1_xxx to studentxxx_A;
Grant Role2_xxx to studentxxx_B;
Grant Role2_xxx to Role3_xxx;
Grant Role3_xxx to studentxxx_B;
```

4. Modify your two users to set their default role as follows:

User name:	studentxxx_A (where xxx is your student number)
Default Role:	Role1_xxx
User name:	studentxxx_B (where xxx is your student number)
Default Role:	Role2_xxx

Lab Exercise 47-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

DBC.ProfileInfo[V][X]

ProfileName	**DefaultAccount**	**DefaultDB**
SpoolSpace	**TempSpace**	**ExpirePassword**
PasswordMinChar	**PasswordMaxChar**	**PasswordDigits**
PasswordSpecChar	**PasswordRestrictWords**	**MaxLogonAttempts**
LockedUserExpire	**PasswordReuse**	**CommentString**
CreatorName	**CreateTimeStamp**	**LastAlterName**
LastAlterTimeStamp		

Lab Exercise 47-2 (cont.)

5. Logon as "studentxxx_A" and execute the following SQL statements and indicate if SELECT is allowed or not.

   SELECT COUNT(*) FROM Orders; Permitted or not? _____
   SELECT COUNT(*) FROM Orders_PPI; Permitted or not? _____

6. As "studentxxx_A", use the DBC.RoleMembersVX and DBC.UserRoleRightsV views to view role information about this user.

   How many roles are available to studentxxx_A? _____

   What is the default role for studentxxx_A? _____

   Does studentxxx_A have the "With Admin" option on any roles? ____

   How many user role rights are available to studentxxx_A? _____

OPTIONAL

7. Logon to Teradata as "studentxxx_B".

   If prompted, set the password to a new value.

Lab Exercise 47-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

DBC.UserRoleRights[V]

RoleName	DatabaseName	TableName
ColumnName	AccessRight	GrantorName
CreateTimeStamp		

DBC.RoleMembers[V][X]

RoleName	Grantee	GranteeKind
Grantor	WhenGranted	DefaultRole
WithAdmin		

Lab Exercise 47-2 (cont.)

OPTIONAL

8. As "studentxxx_B", execute the following SQL statements and indicate if SELECT is allowed or not.

   SELECT COUNT(*) FROM Orders; Permitted or not? _____
   SELECT COUNT(*) FROM Orders_PPI; Permitted or not? _____
   DELETE Orders_PPI; Permitted or not? _____

9. As "studentxxx_B", use the DBC.RoleMembersVX and DBC.UserRoleRightsV views to view the current role of the user, any nested roles, and access rights for the roles.

   How many roles are available to studentxxx_B? _____
   What is the default role for studentxxx_B? _____
   Does studentxxx_B have the "With Admin" option on any roles? ____
   How many user role rights are available to studentxxx_B? _____

10. As "studentxxx_B", use the SET ROLE command to set the current role to "Role3_xxx".

    SELECT COUNT(*) FROM Orders; Permitted or not? _____
    SELECT COUNT(*) FROM Orders_PPI; Permitted or not? _____
    DELETE Orders_PPI; Permitted or not? _____

11. Log off as "studentxxx_A" and "studentxxx_B". Using your initial user logon name, DROP the two users and the profile you created.

Notes

Module 48

System Access Controls

After completing this module, you will be able to:

- **Describe where and how to control and log user access to the Teradata database.**
- **Use views and macros to limit user access to data.**
- **Design your system hierarchy structures for better security and easier maintenance.**

Notes

Table of Contents

System Access Control Levels 48-4
Teradata Access Control Mechanisms 48-6
Teradata Password Encryption 48-8
Password Security Features 48-10
Teradata Connectivity 48-12
Host Logon Processing 48-14
Objects used in Host Logon Processing 48-16
GRANT/REVOKE LOGON Statements 48-18
GRANT/REVOKE LOGON Example 48-20
Session Related Views 48-22
LogonRules View 48-24
LogOnOff View 48-26
SessionInfo View 48-28
Additional Utilities to View Sessions 48-30
Viewpoint – Query Monitor 48-32
Teradata Manager Sessions 48-34
Remote Console – Viewpoint 48-36
Structure the System 48-38
A Recommended Access Rights Structure 48-40
A Recommended Structure Using Roles 48-42
A Recommended System Hierarchy 48-44
System Access Controls Summary 48-46
Module 48: Review Questions 48-48

System Access Control Levels

The mission of security administration on a Teradata system is to:

- Prevent unauthorized persons from gaining access to the RDBMS and its resources.
- Permit legitimate users access to only those resources they are authorized to use.

A variety of mechanisms provide security to the data stored on a Teradata system.

Access Control Levels

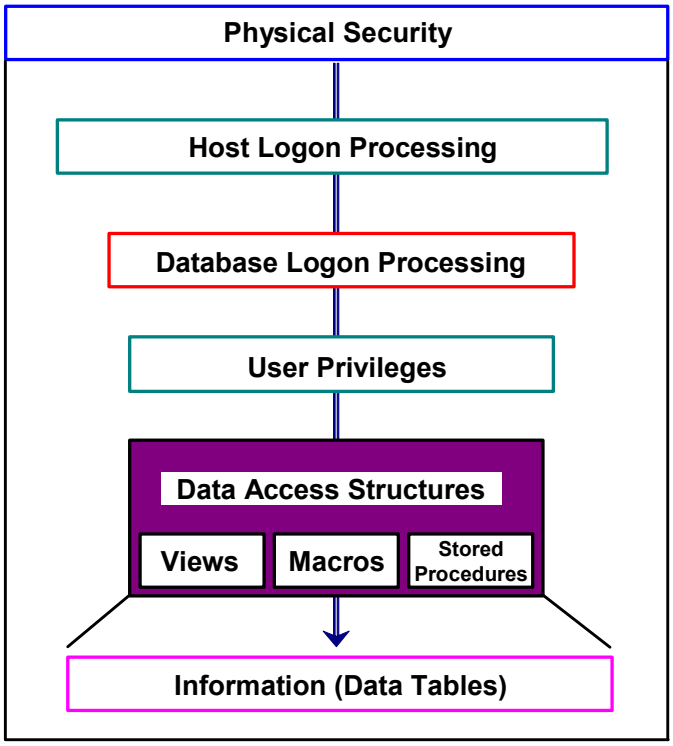
This lesson introduces a guideline for how to determine user access rights and explains how Teradata verifies user access rights.

There are three levels of access controls for the Teradata database:

Physical Security	Physical security pertains to the actual building or computer room in which the Teradata system resides. The system's owner designs and implements physical security.
Host Logon Processing	Host logon processing is the first level of access control and allows or disallows connection between the host and database systems. It involves a host ID and password. This level controls access to the host system.
Database Logon Processing	Database logon processing is the second level of access control and determines access to the Teradata system. This level employs a username and password. It controls access to the Teradata system itself.

Data access structures (views, macros and tables) are discussed later in this module.

System Access Control Levels



Teradata Access Control Mechanisms

You can control user access by granting access to specific views and macros. Views limit user access to table columns or rows that may contain sensitive information. Macros limit the types of actions a user can perform on the columns and rows.

User Privileges

During a session, the Teradata Database system accesses the user's default database to search for or store the occurrence of an object whose reference in the SQL statement is not qualified with a database name.

The user can override the default for a particular object by qualifying statement references with a database name (in the form databasename.objectname).

At any time during the session, the user can override the current default by executing the SQL DATABASE statement. The system uses the space associated with the specified or default database as the default until the user executes another DATABASE statement or logs off.

An arrangement of predefined privileges or access rights control the user's activities during a session. Access rights are associated with a user, a database, and an object (table, view, or macro).

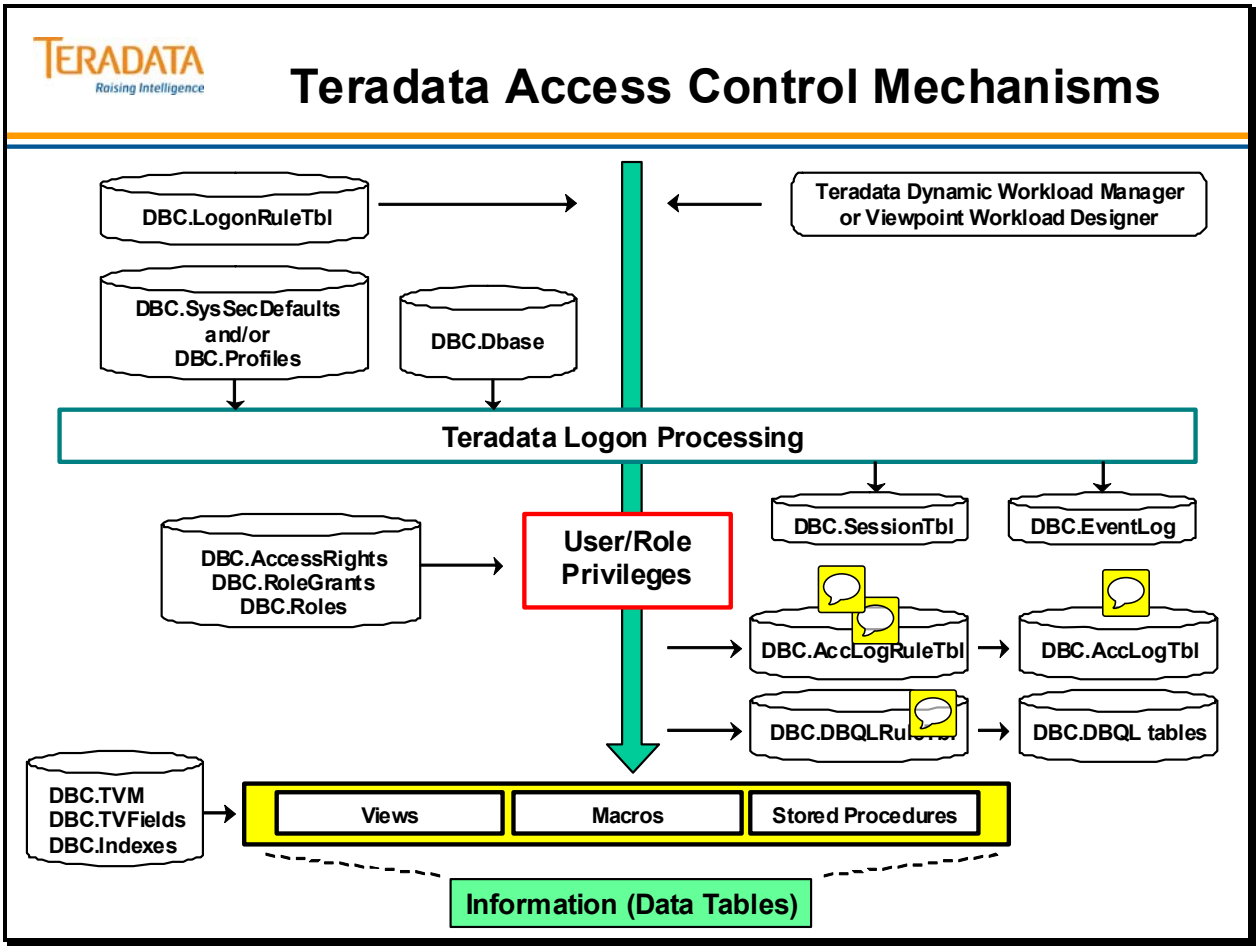
The system verifies a user's access rights when the user attempts to access or execute a function that accesses an object. Teradata stores access rights information in the system table DBC.AccessRights. You can retrieve this information by querying the DBC.UserRights view.

As the administrator, there are two additional methods you can use to limit user access to the Teradata Database:

- Create views
- Create macros and/or stored procedures

The facing page shows a diagram of access control mechanisms in Teradata.

Teradata Access Control Mechanisms



Teradata Password Encryption

You can give access to the Teradata database with the CREATE USER statement, which identifies a username and, usually, a password value.

Although the username is the basis for identification to the system, it is not usually protected information. Often the username is openly displayed during interactive logon, on printer listings, and when session information is queried.

To protect system access, associate a password with the username. Teradata does not display or print passwords on listings, terminals or PC screens.

> **Note:** Neither you nor other system users should ever write down passwords or share them among users.

Teradata stores password information in encrypted form in the DBC.Dbase system table. Information stored in the table includes the date and time a user defined a password, along with the encrypted password. As the administrator, you may modify passwords temporarily when the PasswordLastModDate plus a fixed number has been reached. This allows you to ensure that users change their passwords regularly.

To establish a session on the Teradata system, a user must enter a username at logon. Upon successful logon, the username is associated with a unique session number until the user logs off.

To supervise and enforce users' access rights to stored data, the system associates each username with a default storage area and an arrangement of access rights.

Displaying Passwords

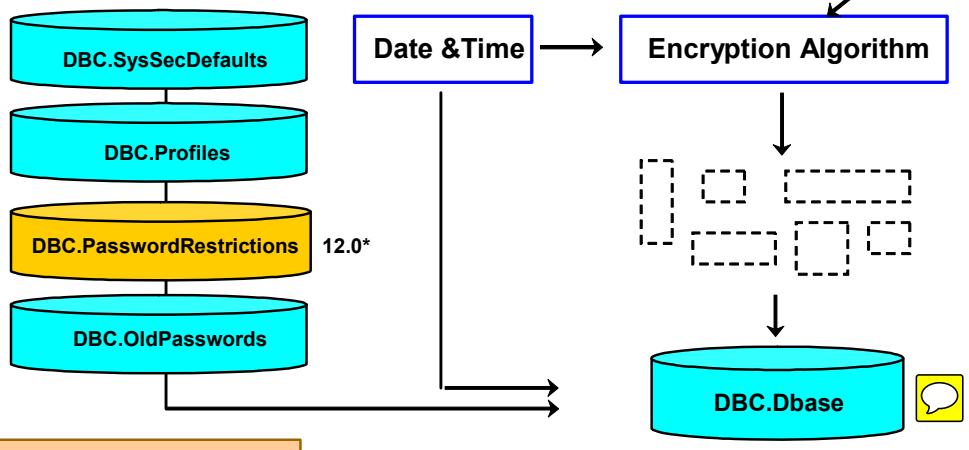
The PasswordString column from the DBC.Dbase table displays encrypted passwords. The SQL request on the facing page demonstrates how you can access an encrypted password. Note that a password cannot be decrypted.

DBC.Users View

The DBC.Dbase table stores the date and time a user defines or modifies a password. The DBC.Users[V] view displays PasswordLastModDate and PasswordLastModTime. A user can modify his or her password without additional access privileges.

Teradata Password Encryption

CREATE USER tfact03 AS PERM=0, SPOOL=500E6, PASSWORD = secure123 ;



* Table of words that cannot appear in a password.

```
SELECT    DatabaseName,
          EncryptedPassword
FROM      DBC.Dbase
WHERE     DatabaseName = 'tfact03' ;
```

DatabaseName	EncryptedPassword
tfact03	-†kTªí□Dzpl‘0]ùs?f¸˜□□ô□Ô³pd@%Û

Password Security Features

Teradata password security features allow you to:

- Expire passwords after a specific number of days.
- Define the amount of time to elapse before a password can be reused.
- Control minimum/maximum length of password.
- Disallow digits/special characters in a password.
- Limit the number of erroneous logon attempts before the system locks a user's access.
- Automatically unlock users after a specific period of time.

You can enable these features by updating the appropriate row in the DBC.SysSecDefaults table as shown on the facing page. The DBC.SecurityDefaults[V] view can also be used to view/update this table. After modifying this table, it is necessary to restart Teradata for the changes to be in effect.

When you create a new user, you also create a temporary password for the user. When the user logs on for the first time, he or she is prompted to change the password.

If a user forgets the password, you can assign a new temporary password. [As another option, you can set user passwords not to expire.]

If you attempt to set the PasswordMinChar attribute equal to 0, Teradata will assume a value of 1.

Note: If MaxLogonAttempts is set to a value other than zero, and if the time interval for locking users after erroneous attempts is left at zero, then the user is never locked.

Options that can be placed in the PasswordSpecChar column include:

Option **PasswordSpecChar**	N	Y	A	B	C	D	E	F	G	H	I	J	K	L	M	O	P	R
RULE **Username**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**
RULE **Upper/ Lower**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**R**	**R**	**R**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**R**	**R**	**R**
RULE **One Alpha**	**Y**	**Y**	**Y**	**R**	**R**	**R**	**R**	**R**	**R**	**Y**	**Y**	**Y**	**R**	**R**	**R**	**R**	**R**	**R**
RULE **Spec Chars**	**N**	**Y**	**R**	**N**	**Y**	**R**	**N**	**Y**	**R**	**N**	**Y**	**R**	**N**	**Y**	**R**	**N**	**Y**	**R**

Password Security Features

The DBC.SecurityDefaults[V] (view) can be used to view/update DBC.SysSecDefaults table.

ExpirePassword	Number of days to elapse before the password expires. Zero (0) indicates passwords do not expire; default is 0.
PasswordMinChar	Minimum number of characters in a valid password string; default is 1.
PasswordMaxChar	Maximum number of characters in a valid password string; default is 30.
PasswordDigits	Indicate if digits are to be allowed in the password (Y, N, or R); default is Y; (R – one or more digits are required in password).
PasswordSpecChar	Indicate if special characters are allowed in the password (Y or N); default is Y; (Options – A to P, R – options provide for more secure passwords).
PasswordRestrictWords	Indicate whether or not a password is subject to the content restrictions (Y or N); default is N.
MaxLogonAttempts	Number of erroneous logons allowed before locking user. Zero (0) indicates that user is never locked; default is 0 - max is 32,767.
LockedUserExpire	Number of minutes to elapse before a locked user is unlocked. Zero (0) indicates immediate unlock; -1 = locked indefinitely; default is 0 - max is 32,767.
PasswordReuse	Number of days to elapse before a password can be reused. Zero (0) indicates immediate reuse; default is 0 - max is 32,767.

To change system-wide password security features:

1. UPDATE this view or table with the desired values
2. Restart Teradata (required)

Teradata Connectivity

Teradata utilities and software programs support Teradata database access in both mainframe and LAN environments. Utilities and programs run under the client's operating system and provide the functionality for a user to access the database system.

When a system is configured, host numbers are assigned to different channel and LAN connections. It is possible to enable/disable user access from specific host numbers.

Channel Environment

The Teradata Channel Interface is an architecture that enables communication between a mainframe client and a Teradata server using a channel with either an ESCON or FICON channel interface.

With Teradata servers, the nodes use I/O adapters such as the PXSA4 (PCI-X Bus ESCON Adapter) to connect to an ESCON channel or the PCI-X FICON Adapter (PXFA) to connect to a FICON channel. TDP software executes on the mainframe and communicates with the PE software executing within Teradata.

LAN Environment and Teradata Gateway Software

In a local area network (LAN) environment, each workstation on the network will have the utilities and programs needed to access the Teradata database. A network interface card connects workstations directly to the LAN. An Ethernet card in a PCI slot within the processing node connects the node directly to the LAN. These connections provide the workstation operating system access to the gateway software in the node.

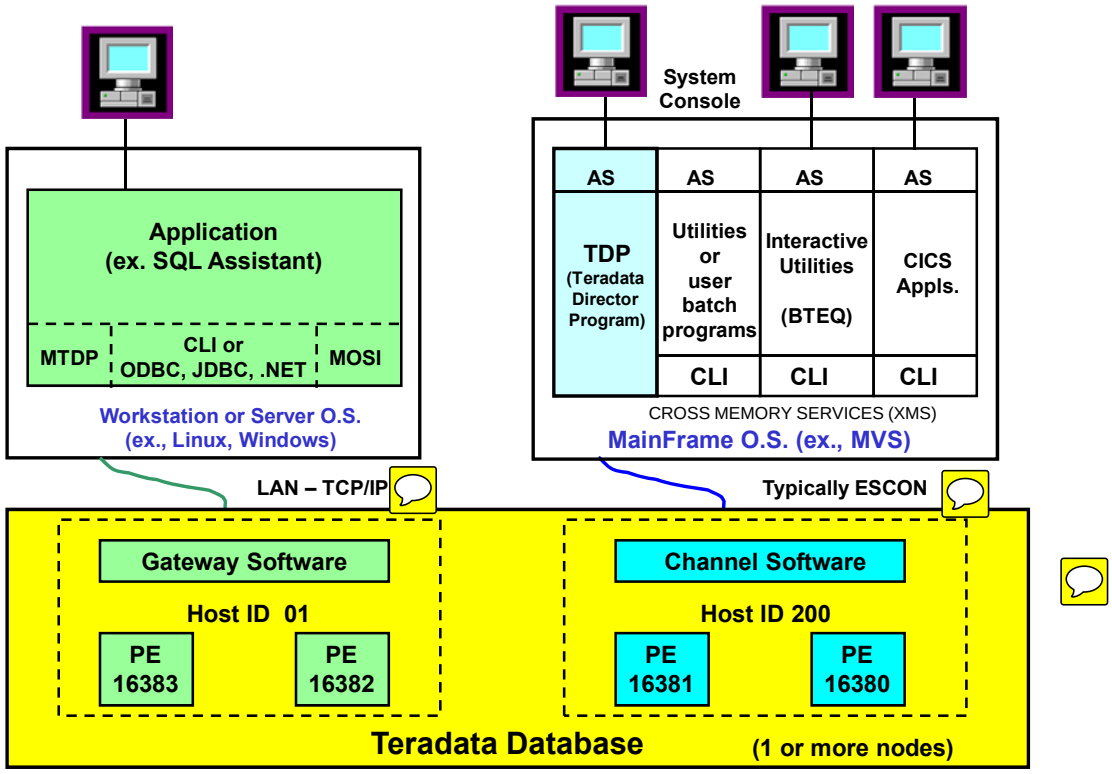
The gateway software runs on the Teradata server that is running the Teradata Database. Client programs that communicate through the gateway to the Teradata Database may be resident on the system, or may be installed and running on network-attached workstations.

When a system is configured, it is possible to assign different hostids to different LANs (Ethernet connections) coming into a system. By having multiple hostids (for LANs), a customer can enable/disable a specific LAN. An example might be that you have east/west coast users on different LANs. You can disable the west coast users as a group. If you have multiple LAN hostids, you are effectively setting up "multiple" gateways. Gateway software will balance the number of sessions between the PEs assigned to the hostid for the LAN.

Most customers have multiple Ethernet connections across multiple nodes, but only one hostid is assigned to all LAN connections and there is effectively one gateway in the system. Usually the hostid for LANs has a value of 1; older systems often used a value of 52.

Teradata’s gateway software supports up to 1200 sessions per node, depending on available system resources. Gateway errors are handled in the same manner as other database errors.

Teradata Connectivity



Host Logon Processing

The Teradata system default is that any defined user with a valid password who is logged on to a host machine has permission to access the Teradata server through any identified client connection. After installing the software, you may restrict access to the server by associating individual users with specific hosts.

GRANT/REVOKE LOGON Statements

Use the GRANT LOGON statement to give users permission to log on to the Teradata RDBMS from one or more specific client systems. Use the REVOKE LOGON command to retract permission to log on to the Teradata database from one or more specific client systems. These two commands store rows in the DBC.LogonRuleTbl.

You must have EXECUTE privileges on the macro DBC.LogonRule to execute either of these commands.

After installation, use the REVOKE LOGON statement to change the system default by first removing access privileges from all users from all hosts. Then, you can submit the GRANT LOGON statement to assign individual users to specific host IDs.

You can execute the GRANT or REVOKE LOGON statements any time after installation to add or remove user names on individual host connections as needed.

Host Logon Processing

The default is that any authorized user can access Teradata through any identified client connection only if they provide a valid password.

Optionally, an administrator can ...

- **grant or deny users permission to logon to Teradata from specific client connections.**
- **give users permission to logon to Teradata from specific host connections using a NULL password.**

The following statements are used to control access from specific "host ids".

- **GRANT LOGON statement**
  - **Gives users permission to logon to Teradata from specific client connections and optionally use a pre-validated logon request.**
- **REVOKE LOGON statement**
  - **Denies users permission to logon to Teradata from client system(s).**

Objects used in Host Logon Processing

You must have EXECUTE privileges on the macro DBC.LogonRule to execute the GRANT LOGON and REVOKE LOGON statements.

Note: DBC.LogonRule is a “dummy macro”. It only has a ; in it.

The GRANT LOGON and REVOKE LOGON statements store rows in the DBC.LogonRuleTbl.

To view the rows in this table use the DBC.LogonRules view.

Objects used in Host Logon Processing

Users who are granted the EXECUTE permission on the following macro can use the GRANT LOGON and REVOKE LOGON statements.

Example:

```
GRANT EXECUTE ON DBC.LogonRule
TO Sysdba;
```

This allows "Sysdba" to execute the GRANT LOGON and REVOKE LOGON statements.

DD/D Macro

DBC.LogonRule

Execution of GRANT LOGON or REVOKE LOGON statements causes rows (representing the rules) to be added or updated in ...

DD/D Table

DBC.LogonRuleTbl

To view the rules in this table, SELECT from this view.

DD/D View

DBC.LogonRules[V]

GRANT/REVOKE LOGON Statements

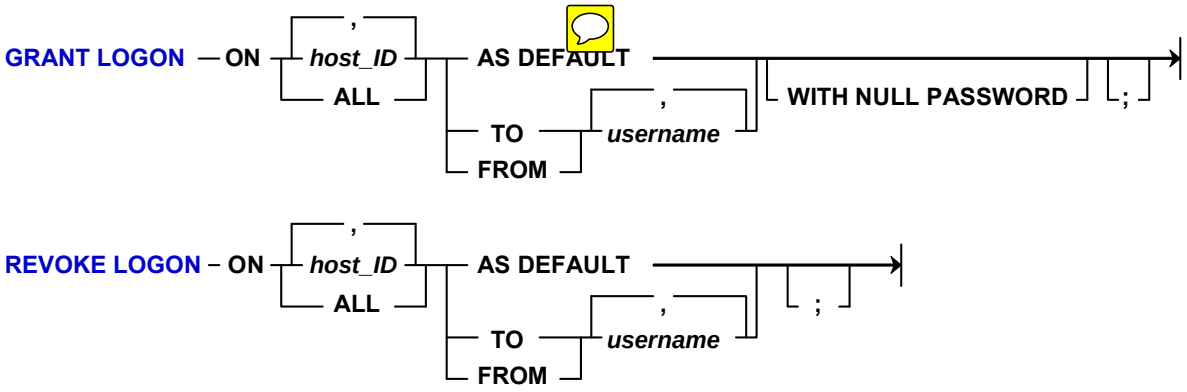
GRANT LOGON and REVOKE LOGON are flagged as non-ANSI when the SQL Flagger is enabled.

Keywords

Keywords you can use with the GRANT and REVOKE LOGON commands include:

HostID	Identifies a mainframe channel connection or a local area network connection that is currently defined to the Teradata RDBMS by the hardware configuration data. The host ID for the Teradata database console is zero (0). For any other connector, the host ID is a value from 1 to 1023.
ALL	The ALL keyword, used in place of a host ID, applies to any source through which a logon is attempted, including the Teradata database console. This is shown as host ID 1024.
AS DEFAULT	Specifies that the current default for the specified host ID(s) is to be changed as defined in this GRANT LOGON statement. A statement with AS DEFAULT has no effect on the access granted to or revoked from particular user names.
TO or FROM username(s)	Overrides the current default for the specified username(s) on the specified host ID(s). The name DBC cannot be specified as a username in a GRANT LOGON statement. A statement that includes this name will return an error message.
WITH NULL PASSWORD	The initial Teradata database default is that all logon requests must include a password. The WITH NULL PASSWORD option, in conjunction with a TDP security exit procedure, permits a logon string that has no password to be accepted on a Teradata system.

GRANT/REVOKE LOGON Statements



host_ID	Host number from configuration data. The database console is host number "0" (zero). ALL is represented as "1024".
AS DEFAULT	Changes the default for the specified host.
username	You can specify up to 25, but not "DBC".
WITH NULL PASSWORD	When used in conjunction with a TDP exit or with single sign-on in Windows 2000, overrides the system default that a password is required.

To execute a GRANT or REVOKE LOGON statement, you must hold execute privileges on the DBC.LogonRule macro.

GRANT/REVOKE LOGON Example

The facing page contains an example of using the REVOKE and GRANT LOGON statements.

COP Entries for LAN Connections

CLI clients work a little differently than ODBC clients. Any CLI based utility will dynamically generate a set of “cop” names at the time you try to make a connection. Example of entries in a “hosts” file:

141.206.28.01	SMP001-7	educ1	**educ**cop1
141.206.28.02	SMP001-8	educ2	**educ**cop2
141.206.28.03	SMP001-9	educ3	**educ**cop3
141.206.28.04	SMP001-10	educ4	**educ**cop4

For example, if you have a 4-node system, you can have four entries for the hostid: TDPIDcop1, TDPIDcop2, TDPIDcop3, TDPIDcop4. Where you put these “cop” entries for address resolution is up to you. COP entries for multiple hosts can be placed in the local hosts file OR in the DNS server file. Most people use DNS, since it is a central repository. If you have “cop” entries in the local hosts file AND they are also in the DNS server file, which are used?

The usual order is to first look in the local /etc/hosts file and then look at DNS server files. With UNIX MP-RAS, you can specify the order of resolution in the "/etc/netconfig" file. With Windows, the default order is to first look in the local hosts file and then escalate to the DNS.

Typically, the place to manage these cop entries is definitely the DNS server. When there are changes, it is much easier to do them in one place rather than on every machine that connects to Teradata.

When you specify a hostid in your logon, the first attempt at connection is to establish the size of the COP pool. First it looks for TDPIDcop1, then TDPIDcop2 ... when an attempt for TDPIDcopn+1 fails, the pool is established as n cops. This "cop" pool is only used to do connection balancing. Another reason for a "cop" pool is to help avoid a single point of connection failure. If the user has the host aliases in the local host file, then the DNS server doesn't get involved until copn if one has local name resolution selected before DNS resolution.

ODBC requires you to create a DSN entry that specifies the machine to connect to. When you create the DSN entry you can give the TDPID and ODBC will resolve the cop names as they exist at that time and cache them in the registry. There is an option on the screen to create the DSN that says do NOT resolve. In that case, ODBC will behave like CLI by dynamically generating the list of cops to choose from when you make a connection. Something to remember is that most user access to Teradata is via ODBC tools and most DSN entries have those IP addresses cached, so for normal client traffic, there is not a lot of copname resolution that has to be done.

GRANT/REVOKE LOGON Example

GRANT LOGON ON 01 TO tfact08;

```
Teradata BTEQ 13.10.00.04 for LINUX.
Enter your logon or BTEQ command:
.logon tdt6-1/tfact08

.logon tdt6-1/tfact08
Password:

*** Logon successfully completed.
*** Transaction Semantics are BTET.
*** Character Set Name is 'ASCII'.

*** Total elapsed time was 1 second.

BTEQ -- Enter your DBC/SQL request or BTEQ ...
.logoff

*** You are now logged off from the DBC.
```

Notes: This GRANT LOGON creates a specific logon rule in DBC.LogonRuleTbl.

If "REVOKE LOGON ON 01 AS DEFAULT;" is executed, tfact08 can still logon since individual rules override AS DEFAULT.

REVOKE LOGON ON 01 TO tfact09;

```
Teradata BTEQ 13.10.00.04 for LINUX.
Enter your logon or BTEQ command:
.logon tdt6-1/tfact09

.logon tdt6-1/tfact09
Password:

*** Error 3026 The user's right to log on
    has been revoked.
*** Error: Logon failed!

*** Total elapsed time was 3 seconds.

Teradata BTEQ 13.10.00.04 for LINUX.
Enter your logon or BTEQ command:
```

Notes: This REVOKE LOGON creates a specific logon rule in DBC.LogonRuleTbl.

A "GRANT LOGON ON 01 TO tfact09;" can be executed to allow tfact09 to logon.

Session Related Views

There are three system views that you can use to monitor database access. They are:

DBC.LogonRules[V]	Retrieves information about logon rules generated as a result of successfully processed GRANT/REVOKE LOGON statements. This view uses columns from the DBC.LogonRuleTbl.
DBC.LogOnOff[V][X]	Supplies information about logon and logoff activity. This view uses columns from the DBC.EventLog table, which records both successful and unsuccessful logon attempts.
DBC.SessionInfo[V][X]	Provides information about users who are currently logged on. This view uses columns from DBC.SessionTbl.

Dictionary Tables accessed include:

- DBC.LogonRuleTbl
- DBC.EventLog
- DBC.SessionTbl

Session Related Views

DBC.LogonRules[V]	Returns a list of logon rules generated by GRANT and REVOKE statements.
DBC.LogOnOff[V][X]	Provides information about logon attempts (successful or unsuccessful) and logoffs.
DBC.SessionInfo[V][X]	Provides information about the current user or all users currently logged on.

LogonRules View

The LogonRules view retrieves information about logon rules generated as a result of successfully processed GRANT LOGON statements. This information is stored as rows in the system table DBC.LogonRuleTbl.

This view returns information about the defined rules that you, as the administrator, specify with the GRANT LOGON statement. This statement controls access to the Teradata Database from any server or host.

System administrators or security administrators must specifically authorize user logon requests without passwords.

Example

The SQL statement on the facing page requests a list of the logon rules sorted by username. The response displays that user “tfact06” cannot log on using host ID 200. The users “tfact05 and tfact07” can log on to the database without a password.

LogonRules View

Provides information about logon rules that are created by GRANT LOGON and REVOKE LOGON statements.

Returns rules from the DBC.LogonRuleTbl.

DBC.LogonRules[V]

UserName	LogicalHostID	LogonStatus
NullPassword	CreatorName	CreateTimeStamp

Example:
List logon rules for TFACT users.

```
SELECT      *
FROM        DBC.LogonRulesV
WHERE       UserName LIKE 'tfact%'
ORDER BY    UserName ;
```

Example Results:

Username	LogicalHostID	LogonStatus	NullPassword
tfact05	102	G	T
tfact06	200	R	F
tfact07	200	G	T
tfact08	1	G	F
tfact09	1	R	F

LogOnOff View

The DBC.LogOnOff[V][X] views provide information about users who have logged on and off. You can also use this view when you need to know about a user's failed attempts to logon. The facing page shows an example of the DBC.LogonOff view.

DBC.LogOnOff event column definitions include:

	Event	**Result**
Logon	Bad User Bad Password Bad Account	Logon failed.
Logoff	Forced Off	User had their session aborted.

LogOnOff View

Provides information about logon and logoff activity, including bad logon attempts and sessions forced off.

DBC.LogOnOff[V][X]

LogDate	LogTime	UserName	AccountName
Event	LogicalHostId	IFPNo	SessionNo
LogonDate	LogonTime	LogonSource	

Example:
List "bad" logon attempts and sessions forced off during the last seven days.

```
SELECT    CAST (LogDate      AS FORMAT 'YYYY-MM-DD')
          ,LogTime
          ,CAST (UserName    AS FORMAT 'X(12)')
          ,Event
FROM      DBC.LogOnOffV
WHERE     (Event LIKE 'Bad%'
OR         Event LIKE 'Forced%')
AND       LogDate > CURRENT_DATE - 7
ORDER BY  LogDate, LogTime ;
```

Example Results:

LogDate	LogTime	UserName	Event
2011-09-23	10:02:54.84	student30	Bad User
2011-09-23	10:05:22.14	student130	Bad Password
2011-09-23	12:10:13:11	student125	Bad Account
2011-09-24	08:14:11:71	student117	Forced Off

SessionInfo View

The facing page shows an example of the DBC.SessionInfo[V][X] view.

This view provides information about users who are currently logged on the system. You can use the [X] option of this view to obtain information about the current user.

Example

LogonSource	May contain up to 11 fields, depending on the values returned. Operating system name (e.g., VM or MVS) followed by: TDP name VM user ID or MVS job name Environment name (e.g., TSO, CICS) etc. Transaction mode: T = TDBS A = ANSI Two PC mode: 2 = 2PC mode N = Non-2PC mode
Partition	DBC/SQL = an SQL session EXPORT = a FASTEXPORT session FASTLOAD = a FASTLOAD session HUTPARSE = an ARC data session MLOAD = a MULTILOAD session MONITOR = sessions running in a performance monitoring application NONE = session is recognized but not yet assigned

To get a count of load jobs that are currently executing, you can use the following SQL.

```
SELECT    COUNT(DISTINCT LogonSequenceNo) AS Utility_Cnt
FROM      DBC.SessionInfo
WHERE     Partition IN ('Fastload', 'Export', 'MLoad');
```

The LogonAcct and AccountName columns will usually have the same account id for a user. If a user changes the account id within a session, the AccountName column will reflect the current account id and the LogonAcct will have the logon (or initial) account id.

Both the LogonAcct and AccountName columns will have the actual logon account id (e.g., '$M_9038_&S&D&H'), not an expanded account id.

SessionInfo View

Returns information about the current users or all users currently logged on.

DBC.SessionInfo[V][X]

UserName	AccountName	SessionNo	DefaultDataBase
IFPNo	Partition	LogicalHostId	HostNo
CurrentCollation	LogonDate	LogonTime	LogonSequenceNo
LogonSource	ExpiredPassword	TwoPCMode	Transaction_Mode
ProfileName	CurrentRole	LogonAcct	LDAP
AuditTrailID	CurIsolationLevel (12.0)	QueryBand (12.0)	ProxyUser (13.0)
ProxyCurRole (13.0)			

Example: List all users currently logged on, their session source, the logon date, and connect time.

```
SELECT    UserName, 'from ' || CAST (LogonSource AS CHAR(55)) AS "Logon Info"
          ,CAST (LogonDate AS FORMAT 'YYYY-MM-DD')
          ,CAST (TIME - LogonTime AS FORMAT '99:99:99') AS ConnectTime
FROM      DBC.SessionInfoV
ORDER BY  UserName ;
```

UserName	Logon Info	LogonDate	ConnectTime
DBC	from (TCP/IP) BCB6 127.0.0.1 TDT6-1 13866 ROOT BTEQ 0	2011-09-23	00:25:40
DBCMANAGER	from (TCP/IP) 0540 153.65.42.35 TDT6-1 864 SYSTEM IS	2011-09-23	03:04:22
STUDENT102	from (TCP/IP) 8A84 127.0.0.1 TDT6-1 3091 LINUX102 BTE	2011-09-23	03:16:21
STUDENT103	from (TCP/IP) 0521 153.65.42.41 153.64.24.65 3624 IM	2011-09-23	03:33:11
STUDENT103	from (TCP/IP) 0507 153.65.42.41 153.64.24.65 4824 IM	2011-09-23	03:35:34
STUDENT104	from (TCP/IP) 04ED 153.65.42.202 153.64.24.65 4460 M	2011-09-23	03:19:43

Additional Utilities to View Sessions

Additional tools that may be used to view session activity are listed on the facing page.

Additional information about the gateway utility follows.

Gateway Global Utility

Gateway Global is not as commonly used as it once was because the Sessions display of Teradata Manager is easier to use. However, you can still access the Gateway Global Utility by invoking the following commands:

Command-line version: ***gtwglobal***
Command for X-version: ***xgtwglobal***

Session Control

The Gateway Global utility allows you to monitor and control Teradata database network-attached users and their sessions. For example, by starting the utility and issuing utility commands with this utility, you can monitor network sessions and traffic, disable logons, force users off the Teradata database and diagnose gateway problems.

Disconnect and Kill Commands

The Disconnect User/Session and Kill User/Session commands are similar in that they both disconnect sessions from the database. The Kill command will abort one session immediately or all sessions of a particular user, then log the user off. The Disconnect command simply puts the sessions in a disconnect state and does not log the user off. The database is still aware of the sessions, and if the user re-establishes the connection from their client workstation, the sessions are allowed to re-connect.

Examples of Gateway Global Commands

Network and Session Information

DISPLAY NETWORK	Displays your network configuration.
DISPLAY GTWALL	Displays all sessions connected to the gateway.
DISPLAY SESSION	Displays information about a specific session on the gateway.

Administering Users and Sessions

DISABLE LOGONS	Disable logons to the RDBMS through the gateway.
ENABLE LOGONS	Enable logons to the RDBMS via the gateway
DISCONNECT USER	Disconnects all sessions owned by a user.
DISCONNECT SESSION	Disconnects a specific session. Must provide the session number in the command syntax.
KILL USER	Terminates all sessions of a specific user.
KILL SESSION	Terminates a specific session. Must know session number.

Additional Utilities to View Sessions

Viewpoint – Query Monitor and My Queries

- Provides functions to view sessions and details about user sessions
- Optionally, sessions can be aborted or have their priority changed

Teradata Manager and/or Performance Monitor – Windows utilities

- Both of these utilities provide functions to view and abort sessions
- Can be used to change a session's priority
- Performance Monitor may be executed independently or via Teradata Manager

QrySessn – utility started via Supervisor (e.g., Viewpoint – Remote Console)

- Provides display of sessions; Supervisor is used to abort sessions

gtwglobal – system utility

- This utility can be used to monitor/abort only gateway or LAN-based sessions
  - gtwglobal or xgtwglogal (X Windows)
  - Not as commonly used because Viewpoint or Teradata Manager provides an easier interface

Viewpoint – Query Monitor

Teradata Viewpoint enables database and system administrators and business users to monitor and manage Teradata Database systems from anywhere using a standard web browser.

Teradata Viewpoint allows users to view system information, such as query progress, performance data, and system saturation and health through preconfigured portlets displayed from within the Teradata Viewpoint portal. Portlets can also be customized to suit individual user needs. User access to portlets is managed on a per-role basis.

Database administrators can use Teradata Viewpoint to determine system status, trends, and individual query status. By observing trends in system usage, system administrators are better able to plan project implementations, batch jobs, and maintenance to avoid peak periods of use. Business users can use Teradata Viewpoint to quickly access the status of reports and queries and drill down into details.

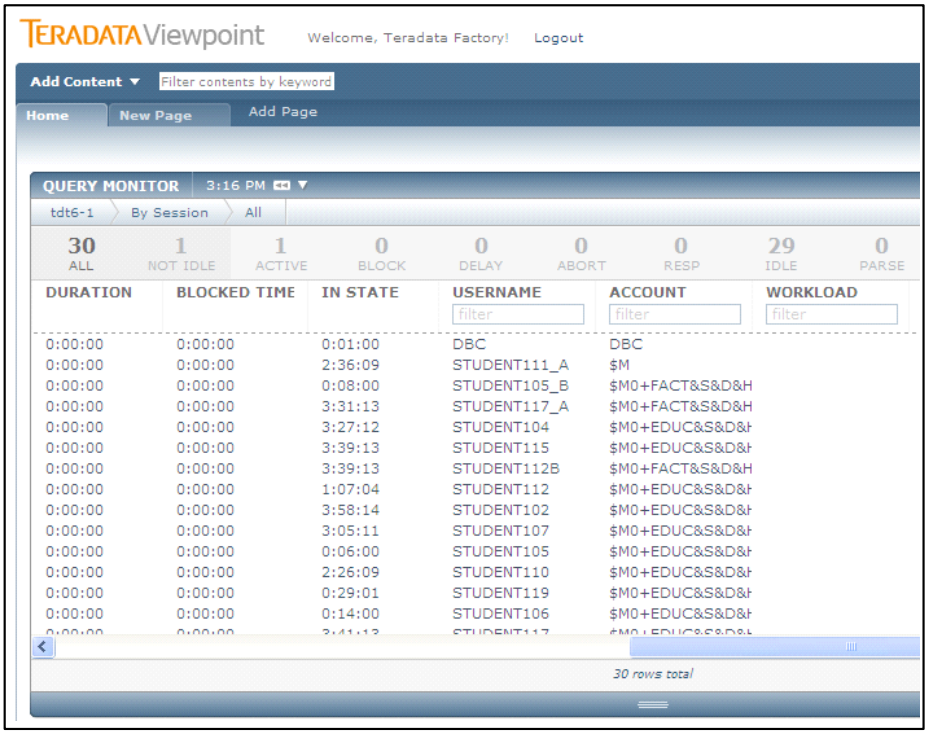
The Query Monitor portlet allows you to view information about queries running in a Teradata Database system so you can spot problem queries. You can analyze and decide whether a query is important, useful, and well written. After you have identified a problem query, you can take action to correct the problem by changing the priority or workload, releasing the query, or aborting the query or session. You can take these actions for one query or session, or multiple queries or sessions at a time.

Viewpoint – Query Monitor

The Query Monitor portlet of Viewpoint allows you to view and control sessions.

To access this portlet:

Add Content > Monitoring > Query Monitor



Teradata Manager Sessions

An example of the **Sessions** display from Teradata Manager is shown on the facing page.

From the **Teradata Manager** menu, click **Monitor > Sessions**, and choose the filter for the types of sessions to view:

- **All** - shows all sessions currently on the Database
- **Active** - shows all active sessions
- **Blocked** - shows sessions that are blocked by other sessions
- **Idle** - lists information about inactive and idle sessions
- **Parsing** - lists information about parsing sessions
- **Responding** - lists information about responding sessions
- **Aborting** - shows sessions that are in the process of aborting
- **Other** - lists information about sessions when there is a difference in the state of the AMP and PE or if the state is not Idle, Active, Blocked, Parsing, Responding, Aborting or Delayed, including those that are currently logged on to the Monitor Partition.
- **Delayed** - shows sessions that are delayed

To view session details, either double-click the session number, or right-click the number of the session to display the shortcut menu, and click Session Details.

Other options include:

- **Modify Session Priority** – modify session's account to change priority.

  Note: Modifying accounts is allowed only in the DBC/SQL partition; therefore, this option is enabled only when the partition is DBC/SQL.

- **Abort Session**- abort this session
- **Blocked By** - display a report showing which sessions (if any) are blocking this session
- **Blocking** - display a report showing which sessions this one is blocking
- **Current SQL** - view the SQL statements currently being executed by this session, along with job step information and associated Explain text
- **Skew** - display a report showing skew (workload imbalance) for this session
- **Modify Session Workload** - modify session workload settings
- **Release Request** - release a request that is on hold
- **Query Band** - display a report showing the query band pairs applied to this session

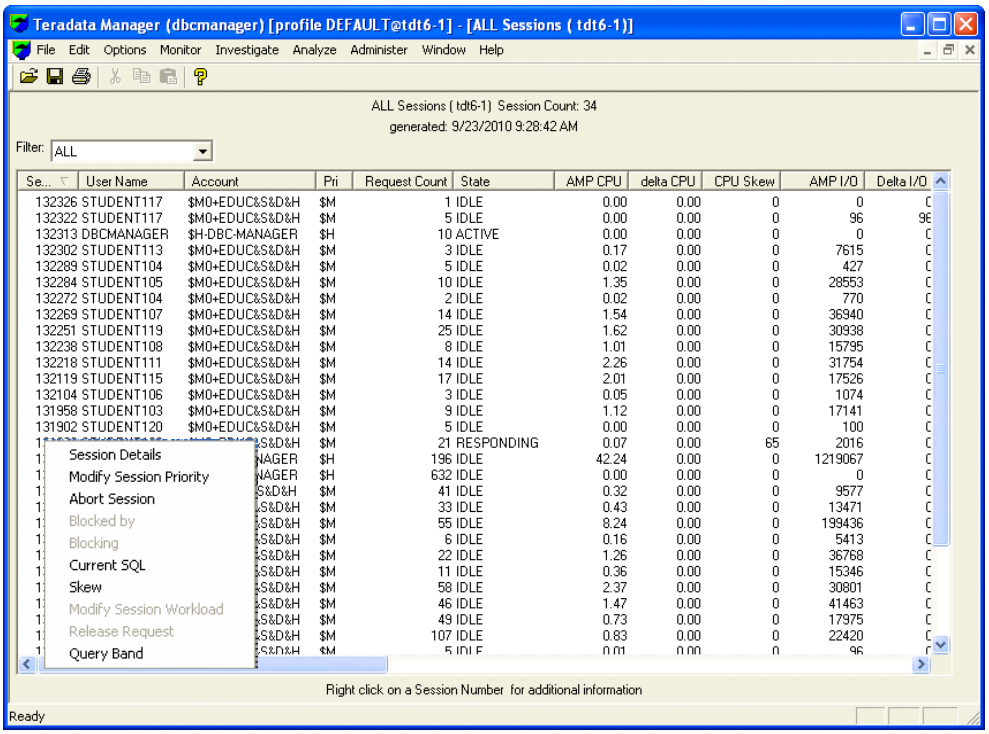
Teradata Manager Sessions

The Sessions display of Teradata Manager allows you to view and control sessions.

To access this display:

Monitor Menu > Sessions

This pull down menu can be accessed by right-clicking on the session number.



Remote Console – Viewpoint

The Query Session utility provides information about all Teradata sessions. To start the utility, enter ***START QRYSESSN*** in the Supervisor window. A Teradata session may be in one of several possible states. They include:

State	Description
Unknown	The session number is not recognized.
Idle	Process is not taking place at this time.
Delay	The query is on a Teradata DWM Delay queue.
Parsing	Session is in the DBC/SQL parser phase.
Active[1]	Session has sent steps to the dispatcher/AMPs
Aborting[1]	Session is aborting the latest request.
Blocked[2]	Session is waiting for a database lock to release.
Response	Response to session request is in process.

Archive/Recovery, FastLoad, MultiLoad, FastExport, and other utility status information is also provided.

The Query Session utility can also be started from HUTCNS by entering the following command: **SES** or **ses**

Example

The example on the facing page illustrates using the Remote Console portlet of Viewpoint to access the Query Session utility. A sample Query Session Report and the headings displayed by Query Session for this report. You can use an asterisk (*) as a wild card symbol to depict all hosts and/or all sessions. If a session is idle, only the session identifier information would be displayed.

The complete prompt for detailed information is …

"Is detailed information needed for HUT/FASTLOAD/MLOAD/EXPORT? y-yes, n-no

Answering yes to this prompt provides more details for the archive and load utilities.

[1]Shows CPU time (all AMPS) in 100ths of a second and total segment access calls.
[2]Shows if lock was requested and if the lock encountered was a host utility lock (archive).

Viewpoint
Remote Console – Query Session

Query Session prompts for:

```
Please enter logical host id?   1
Please enter session ids?       *
Is detail information needed?   y
```

From Viewpoint Operator Console, you can also abort sessions.

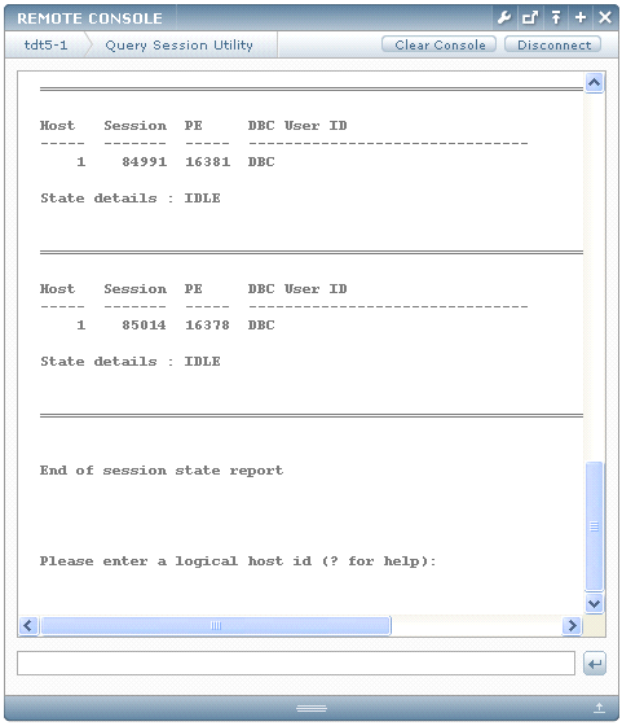
- The ABORT SESSION command aborts the currently running SQL transaction in progress.
- The LOGOFF option also terminates the user session.

```
ABORT SESSION  hostid:ses# [LOGOFF]
               hostid.username
                    *.username
               hostid.*
                    *.*
```

Why use operator console to abort sessions?

You can abort a large number of sessions quickly.

```
ABORT SESSION *.* LOGOFF
```



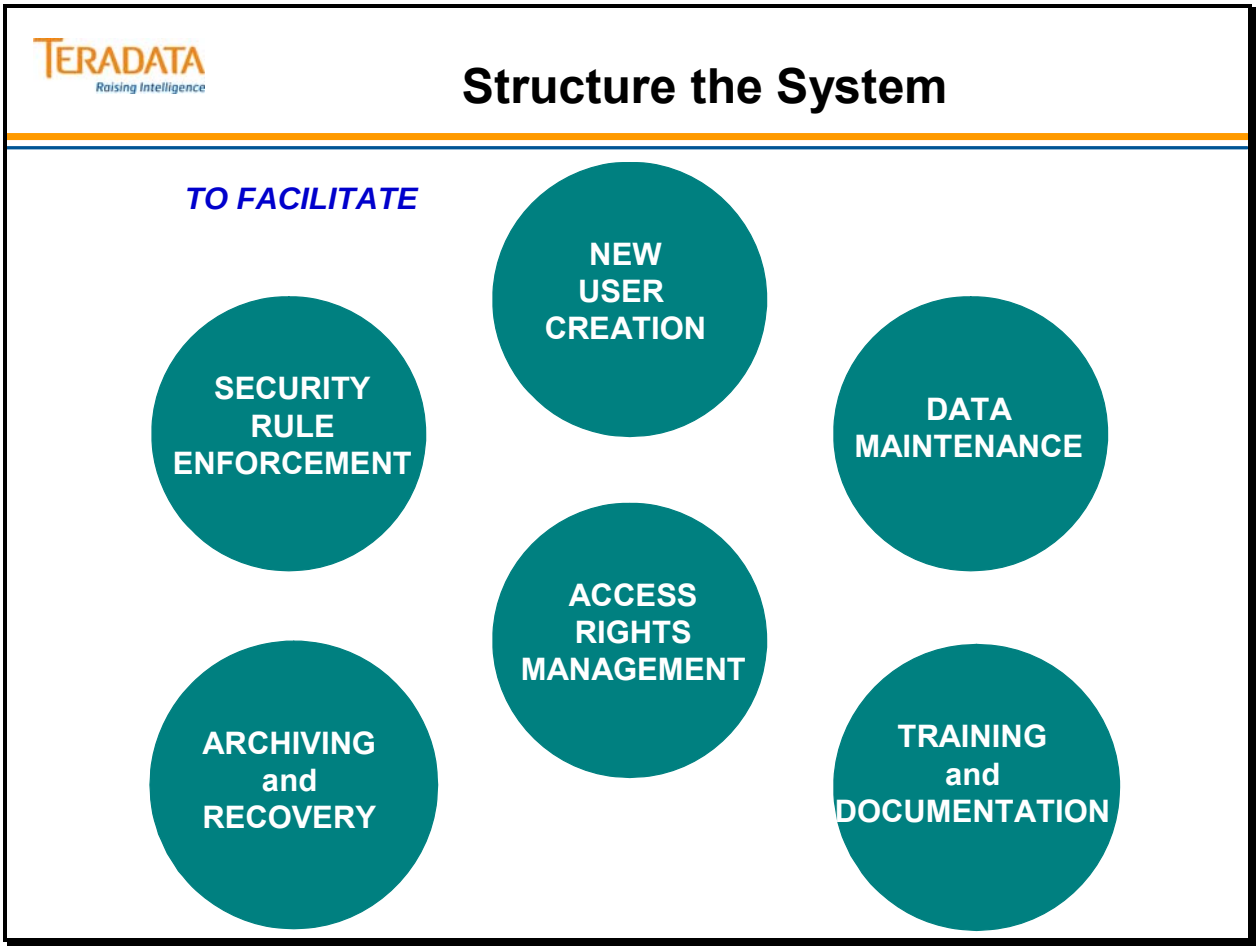
From Remote Console, Select Query Session

Structure the System

As the administrator, it is your responsibility to manage access rights. Managing access rights is important for:

- New user creation
- Security rule enforcement
- Data maintenance
- Training and documentation
- Archiving and recovery

Structure the System



A Recommended Access Rights Structure

An access rights structure recommended for the Teradata database has the following characteristics:

- All users belong to a database and inherit their access rights.
- Users do not have direct access to data tables, unless they are performing batch operations.
- Users access databases that contain only views and macros.
- VMDB databases contain *only* views and macros.
- TABLE databases contain *only* tables.
- Access rights are *only* extended at the database or user level, not at the individual table level.

Example

The diagram on the facing page illustrates an example of the suggested Teradata access rights scheme. This scheme has three user databases:

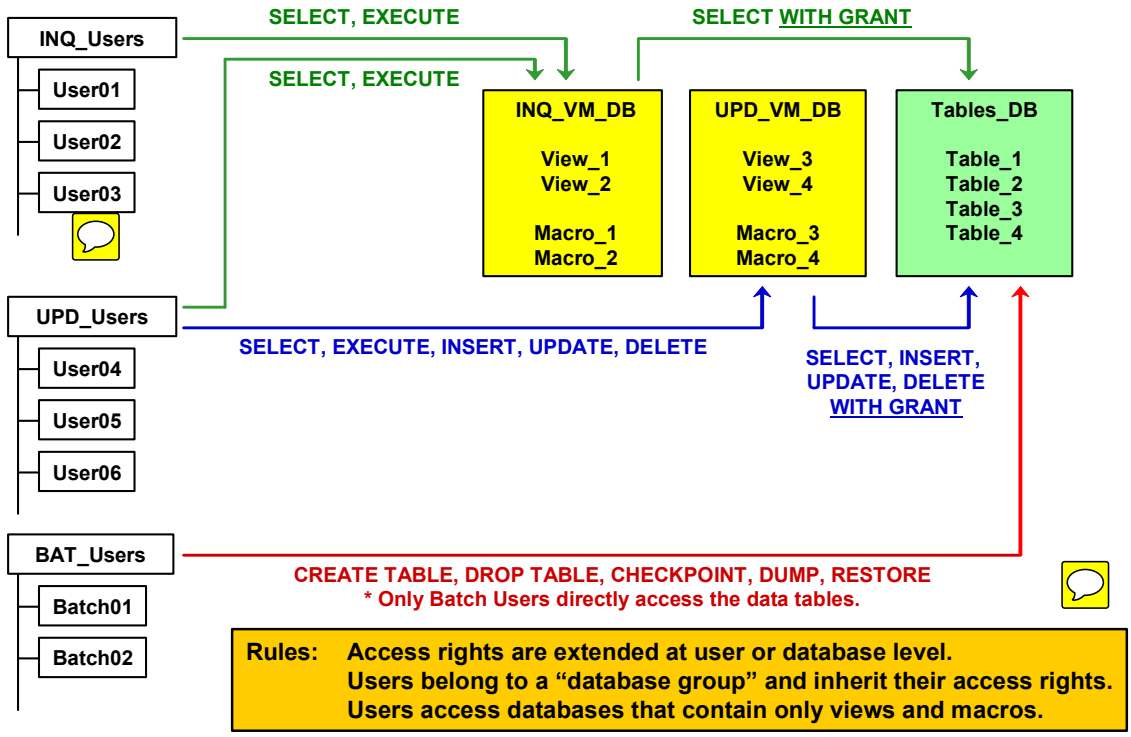
INQ_Users	Users that belong to the Inquiry database inherit SELECT and EXECUTE privileges when you create them.
UPD_Users	Users that belong to the Update database inherit SELECT, EXECUTE, INSERT, DELETE and UPDATE privileges when you create them.
BAT_Users	Users that belong to the Batch database inherit DROP and CREATE TABLE, CHECKPOINT, DUMP, and RESTORE privileges when you create them.

In addition to the access rights stored in each user database, the Inquiry VM and Update VM databases also contain a set of access rights. Both are discussed below:

INQ_VM_DB	The Inquiry VM Database contains views and macros that give Inquiry users access to information. The database has the SELECT privilege with GRANT OPTION.
UPD_VM_DB	The Update VM Database contains views and macros that enable Update users to modify information. This database has the SELECT, INSERT, DELETE and UPDATE privileges with GRANT OPTION.

The WITH GRANT option enables the Upd_VM_Database to give the necessary privileges to the update users.

A Recommended Access Rights Structure



Rules: Access rights are extended at user or database level.
Users belong to a "database group" and inherit their access rights.
Users access databases that contain only views and macros.

A Recommended Structure Using Roles

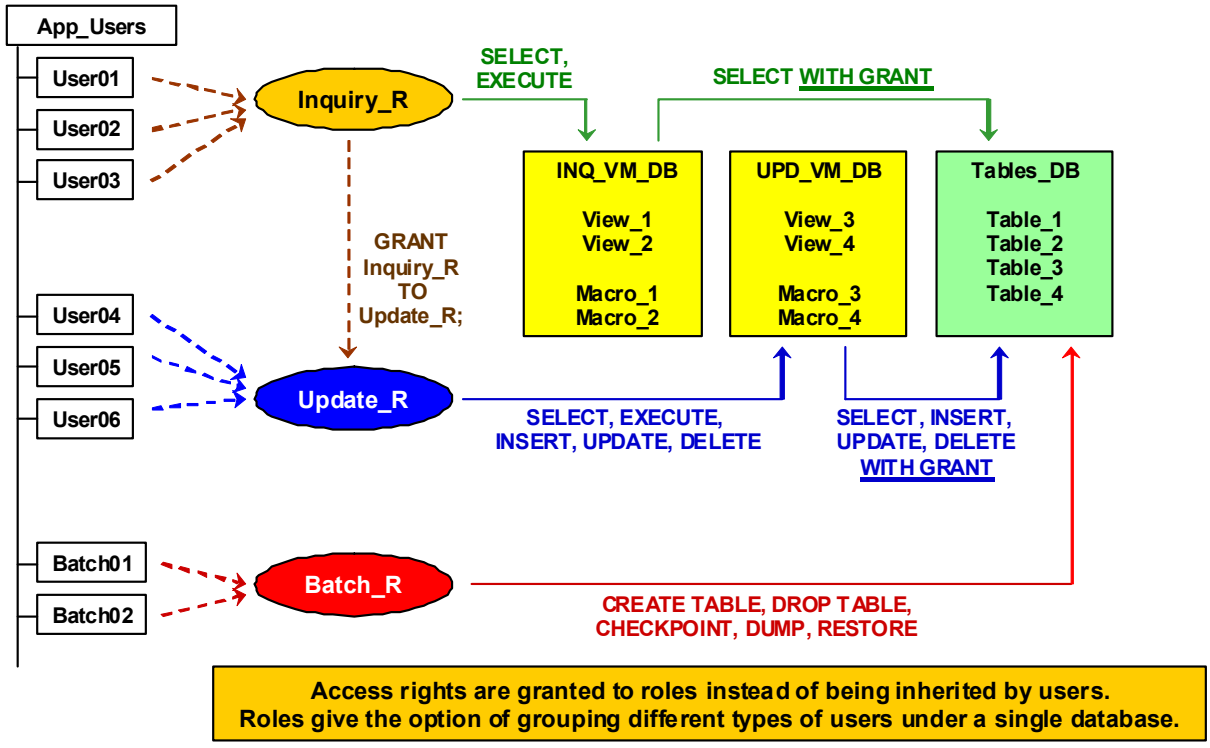
The example on the facing page illustrates an example of the suggested Teradata access rights scheme utilizing roles. This scheme has three roles:

Inquiry_R	Users granted to the Inquiry role get SELECT and EXECUTE privileges associated with this role.
Update_R	Users granted to the Update role get SELECT, EXECUTE, INSERT, DELETE and UPDATE privileges associated with this role.
Batch_R	Users granted to the Batch role get DROP and CREATE TABLE, CHECKPOINT, DUMP, and RESTORE privileges associated with this role.

In addition to the access rights assigned to the roles, the Inquiry VM and Update VM databases also contain a set of access rights. Both are discussed below:

INQ_VM_DB	The Inquiry VM Database contains views and macros that give Inquiry users access to information. The database has the SELECT privilege with GRANT OPTION.
UPD_VM_DB	The Update VM Database contains views and macros that enable Update users to modify information. This database has the SELECT, INSERT, DELETE and UPDATE privileges with GRANT OPTION.

A Recommended Structure Using Roles



Access rights are granted to roles instead of being inherited by users.
Roles give the option of grouping different types of users under a single database.

A Recommended System Hierarchy

A system structure recommended for the Teradata database is shown on the facing page. Each major application function has an associated administrator that would have control of the users and databases within that application function.

Keys to the hierarchy on the facing page are:

INQ_Users – database or set of “inquiry” users

UPD_Users – database or set of “update” users

INQ_VM_DB – database of views and macros that access tables in Tables_DB

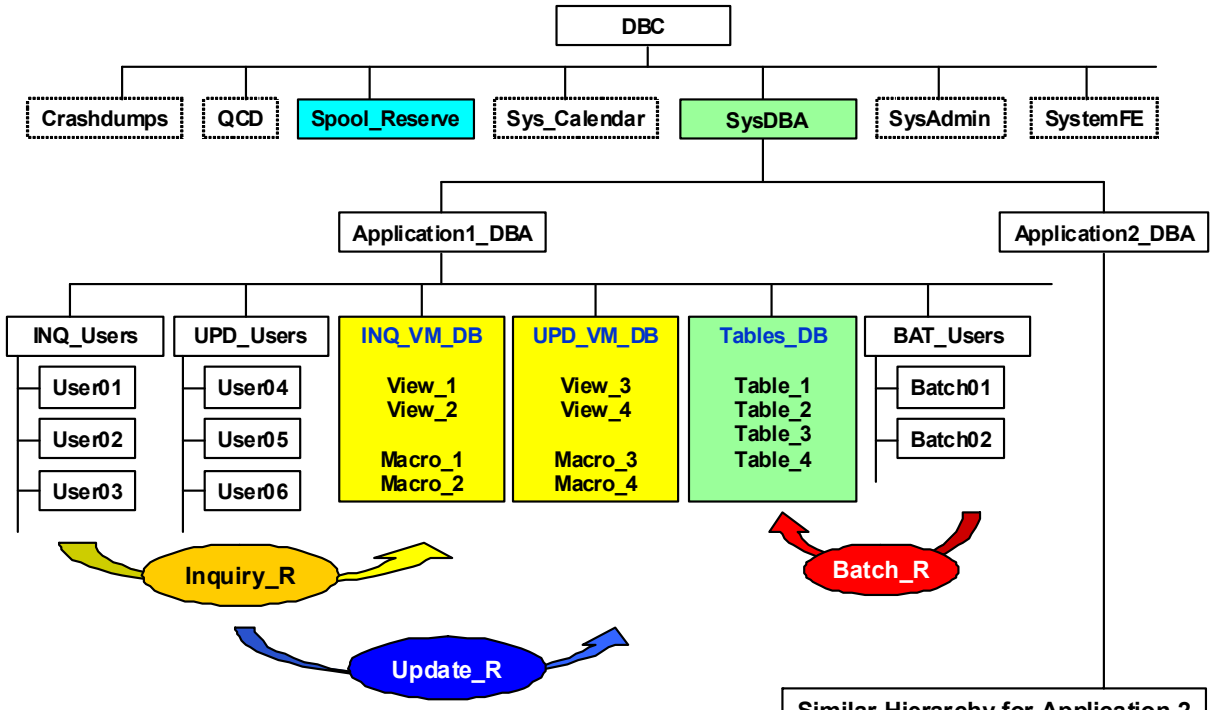
UPD_VM_DB – database of views and macros that update tables in Tables_DB

Tables_DB – database of user data tables

BAT_Users – database or set of “batch” users; operational users that execute utilities that directly access the tables (e.g., FastLoad)

Optionally roles can be used to easily maintain access rights and reduce the number of access rights.

A Recommended System Hierarchy



Ex. Roles are granted to users and only the roles have access rights on the views and macros.

System Access Controls Summary

The facing page summarizes some important concepts regarding this module.

System Access Controls Summary

- The mission of security administration is to prevent unauthorized user access to the database and its resources.
- To protect system access:
  - Associate passwords with usernames.
  - Associate application users with specific hosts.
- You can control database access by granting access to views and macros.
  - Views can limit access to certain columns and rows.
  - Macros can limit the actions a user can perform.
- Good access rights management facilitates your role as system administrator in security rule enforcement, data maintenance, archive and recovery, and other areas.
- Characteristics of a good database structure include:
  - Users belong to a database group and get their access rights from roles.
  - Users do not have direct access to tables.
  - Access rights are extended at the database, user, or role level (not at the individual table level).

Module 48: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 48: Review Questions

1. True or False. Although usernames are the basis for identification in the system, username information usually is not protected information.
2. True or False. Once users have a username and password, they can access any information in the system.
3. True or False. To change the minimum number of characters in a valid password from 6 to 8, you would update the DBC.LogonRules table.
4. What does 1024 represent in the DBC.LogonRules view? _____
5. Which choices can be used to view host or mainframe sessions? _____
   a. Sessions utility
   b. QrySessn utility
   c. Gtwglobal utility
   d. DBC.SessionInfo view
6. Which choice is used to determine why a user logon has failed? ____
   a. DBC.Logons view
   b. DBC.LogOnOff view
   c. DBC.AccessLog view
   d. DBC.SessionInfo view
   e. DBC.LogonEvents view

Notes

Module 49

Access and Query Logging

After completing this module, you will be able to:

- **Describe how the BEGIN/END LOGGING statements capture information about user access to Teradata.**
- **Describe how to set up user access logging.**
- **Use system views to gather information about data access.**
- **Identify the reasons for using the Database Query Log.**
- **Identify the tables and views that make up the DBQL facility.**

Notes

Table of Contents

Access and Query Logging 49-4
Access Logging 49-6
Objects used in Access Logging 49-8
BEGIN LOGGING Statement 49-10
END LOGGING Statement 49-12
Setting up Access Logging 49-14
  Log Entries 49-14
  Keywords and Object-Names 49-14
Access Log Views 49-16
AccLogRules View 49-18
BEGIN LOGGING – Example 49-20
AccessLog View 49-22
AccessLog View – Example 49-24
END LOGGING – Example 49-26
Teradata Administrator – Tools Menu > Access Logging 49-28
Query Logging (DBQL) Concepts 49-30
  Provides Collection of Historical records based on Rules 49-30
Objects used in Defining Rules for DBQL 49-32
  Rules 49-32
Objects used in DBQL (cont.) 49-34
BEGIN QUERY LOGGING Statement 49-36
  BEGIN QUERY LOGGING 49-36
BEGIN QUERY LOGGING WITH … (cont.) 49-38
BEGIN QUERY LOGGING LIMIT … (cont.) 49-40
BEGIN QUERY LOGGING Examples 49-42
  Hierarchy of Applying Database Query Logging Rules 49-42
BEGIN QUERY LOGGING Examples (cont.) 49-44
BEGIN QUERY LOGGING Examples (cont.) 49-46
END QUERY LOGGING Statement 49-48
REPLACE QUERY LOGGING (13.10) Statement 49-50
DBQLRules View 49-52
QryLog View – Example 49-54
QryLogSummary View – Example 49-56
Teradata Administrator – Tools Menu > Query Logging 49-58
Access and Query Logging Summary 49-60
Module 49: Review Questions 49-62
Lab Exercise 49-1 49-64
  Lab Exercise 49-1 (cont.) 49-66
Lab Exercise 49-2 49-68
  Lab Exercise 49-2 (cont.) 49-70
  Lab Exercise 49-2 (cont.) 49-72

Access and Query Logging

Use Access Logging for security and auditing purposes.

Use DBQL to capture details needed for workload analysis, performance tuning, and resource usage analysis.

Access Logging Facility

The Access Logging facility provides an administrator with the capability to monitor data access requests in the system and log granted and/or denied requests.

With Access Logging, logged rows are written immediately to disk before the SQL is executed, which can slow down short query work. At the same time, this approach to logging does offer higher reliability and can register negative accesses (attempts to view data that did not succeed), which would not show up in DBQL.

Query Logging Facility

The Database Query Log (DBQL) is a feature (starting with Teradata V2R5) that you can employ to log query processing activity for later analysis. Query counts and response times can be charted and SQL text and processing steps can be compared to fine-tune your applications for optimum performance.

DBQL provides a series of predefined tables that can store, based on rules you specify, historical records of queries and their duration, performance, and target activity.

DBQL is flexible enough to log information on the variety of SQL requests that run on Teradata, from short transactions to longer-running analysis and mining queries. You begin and end collection for a user or group of users and/or one or a list of accounts.

DBQL is streamlined for collection efficiency and provides more detail about the query than the Access Log. On the other hand, because DBQL caches data in memory before writing it periodically to disk, there is some delay in seeing the logged data, and it is possible to lose data that is still in the cache on a restart.

Access and Query Logging

There are two logging facilities available to the database and/or security administrator.

- **Access Logging Facility** 
  - **Used for access and security audit analysis.**
  - **May be used to monitor data access requests (via access rights checks) and log entries for requests that are granted and/or denied.**
- **Query Logging Facility (DBQL)**
  - **Used for query activity and workload analysis.**
  - **Can be used to track processing behavior and/or capture detailed information about the queries that are running on a system.**
  - **Workloads can be utilized with Teradata Analyst tools such as Teradata Index Wizard.**

Access Logging

The Access Logging facility provides an administrator with the capability to monitor data access requests in the system and log granted and/or denied requests.

The DDL statements BEGIN LOGGING and END LOGGING are used to control the monitoring of access rights checks performed by the Teradata Database. Each time you execute a BEGIN LOGGING statement, the system table DBC.AccLogRuleTbl receives applicable rule entries. (The system view DBC.AccLogRules offers access to the contents of this table.)

When a user named in a BEGIN LOGGING statement attempts to execute a specified action against a specified object, the Teradata Database checks the access rights necessary to execute the statement according to the rules in DBC.AccLogRuleTbl. The privilege checks made and/or the access results are logged in the system table DBC.AccLogTbl. (The system view DBC.AccessLog offers access to the contents of this table.)

A logging entry does not indicate that a statement was executed; rather, it indicates that the system checked the privileges necessary to execute the statement.

You can terminate logging by submitting an END LOGGING statement for any action, user, or object for which logging is currently active. Note that you cannot end logging begun for a specific username by omitting the BY *username* option.

Access Logging

An administrator can ...

- use the Access Logging facility to monitor data access requests and log entries for requests that are granted and/or denied.
- optionally capture the SQL text along with the access right check.

The following statements are used to specify objects and/or SQL requests to monitor for specific or all users.

- **BEGIN LOGGING statement**
  - Starts the monitoring of data access requests by Teradata.
- **END LOGGING statement**
  - Ends the monitoring of data access requests by Teradata.

Objects used in Access Logging

You must have EXECUTE privileges on the macro DBC.AccLogRule to execute the BEGIN LOGGING and END LOGGING statements.

Note: DBC.AccLogRule is a “dummy macro”. It only has a ; in it.

The BEGIN LOGGING and END LOGGING statements start and stop the auditing of data access requests. The BEGIN LOGGING and END LOGGING statements store rows in the DBC.AccLogRuleTbl. To view the rows in this table, use the DBC.AccLogRules[V] views.

If the user does not submit a BEGIN LOGGING statement, then by default the system does not generate any entries on any user action.

When an object or SQL request (identified in one of the access log rules) is accessed, an entry is logged in the DBC.AccLogTbl. To view the rows in this table, use the DBC.AccessLog[V] views.

Objects used in Access Logging

Users who are granted EXECUTE permission on the following macro can use the BEGIN LOGGING and END LOGGING statements.

Example:

GRANT EXECUTE ON DBC.AccLogRule TO SecAdmin;

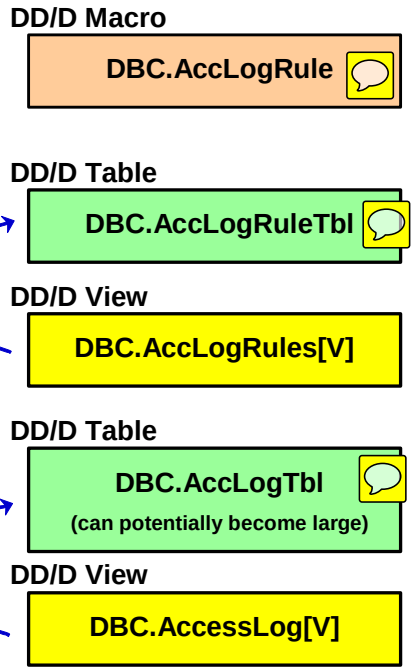
This allows "SecAdmin" to execute the BEGIN LOGGING and END LOGGING statements.

Execution of BEGIN LOGGING or END LOGGING statements causes rows (representing the rules) to be added or updated in …

To view the rules in this table, SELECT from these views.

Based on the rules, access of specified objects or SQL statements cause entries to be placed in …

To view the log of entries, SELECT from these views.



BEGIN LOGGING Statement

Teradata verifies a user's access rights when the user attempts to access an object. As the administrator, you can capture information about checks performed on a user's access rights with the BEGIN LOGGING statement.

When you activate logging, the specified privilege check performed by the Teradata Database generates a row in the DBC.AccLogTbl. Later, you can use system-supplied views to monitor and analyze the information stored there.

The BEGIN LOGGING statement has a number of options. Several are described below:

DENIALS — Tracks only those entries when statement execution fails because the user does not have the privilege(s) necessary to execute the statement.

FIRST, LAST, EACH — Defines the frequency with which log entries are made. The default for BEGIN LOGGING is FIRST.

ALL — Tells the system to make a log entry when the user attempts certain actions against the specified object, including:

CD	CREATE DATABASE	DP	DUMP	I	INSERT
CM	CREATE MACRO	D	DELETE	S	RETRIEVE/SELECT
CP	CHECKPOINT	CU	CREATE USER	RS	RESTORE
CT	CREATE TABLE	DU	DROP USER	U	UPDATE
CV	CREATE VIEW	G	GRANT	E	EXECUTE
DD	DROP DATABASE/USER				

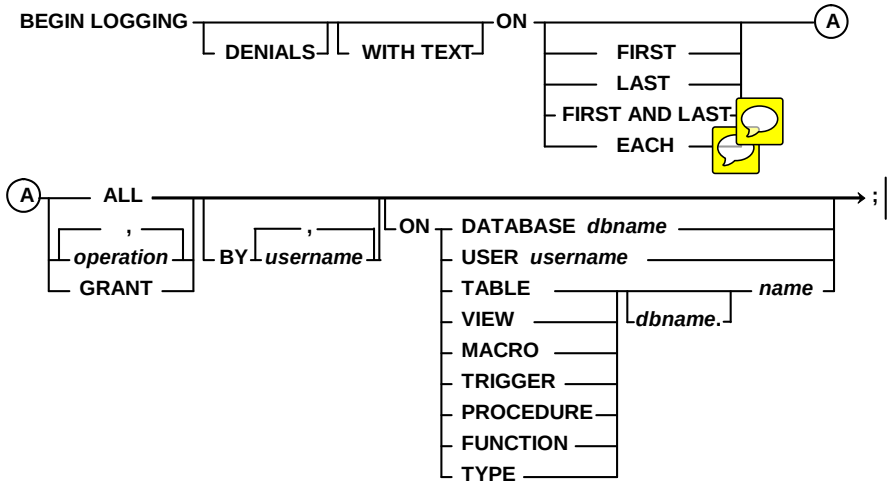
BY username — Lists the users for which the system will make log entries. The default is all users.

ON keyword object-name — Defines which objects will generate rows in the log table when a user attempts to access them. The keyword object-name combinations must be one of the following:

USER	User name	TRIGGER	Trigger name
DATABASE	Database name	MACRO	Macro name
TABLE	Table name	PROCEDURE	Stored procedure name
VIEW	View name	FUNCTION	User-defined function name

Absence of the ON keyword object name option implies all entities that the user attempts to access. A single logging statement may contain up to 20 objects.

BEGIN LOGGING Statement



Operation Any function for which an access right can be granted (e.g., GRANT).

BY ***username*** – implies all users, if not specified.

ON ***object-name*** – implies all entities, if not specified. Valid object-names are:

DATABASE	database_name	USER	user_name
TABLE	table_name	VIEW	view_name
MACRO	macro_name	PROCEDURE	procedure_name
TRIGGER	trigger_name	FUNCTION	function_name

END LOGGING Statement

Stops the auditing of SQL requests that attempt to access data that was started with a BEGIN LOGGING statement.

The END LOGGING statement erases only the frequency or text flags for the specified actions and user or object. However, if erasing a frequency leaves all logging blank for a particular user, database, and table, then the row is deleted from the AccLogRuleTbl table.

Use of the END LOGGING statement results in an error if BEGIN LOGGING is not currently in effect for the community for which logging is to be ended.

The END LOGGING statement has a number of options. Several are described below:

DENIALS — tracks only those entries when statement execution fails because the user does not have the privilege(s) necessary to execute the statement.

ALL — tells the system to make a log entry when the user attempts certain actions against the specified object, including:

CD	CREATE DATABASE	DP	DUMP	I	INSERT
CM	CREATE MACRO	D	DELETE	S	RETRIEVE/SELECT
CP	CHECKPOINT	CU	CREATE USER	RS	RESTORE
CT	CREATE TABLE	DU	DROP USER	U	UPDATE
CV	CREATE VIEW	G	GRANT	E	EXECUTE
DD	DROP DATABASE/USER				

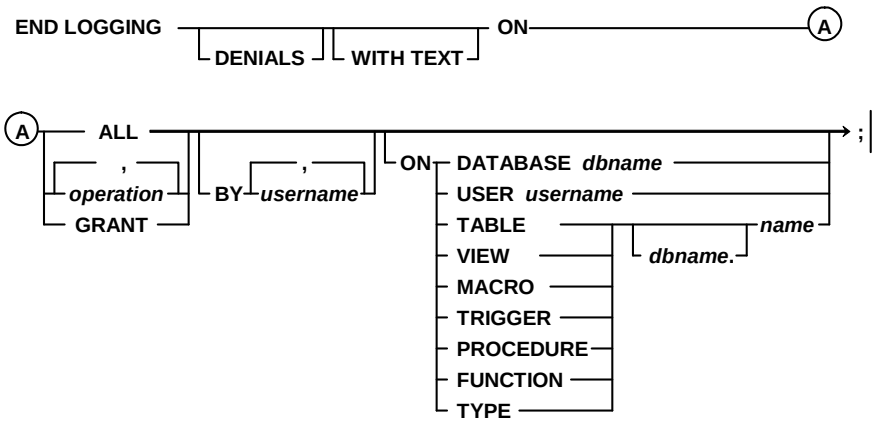
BY username — lists the users for which to end logging on. The default is all users.

ON keyword object-name — defines which objects to end logging for in the log table when a user attempts to access them. The keyword object-name combinations must be one of the following:

USER	User name	TRIGGER	Trigger name
DATABASE	Database name	MACRO	Macro name
TABLE	Table name	PROCEDURE	Stored procedure name
VIEW	View name	FUNCTION	User-defined function name

Absence of the ON keyword object name option implies all entities that the user attempts to access. A single logging statement may contain up to 20 objects.

END LOGGING Statement



Operation Any function for which an access right can be granted (e.g., GRANT).

BY ***username*** – implies all users, if not specified.

ON ***object-name*** – implies all entities, if not specified. Valid object-names are:

DATABASE	database_name	USER	user_name
TABLE	table_name	VIEW	view_name
MACRO	macro_name	PROCEDURE	procedure_name
TRIGGER	trigger_name	FUNCTION	function_name

Setting up Access Logging

Before you can execute BEGIN/END LOGGING statements, you must run DIP script located on the software release medium. The DIP script creates a special security macro called DBC.AccLogRule. After you run the script, you must reset the system to initialize the logging software.

Log Entries

A logging entry does not indicate that the system successfully executed an SQL statement. It only indicates that the system checked the privileges necessary to execute the statement.

Keywords and Object-Names

By default, access logging inserts a row whenever a user accesses any database object. To restrict the scope of the log entries, you can include one of the following combinations in the BEGIN LOGGING statement:

DATABASE databasename	USER username
TABLE databasename.tablename	VIEW databasename.viewname
MACRO databasename.macroname	TRIGGER databasename.triggername
PROCEDURE databasename.procedurename	
FUNCTION databasename.functionname	TYPE databasename.datatypename

To activate access logging:

- Install Access Logging on the system with the DIP script DIPACC.
- Create and empower a security administrator.
- Submit the BEGIN LOGGING statement to define an access logging rule. You must submit this statement for each rule you define.

Example

Step 3 on the facing page is an example of three access logging rules. Each rule is described below:

- Log all attempts to access the security macros — Creates a new row in the log table each time a user attempts to access DBC.LogonRule or DBC.AccLogRule security macros. The ALL keyword indicates that any one of 27 user actions triggers an entry in the log table. The WITH TEXT option stores SQL statement contents in the log table.

- Log all denied attempts to access system user DBC — Logs all denied attempts made by any user to access system user with any one of the 27 defined actions. Teradata stores the SQL text in the log table.

- Log any SQL statements that involve CREATE or DROP USER/DATABASE or GRANT.

Setting Up Access Logging

STEP 1 **Install Access Logging on the system:**

1. Run the DIP script DIPACC to install the DBC.AccLogRule macro in system user DBC.
2. Restart the database to activate the code.

STEP 2 **Create and empower a security administrator:**

```
CREATE   USER SecAdmin  AS PASSWORD = secpasswd, PERM = 0, SPOOL = 500E6;
GRANT    EXECUTE ON DBC.AccLogRule TO SecAdmin;
GRANT    EXECUTE ON DBC.LogonRule  TO SecAdmin;
```

STEP 3 **Define access logging rules. Examples:**

1. Log all attempts to access security macros.
2. Log all denied attempts to access DBC User.
3. Log any CREATE or DROP USER/DATABASE or GRANT commands.

```
BEGIN LOGGING WITH TEXT              ON   EACH ALL
                                     ON   MACRO DBC.LogonRule,
                                          MACRO DBC.AccLogRule;
BEGIN LOGGING DENIALS WITH TEXT      ON EACH ALL
                                     ON USER DBC;
BEGIN LOGGING WITH TEXT              ON EACH DATABASE, USER, GRANT;
```

Access Log Views

There are actually four system-supplied views that provide information about the entries in the DBC.AccLogRuleTbl[V] and the DBC.AccLogTbl[V]. They are:

DBC.AccLogRules[V]	Teradata maintains entries in this view's underlying table as the result of executing BEGIN/END LOGGING statements. The system uses these entries to determine which privilege checks should generate rows in the DBC.AccLogTbl.
DBC.AccessLog[V]	The underlying table for this view is DBC.AccLogTbl. Each entry in DBC.AccLogTbl indicates the results of a privilege check performed against a Teradata SQL request, based on the criteria defined by the BEGIN LOGGING statement.

Underlying DD/D Tables

- DBC.AccLogRuleTbl
- DBC.AccLogTbl

Access Log Views

DBC.AccLogRules[V] — Contains current logging rules generated by BEGIN and END LOGGING statements.

DBC.AccessLog[V] — Contains log entries collected as a result of applying access log rules.

Dictionary Tables Accessed:

- DBC.AccLogRuleTbl
- DBC.AccLogTbl

AccLogRules View

The DBC.AccLogRules[V] views provide information about logging rules currently in effect on the system. These rules were put into effect by successfully processing BEGIN LOGGING statements.

Example

The SQL statement on the facing page requests a list of the current rules stored in the DBC.AccLogRules table. It limits the rules to CREATE and DROP database and user, GRANT, SELECT, and EXECUTE.

The response produces four rows. Each contains a series of codes under each privilege column. There are three positions under each privilege. The first position indicates how often to log privilege checks. The second position indicates how often to log denials. The third position indicates when to save text. The following codes are used for positions 1 and 2:

B	Log FIRST and LAST occurrences.
E	Log each occurrence.
F	Log the FIRST occurrence.
L	Log the LAST occurrence.
Blank	No logging

The third position for text uses the following codes:

-	Save text only for Denial entries.
+	Save text for all entries.
=	Save text for all entries specified in multiple BEGIN LOGGING Statements

The code E+ means insert a row for each occurrence and save the text. The code E- means insert a row for each occurrence but only save the text for denials.

AccLogRules View

DBC.AccLogRules[V] views – return information about current access logging rules.

UserName	DatabaseName	TVMName	AcrAlterFunction (AFN)
AcrCheckpoint (CPT)	AcrCreateDatabase (CDB)	AcrCreateFunction (CFN)	AcrCreateMacro (CMC)
AcrCreateTable (CTB)	AcrCreateUser (CUS)	AcrCreateView (CVW)	AcrCreateProcedure (CSP)
AcrCreExtProcedure (CXP)	AcrDelete (DEL)	AcrDropDatabase (DDB)	AcrDropFunction (DFN)
AcrDropMacro (DMC)	AcrDropTable (DTB)	AcrDropUser (DUS)	AcrDropView (DVW)
AcrDropProcedure (DSP)	AcrDump (DMP)	AcrExecute (EXE)	AcrExecuteFunction (EFN)
AcrExecuteProcedure (ESP)	AcrGrant (GRT)	AcrIndex (IDX)	AcrInsert (INS)
AcrReference (REF)	AcrRestore (RST)	AcrSelect (SEL)	AcrUpdate (UPD)
AcrCreateTrigger (CTG)	AcrDropTrigger (DTG)	AcrCreateRole (CRO)	AcrDropRole (DRO)
AcrCreateProfile (CPR)	AcrDropProfile (DPR)	AcrAlterProcedure (ASP)	AcrRepControl (REP)
AcrAlterExtProcedure (AXP)	AcrUDTUsage (USG)	AcrUDTType (UDT)	AcrUDTMethod (UDM)
AcrCreAuthorization (CAU)	AcrDropAuthorization (DAU)	AcrStatistics (STA)	AcrShow (SHO)
AcrCreOwnerProcedure (COP)	AcrConnectThrough(CTH)	CreatorName	CreateTimeStamp
AcrCreateGLOP (CGL)	AcrDropGLOP (DGL)	AcrGLOPMember (MGL)	

ACR Columns are positional:

Position #1 = How often to log requests (F, L, B, E, blank = First, Last, Both, Each, None)

Position #2 = How often to log denials (F, L, B, E, blank = First, Last, Both, Each, None)

Position #3 = How often to save text (+ All entries, - Denials, = All Specified)

```
SELECT  UserName        (CHAR (6))   AS "User//Name"
       ,DatabaseName    (CHAR (6))   AS "Dbase//Name"
       ,TVMName         (CHAR (10))  AS "TVM//Name"
       ,AcrCreateDatabase, AcrCreateUser, AcrDropDatabase
       ,AcrDropUser, AcrGrant, AcrSelect, AcrExecute
FROM    DBC.AccLogRulesV;
```

Results:

User Name	Dbase Name	TVM Name	CDB	CUS	DDB	DUS	GRT	SEL	EXE
All	All	All	E +	E +	E +	E +	E +		
All	DBC	LogonRule					E +		E +
All	DBC	All	E -	E -	E -	E -	E -	E -	E -
All	DBC	AccLogRule					E +		E +

BEGIN LOGGING – Example

The facing page provides an additional example of entries that may appear in the DBC.AccLogRules view. This view provides information about logging rules currently in effect on the system. These rules were put into effect by successfully processing BEGIN LOGGING statements.

BEGIN LOGGING – Example

```
BEGIN LOGGING DENIALS  WITH TEXT  ON EACH SELECT          ON TABLE PD.Employee;   1
BEGIN LOGGING          WITH TEXT  ON FIRST INSERT         ON TABLE PD.Employee;   2
BEGIN LOGGING          WITH TEXT  ON FIRST AND LAST DELETE ON TABLE PD.Employee;   3
BEGIN LOGGING          WITH TEXT  ON FIRST UPDATE         ON TABLE PD.Employee;   4
BEGIN LOGGING DENIALS  WITH TEXT  ON LAST UPDATE          ON TABLE PD.Employee;   4

SELECT      UserName              (CHAR (6))     AS "User//Name"
           ,DatabaseName          (CHAR (6))     AS "Dbase//Name"
           ,TVMName               (CHAR (10))    AS "TVM//Name"
           ,AcrSelect, AcrInsert, AcrDelete, AcrUpdate
FROM        DBC.AccLogRulesV
WHERE       DatabaseName = 'PD';
```

			1	2	3	4
User Name	Dbase Name	TVM Name	SEL	INS	DEL	UPD
All	PD	Employee	E-	F +	B +	FL=

Position #1 = How often to log requests (F, L, B, E, blank = First, Last, Both, Each, None)

Position #2 = How often to log denials (F, L, B, E, blank = First, Last, Both, Each, None)

Position #3 = How often to save text (+ All entries, - Denials, = All Specified)

AccessLog View

The DBC.AccessLog[V] views display the entries made in the DBC.AccLogTbl system table. It returns information on the results of privilege checks performed against user requests to access data, which are logged as determined by the access logging rules.

Administrators may use this view to analyze application performance. This view would provide information about SQL requests (the text), tables and views accessed, embedded view (view of views), etc.

- Access Type
  The same codes are used to indicate an access right, but with CUS and DUS for CREATE/DROP USER, AN for any privilege (validated for HELP and SHOW commands), HR for HOST UTILITY LOCK, and WL for WRITE LOCK.

- Frequency
  F, L, B, E = First, Last, Both or Each.

Once logging begins, the access log grows very quickly. To keep space consumption under control, you should archive and empty the log regularly using the DBC.DeleteAccessLog view.

Examples:

DELETE FROM DBC.DeleteAccessLogV;

– Deletes entries from DBC.AccLogTbl older than 30 days.

DELETE FROM DBC.DeleteAccessLogV WHERE LOGDATE < (DATE – 90);

– Deletes entries from DBC.AccLogTbl older than 90 days.

AccessLog View

These views display entries made to DBC.AccLogTbl.

DBC.AccessLog[V]

LogDate	LogTime	LogonDate	LogonTime
LogicalHostID	IFPNo	SessionNo	UserName
AccountName	OwnerName	AccessType	Frequency
EventCount	AccLogResult	Result	DatabaseName
TVMName	ColumnName	StatementType	StatementText
QueryBand	ProxyUser		

Access Type	The same codes are used that indicate an access right.
Frequency	F, L, B, E = First, Last, Both or Each.

To delete entries in the DBC.AccLogTbl, use the DBC.DeleteAccessLog[V][X] views.

- **DELETE FROM DBC.DeleteAccessLogV;**
  - **Deletes entries older than 30 days.**
- **DELETE FROM DBC.DeleteAccessLogV WHERE LOGDATE < (CURRENT_DATE – 90);**
  - **Deletes entries older than 90 days.**

AccessLog View – Example

The SELECT statement on the facing page requests the contents of the DBC.AccLogTbl via the DBC.AccessLog view. The response shows seven separate entries.

User TFACT01 executed the following SQL statements at the listed times:

09:04:25 SELECT * FROM PD.Employee;

09:17:04 UPDATE PD.Employee SET Salary_Amount = 51000 WHERE Employee_Number = 100996;

09:24:31 UPDATE PD.Employee SET Salary_Amount = 51000 WHERE Employee_Number = 100995;

09:32:50 UPDATE PD.Employee SET Salary_Amount = 51000 WHERE Employee_Number = 100994;

Note that only the last UPDATE denial for TFACT01 appears on the following page. The rule specified to log the Last Update denial.

User Sysdba executed the following SQL statements at the listed times:

09:10:17 INSERT INTO PD.Employee VALUES
(101001, 1060, 100991, 3054, 'Scott', 'Bill', 50000.00);

09:12:22 DELETE FROM PD.Employee
WHERE Employee_Number = 100900;

09:12:25 DELETE FROM PD.Employee
WHERE Employee_Number = 100901;

09:12:31 DELETE FROM PD.Employee
WHERE Employee_Number = 100902;

09:15:54 UPDATE PD.Employee SET Salary_Amount = 51000
WHERE Employee_Number = 101001;

09:57:20 UPDATE PD.Employee SET Salary_Amount = 51000
WHERE Employee_Number = 100800;

Note that only the first and last DELETEs for Sysdba appear on the following page based on the Access Log Rules. Also notice that only the first UPDATE appears on the facing page.

Note about uppercase and lowercase Frequency values:

When a FIRST and LAST are both specified for an entry, a lower case ‘l’ or ‘f’ are used to identify which entry this one is. An uppercase ‘L’ and ‘F’ is used if one is specified without the other.

AccessLog View – Example

Example:

List all of the entries in the Access Log table for the current date.

```
SELECT    LogTime
          ,UserName (CHAR (10))    AS "User//Name"
          ,AccessType              AS "Access//Type"
          ,Frequency               AS "Log//Freq"
          ,AccLogResult            AS "Granted//Denied"
          ,StatementText           AS "Statement//Text"
FROM      DBC.AccessLogV
WHERE     LogDate = CURRENT_DATE
ORDER BY  LogDate, LogTime ;
```

Results:

LogTime	User Name	Access Type	Log Freq	Granted Denied	Statement Text
09:04:25	TFACT01	S	E	D	SELECT * FROM PD.Empl...
09:10:17	SYSDBA	I	F	G	INSERT INTO PD.Employe ...
09:12:22	SYSDBA	D	f	G	DEL FROM PD.Employee ...
09:12:31	SYSDBA	D	l	G	DEL FROM PD.Employee ...
09:15:54	SYSDBA	U	F	G	UPDATE PD.Employee SE ...
09:17:04	TFACT01	U	f	D	UPDATE PD.Employee SE ...
09:32:50	TFACT01	U	l	D	UPDATE PD.Employee SE ...

Note: The facing page contains the SQL that generated this report.

END LOGGING – Example

The facing page provides the END LOGGING statements to remove the logging for the PD.Employee table.

The BEGIN LOGGING statements (that were previously executed) are:

```
BEGIN LOGGING DENIALS WITH TEXT
    ON EACH SELECT
    ON TABLE PD.Employee;

BEGIN LOGGING WITH TEXT
    ON FIRST INSERT
    ON TABLE PD.Employee;

BEGIN LOGGING WITH TEXT
    ON FIRST AND LAST DELETE
    ON TABLE PD.Employee;

BEGIN LOGGING WITH TEXT
    ON FIRST UPDATE
    ON TABLE PD.Employee;

BEGIN LOGGING DENIALS WITH TEXT
    ON LAST UPDATE
    ON TABLE PD.Employee;
```

END LOGGING – Example

Previously, these rules were created for logging on PD.Employee table.

```
BEGIN LOGGING DENIALS  WITH TEXT  ON EACH SELECT                ON TABLE PD.Employee;
BEGIN LOGGING          WITH TEXT  ON FIRST INSERT               ON TABLE PD.Employee;
BEGIN LOGGING          WITH TEXT  ON FIRST AND LAST DELETE      ON TABLE PD.Employee;
BEGIN LOGGING          WITH TEXT  ON FIRST UPDATE               ON TABLE PD.Employee;
BEGIN LOGGING DENIALS  WITH TEXT  ON LAST UPDATE                ON TABLE PD.Employee;
```

To end the logging for PD.Employee table, the following statements can be executed:

```
END LOGGING DENIALS    ON SELECT, UPDATE              ON TABLE PD.Employee;
END LOGGING            ON INSERT, DELETE, UPDATE      ON TABLE PD.Employee;
```

To verify the rules have been removed, use the DBC.AccLogRules view:

```
SELECT      UserName              (CHAR (6))    AS "User//Name"
           ,DatabaseName          (CHAR (6))    AS "Dbase//Name"
           ,TVMName               (CHAR (10))   AS "TVM//Name"
           ,AcrSelect, AcrInsert, AcrDelete, AcrUpdate
FROM        DBC.AccLogRulesV
WHERE       DatabaseName = 'PD';
```

User Name	Dbase Name	TVM Name	SEL	INS	DEL	UPD

Rules for PD.Employee have been removed.

Teradata Administrator – Tools Menu > Access Logging

The Tools menu provides the following options.

Menu Selection	Function / Options
Create	Create an entirely new object – Database, Table, User, Profile, or Role.
Grant/Revoke	Grant or revoke general access privileges to users. Options include Object Rights, System Rights, Logon Rights, or Column Rights.
Administer Profiles	Create and manage Profiles for users.
Administer Roles	Create and manage Roles.
Clone User	Create a new user either identical or closely related to an existing user.
Modify User	Change the specifications of an existing user.
Access Logging	**Create and manage Access Log rules.**
Query Logging	Create and manage Query Log rules.
Move Space	Reallocate permanent disk space from one database to another (efficient if not a direct descendant or parent).
Query	Create, modify, test, or run SQL query scripts.
Options	Configure the operational preferences for Teradata Administrator.

The example on the facing page effectively causes the following BEGIN LOGGING statement to be executed.

```
BEGIN LOGGING WITH TEXT ON EACH
CREATE DATABASE, CREATE USER, CREATE PROFILE, CREATE ROLE, DROP
DATABASE, DROP USER, DROP PROFILE, DROP ROLE;
```

Teradata Administrator
Tools Menu > Access Logging

Teradata Administrator can be used to Begin and End Access Logging – effectively managing Access Log rules.

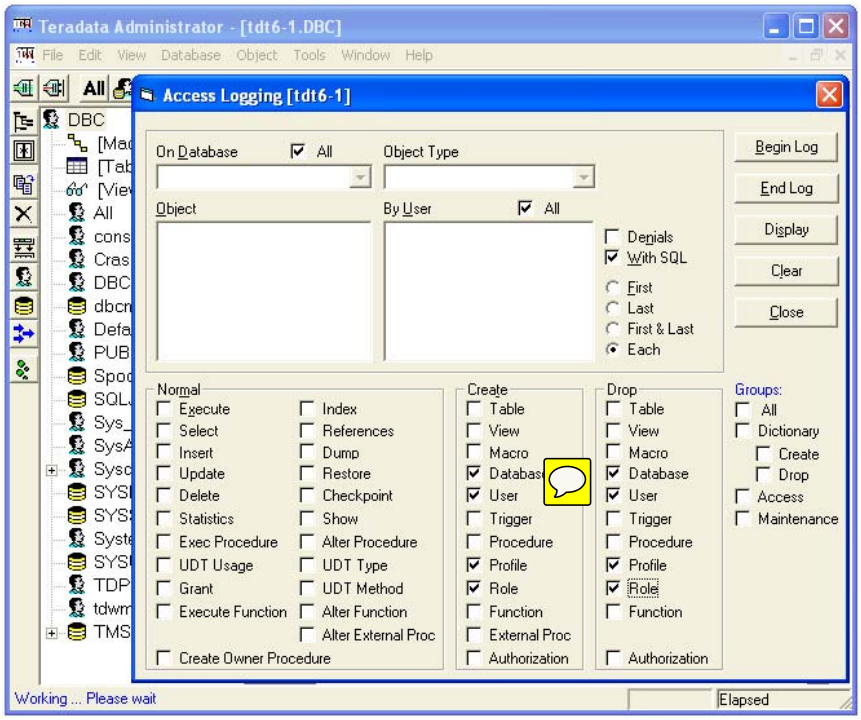
To select …

Tools >
Access Logging

Group Options:

Selecting different Groups will automatically choose specific Normal, Create, or Drop functions.

The corresponding BEGIN LOGGING statement is provided on the facing page.



Query Logging (DBQL) Concepts

Provides Collection of Historical records based on Rules

DBQL provides a series of predefined tables that can store, based on rules you specify, **historical records** of queries and their duration, performance, and target activity. DBQL is flexible enough to log information on the variety of SQL requests that run on Teradata, from short transactions to longer-running analysis and mining queries. **You begin and end collection for a user or group of users and/or one or a list of accounts.**

Performance

The performance of collecting query activity with DBQL is much better that attempting to capture query activity with the Access Logging facility because DBQL stores information in cache memory and will write to disk only when cache is full or when you use END QUERY LOGGING.

Notes about performance:

- The impact of turning on DBQL for all-AMP operations is negligible in comparison to the time an all-AMP operation takes to complete.

- The impact of turning on DBQL for single-AMP (PI) and two-AMP (USI) operations does impact the response time of the operation by a small amount. These types of operations are best suited for summary logging where the overhead is negligible.

- Specific information from the San Diego performance group follows:

  With a single session of single-AMP queries (a total of 40,000 were executed), with default logging, the total response time increased by 6.5%. The performance report shows an average CPU path length increase of 4.6 ms for similar type work, from a baseline (no logging) path length of 4.3 ms, essentially a doubling of the path length.

  While the response time tests indicated summary level logging on single AMP queries to have only a 1.6% increase in total response time, the performance report showed less than 1% increase in path length with summary logging.

  The clear conclusion to be drawn is that single or few AMP queries will be best suited for summary logging, and will exhibit negligible overhead if logging is at that level.

  With all-AMP queries, even very short ones, the response time tests were not able to pick up any overhead at all with any level of query logging enabled, including "all".

Query Logging (DBQL) Concepts

- **DBQL is a feature that is used to log historical query information.**
  - DBQL caches and eventually stores query information in multiple Teradata Data Dictionary tables as the queries are executed.
  - Key use is to track SQL for query and workload analysis.
  - Not intended for live review of queries and query steps.
- Logging is invoked and revoked via SQL statements – **BEGIN QUERY LOGGING, END QUERY LOGGING,** and **REPLACE QUERY LOGGING.**
  - Logging can be invoked for an application name (e.g., FASTLOAD), all users, a list of users, a list of accounts, or a specific user/account.
  - You can also selectively exclude users, applications, and so on from logging through the use of the WITH NONE option (13.0 feature).
  - 13.10 Feature – REPLACE QUERY LOGGING allows you to replace a logging rule set.
- **By default, 1 row per query is logged that contains user id information and some statistics for that query.**
  - Options are available to expand the amount and kind of information to be logged.
- DBSControl option (**DBQLFlushRate**) – determines the frequency (default is 10 minutes) for writing DBQL cache entries to DBQL dictionary tables.

Objects used in Defining Rules for DBQL

The DBQL logs are a series of system tables created in database DBC during the Teradata Database installation process. The suite of DBQL components includes a security macro and a view for each table, which are created in database DBC by the DIP utility during installation.

Rules

You define rules that identify for which users and how much data to log for queries. For instance, you can log the first 5,000 characters of any query that runs during a session invoked by a specific user under a specific account. This rule can also be qualified so that only queries that exceed a specified time threshold are logged and those queries that execute in less than the threshold time are simply counted.

The DBC.DBQLRuleTbl table stores the rules resulting from each BEGIN QUERY LOGGING statement. One row exists for each set of specifications, which are made up of *user* and/or *account* plus any options or limits set for the user.

The DBC.DBQLRuleCountTbl table is an internal table that stores the cardinality of DBC.DBQLRuleTbl table.

The DBC.DBQLRules[V] views are used to display DBQL rules that are in effect.

Objects used in Defining Rules for DBQL

Users who are granted EXECUTE permission on the following macro can use the BEGIN QUERY LOGGING, END QUERY LOGGING, and REPLACE QUERY LOGGING statements.

Example:

GRANT EXECUTE ON DBC.DBQLAccessMacro TO Sysdba;

Initially, only DBC and SystemFE users are allowed to issue BEGIN/END QUERY LOGGING statements.

DD/D Macro

DBC.DBQLAccessMacro

Execution of BEGIN QUERY LOGGING, END QUERY LOGGING, or REPLACE QUERY LOGGING (13.10) statements causes rows (representing the rules) to be added or updated in …

DD/D Tables

DBC.DBQLRuleTbl

To view the rules in this table, SELECT from these views.

DD/D Views

DBC.DBQLRules[V]

Note: There are additional tables and views that are used to hold and view captured query data – these are shown on next page.

Objects used in DBQL (cont.)

The DBQL logs are a series of system tables created in database DBC during the Teradata Database installation process. The suite of DBQL components includes a security macro and a view for each table, which are created in database DBC by the DIP utility during installation.

Like other system tables, the predefined DBQL logs are created as relational tables in database DBC during normal Teradata Database installation. However, while most system tables are populated automatically, you can choose whether you want to populate the DBQL tables.

If you choose not to use the feature, the tables remain empty. If you want to use the feature, simply submit a BEGIN/END QUERY LOGGING statement, with or without options, to control the start, magnitude, and end of logging activity. The options enable you to control the volume and detail of the logged data. You can define rules, for instance, that log the first 5,000 characters of any query that runs during a session invoked by a specific user under a specific account, if the time to complete that query exceeds the specified time threshold.

The key or foundation table in DBQL is the DBC.DBQLogTbl table that holds the default rows. When you specify options that result in more information being captured, a default row is still generated in this table plus one or more additional logs (tables) will get rows.

- Exceptions to this are when you use the SUMMARY option or a query completes within a THRESHOLD. In these cases, default rows won't be placed into DBC.DBQLogTbl.

The facing page summarizes the tables and views used by DBQL to hold query data. Details on these views can be found in Appendix C.

Acronym: TDWM – Teradata Dynamic Workload Manager

Objects used in DBQL (cont.)

Key views and tables used to hold query data are ...

DD/D Views	DD/D Tables	DBQL Purpose
DBC.QryLog[V]	DBC.DBQLogTbl	Stores default rows (key table)
DBC.QryLogSteps[V]	DBC.DBQLStepTbl	One row per step
DBC.QryLogObjects[V]	DBC.DBQLObjTbl	One row per object referenced in query
DBC.QryLogSQL[V]	DBC.DBQLSqlTbl	Stores full SQL text – multiple rows may be needed
DBC.QryLogSummary[V]	DBC.DBQLSummaryTbl	Queries meeting Summary or Threshold rules
DBC.QryLogExplain[V]	DBC.DBQLExplainTbl	Stores EXPLAIN steps of query
DBC.QryLogXMLV (13.0)	DBC.DBQLXMLTbl	Stores Optimizer query plan for the logged query as an XML document

Examples of additional DBQL views that provide TDWM information include:

- **DBC.QryLogEvents[V] – TDWM events that could affect system performance**
- **DBC.QryLogEventHis[V] – history of TDWM events**
- **DBC.QryLogExceptions[V] – query exceptions as defined by Workload Definitions (WD)**

BEGIN QUERY LOGGING Statement

The DBQL facility is controlled by the Teradata SQL statements BEGIN QUERY LOGGING and END QUERY LOGGING.

There are numerous collection options using the WITH and LIMIT options. These options will be described on the following pages.

BEGIN QUERY LOGGING

When submitted by a user with EXECUTE privileges on DBC.DBQLAccessMacro, enables logging for the named users and/or accounts. For active sessions, logging begins when the next query is received. (Teradata recommends a maximum of 100 user/account pairs per statement.)

When you do not specify a LIMIT option, one default row of query-level information is logged in DBQLogTbl for each query processed during a session that is initiated by any user for whom a query logging rule exists.

Default rows are stored in DBQLogTbl, the foundation of the DBQL feature. If you specify options that result in more detailed information, a default row is still generated in DBQLogTbl (except with the SUMMARY option or a query that completes within the limit specified with the THRESHOLD option), plus one or more additional logs are populated with one or more additional rows.

Examples of valid application names (Teradata 13.0 feature) include FASTLOAD, FASTEXP, MULTLOAD, ARC, etc.

You can determine the application name used in the LogonSource string for a running application by querying DBC.SessionInfo as follows:

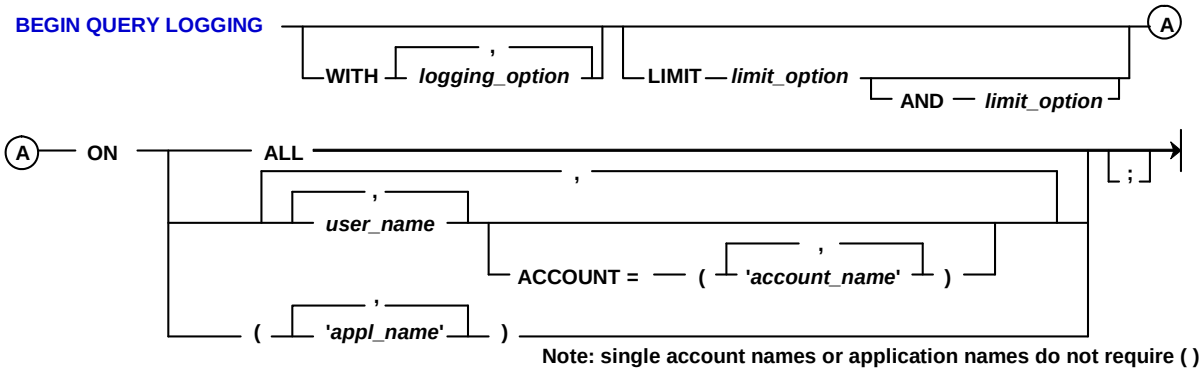
```
SELECT DISTINCT (LogonSource) FROM DBC.SessionInfo;
```

The Default Row

The fields of the default row provide general query information that is usually adequate for investigating a query that is interfering with performance. When no options are specified, key fields in a default row includes the following (not a complete list).

- User name under which the session being logged was initiated
- Unique ID for the process, session, and host (client) connection
- Account string, expanded as appropriate, that was current when the query completed
- First 200 characters of the query SQL statement.
- Time of receipt
- Number of processing steps completed
- Time the first step was dispatched, and
- Times of the first and last response packets were returned to the host.

BEGIN QUERY LOGGING Statement



Note: single account names or application names do not require ( )

- A BEGIN QUERY LOGGING statement without the WITH or LIMIT options causes default rows to be placed in the DBQLogTbl. A default row contains:
  - User name, account string (expanded), time stamp information
  - Unique ID for process, session, and client (host) connection
  - First 200 characters of SQL statement
- Use of the WITH option(s) cause a default row to be placed in DBC.DBQLogTbl plus additional rows in other DBQL tables or can cause query logging to be turned off.
- The LIMIT option may be used to limit the amount of SQL text captured, set thresholds, or just capture summary information.

BEGIN QUERY LOGGING WITH … (cont.)

… NONE (13.0)

You can specify any of the following items with NONE:

- account:user pair or account:user list
- application name or application name list
- user name list
- ALL:account name or account name list
- ALL (which specifies all accounts)

No additional options are valid if you specify WITH NONE.

… ALL

Logs all information generated by all the WITH rules (EXPLAIN, OBJECTS, SQL, & STEPINFO) and the default row.

… EXPLAIN

Use this option selectively because the performance cost of generating EXPLAINs can be expensive. This option generates and logs the unformatted EXPLAIN text for each query. It does not generate EXPLAIN text for queries preceded by the EXPLAIN modifier. This option logs one or more rows into DBC.DBQLExplainTbl

… OBJECTS

Use this option selectively. Object data is useful for analyzing queries that make heavy use of join indexes and indexed access, but can generate many rows. Inserts one row per target object per query in DBC.DBQLObjTbl.

… SQL

This option logs the entire SQL statement in the DBC.DBQLSqlTbl table. Large statements can cause multiple rows to be written in order to log the full query text.

… STEPINFO

Use this option selectively. Although step data is useful for analyzing queries, this option can generate many rows. Inserts one row per step per query in the DBC.DBQLStepTbl.

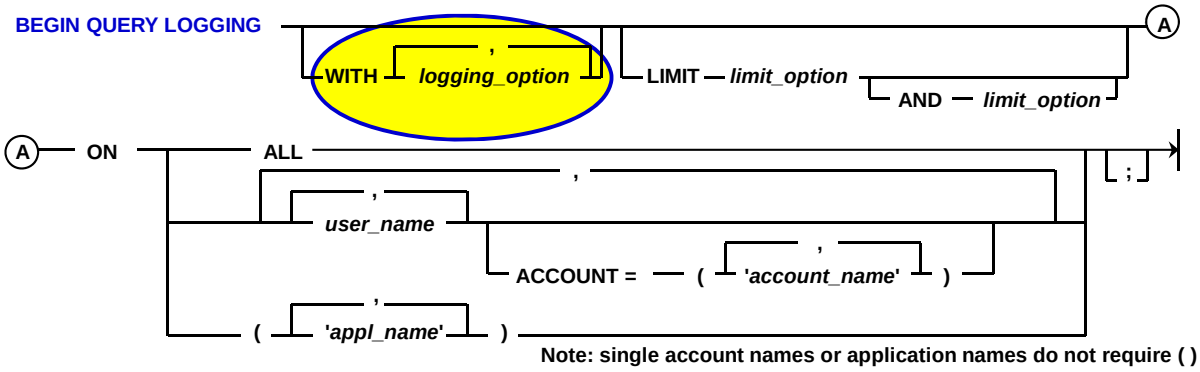
… XMLPLAN (13.0)

Logs the query plan generated by the Optimizer for all SQL DML requestsa as an XML document in system table DBC.DBQLXMLTbl. Because the XML document includes the query and step text, you generally do not need to specify the EXPLAIN and SQL options if you specify XMLPLAN. You should also specify a value of 0 for SQLTEXT to avoid redundant logging when you specify XMLPLAN. This option logs one or more rows into DBC.DBQLXMLTbl.

… STATSUSAGE (14.0)

Logs statistics usage data and detailed query plan information for all DML and selected DDL requests as an XML document in DBC.DBQLXMLTbl. If you specify both XMLPLAN and STATSUSAGE, Teradata logs the collected data into a single integrated document.

BEGIN QUERY LOGGING WITH ... (Cont.)



Note: single account names or application names do not require ( )

WITH options

- **NONE – turns off query logging for the specified user, account, or application name.**
- **ALL – logs the default row plus includes EXPLAIN, OBJECTS, SQL and STEPINFO options.**
- **EXPLAIN – logs the default row plus the unformatted EXPLAIN text for the query in DBQLExplainTbl.**
- **OBJECTS – logs one row per target object per query in DBQLObjTbl plus default row in DBQLogTbl.**
- **SQL – logs the entire SQL for each request DBQLSqlTbl plus default row in DBQLogTbl.**
- **STEPINFO – inserts one row per step per query in DBQLStepTbl plus default row.**
- **STATSUSAGE (14.0) – logs statistics usage data and detailed query plan information for all DML and selected DDL requests as an XML document in DBQLXMLTbl.**
- **XMLPLAN – stores Optimizer query plan for the query in DBQLXMLTbl as an XML document.**

BEGIN QUERY LOGGING LIMIT … (cont.)

… SQLTEXT

Use this option if you want to capture less than or more than the first 200 characters in the default row. To turn off text capture in the default row completely, specify 0 (zero). The maximum limit is 10,000 characters. If you specify the option keyword but not a value, up to 10,000 characters are logged in DBQLogTbl.

To store the complete statement regardless of length, specify the SQL option, as many rows as needed to contain the full text will be logged in DBQLSqlTbl. (If you do this, define LIMIT SQLTEXT=0 to avoid redundant logging in both the default row and DBQLSqlTbl.)

Note: Also set LIMIT SQLTEXT=0 if you specify either the WITH ALL or the WITH SQL option, which also logs SQL.

… SUMMARY

SUMMARY is useful for tracking voluminous short queries, such as for OLTP applications, because it does not grow the DBQLogTbl. It simply counts queries based on specified time differentials and stores the count results in DBQLSummaryTbl.

The SUMMARY option is unique in that it:

- Does not generate default rows in DBQLogTbl
- Summary information is flushed at system-controlled intervals of 10 minutes
- If no data has been collected for a summary category in a 10-minute interval, no rows will be written for it.

… THRESHOLD

THRESHOLD also is useful for short, high-volume queries, but in addition to incrementing a count for qualifying queries, THRESHOLD logs a default row for any query that exceeds the specified time. This enables you examine the processing timestamps and the query structure. You can combine THRESHOLD with SQLTEXT if you want to capture more than the first 200 characters of a query that runs longer than THRESHOLD seconds for identification of the longer running queries.

You define the threshold of execution time, in seconds, which determines whether to log a query or just count it, as follows:

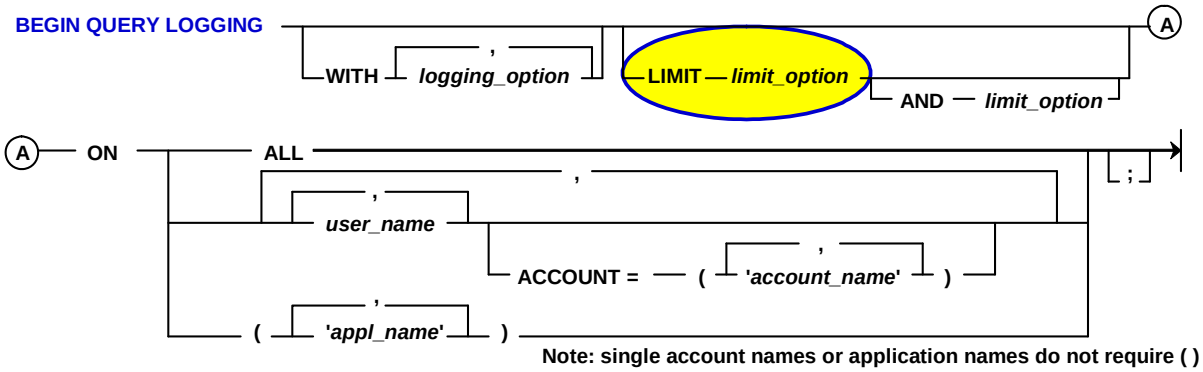
IF a query completes at or under the threshold time

- Increments the query count and the query seconds
- Stores the final count for the session as a row in DBQLSummaryTbl
- In the summary row, sets the value in the LowHist field to the THRESHOLD time and in the HighHist field to 0 (to identify it as a THRESHOLD row)

IF a query runs beyond the threshold time

- DBQL logs a default row for the query in DBQLogTbl so you can examine its structure and the number and level of processing steps.

BEGIN QUERY LOGGING LIMIT … (cont.)



Note: single account names or application names do not require ( )

LIMIT options

- **SQLTEXT** – specify the amount of SQL text to capture in the default row of DBQLogTbl. (Default is 200 char., 0 = off, max = 10,000 characters)
- **SUMMARY** (for high volume queries or tactical queries – doesn't log default rows)
  - Counts queries; count is written in DBQLSummaryTbl every 10 min
- **THRESHOLD** (Also for short high-volume queries – example tactical queries)
  - Similar to SUMMARY, but default rows are generated in DBQLogTbl.
  - Threshold, in seconds, determines whether to log a query or just count it.
    - Query that complete <= threshold (sec.) are counted in DBQLSummaryTbl.
    - Query that complete > threshold (sec.), DBQL logs the default row.

BEGIN QUERY LOGGING Examples

The facing page contains a number of BEGIN QUERY LOGGING examples.

Prior to Teradata 13.0, when you specify **BEGIN QUERY LOGGING ON ALL;**, you effectively create a rule for "everyone". Therefore, you cannot create rules for specific users. The opposite is also true. If you create rules for specific users, you cannot create a rule for ALL.

Starting with Teradata 13.0, there is more flexibility with query logging. Also starting with Teradata 13.0, there is a hierarchy of applying database query logging rules.

Hierarchy of Applying Database Query Logging Rules

Database Query Logging works from a hierarchical foundation that allows BEGIN QUERY LOGGING requests to be submitted for individual users even if a rule exists for ALL users.

However, if a rule exists for a specific account:user pair, you must submit an appropriate END QUERY LOGGING request to delete the rule before you can issue a new rule for that account:user pair. Teradata Database applies the rules in the following order:

Order	Type of Rule
1	A rule based on an application name.
2	A rule for this specific user and specific account.
3	A rule for this specific user and any account.
4	A rule for all users and this specific account.
5	A rule for all users and any account.

DBQL first searches for a rule based on an application name. If no such rule exists, DBQL then looks for a rule specific to the user and account, and so on down the hierarchy.

The rules cache contains rules either for an application or for a specific account:user combination. As each user logs on, DBQL first searches the rules cache in hierarchical order for a rule. If there are no specific rules in the rules cache for level 1 or 2, DBQL searches *DBC.DBQLRuleTbl* in hierarchical order for the best fit. DBQL makes an entry in the rules cache for the account:user pair: either a rule that DBQL is not enabled for the account:user or the DBQL rule that applies with its options. If a match is made on the rules table at level 1, DBQL makes an application name entry in the rules cache.

For example, you can submit a BEGIN QUERY LOGGING request for default logging on ALL users, and DBQL can also be enabled for *user1* with objects and steps. If *user1* logs on and executes queries, DBQL collects objects and steps. When users other than *user1* log on and execute queries, DBQL only logs default row information for them.

BEGIN QUERY LOGGING Examples

```
BEGIN QUERY LOGGING                    ON ALL;
```

- Creates a rule to log default query information on all users and accounts.

```
BEGIN QUERY LOGGING                    ON ALL;
BEGIN QUERY LOGGING    WITH NONE       ON tfact01, tfact02;
```

- Creates a rule to log default query information on all users and accounts.
- Creates two additional rules to disable query logging for tfact01 and tfact02.

```
BEGIN QUERY LOGGING    ON    tfact03    ACCOUNT = ('$L_&S&D&H', '$M_&S&D&H');
```

- Creates two rules for a specific user – each rule identifies a specific account id.

```
BEGIN QUERY LOGGING    ON    APPLNAME = ('MULTLOAD', 'TPTUPD');
```

- Creates a rule to enable logging for any MultiLoad or TPT Update job. (no 'I' in MULTLOAD)

```
BEGIN QUERY LOGGING    WITH SQL       ON ALL;
BEGIN QUERY LOGGING    WITH NONE      ON APPLNAME = ('FASTLOAD', 'TPTLOAD');
```

- Creates a rule to log default query information and full SQL on all users and accounts.
- Creates a rule to disable logging for any FastLoad or TPT Load job.

BEGIN QUERY LOGGING Examples (cont.)

The facing page contains additional BEGIN QUERY LOGGING examples.

Limits that may be used include:

SQLTEXT: option to control the number of SQL statement characters to log

- 200 characters of SQL logged in default row
- SQLTEXT values range from 0 to 10,000
- "SQLTEXT" without a value logs 10,000 characters

THRESHOLD: option to limit the queries logged by elapsed time

- THRESHOLD values are specified in seconds.
- "THRESHOLD" without a value results in a 5 second value.
- Queries greater than Threshold value generate a default row.
- Maximum THRESHOLD value is 32,767 seconds.

Notes about Threshold logging in Teradata 13.10:

- Starting with Teradata 13.10, you can use THRESHOLD logging along with the SQL, STEPINFO, and EXPLAIN options.

- With threshold logging, DBQL cannot log to separate Explain and XML tables, even for those queries taking longer than the specified criteria. SQL, STEPINFO, and OBJECTS can be logged during threshold logging, even for those queries taking longer than the specified clock seconds.

- By filtering logging with THRESHOLD will lower the performance overhead of detailed logging of unnecessary queries.

- For example, to log into DBQLogTbl, DBQLSqlTBL, DBQLStepTbl and DBQLObjTbl queries that run beyond 1 CPU and summarize all other queries in the DBQLSummaryTbl:

  BEGIN QUERY LOGGING with SQL, STEPINFO, OBJECTS LIMIT THRESHOLD=100 CPUTIME on ALL;

SUMMARY: option to only count running queries based on elapsed time.

- 3 values (in seconds) are needed; no verification done on order of values
- 4 count intervals are created
  <= n1; > n1 <= n2; > n2 <= n3; > n3
- Data is logged in the DBC.DBQLSummaryTbl every 10 minutes (or if the cache should get full). There is no parameter to change the flush period of 10 minutes.
- 1 row for each count > 0
- **SUMMARY cannot be used with any other "limits".**

BEGIN QUERY LOGGING Examples (cont.)

```
BEGIN QUERY LOGGING   WITH ALL                                  ON   ALL;
```

- ALL options are logged for ALL users (probably generates too much information).

```
BEGIN QUERY LOGGING   WITH OBJECTS, SQL                         ON   ALL;
BEGIN QUERY LOGGING   WITH OBJECTS, SQL, EXPLAIN                ON   tfact07, tfact08;
```

- Default rows are logged as well as complete SQL text and objects used in queries for all users.
- The default row has the first 200 bytes of SQL text.
- For tfact07 and tfact08, Explain text is also captured.

```
BEGIN QUERY LOGGING   LIMIT SUMMARY = 5, 60, 600                ON   ALL;
```

- Default Summary option is to only count running queries based on elapsed time. 3 values (in sec.) are required. 4 count intervals are logged (<=5, <=60, <=600, >600)
- Summary limit cannot be used with any other limits.

```
BEGIN QUERY LOGGING   WITH SQL, OBJECTS   LIMIT THRESHOLD = 60   ON   ALL;
```

- If a query runs for less than or equal to 60 seconds (1 minute), increment the count.
- If a query runs longer than 1 minute, log a default row.
- Teradata 13.10 feature – ability to also log SQL, Objects, and StepInfo with Threshold logging.

In these examples, the ON option can specify users, account ids, or application names.

BEGIN QUERY LOGGING Examples (cont.)

The SUMMARY and THRESHOLD limits have additional options. The default for each of these limits is elapsed time (ELAPSEDSEC).

SUMMARY

SUMMARY = *n1, n2, n3* [ELAPEDSEC, ELAPSEDTIME, CPUTIME, CPUTIMENORM or IOCOUNT]

The SUMMARY option is designed for use with short, OLTP-like, queries. This option counts the number of queries for the session that fall into each of four time intervals. Interval values can be specified in seconds, CPU time, or I/O counts.

If you specify SUMMARY, then you cannot specify any other options. You must specify the first three intervals explicitly. The fourth interval is created by default which is 32,767 seconds or 9 hours.

THRESHOLD

THRESHOLD [=n] [ELAPEDSEC, ELAPSEDTIME, CPUTIME, CPUTIMENORM or IOCOUNT]

This option is also designed for use with tactical queries. This option sets a threshold time in seconds that determines whether a query is to be logged fully or just counted.

If a query completes earlier than or equal to the threshold value, then it is only logged as a count in DBQLSummaryTbl. The Threshold row in DBQLSummaryTbl is identified by a HighHist field value of 0. If a query completes later than the threshold value, then a full entry is logged for it in DBQLogTbl with values for all fields of the row

Options for both SUMMARY and THRESHOLD

ELAPSEDSEC (default) – use this option to set ranges and to summarize counts of the number of requests that fall into an elapsed time interval. This value is expressed in units of 1.00 seconds.

ELAPSEDTIME – use this option to set ranges and to summarize counts of the number of requests that fall into an elapsed time interval. The value is expressed in units of 0.01 seconds so it provides finer granularity for elapsed time than ELAPSEDSEC.

CPUTIME – use the CPU time for the query to set ranges and to summarize. The value is in units of 0.01 second. For example, if you specify 500 for one of the intervals, then the value used to make the determination is 5 CPU seconds.

CPUTIMENORM – use this option to set ranges and to summarize counts of the number of requests that fall into a normalized CPU time interval. This option is designed for use with coexistence systems to aid in managing mixed nodes more efficiently, but it can be used with any system. The value is expressed in units of 0.01 second.

IOCOUNT – use the I/O count for the query to set ranges and to summarize.

BEGIN QUERY LOGGING Examples (cont.)

The THRESHOLD and SUMMARY Limit options include:

- **ELAPSEDSEC (default)** – values are in units of 1.0 seconds; counts the number of requests that fall into elapsed wall-clock time intervals
- **ELAPSEDTIME** – values are specified in units of 0.01 sec – provides finer granularity than ELAPSEDSEC
- **CPUTIME** – uses actual CPU time to set ranges and summarize – represented in units of 0.01 sec.
- **CPUTIMENORM** – similar to CPUTIME, but with normalized time intervals (use with coexistence systems)
- **IOCOUNT** – uses the I/O count for the query to set ranges and to summarize

```
BEGIN QUERY LOGGING  LIMIT SUMMARY = 100, 500, 2000 CPUTIME                ON   ALL;
```

- **This Summary example only counts queries based on actual CPU time.**
- **100 is 1 CPU second, 500 is 5 CPU seconds, and 2000 is 20 CPU seconds. 3 values are required. 4 count intervals are logged (<=100, <=500, <=2000, >2000)**
- **Summary limit cannot be used with any other limits.**

```
BEGIN QUERY LOGGING        LIMIT THRESHOLD = 50  IOCOUNT                  ON   ALL;
```

- **If a query generates less than or equal to 50 IO's, increment the count.**
- **If a query generates more than 50 IO's, log a default row.**

In these examples, the ON option can also specify user name and/or account IDs.

END QUERY LOGGING Statement

… ***ON ALL*** **–** to stop logging query information for all users specified by a rule created by a BEGIN QUERY LOGGING ON ALL statement.

…user_name **–** the name of a specific user or set of users for whom logging of SQL query information is to be stopped.

…account_name **–** the name of one or more specific accounts for which logging of SQL query information is to be stopped.

Account names must be enclosed by LEFT and RIGHT PARENTHESIS characters. When you specify a list of accounts, each account name must be delimited by APOSTROPHE characters and separated by COMMA characters.

If you begin query logging on a specific user-account pair, then you must also specify that user-account pair to end query logging.

When this statement is submitted by a qualified user (with EXECUTE privileges on DBQLAccessMacro), logging is stopped for the named users and/or accounts. This command can be used for up to 100 active sessions. When this command is used, a routine is called that commits the data and flushes the cache.

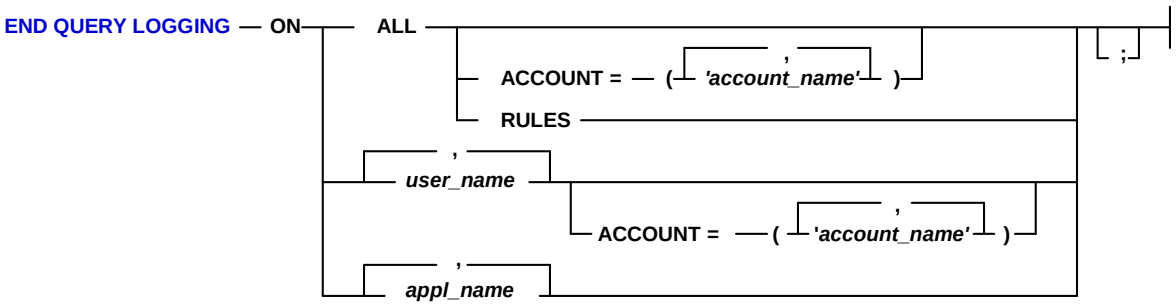
When you enable or disable query logging, the change has an immediate effect on active sessions where the user or account being logged appears within the first 100 names you specify in the user and/or account list of a single BEGIN/END QUERY LOGGING statement. For users listed beyond the first 100, the user must log off from the Teradata Database and restart the session.

Note: If you need to enable or disable large volumes of users or accounts, the best practice is to submit an additional BEGIN/END QUERY LOGGING statement for each block of 100 names.

When you disable logging (submit an END QUERY LOGGING statement) for an active session (a query for that session is in process) and data is already cached, the following occurs:

- The data is committed immediately
- One or more DBQL rows are written (but may be incomplete)
- The cache is flushed
- Subsequent queries during that session are not logged

END QUERY LOGGING Statement



- Prior to Teradata 13.0, if "ON ALL" was used in the BEGIN statement, "ON ALL" must be used in the END statement.
- If a list of users or a list of account strings was given in the BEGIN statement, logging can be ended on an individual basis.
- The "END QUERY LOGGING" statement will cause DBQL cache to be written to the tables except for Summary cache.

END QUERY LOGGING ON ALL RULES;	Removes all rules.
END QUERY LOGGING ON tfact01;	(You can end logging for a specific user.)
END QUERY LOGGING ON tfact03 ACCOUNT=('$L_&D&H');	(You can end logging for a specific account of a user.)

REPLACE QUERY LOGGING (13.10) Statement

The REPLACE QUERY LOGGING statement allows the customer to modify their query logging for active users without having to end query logging and begin query logging. This will help prevent missing logging data between the end logging and begin logging sequence of statements.

This statement can also be used to avoid the end query logging and begin query logging statement pairs and unnecessary flushing of DBQL caches.

The REPLACE QUERY LOGGING statement will replace an existing rule or will create a new rule if one did not exist.

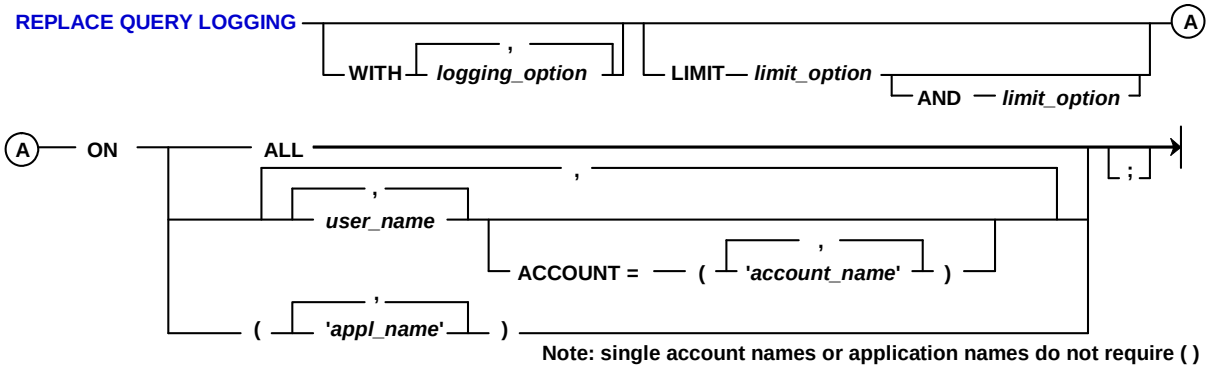
Additional 13.10 DBQL Enhancement

An additional Teradata 13.10 DBQL enhancement is that TypeOfUse column in the DBQLObjTbl table will be populated

The TypeOfUse column contains the following numeric values:

1 = Found in the resolver
2 = Accessed during query processing
4 = Found in a conditional context
8 = Found in inner join condition
16 = Found in outer join condition
32 = Found in a sum node
64 = Found in a full outer join condition

REPLACE QUERY LOGGING (13.10) Statement



- **A REPLACE QUERY LOGGING (13.10) statement has the same options as the BEGIN QUERY LOGGING statement.**
  - **Replaces an existing rule. If no rule exists, it will be created.**
- **This statement allows you to replace a logging rule set. Formerly, this action required two statements: END QUERY LOGGING and BEGIN QUERY LOGGING.**
  - **This will help prevent missing logging data between the end logging and begin logging sequence of statements.**
  - **This statement also avoids the unnecessary flushing of DBQL caches.**

DBQLRules View

A description of the columns in the DBC.DBQLRules[V] views follows:

UserName:	Identifies the name associated with the rule.
AccountString:	Contains the default account for the user, or the specified Account.
ApplName	Contains the application name for the rule.
TypeOfRule	Indicates whether logging is enabled or disabled by this rule.
ExplainFlag:	T = Explain text will be stored; F = No explain text is provided.
ObjFlag:	T = Object data (Columns, Indexes) will be stored; F = No object data.
SqlFlag:	T = SQL text will be stored; F = No SQL text is provided.
StepFlag:	T = Step level data will be stored; F = No step level data is provided.
XMLPlanFlag	T = XML plans will be stored, F = No XML plans.
SummaryFlag:	T = Summary information will be stored; F = not summarized.
ThresholdFlag:	T = Count queries shorter than ThreshValue seconds; detailed data on long queries.Detailed data for all queries
TextSizeLimit:	Indicates the number of characters of SQL text to be stored.
SummaryVal1:	If Summary or Threshold is T, low history in seconds, CPU or IO.
SummaryVal2:	Group 2: SummaryVal1 to SummaryVal2 seconds.
SummaryVal3:	Group 3: SummaryVal2 to SummaryVal3 seconds.
TypeOfCriterion:	Indicates if values are elapsed time, CPU seconds, or IO counts 0 = Summary values are in 1.0 seconds 1 = Values in CPU hundredths of seconds 2 = Values in IO count 3 = Values in normalized CPU time (hundredths of seconds) 4 = Summary values are in 0.01 seconds

The BEGIN QUERY LOGGING statements that generated the rules on the facing page are as follows:

```
BEGIN QUERY LOGGING ON ALL ACCOUNT = ('$H');
BEGIN QUERY LOGGING LIMIT SQLTEXT=500 ON tfact01, tfact02;
BEGIN QUERY LOGGING ON tfact03 ACCOUNT = ('$M_&D&H');
BEGIN QUERY LOGGING LIMIT SUMMARY = 5, 30, 60 ON tfact04;
BEGIN QUERY LOGGING LIMIT THRESHOLD = 200 CPUTIME ON tfact05;
BEGIN QUERY LOGGING WITH SQL LIMIT SQLTEXT=0 ON tfact06;
```

The END QUERY LOGGING statements that correspond to these BEGIN QUERY LOGGING statements are as follows:

```
END QUERY LOGGING ON ALL ACCOUNT = ('$H');
END QUERY LOGGING ON tfact01, tfact02;
END QUERY LOGGING ON tfact03 ACCOUNT = ('$M_&S&D&H');
END QUERY LOGGING ON tfact04;
END QUERY LOGGING ON tfact05;
END QUERY LOGGING ON tfact06;
```

Note: Simply replacing the BEGIN with END will also remove the rules.

DBQLRules View

DBC.DBQLRules[V] views – return information about current query logging rules.

UserName	AccountString	ApplName (13.0)	TypeofRule (13.0)	ExplainFlag
ObjFlag	SqlFlag	StepFlag	XMLPlanFLag (13.0)	SummaryFlag
ThresholdFlag	TextSizeLimit	SummaryVal1	SummaryVal2	SummaryVal3
ThreshValue	TypeofCriterion (13.0)			

Example:

```
SELECT    UserName          (CHAR (8))    AS "User"
          ,AccountString    (CHAR (8))    AS "Acct_ID"
          ,SqlFlag          AS "Sql" ,    TextSizeLimit   AS "Size"
          ,ThresholdFlag    AS "T_Flag" , SummaryFlag  AS "S_Flag"
          ,TypeofCriterion  AS "Type"
          ,SummaryVal1      AS "V1"  , SummaryVal2  AS "V2" , SummaryVal3  AS "V3"
FROM      DBC.DBQLRulesV
ORDER BY 1;
```

Notes:

BEGIN QUERY LOGGING statements are on facing page.

Type of Criterion:

- 0 – Elapsed (1.0 sec)
- 1 – CPUTIME
- 2 – IOCOUNT
- 3 – CPUTIMENORM
- 4 – Elapsed (0.01 sec)

User	Acct_ID	Sql	Size	T_Flag	S_Flag	Type	V1	V2	V3
All	$H	F	200	F	F	?	?	?	?
tfact01		F	500	F	F	?	?	?	?
tfact02		F	500	F	F	?	?	?	?
tfact03	$M_&S&D&H	F	200	F	F	?	?	?	?
tfact04		F	200	F	T	0	5	30	60
tfact05		F	200	T	F	1	200	?	?
tfact06		T	0	F	F	?	?	?	?

QryLog View – Example

The facing page contains an example of using the DBC.QryLog[V] view.

One of the key columns is the CacheFlag column. This column can be one of the following:

- Blank indicates the query is not found in step cache.
- **"S"** if the query is a parameterized query and a Specific Plan is generated.
- **"G"** if the query is a parameterized query and a Generic Plan is generated.
- "**T**" if the query is found in step cache.
- **"A"** if the query is a parameterized query and a SpecAlways decision is taken. That is, each time a query submitted, the USING values are peeked at and the query is parsed.

The first time a parameterized request is seen, the parser peeks at the values in the USING clause, and a plan specific to those values is produced. This will cause the cache flag for that query's execution to be set to 'S' for 'specific plan'. At this point in time, no plan is actually cached for re-use, because a value-specific plan was produced.

After this first 'specific plan' execution is complete, metrics from the execution, as well as the estimates that were produced, are saved in the request cache, in preparation for additional decisions that will be made should this request be seen again. If the parsing engine (PE) time that resulted from the specific plan was a very small percent of the total query execution time, then the optimizer may set a flag to always use a specific plan for this query. If that decision is made, then the cache flag going forward will contain an 'A' for 'always specific' and no generic plans will be generated for this request as long as its metrics remain in the request cache.

The second time that parameterized request is seen (assuming that the PE time that resulted from the specific plan is non-trivial), a generic plan is produced resulting in a cache flag 'G'. Once both a specific and a generic plan have been generated, the estimates and run-time metrics they each produced can be compared side by side, and further decisions can be made whether to always produce the specific or always produce the generic plan.

Assuming the same parameterized request is repetitively executed on the same PE, a common pattern that may occur, is a cache flag sequence of 'S', 'G', 'T', 'T', 'T'...

The first time the query is seen, a specific plan is produced ('S'), the second time a generic plan ('G'), and from that point on, the same generic plan is executed ('T') until that plan is flushed from the cache.

Another column in this view (not listed on facing page) is AppId. Examples of names in this column are BTEQ, EXECUTOR, FASTEXP, FASTLOAD, SAS, SQLA, TBUILDEXE, TDLOAD, TPTEXP, TPTLOAD, and TPTUPD.

DBC.QryLog View – Example

DBC.QryLog[V] views – return information about default rows in the DBQLogTbl.

Example of the data in a default row with a QueryID of 163811496961777848.

```
SELECT    ProcID, SessionID, UserName, QueryID, UserID, AcctString, ExpandAcctString,
          StartTIme, FirstStepTime, ElapsedTime, CacheFlag, NumResultRows,
          SpoolUsage, QueryText
FROM      DBC.QryLogV
WHERE     QueryID = 163811496961777848;
```

Shown in BTEQ with SIDETITLES and FOLDLINE on.

Result:

```
            ProcID  16381
         SessionID  116038
          UserName  tfact07
           QueryID  163811496961777848
            UserID  00007506
        AcctString  $M0+EDUC&S&D&H
  ExpandAcctString  $M0+EDUC00011603811092219
         StartTime  2011-09-12 19:14:52
     FirstStepTime  2011-09-12 19:14:52
       ElapsedTime  0:00:00.070000
         CacheFlag  T
     NumResultRows  1137
        SpoolUsage  313,856
         QueryText  SELECT * FROM DBC.AccLogTbl WHERE Username LIKE ...
```

QryLogSummary View – Example

An example of the output from the DBC.QryLogSummary[V is provided on the facing page. This summary data was collected for the tfact04 user who had established multiple sessions on the system. The summary rule was created as follows:

BEGIN QUERY LOGGING LIMIT SUMMARY = 5, 30, 60 ON tfact04;

QryLogSummary View – Example

Returns information about summary rows in the DBQLSummaryTbl.

This example is based on the summary rule:

```
BEGIN QUERY LOGGING LIMIT SUMMARY = 5, 30, 60 ON tfact04;
```

```
SELECT    UserID, CollectTimeStamp, QueryCount, QuerySeconds,
          LowHist, HighHist
FROM      DBC.QryLogSummaryV
ORDER BY  2, 3, 6;
```

For #2, the average time of queries can be calculated:

```
407 / 1060 =    .38
748 /   41 =  18.24
351 /    8 =  43.88
2478 /   3 = 826.00
```

Result:

UserID	CollectTimeStamp	QueryCount	QuerySeconds	LowHist	HighHist	
00003305	2011-09-12 19:32:06	60	18	0	5	1
00003305	2011-09-12 19:32:06	2	27	5	30	1
00003305	2011-09-12 19:42:06	1060	407	0	5	2
00003305	2011-09-12 19:42:06	41	748	5	30	2
00003305	2011-09-12 19:42:06	8	351	30	60	2
00003305	2011-09-12 19:42:06	3	2478	60	32767	2
00003305	2011-09-12 19:52:06	258	80	0	5	3
00003305	2011-09-12 19:52:06	16	251	5	30	3
00003305	2011-09-12 19:52:06	1	821	60	32767	3

1 – In this summary collection, no queries were executed that exceeded 30 seconds.
2 – In this summary collection, queries were executed in all 4 summary intervals.
3 – In this summary collection, no queries were executed that ran between 30 and 60 seconds.

Teradata Administrator – Tools Menu > Query Logging

The Tools menu provides the following options.

Menu Selection	Function / Options
Create	Create an entirely new object – Database, Table, User, Profile, or Role.
Grant/Revoke	Grant or revoke general access privileges to users. Options include Object Rights, System Rights, Logon Rights, or Column Rights.
Administer Profiles	Create and manage Profiles for users. (V2R5 feature)
Administer Roles	Create and manage Roles. (V2R5 feature)
Clone User	Create a new user either identical or closely related to an existing user.
Modify User	Change the specifications of an existing user.
Access Logging	Create and manage Access Log rules.
Query Logging	**Create and manager Query Log rules.**
Move Space	Reallocate permanent disk space from one database to another (efficient if not a direct descendant or parent).
Query	Create, modify, test, or run SQL query scripts.
Options	Configure the operational preferences for Teradata Administrator.

The example on the facing page effectively causes the following BEGIN QUERY LOGGING statement to be executed.

```
BEGIN QUERY LOGGING LIMIT THRESHOLD=60
ON student301, student302, student303, student304, student305;
```

Teradata Administrator
Tools Menu > Query Logging

Teradata Administrator can be used to Begin and End Query Logging – effectively managing DBQL rules.

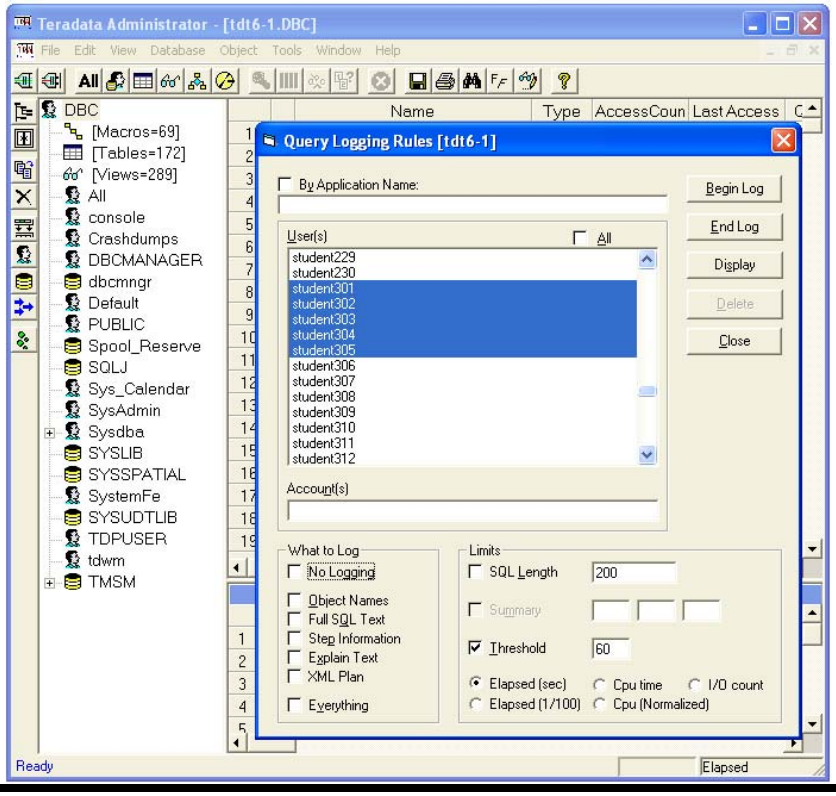
To select …

Tools > Query Logging

Options:

DISPLAY will show query log rules for the selected users.

The corresponding BEGIN QUERY LOGGING statement is provided on the facing page.



Access and Query Logging Summary

The facing page summarizes some important concepts regarding this module.

Access and Query Logging Summary

There are two logging facilities available to the database and/or security administrator.

- **Access Logging Facility**
  - **Used for access and security audit analysis**
- **Query Logging Facility (DBQL)**
  - **Used for query activity and workload analysis**
- **Additional DBQL Enhancements:**
  - **Macros, views, triggers, stored procedures, and User-DefinedFunctions (UDFs) are logged in DBQLObjTbl if the WITH OBJECTS option is used.**
  - **By querying DBQLObjTbl information, Database Administrators (DBAs) are able to see which views and macros users access.**
  - **If the WITH OBJECTS option is used, FastLoad and MultiLoad target tables are also logged in the DBQLObjTbl.**

Module 49: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 49: Review Questions

1. In order to use the BEGIN/END LOGGING commands, what is the name of the system macro you need execute permission on?

   _____ 

2. How is this macro initially created?

   _____ 

3. What is a negative impact of the following statement?

   BEGIN LOGGING WITH TEXT ON EACH ……

   _____

4. With DBQL, what is the size of the default text captured for queries? _____

5. True or False. With DBQL, the LIMIT SUMMARY option cannot be used with any other LIMIT.

6. True or False. With DBQL, use WITH SQL option only captures a maximum of 10,000 characters.

7. True or False. With DBQL, the option WITH ALL ON ALL is typically a good choice.

8. True or False. With DBQL, default rows are logged in the DBC.DBQLogTbl.

Lab Exercise 49-1

The following page contains the start of this lab exercise.

Lab Exercise 49-1

Lab Exercise 49-1

Purpose

In this lab, you will use Teradata SQL Assistant to view information in the data dictionary about security defaults and invalid logons in the system (use Appendix D).

Tasks

1. What system security defaults are in effect for your system?

Number of days to expire password:	_____
Minimum number of characters required:	_____
Maximum number of characters required:	_____
Are digits allowed?	Yes____ No____
Are special characters allowed?	Yes____ No____
Maximum failed logons permitted (0=never lock):	_____
Minutes to elapse before unlocking:	_____
Days to expire before password reuse:	_____

2. Are these the security defaults that are in effect for your username? Yes or No.

3. Is a Profile in effect for your username? If so, what is the name of your Profile? _____

4. If a Profile is being used, which attributes in the Profile override the system security defaults?

_____

Lab Exercise 49-1 (cont.)

The following page continues this lab exercise.

Lab Exercise 49-1 (cont.)

5. Using the DBC.LogOnOffV view, list the "BAD" logon attempts on your system that have occurred during the last ten days. Qualify the SELECT using LIKE 'BAD%'.

   Number of Bad Logons (System) _____
   Number of Bad Logons (Your UserName) _____

6. Using the DBC.SessionInfoV, list the sessions currently logged on your system.

   Total number of Sessions (System) _____
   Total number of sessions (Your UserName) _____

Lab Exercise 49-2

The following page contains the start of this lab exercise.

Lab Exercise 49-2

Lab Exercise 49-2

Purpose

In this lab, you will use Teradata SQL Assistant to list access and query logging rules that are being used and the entries in these logs for your username.

Tasks

1. Using the DBC.AccLogRulesV view, list the access log rules that are in effect for your username.

   Which codes are being logged and what type of logging is being captured?

	Code	Type of Logging	SQL Function Being Logged
Ex.	*CDB*	*B +*	*Create Database*

2. How many different access logging rules are there for all users? Count _____

Lab Exercise 49-2 (cont.)

The following page continues this lab exercise.

Lab Exercise 49-2 (cont.)

3. Execute the following statement.

   CREATE DATABASE Test FROM DBC AS PERM=0;

   (this command should fail – access right violation)

4. Using the DBC.AccessLogV view, list the access log entries for the last 2 weeks for your username.

   How many entries are in this log have been granted? _____

   How many entries are in this log have been denied? _____

   What is the difference between the Create Table and the Execute command log entries?

   _____

Lab Exercise 49-2 (cont.)

The following page continues this lab exercise.

Lab Exercise 49-2 (cont.)

5. **Using the DBC.DBQLRulesV view, list the attributes of query log rule that is in effect for your username.**

   Explain Text Logged ____
   Objects Logged ____
   Full SQL Logged ____
   Execution Steps Logged ____
   Summary ____ **If Summary, times** ____ ____ ____
   Threshold ____ **If threshold, time** ____ **Type of Criterion** ____
   SQL Text Size ____

6. **Using the DBC.QrylogV view, list the logged queries for your username and how many are there?**

   Count = _____

   Using this view, how many queries are logged for all of the users with usernames like 'student%'?

   Count = _____

7. **(Optional) Using the DBC.QrylogSummaryV view, what is the count of queries that have been executed for your username? (Hint: Join the DBC.QrylogSummaryV to the DBC.QryLogV)**

   Count = _____

   (Optional) Using this view, what is the count of queries that have executed for all of the users with usernames like 'student%'?

   Count = _____

Notes

Module 50

Priority Scheduler

After completing this module, you will be able to:

- **Explain the purpose of Resource Partitions, Performance Groups, and Allocation Groups.**
- **Determine the minimum % of resources a user can expect with a specific performance group.**

Notes

Table of Contents

Levels of Workload Management 50-4
Priority Scheduler Facility 50-6
Priority Scheduler Architecture 50-8
Priority Scheduler Architecture with TASM Workloads 50-10
Priority Scheduler Concepts 50-12
Resource Partitions and Performance Groups 50-14
Relative Weights 50-16
Performance Periods and Milestones 50-18
CPU Usage Limits with Priority Scheduler 50-20
Use of Performance Groups 50-22
Getting Started with Priority Scheduler 50-24
Schmon Utility 50-26
Schmon Example 50-28
Priority Scheduler Administrator 50-34
Summary 50-36
Module 50: Review Questions 50-38

Levels of Workload Management

The facing page illustrates three tiers of workload management. This module provides details on Priority Scheduler. The Teradata DWM, Teradata QS, and Database Query Log facility are covered in other modules.

Teradata Dynamic Workload Manager (TDWM) or Viewpoint Workload Designer

Teradata Dynamic Workload Manager (also known as Teradata DWM or TDWM) or Viewpoint Workload Designer provides a graphical user interface (GUI) for creating rules that manage database access, increase database efficiency, and enhance workload capacity. Via the rules, queries can be rejected, throttled, or executed when they are submitted to the Teradata Database.

Teradata Query Scheduler (QS) is designed to provide a facility to submit Teradata SQL jobs to the Teradata Database. TQS is not shown on the facing page, but is an external tool that simply is submits SQL to Teradata.

Priority Scheduler

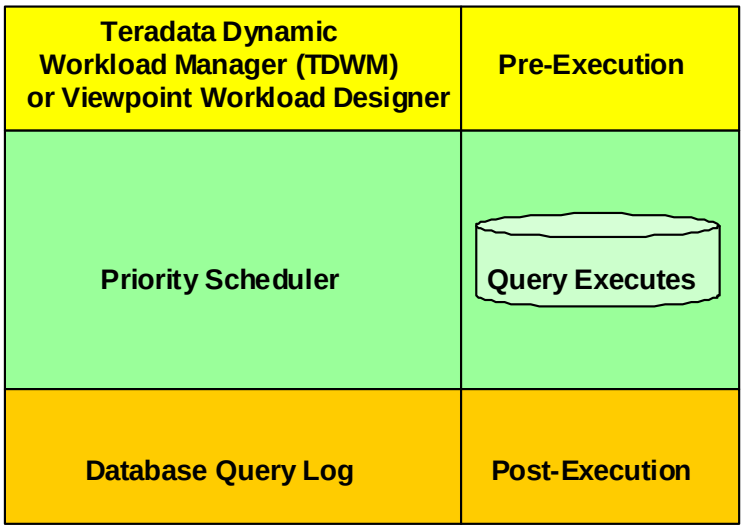
The Priority Scheduler is a resource management tool that controls how compute resources (e.g., CPU) are allocated to different users in a Teradata Database system. This resource management function is based on scheduler parameters that satisfy your site-specific requirements and system parameters that depict the current activity level of the Teradata Database system. You can provide Priority Scheduler parameters to directly define a strategy for controlling compute resources.

Database Query Log

The Database Query Log (DBQL) is a feature that lets you log query processing activity for later analysis. Query counts and response times can be charted and SQL text and processing steps can be compared to fine-tune your applications for optimum performance.

Levels of Workload Management

Three Tiers of Workload Management



TDWM or Viewpoint Workload Designer
Control what and how much is allowed to begin execution.

Priority Scheduler
Manage the level of resources allocated to different priorities of executing work.

Database Query Log
Analyze query performance and behavior after completion.

Priority Scheduler Facility

The Priority Scheduler Facility (PSF) provides a resource partition hierarchy that allows you to control system resources, specifically the CPU resource. With this utility, processes have an externally assigned priority associated with their database session that the Priority Scheduler Facility uses to allocate CPU and I/O resources. Characteristics include:

- Automatic change of priority if needed
  - Time of day
  - Resource usage at the session or query level
- All work in the database treated equal
  - Not biased toward the short and the quick
  - No punishment for the lengthy
- Flexible: When activity is sparse, lower priority jobs get more resources.
- Offers utilities to define scheduling parameters and to monitor your current system activity.

Why create a customized priority environment?

- Assign very high priority users to a very high priority level to support Active Data Warehouse applications.
- Establish priorities to control the impact of TPump load jobs on short, medium and long DS queries that are running at the same time.
- Create a consistent service level for web requests supported in a database doing a mix of decision-making.
- Provide better service for your more important work.
- Control resource sharing among different applications.
- Automate changes in priority by time of day or by amount of CPU used.
- Place a ceiling on Teradata Database system resources for specific applications.

Priority Scheduler Facility

- Teradata's facility to mandate how database resources will be shared.
  - It is a weight-based system that uses relative weights to control how frequently and quickly CPU resources will be available for different categories of work.
  - Does not provide more capacity, but simply allows more control over how the available capacity is used.
- Runs independently on each node in the configuration.
  - Accumulates CPU usage and I/O demand independently on each node and is not coordinated across nodes.
- Priority can automatically change if needed (defined by performance periods or milestones).
  - Time of day
  - CPU Resource usage at query or session level
- Should be used AFTER application-level tuning has been done.
- To configure Priority Scheduler, use:
  - schmon command-line utility
  - Priority Scheduler Administrator (PSA) via Teradata Manager
  - Teradata Dynamic Workload Manager (TDWM) or Viewpoint Workload Designer when TASM workloads are enabled

Priority Scheduler Architecture

Priority Scheduler is a resource management tool that controls the dispersal of computer resources in a Teradata Database system. This resource management tool uses scheduler parameters that satisfy site-specific requirements and system parameters that depict the current activity level of the Teradata Database system. You can provide Priority Scheduler parameters to directly define a strategy for controlling computer resources.

The Priority Scheduler does the following:

- Allows you to define a prioritized weighting system based on user logon characteristics.
- Balances the workload in your data warehouse based on this weighting system.

Priority Scheduler includes default parameters that provide four priority levels with all users assigned to one level using performance groups of L, M, H, and R. Additional performance groups can be created and a performance group (e.g., DM) is specified as part of the Account ID.

If TDWM workloads are NOT enabled, when a SQL request is issued, it enters Teradata, is parsed, and broken into steps. Each step is then dispatched as one or many individual processes. Each process active on behalf of a query executes at the same priority. The priority will depend on how the administrator has put together the several components in the priority framework.

V2R6 Priority Scheduler Changes

A number of enhancements have been made with Priority Scheduler with Teradata V2R6.

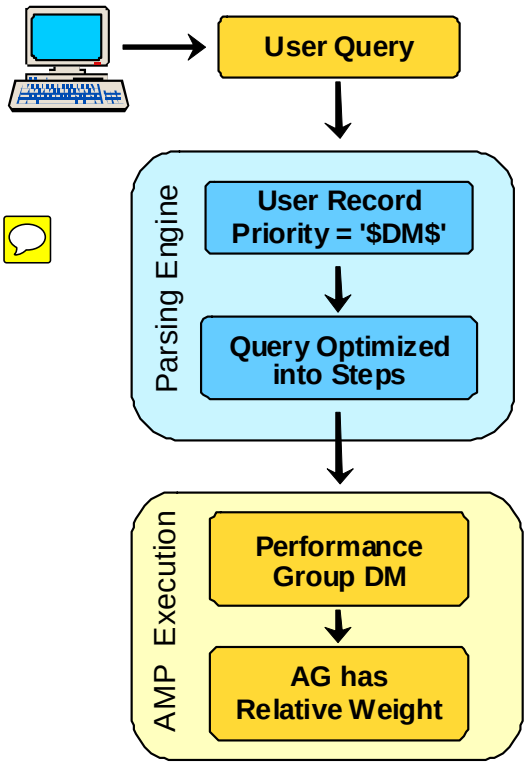
Several options/parameters have been removed, including:

- Internal Performance Groups
- Attributes such as Priority ON/OFF, I/O Prioritization, Throughput vs. Response
- The VALUE parameter of the performance group, which in V2R5 controlled ranking within a resource partition (0-7)
- The allocation group POLICY (default, immediate, relative, absolute)
- The limit on the number of performance groups within one resource partition (was 8 in V2R5)

New enhancements include:

- DBS-generated critical work has been disassociated from the default resource partition. This internal critical work now runs outside the user-controlled priorities as “system” work, and means you no longer have to keep RP0 as the highest-weighted resource partition.
- New allocation group parameter to limit CPU given to an AG.
- Default time quantum for UNIX is now 10 ms., was 20 ms.

Priority Scheduler Architecture



If TASM workloads are not enabled, priorities are assigned by Priority Scheduler in this manner.

- User Logs on a session with an Account ID.
- Access of USER record confirms Account ID which establishes the Performance Group.

  The "priority" for the session is effectively determined by the Account ID.
- A query is submitted and the Optimizer breaks query into processing steps.
- Each step is sent to the AMP as a process, belonging to the User's Performance Group (PG).
- Each PG associates the process to one Allocation Group (AG).
- The relative weight in the AG controls the priority of the process.

Priority Scheduler Architecture with TASM Workloads

TDWM (Teradata Dynamic Workload Manager) allows for the creation of workloads. This capability is part of a new concept called **Teradata Active System Management** (TASM). TASM is made up of several products/tools that assist the DBA or application developer in defining and refining the rules that control the allocation of resources to workloads running on a system. These rules include filters, throttles, and "workload definitions".

The concept of workloads is new with Teradata V2R6.1. To create workloads, you need to use TDWM Release 6.1 (or later) with the following Teradata releases.

- UNIX MP-RAS – Teradata V2R6.0.2 or later
- Windows 2003 or Linux – Teradata V2R6.1.0 or later

TASM and workload definitions will be covered in more detail later in this course.

What is a Workload Definition?

A workload represents a portion of the queries that are running on a system. A Workload Definition (WD) is a workload grouping and its operating rules to assist in managing queries. The requests that belong to the same workload will share the same resource priority and exception conditions. It consists of:

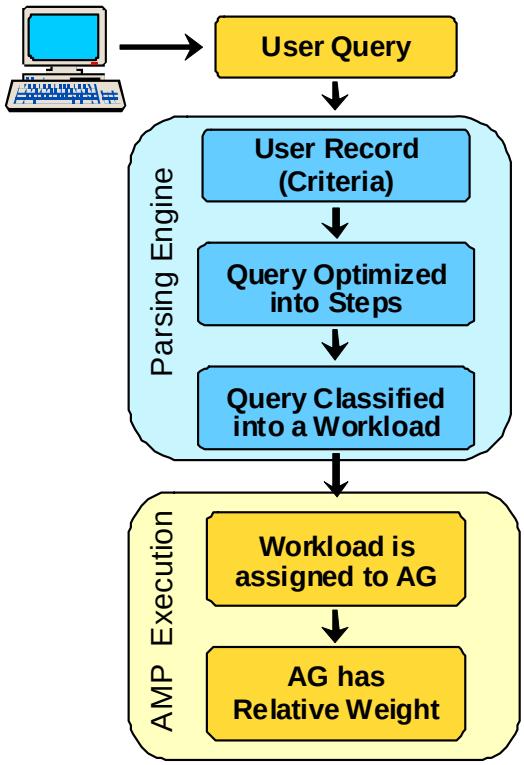
- Classification Criteria: criteria to determine which queries belong to the workload. This criteria defines characteristics which are detectable prior to query execution. This is also known as the *"who"*, *"where"*, and *"what"* criteria of a query. For example, *"who"* may be an account name, *"where"* is the database tables being accessed, and *"what"* may be the type of statement (UPDATE) being executed.

- Exception Criteria: criteria to specify "abnormal" behavior for queries in this workload. This criterion is only detectable after a query has begun execution. If an exception criterion is met, the request is subject to the specified exception action which may be to lower the priority or abort the query.

- Operating Periods: a description of hours of the day and/or days of the week (or month). Directives may be specified for exception handling and Priority Scheduler settings can be changed for each operating period.

A Workload Definition is mapped to an Allocation Group (AG) of Priority Scheduler.

Priority Scheduler Architecture
with TASM Workloads



If TASM workloads are enabled, priorities are assigned by Priority Scheduler in this manner.

- User Logs on a session.
- A query is submitted and the Optimizer breaks query into processing steps.
- Queries are classified based on criteria (who, where, and what) and the query is assigned to a Workload.

  Different types queries (e.g., Tactical and DSS) from the same user can automatically be assigned to different workloads and effectively different priorities.
- Each Workload is assigned to an Allocation Group (AG).
- Each step is sent to the AMP as a process, effectively associated with an Allocation Group (AG).
- The relative weight of the AG controls the priority of the process.

Priority Scheduler Concepts

To establish the partition hierarchy, establish values for:

- Resource partitions (up to 5)
- Performance groups (up to 40)
- Performance periods (up to 8 per performance group)
- Allocation groups (up to 200)

Resource Partitions

A Resource Partition is a collection of prioritized Performance Groups. A Resource Partition carries a weight that will be compared to other Resource Partition weights.

You can divide your system and user base into resource partitions (RP) that you usually distinguish by use or by accounting strategy. You must number and name the partitions, and assign a weight to each partition that determines the total system resources they receive.

Performance Groups

You define Performance Groups within each Resource Partition. The Performance Group names are used in user Account IDs and determine the priority level for the user.

- Performance Group names match the Account ID string and must be unique.

The default resource partition has a set of 4 performance groups (L, M, H, and R). You can define as many performance groups in a resource partition as you wish. However, the total number of performance groups that can be defined in the system is 40.

Performance Periods

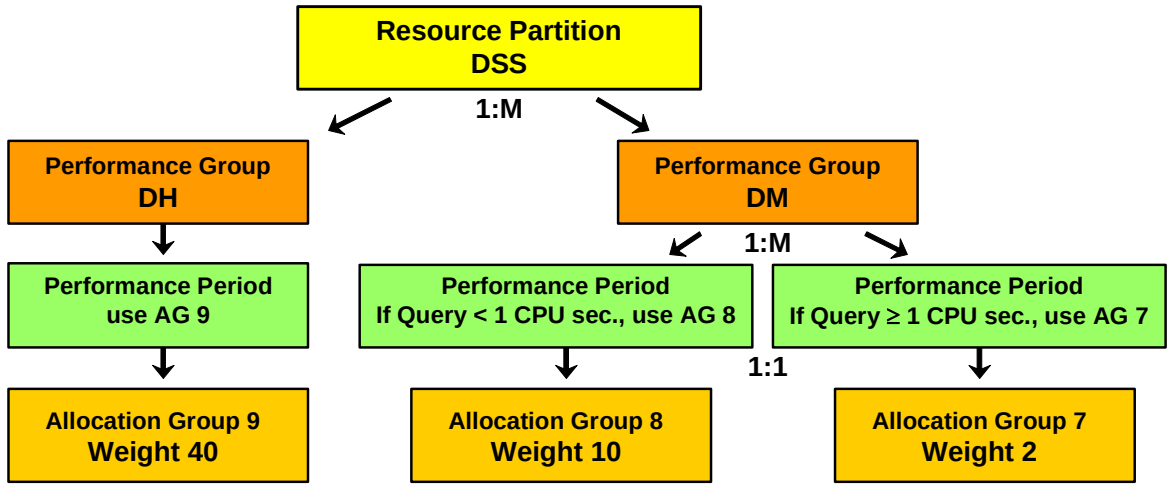
Performance periods link a performance group to an allocation group. Up to 8 performance periods can be assigned to one performance group. Performance periods are based on milestones and are one of three types.

- T for time-of-day
- Q for query resource usage
- S for session resource usage

Allocation Groups

An allocation group (AG) defines a weight that is compared to other Allocation Group weights. These weights determine the amount of scheduling resources allocated. Allocation groups may be referenced by more than one performance period and performance group. However, all references to an allocation group must come from the same resource partition.

Priority Scheduler Concepts



- Maximums of 5 Resource Partitions, 40 Performance Groups, and 200 Allocation Groups
  - Starting with V2R6, all Performance Groups can be placed in a single resource partition (flattened approach) and may be easier to manage.
- Maximum of 8 Performance Periods per Performance Group
  - One Performance Period is assigned to one Allocation Group (1:1 relationship)
- User accounts specify performance group names – for DH, specify 'DH' in account id.

Resource Partitions and Performance Groups

Teradata comes with Resource Partition 0 defined for customer use. Many customers may only need Resource Partition 0 and will not need to define additional Resource Partitions.

Resource Partition 0 uses the standard Teradata priorities – L, M, H, and R. The default allocation weights are shown on the facing page. These priorities are used at the beginning of the Account IDs and are specified as: $L, $M, $H, and $R.

The complete description of Resource Partition 0 is shown below for Teradata V2R6.0.

Perf Group #	Perf Group Name	Alloc Group #	Alloc Group Weight
0	L	1	5
1	M	2	10
2	H	3	20
3	R	4	40

Note: With Teradata V2R5.1 (and previous release), RP# 0 is also used by internal Teradata software (rollbacks, deadlock detection, etc.). Therefore, RP# 0 is usually given the highest resource partition weight with V2R5.1 and previous releases of Teradata. Starting with Teradata V2R6.0, internal work is no longer assigned to the Default partition.

What is a weight?

Weights (not percentages) are assigned at Resource Partition level and to Allocation Groups within a Resource Partition. Weights are used at the Resource Partition and Allocation Group levels to determine the relative proportion of resources to allocate to the user. Weights are:

- A numeric value used at the Resource Partition Level to compute a relative weight (compared to other Resource Partitions) to determine the proportion of resources the processes of the entire Resource Partition are to receive.
- A numeric value used at the Allocation Group Level to compute a relative weight (within the Resource Partition) to determine the proportion of resources the processes of the Allocation Group are to receive.

Additional Resource Partitions

Additional resource partitions may be added to the Priority Scheduler. The maximum number of resource partitions is 5. The facing page contains an example with a second resource partition named **Tactical.**

For example, you may assign the following Performance Groups to different types of users.

TL	Tactical queries that are all AMP operations – possibly utilize a NUSI
TH	Tactical queries that are one or two AMP operations – possibly utilize a PI or USI

Resource Partitions and Performance Groups

RP 0 – Default Weight – 20	RP 1 – Tactical Weight – 60	RP 2 – DSS Weight – 20

PG #	Name	AG #	Weight
0	L	1	5
1	M	2	10
2	H	3	20
3	R	4	40

PG #	Name	AG #	Weight
4	TL	5	10
5	TH	6	40

PG#	Name	AG #	Weight
6	DL	7	2
7	DM	8	10
8	DH	9	40

Resource Partition 0 is provided with Teradata.

To create this partition, schmon -b 1 Tactical 60

To create this partition, schmon -b 2 DSS 20

Example:

- L, M, H, or R – assigned to Batch and Load jobs
- TL – assigned to Tactical queries that are all AMP operations (e.g., NUSI)
- TH – assigned to Tactical queries that are one or two AMP operations (e.g., PI)
- DM – assigned to unknown DSS queries (e.g., ad hoc)
- DH – assigned to known DSS queries

Assuming queries are active in all 3 partitions, system resources are effectively divided between the three partitions as follows:

Default (20/100) = 20% Tactical (60/100) = 60% DSS (20/100) = 20%

Relative Weights

Resource partition weight is a relative weight, since its value is relative to the weights of the currently active partitions. Dividing the weight of a partition by the sum of the weights of all active partitions gives the percentage of total Teradata Database system resources for that partition. The concept applies to performance groups within a resource partition.

The facing page contains two examples of how the system resource is divided between resource partitions and between performance groups within a resource partition.

Relative Weights

Relative weights are based on the assigned weights of the ACTIVE Allocation Groups. Fewer ACTIVE AG's will result in more CPU priority for those AG's that are ACTIVE.

	PG #	PG Name	AG #	AG Weight	Active Sessions	Relative Weight Calculation	AG %
RP 0 Weight 20	0	L	1	5	Y	20/100 x 5/35	2.9
	1	M	2	10	Y	20/100 x 10/35	5.7
	2	H	3	20	Y	20/100 x 20/35	11.4
	3	R	4	40	N		
RP 1 Weight 60	4	TL	5	10	Y	60/100 x 10/50	12.0
	5	TH	6	40	Y	60/100 x 40/50	48.0
RP 2 Weight 20	6	DL	7	2	N		
	7	DM	8	10	Y	20/100 x 10/10	20.0
	9	DH	9	40	N		

	PG #	PG Name	AG #	AG Weight	Active Sessions	Relative Weight Calculation	AG %
RP 0 Weight 20	0	L	1	5	Y	20/80 x 5/15	8.3
	1	M	2	10	Y	20/80 x 10/15	16.7
	2	H	3	20	N		
	3	R	4	40	N		
RP 1 Weight 60	4	TL	5	10	N		
	5	TH	6	40	Y	60/80 x 40/40	75.0
RP 2 Weight 20	6	DL	7	2	N		
	7	DM	8	10	N		
	9	DH	9	40	N		

Performance Periods and Milestones

A performance period type defines a type of threshold used to change performance periods for a Performance Group. The milestone limit represents that threshold. The milestone limit triggers an automatic change in Allocation Group when a threshold you define is reached.

You can express milestone limits in the following units:

- Time-of-day – minutes of military time (0 – 2359) and represent time periods during a 24-hour day.
- Session resource usage – defined in CPU seconds (0 – 86400) and represents an amount of session CPU resource consumption per node.
- Query resource usage – defined in CPU seconds (0 – 86400) and represents an amount of Query CPU resource consumption per node.

The facing page contains an example of having multiple performance periods and automated priority changes.

Day-of-Week options for Time-of-Day Milestone

Optionally, you can use a day-of-week specification with a time-of-day milestone for a performance period to indicate days when the performance period is to be active. This specification might indicate one or more individual days, or a range of days, but not both.

Days are numbered sequentially from 0 to 6, Sunday through Saturday. A range of days is indicated by two day numbers separated by a hyphen.

If a day-of-week specification is present for one performance period, then day-of-week must be present for all.

A time-of-day milestone for a performance period defines the end of a time period during which the associated Allocation Group is to control processes. The beginning of each period is the end of the preceding period. This might be on a previous day if day of week has been specified.

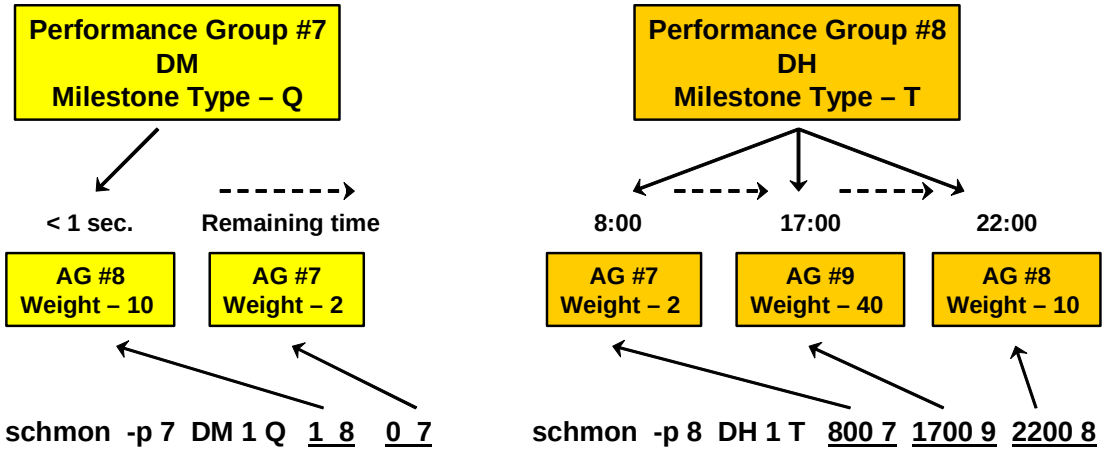
Resource Usage

When you express milestone limits in units of session or query resource usage, their values indicate the total resource usage of all processes working on behalf of a session, or a query submitted by a session, on that node. In this case, a performance period controls processes grouped into sessions. Since some sessions or queries might consume more or less resources than others, several performance periods of a Performance Group might be actively controlling processes of different sessions concurrently.

Performance Periods and Milestones

A performance period type defines a type of threshold (milestone limit) used to change performance periods for a Performance Group. Milestone options are:

- **T – Time-of-day**
- **S – Session resource usage (based on CPU time accumulated in a session by node)**
- **Q – Query resource usage (based on CPU time accumulated by a query by node)**



Milestones cause a change in Allocation Group, not Performance Group.

CPU Usage Limits with Priority Scheduler

It is possible to limit the amount of CPU usage that is available to the system, to a specific resource partition, or to a specific allocation group. A percentage can be included with each of these levels. The general format of the "Limit" follows:

Limit – range is 1 through 100 or the value "none"

- A value of 100 indicates that no limit is to be enforced.
- A character string of "none" indicates that no limit is to be enforced.
- For Resource Partitions and Allocation Groups, if *limit* is not present, any previously defined limit is removed.

System Level (Node level)

To limit the CPU usage at the system level, you can specify a percentage value to limit the amount of CPU usage available to all Teradata Database sessions. This usage does not include non-Teradata work, such as time-share users, I/O or other interrupt services, Gateway processing, or streams work.

For example, to set the Teradata Database system CPU usage limit to 80%, use the following command:

schmon –l 80 (Sets system CPU % limit to 80%)

Resource Partition Level

To limit CPU usage at the resource partition level, you can specify a percentage limit on total CPU usage by all processes controlled by that a resource partition.

For example, to set a 75% limit at a Resource Partition level, use the following command:

schmon –b 1 Tactical 100 75 (Sets RP CPU % limit to 75%)

Note: The Resource Partition weight is 100 and the CPU limit is 75%.

Allocation Group Level

To limit CPU usage at an allocation group level, you may specify a percentage limit on total CPU usage by all processes controlled by the allocation group.

CPU Usage Limits with Priority Scheduler

CPU usage limits may be used to place a "ceiling" on the amount of resources that are available to a specific level.

- **System level**

  Example: To limit system CPU usage to 80%,

  ```
  schmon  –l  80
  ```

  (l: lower case L)

- **Resource Partition level**

  Example: To limit CPU usage to 75% for Resource Partition "Tactical",

  ```
  schmon –b 1  Tactical  60     75
                         ↑      ↑
                      weight  limit
  ```

- **Allocation Group level**

  Example: To limit CPU usage to 50% for Allocation Group #9,

  ```
  schmon –a  9  N    40     50
                     ↑      ↑
                  weight  limit
  ```

Note: All weights and CPU limits are enforced by Priority Scheduler at the node level.

Use of Performance Groups

Performance Group names are specified within the Account parameter when creating or modifying a user. Performance Group names must be delimited with a “starting $” and an “ending $”. For example, DH would be identified as DH.

In earlier Teradata Database versions, a single character (L, M, H, or R) prefixed by the $ character in the Account ID string indicated the Performance Group. To provide backward compatibility, Priority Scheduler provides each of these single character identifiers as a Performance Group name within default Resource Partition 0.

In this special case, the four Performance Group names (L, M, H, and R) do not require an ending $ character in the Account ID string. In this case, the strings $M and M are equivalent.

The logon process assigns each user session to a Performance Group based on the accounted string of the logon command. If a Performance Group cannot be assigned based on the Account ID string, a default assignment is made.

Each session has a designated Performance Group. When a session begins a process, it falls under the control of a performance period whose milestone limit conditions are met.

Use of Performance Groups

Performance Group names are specified within the Account parameter when creating or modifying a user.

```
CREATE USER rjones AS PERM=0, SPOOL=500e6, PASSWORD=rj182130,
    ACCOUNT=('$DM$_TT_&S_&D&H', '$DH$_TT_&S_&D&H');
```

If “rjones” only needs 'DM' as a priority, then only include that account ID.

Users log on to Teradata using valid account IDs which can include Performance Group names.

.logon educ1/rjones, rj182130	Uses DM performance group
.logon educ1/rjones, rj182130, 'DH_TT_&S_&D&H'	Uses DH performance group

Getting Started with Priority Scheduler

To establish the partition hierarchy, establish values for:

- Resource partitions (up to 5)
- Performance groups (up to 40)
- Performance periods (up to 8 per performance group)
- Allocation groups (up to 200)

Resource Partitions

Divide the computer system and user base into resource partitions (RP) that you usually distinguish by use or by accounting strategy. You must number and name the partitions. Assign a weight to each partition that determines the total system resources they receive.

Allocation Groups

An allocation group (AG) defines a weight and a division type that determines the amount of scheduling resources allocated. Allocation groups may be referenced by more than one performance period and performance group in the same resource partition.

When creating an allocation group, it is necessary to set the division type.

- N for NONE – resources are divided amongst processes; better for complex queries and all-AMP queries. N is the default and recommended choice for most environments.

- S for SESSION – resources are divided by # of active sessions, then by processes; may be better for simpler queries and single-AMP queries. (This option is usually not needed or used as the impact it has is negligible.)

Performance Groups

Each resource partition has a set of performance groups. Performance groups divide the resource partition in priority groups. 4 performance groups per resource partition are typical. However, with V2R6, you can have more than 4 performance groups per resource partition and having all of the performance groups in one partition (flattened approach) may be useful in some environments.

Performance Periods

Performance periods link a performance group to an allocation group. Up to 8 performance periods can be assigned to one performance group.

- Performance periods make it possible to have changes in priority weight by time or resource usage.

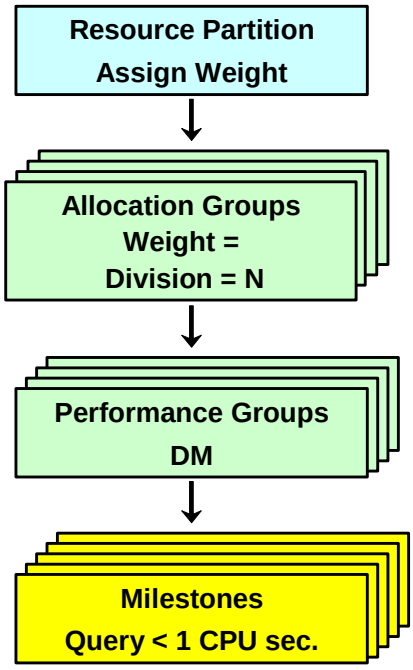
Getting Started with Priority Scheduler

1. **Create and name a Resource Partition and assign a weight.**
   - Optionally assign all Tactical queries to a RP with the highest priority.
2. **Create Allocation Groups.**

   Specify weight and division type (N or S).
   - N – None (default & best choice for most environments)
   - S – Session (normally not used)
3. **Create Performance Groups with unique names.**

   Assign an Allocation Group to the Performance Group or to each Performance Period.

   Optionally define Performance Periods or Milestones. If weight will not change, Session = 0.
4. **Modify users to create Account IDs with Performance Group names – ex. 'DM_TT_&S_&D&H'.**
5. **Use Performance Group names on logons.**



Schmon Utility

The **schmon** utility is used to add and change Priority Scheduler Facility parameters. There are both command-line and an X-Windows versions of this utility.

Schmon – a command-line interface.
Xschmon – a graphical user interface that uses the X-Window system.

The section on the Priority Scheduler Facility in the *Teradata Database User Utilities* reference manual has details for all parameters to all commands you can execute with this utility.

Via Teradata Manager, you can also use the Priority Scheduler Administrator (PSA) utility to manage priority scheduler settings. This is a Windows based utility and provides an easier to use GUI interface.

schmon Utility

- **The schmon utility modifies and displays scheduler parameters and displays current scheduling activity.**
- **schmon -h will display help information about the options.**
- **schmon -d will display …**
  - **current settings including the age and active times**
  - **Allocation Group to Performance Group assignments**
  - **Allocation Group weight and policy settings**
- **schmon -m or -M will display the current scheduling activity.**
  - **This shows the current CPU and I/O usage, the number of processes attached to each Allocation Group and other data.**
- **Priority Scheduler Administrator (PSA) provides a Windows interface to also manage the Priority Scheduler facility.**

Schmon Example

The facing page contains the **schmon** parameters needed to establish the Tactical resource partition shown previously.

Some additional notes:

When creating an allocation group, it is necessary to set the division type.

- N for NONE - Resources are divided amongst processes; better for complex queries and all-AMP queries. N is the default and recommended value for most environments.

- S for SESSION – Resources are divided by # of active sessions, then by processes; better for simpler queries and single-AMP queries.

The example on the facing page shows the division type set to N.

When creating a Performance Group, either T (time), S (session), or Q (Query) is used as the Performance Period. The last milestone for the Session or Query group type must be 0.

Following the T, S, or Q, you can include up to 8 performance period pairs consisting of milestone limits and allocation group #.

To modify the weight of an existing component, use the same commands except with a different weight. The change will be effective immediately. For example:

 # schmon -a 7 S 25 none (changes AG# 7 weight to 25 and removes the limit)

 # schmon -b 1 Tactical 80 none (sets RP# 1 weight to 80 and removes the limit)

schmon Example

The following commands will establish the Tactical resource partition shown earlier.

To create Resource Partitions:

```
# schmon   -b 0   Default     20   100
# schmon   -b 1   Tactical    60   100
# schmon   -b 2   DSS         20   100
           RP#    RP Name     Weight  (Optional % Limit)
```

To create Allocation Groups:

```
# schmon   -a 5   N   10   100
# schmon   -a 6   N   40   100
# schmon   -a 7   N   2    100
# schmon   -a 8   N   10   100
# schmon   -a 9   N   40   100
           AG#    Division Type (N or S)   Weight   (Optional % Limit)
```

To create Performance Groups:

```
# schmon   -p 4   TL   1   S   0   5
# schmon   -p 5   TH   1   S   0   6
# schmon   -p 6   DL   2   S   0   7
# schmon   -p 7   DM   2   Q   1   8   0      7   
# schmon   -p 8   DH   2   T   800 7   1700   9   2200   8
           PG# (unique)   PG Name (unique)   RP#   Session, Query, Time   Mile-stone   AG#   Mile-stone   AG#   Mile-stone   AG#
```

Schmon –d Example Output

An example of **schmon -d** option is shown on the facing page.

Options for **schmon** include:

schmon

-a ['all' | <AG#>] | [<AG#> -x] | [<AG#> -s|-S] | [<AG#> [X] <weight> [limit]]
allows you to set/display allocation groups.

-b ['all' | <RP#>] | [<RP#> -x] | [<RP#> -s|-S] | [<RP#> <RPNAME> <weight> [limit]]
allows you to set/display resource partitions.

-p ['all' | <PG#>] | [<PG#> -x] | [<PG#> -s|-S] | [<PG#> <PGNAME> <R> <T> <PP(i)(i = 0,7)>]
allows you to set/display performance groups.

-c [-p|-b|-a|-w|-t] [-f path]	allows you to dump current settings as commands.
-d	displays the current settings in a multiple line format.
-f [path]	specifies a path for which input is to be read.
-h [specific option(s)]	displays help information for the option specified.
-l [limit]	sets the system CPU usage limit.
-m [-S [delay [reps]]	monitors PS statistics for the current node.
-M [-p] [delay [reps]]	monitors PS statistics for the current node.
-s ['all' | <id>] [-S] | [-S]	displays PS data for specified sessions.
-t <age> <active> <disp> <ioconc>	displays Age, Time, Active Time, and Disp. Time.
-w <reserve> <maximum>	sets/displays the number of processes available for AG with the Expedite attribute.
-X [-p|-b|-a|-w|-t]	dumps all the fields from GDO in hexadecimal format.

schmon -d Example Output

Scheduler Times & Attributes:

Age Time(sec): 60.0 Active Time(sec): 61.0 Limit(%): none I/O Concurrency: 10

Resource Partitions (0 - 4)

Id	Partition Name	Weight	Limit
0	Default	20	none
1	Tactical	60	none
2	DSS	20	none

(Note: Limit of 100% is identified as "none".)

Performance Groups (0 - 39)

Id	Group Name	RP	Type	Milestones & Alloc Groups[0-4]
0	L	0	S	0.00 1
1	M	0	S	0.00 2
2	H	0	S	0.00 3
3	R	0	S	0.00 4
4	TL	1	S	0.00 5
5	TH	1	S	0.00 6
6	DL	2	S	0.00 7
7	DM	2	Q	1.00 8 0.00 7
8	DH	2	T	0800 7 1700 9 2200 8

Allocation Groups (0 - 199)

Id	Type	Weight	Limit
1	N	5	none
2	N	10	none
3	N	20	none
4	N	40	none
5	N	10	none
6	N	40	none
7	N	2	none
8	N	10	none
9	N	40	none

(new starting with V2R6)

AWT Expedited work type limits (new starting with V2R5)

res	max
0	80

Schmon –m Example

Examples of the **schmon –m** are shown on the facing page.

An example of **schmon –m** with a single resource partition is shown below. The **–M** option (not shown) provides information for all of the SMPs or nodes.

Answer to question on facing page:

> Queries running between 17:00 (5:00 PM) and 22:00 (10 PM) in the DH group use AG #8.

Column definitions for AG #3 at the bottom of the facing page are:

- **AG:** The ID of each active allocation group. Only allocation groups that have seen new requests for AMP worker tasks during the most recent age interval (usually the last 60 seconds) will appear in this list.

- **#requests:** This column reflects the number of messages that have been received for this allocation group within the age interval. This represents work that needs to acquire AMP worker tasks to get underway. This may be work belonging to any work type.

- **Avg queue wait (msec):** This column shows the average time per re-quest spent waiting for an AMP worker task, in milliseconds, as captured during the age interval. On UNIX, if this is a number from 0 to 5, it is considered normal and is not pointing to a performance issue.

- **Avg queue length:** This represents the average queue length during the age interval for new requests that were waiting in line to be serviced. A zero in this column means that on average, there was no line of requests waiting for an AWT. Since this is an average, and only whole numbers are represented, a zero could represent a fraction.

- **Avg service time (msec):** This is the average amount of time an AMP worker task was held within the recent age interval. This represents wall-clock time, not milliseconds of CPU consumption, as is reported for CPU usage in the higher portion of the schmon –m output.

schmon -m Example

```
schmon -m
  Stats:  Thu Jul 29 17:00:16 2010

    Rel   Avg CPU    Avg I/O    # of  # of
 RP Wgt  %  (msec)   %  (sblks) Procs Sets
=== === === ======= === ======= ===== ===== ==============================
  0  50   0     311   7     597     8     2
  2  50   2    4718  91   13135     4     2

    Rel   Avg CPU     Avg I/O   # of  # of
 AG Wgt  %  (msec)   %  (sblks) Procs Sets  Performance Groups Affected
=== === === ======= === ======= ===== ===== ==============================
  2   8   0     175   5     393     7     1   M
  3  16   0     136   2     204     1     1   H
  8  24   5    3430  26    4717     2     1   DM, DH
  9  48   2    1288  65    8418     2     1   DH
200 MAX   0      63   2      31    69     1   System

System: %CPU  CPU(msec) I/O(sblks) #procs
         ==== ========= ========== ======
            1      5029      13722     37

                Avg queue   Avg queue  Avg service
 AG #requests   wait(msec)  length     time(msec)
=== ========== ==========  ========== =============
  3          1      57.10        0.00        6832.20
200          5       0.00        0.00         639.80
```

System with 2 Resource Partitions being utilized:

Question:
At this time, what AG is used for queries running with DH priority?

Priority Scheduler Administrator

The Teradata Priority Scheduler Administrator (PSA) is a resource-management tool that provides an easy-to-use graphical interface that allows you to define Priority Definition Sets, generate schmon scripts to implement these sets, and monitor and control the Priority Scheduler environment.

This utility is accessed via Teradata Manager. An example of a PSA display representing the Tactical resource partition is shown on the facing page.

Features of PSA include:

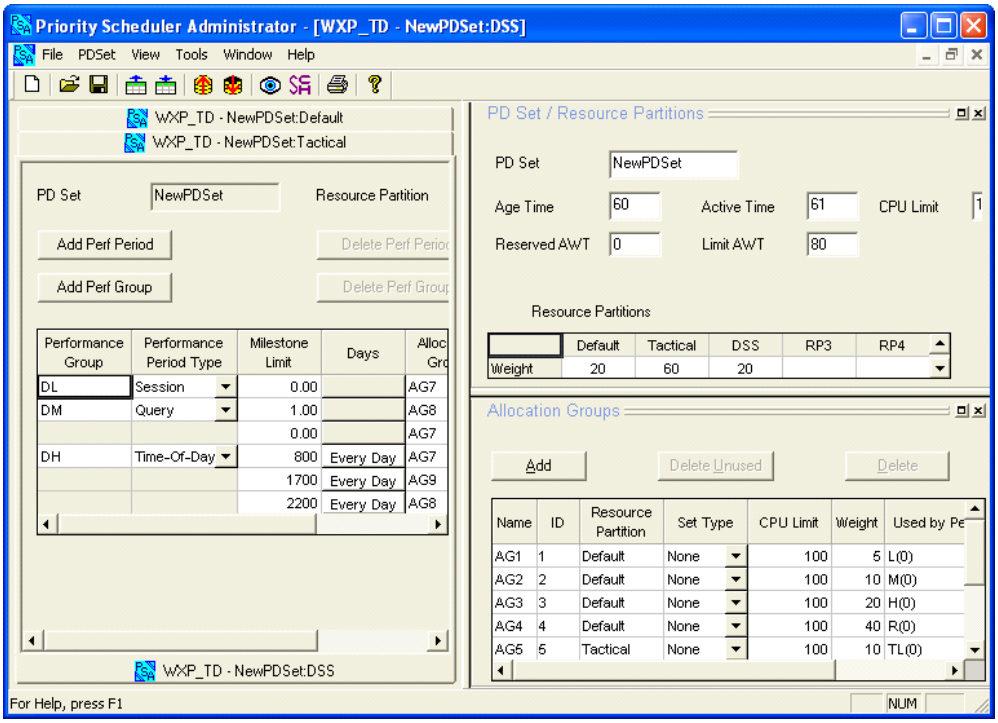
- Makes Priority Scheduler more usable and understandable.
- Provides visualization of Resource Partition and Allocation Group weights and CPU Utilization.
- Eliminates much of your current guesswork about the results of Priority Scheduler changes.
- Easily manage multiple Priority Scheduler configuration profiles.

Corresponding schmon Commands

The following “schmon” commands correspond to the PSA example shown on the facing page.

```
schmon -b 0 Default 20 100
schmon -b 1 Tactical 60 100
schmon -b 2 DSS 20 100
schmon -a 5 N 10 100
schmon -a 6 N 40 100
schmon -a 7 N 2 100
schmon -a 8 N 10 100
schmon -a 9 N 40 100
schmon -p 4 TL 1 S 0 5
schmon -p 5 TH 1 S 0 6
schmon -p 6 DL 2 S 0 7
schmon -p 7 DM 2 Q 1 8 0 7
schmon -p 8 DH 2 T 1700 9 2200 8 800 7
```

Priority Scheduler Administrator



Summary

The facing page summarizes some important concepts regarding this module.

Summary

- **Priority Scheduler is Teradata's facility to mandate how database resources will be shared.**
  - **It is a weight-based system that uses relative weights to control how frequently and quickly CPU resources will be available for different categories of work.**
- **Priority can automatically change if needed (defined by performance periods or milestones).**
  - **Time of day**
  - **CPU Resource usage at query or session level**
- **To configure Priority Scheduler, use:**
  - **schmon command-line utility**
  - **Priority Scheduler Administrator (PSA) via Teradata Manager**
  - **Teradata Dynamic Workload Manager (TDWM) when TDWM workloads are enabled**

Module 50: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 50: Review Questions

1. Given the following, what is minimum % of opportunities to use the CPU resource that the following Performance Groups (PG) can expect.

   TM = _____ DH = _____

Default RP 0 - Weight 20		Tactical RP 1 - Weight 60		DSS RP 2 - Weight 20	
PG Name	AG Wgt	PG Name	AG Wgt	PG Name	AG Wgt
L	5	TL	5	DL	5
M	10	TM	10	DM	10
H	20	TH	30	DH	30
R	40	TR	55	DR	75

2. Without TASM workloads enabled, a user session is associated with a _  _____ which is effectively assigned to an _____.

3. With TASM workloads enabled, a user query is associated with a _  _____ which is effectively assigned to an _____.

Notes

Module 51

Workload Management

After completing this module, you will be able to:

- **Describe the type of workload management rule to apply to limit certain kinds of queries.**
- **List the query attributes that are used to classify a query into a Workload Definition.**
- **Specify the exception actions that are possible for a workload definition.**
- **Place the TASM control options in the proper sequence as they are acted upon by Teradata software.**

Notes

Table of Contents

Levels of Workload Management 51-4
What is TASM? 51-6
TASM Capabilities 51-8
Query Management Architecture 51-10
Query Management Architecture (cont.) 51-12
TDWM Example 51-14
Workload Designer Example 51-16
Filters and Throttles for Query Management 51-18
Object Access and Query Resource Filters 51-20
Object and Load Utility Throttles 51-22
Workload Definitions 51-24
Example of Using Workloads 51-26
Creating Workloads 51-28
WD – Classification Criteria 51-30
Specify Exception Criteria 51-32
Example – Exception Handling 51-34
Teradata Workload Analyzer 51-38
Summary 51-40
Module 51: Review Questions 51-42

Levels of Workload Management

The facing page illustrates three tiers of workload management. This module provides an overview on the types of workload management rules that can be created.

Teradata Dynamic Workload Manager (TDWM) or Viewpoint Workload Designer

Teradata Dynamic Workload Manager (also known as Teradata DWM or TDWM) or Viewpoint Workload Designer provides a graphical user interface (GUI) for creating rules that manage database access, increase database efficiency, and enhance workload capacity. Via the rules, queries can be rejected, throttled, or executed when they are submitted to the Teradata Database.

Teradata Query Scheduler (QS) is designed to provide a facility to submit Teradata SQL jobs to the Teradata Database. TQS is not shown on the facing page, but is an external tool that simply is submits SQL to Teradata.

Priority Scheduler

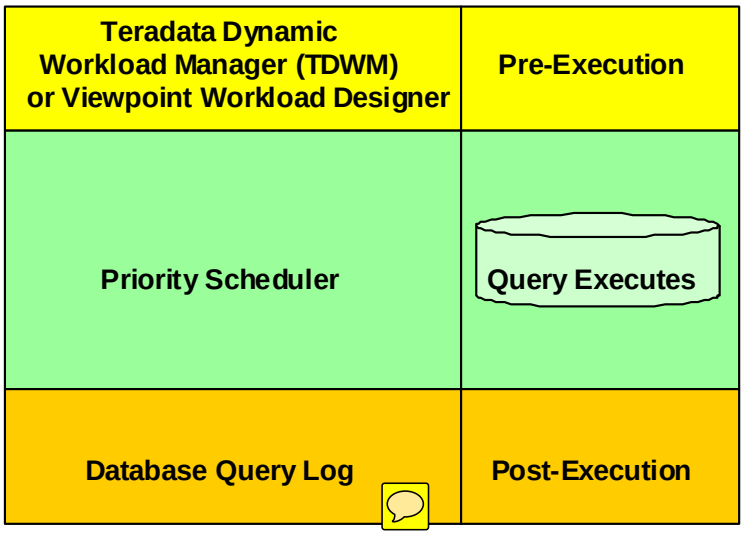
The Priority Scheduler is a resource management tool that controls how compute resources (e.g., CPU) are allocated to different users in a Teradata Database system. This resource management function is based on scheduler parameters that satisfy your site-specific requirements and system parameters that depict the current activity level of the Teradata Database system. You can provide Priority Scheduler parameters to directly define a strategy for controlling compute resources.

Database Query Log

The Database Query Log (DBQL) is a feature that lets you log query processing activity for later analysis. Query counts and response times can be charted and SQL text and processing steps can be compared to fine-tune your applications for optimum performance.

Levels of Workload Management

Three Tiers of Workload Management



TDWM or Viewpoint Workload Designer
Control what and how much is allowed to begin execution.

Priority Scheduler
Manage the level of resources allocated to different priorities of executing work.

Database Query Log
Analyze query performance and behavior after completion.

What is TASM?

Teradata Active System Management (TASM) is made up of several products/tools that assist the DBA or application developer in defining and refining the rules that control the allocation of resources to workloads running on a system. These rules include filters, throttles, and "workload definitions".

Rules to control the allocation of resources to workloads are effectively represented as workload definitions. Tools are also provided to monitor workloads in real time and to produce historical reports of resource utilization by workloads. By analyzing this information, workload definitions can be adjusted to improve the allocation of resources.

The key products to implement TASM "workloads" are:

Teradata Dynamic Workload Manager (TDWM) or Viewpoint Workload Designer
Teradata Manager or Viewpoint
Teradata Workload Analyzer (TWA)

Note: Starting with Teradata Release 13.10, TDWM and Teradata Manager are no longer available.

Teradata Dynamic Workload Manager (known as TDWM or DWM) or Workload Designer is a key supporting product component for TASM. The major functions include:

Define Filters and Throttles
Define Workloads (new) and their operating periods, goals and PSF mapping/weights
Define general TASM controls

The benefit of TASM is to automate the allocation of resources to workloads and to assist the DBA or application developer regarding system performance management. The benefits include:

Fix and prevent problems before they happen. Seamlessly and automatically manage resource allocation; removes the need for constant setup and adjustment as workload conditions change.

Improved reporting of both real-time and long-term trends – Service Level statistics are now reported for each workload. This helps manage Service Level Goals (SLG) and Service Level Agreements (SLA) – applications can be introduced with known response times

Automated Exception Handling – queries that are running in an inappropriate manner can be automatically detected and corrected.

Reduced total cost of ownership – one administrator can analyze, tune, and manage a system's performance.

What is TASM?

What is TASM?

- Teradata Active System Management (TASM) is made up of several products/tools that assist the DBA or application developer in defining (and refining) the rules that control the allocation of resources to workloads running on a system.
- These rules include filters, throttles, and workload definitions.
  - Workload definitions are rules to control the allocation of resources to workloads.

The key products that are used to create, manage, and monitor "workloads" are:

- Teradata Dynamic Workload Manager and Teradata Manager

  or

- Viewpoint Workload Designer and Viewpoint
- Teradata Workload Analyzer (for both TDWM and Viewpoint)

The benefit of TASM is to automate the allocation of resources to workloads.

TASM Capabilities

Teradata Dynamic Workload Manager (also known as Teradata DWM or TDWM) and Viewpoint Workload Designer are products that enable you to effectively manage the access to and utilization of a Teradata Database. These tools provide both Query and Workload Management capabilities.

TDWM or Viewpoint Workload Designer provides a graphical user interface (GUI) for creating rules that manage database access, increase database efficiency, and enhance workload capacity.

What is Query Management?

Query Management is a set of "rules" to determine whether logon and query requests will be accepted by the Teradata Database, and further to determine whether the execution of some query requests should be "delayed" (internally queued). The purpose of "delaying" queries is to limit the number of database resources that are tied up in processing low priority and/or long running queries. Queries that are delayed are still perceived as executing within the user's session.

Why use a Query Management facility?

Enables the DBA to effectively manage access to and the use of Teradata resources.

Allows the processing of logon and query requests from all types of clients sources without any client software requirements.

TASM addresses the key problems of database system overload and network saturation that result from a large number of clients accessing the Teradata Database.

What is Workload Management?

Workload management on a system yields improved workload distribution and customized delegation of resources among the various workloads. A workload represents a portion of the queries that are running on a system. To use workload management in Teradata, a set of workload definitions must be established and enabled. A Workload Definition (WD) is a workload grouping and its operating rules to assist in managing queries. The requests that belong to the same workload will share the same resource priority and exception conditions.

Why use a Workload Management (new with TASM) facility?

Assign queries to the correct workload before they start executing. Fix and prevent problems before they happen. Seamlessly and automatically manage resource allocation; removes the need for constant setup and adjustment as workload conditions change.

A new application (i.e., Teradata Workload Analyzer) is also available to help in migrating existing environments to a "workload" environment.

TASM Capabilities

TASM provides a Query Management (QM) capability.

- A set of user-defined "rules (or filters and throttles)" is created to determine whether logon and query requests will be accepted by the Teradata Database.
  - These rules also determine whether the execution of some query requests should be "delayed" (internally queued).
- Query Management provides "non-workload" filters and throttles:
  - Filters – object access and query resource rules used to reject queries
  - Throttles – object and load utility rules used to delay or reject queries

TASM also provides a Workload Management (WM) capability.

- A set of user-defined "workload definitions" is created to control the allocation of resources to workloads.
- Queries are associated with a "workload" based on who, where, and what criteria.

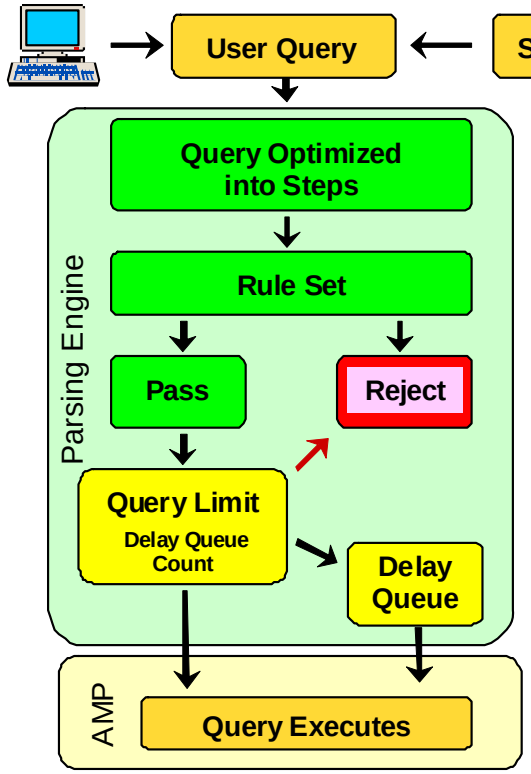
Query Management Architecture

The rules you create are stored in tables in the Teradata database. Unless otherwise specified, every logon and every query in every Teradata Database session is checked against the enabled rules. That includes SQL queries from any supported Teradata Database interface, such as BTEQ, CLIv2, ODBC, and JDBC.

The TASM rules are loaded into the Dispatcher components of the Teradata Database. When a Teradata client application issues a request to the Teradata Database, the request is examined and checked by TASM functions in the Dispatcher before being forwarded to the AMPs to execute the request against the user database.

The Query Management component examines database log on and query requests. It also analyzes the resource criteria of those requests and the objects it references. TASM then compares the requests against the active rules to see if the requests should be accepted, rejected, or delayed

Query Management Architecture



User logs on a session with an Account ID and submits a query.

- A User and/or account may be associated with one or more Query Management Rules.
- Rules-checking happens in the PE (Dispatcher) before the query begins to execute.
- Queries that do not pass Filter Rules are rejected.
- Queries that pass Throttle Rules can also be delayed or rejected if the query would exceed a query limit.
- Delayed queries are managed by Query Workload Manager.
- Queries from all client sources are checked for compliance to Query Management Rules.

Query Management Architecture (cont.)

The illustration on the facing page expands the rules shown in the previous illustration.

Query Management rules have been generally classified into three groups: Filters, Throttles, and Workload Definitions.

Query Management analyzes the incoming requests and compares the requests against the active rules to see if the requests should be accepted, rejected, or delayed.

- Queries that do not pass Filter Rules are rejected
- Queries that do not pass Throttle Rules can be delayed or rejected
- Queries that pass both Filter and Throttle rules are checked against Workload rules. Additional throttles can also be applied at the Workload Definition level.
- As queries execute within their assigned workload, they will be monitored against any exception rules.
- Violations of exception rules can invoke several actions from changing workloads, abort the query, send alert or run a program.

TASM provides two major capabilities.

- With workloads, queries can be rejected, delayed or run in a performance group based on query attributes, not just the account string.

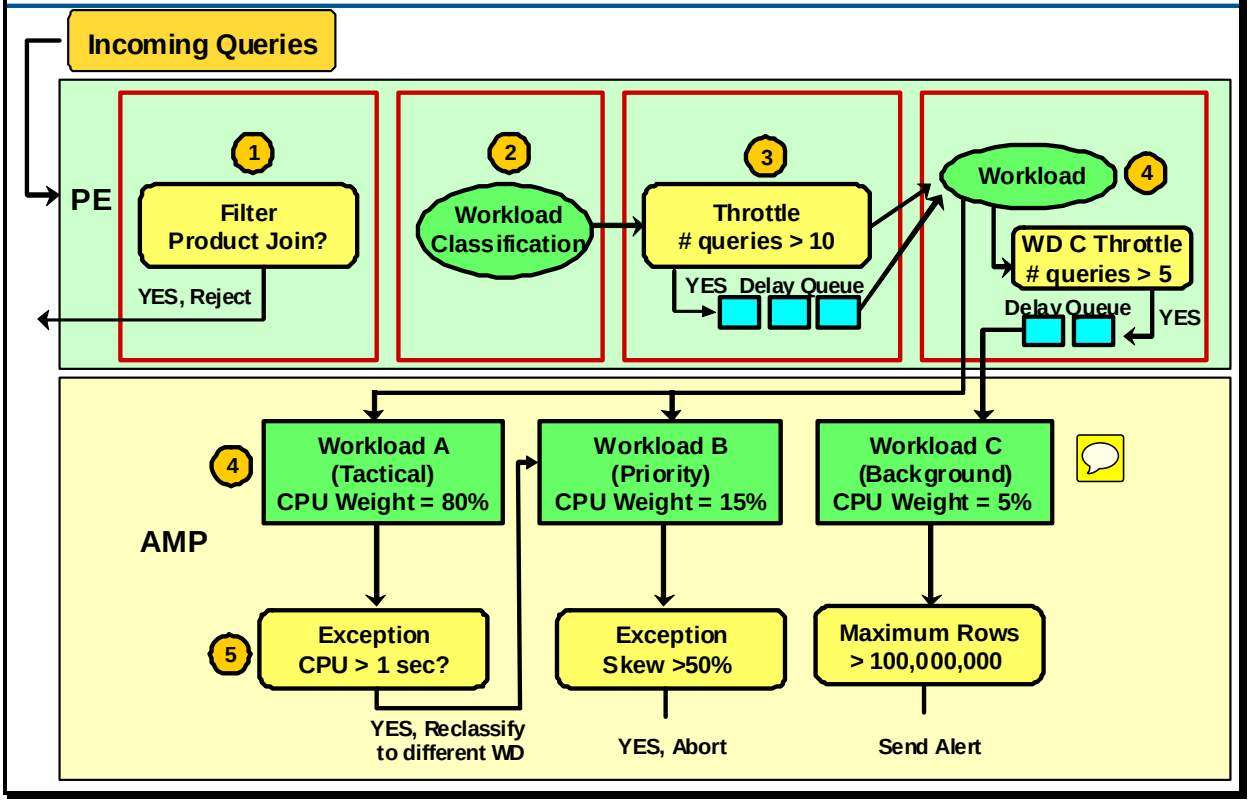
- Executing queries can be monitored and acted on. TASM has the ability to monitor and manage running queries. This is done with exception criteria that is specified in the workload definition and the query with the criteria detected can have its priority changed (e.g., lowered) or even aborted.

How are queries associated with a workload?

- Teradata Parsing Engine software “classifies” a query into a Workload Definition (WD). It takes the query attributes (user name, account, optimizer estimates, etc.) and puts the query into the correct WD.

The example in the illustration shows five ways to control workload resource allocation.

Query Management Architecture (cont.)



TDWM Example

TDWM is a graphical user interface (GUI) client utility that runs on Microsoft Windows. It allows a database administrator (DBA) to control the behavior of TDWM. The following are the main features of query management:

Create, delete, modify, view, enable, and disable filters, throttle, or workloads.
Database Browser window for associating query objects with defined filters.
Grant bypasses privileges to specific users, groups of users, or accounts.
You can configure TDWM to affect how rules are enforced by ignoring EXPLAIN estimates that are below a specified level of confidence.
Apply (or notify the database of) the latest filter, throttle, or workload changes.

The TDWM rules you create are stored in tables in the **tdwm** database. Unless otherwise specified, every logon and every query in every Teradata Database session is checked against the enabled TDWM rules in the **tdwm** database. That includes SQL queries from any supported Teradata Database interface, such as BTEQ, CLIv2, ODBC, and JDBC.

Note: Although every SQL request is subject to TDWM rules, you can set up specific users to bypass TDWM checking. These users are also called “unrestricted users”.

TDWM rules are loaded into the Dispatcher components of the Teradata Database. When a Teradata client application issues a request to the Teradata Database, the request is examined and checked by TDWM functions in the Dispatcher before being forwarded to the AMPs to execute the request against the user database.

There are three sets of workload management rules that are available. Any of theses three sets can enabled or disabled.

System-wide query management filters
System-wide query management throttles
Workload Definitions

Types of Query Management Rules

Object Access filters	Access to and from specific Teradata Database objects and object combinations by some or all users.
Query Resource filters	Which Teradata Database requirements are necessary to execute certain queries; such as limiting row count, processing time, or types of joins?

Object Throttles	How many sessions and/or queries can be running for specified Teradata Database objects?
Load Utilities	How many load utilities can be running on the Teradata Database either individually or collectively? Load utilities include FastLoad, MultiLoad, and FastExport.

TDWM Example

Left pane is Rules Directory Information Tree (DIT).

From here, you can create, delete, enable, or disable rules.

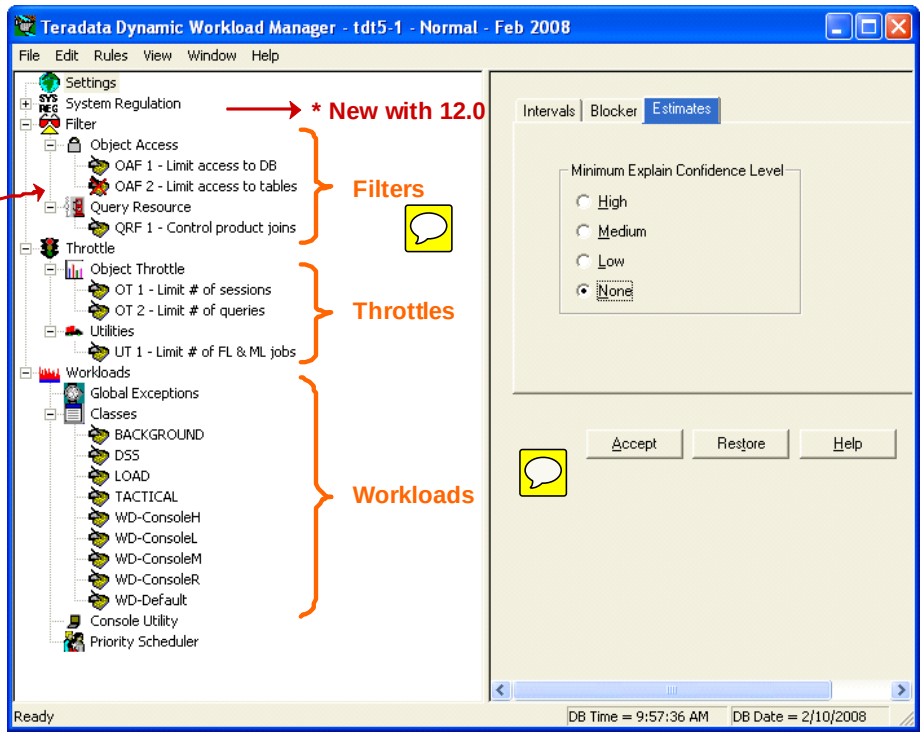
Note the disabled rule.

The Settings option lets you define overall parameters for TDWM.

Types of rules include:

- Filters
- Throttles
- Workloads

* System Regulation – new TD 12.0.

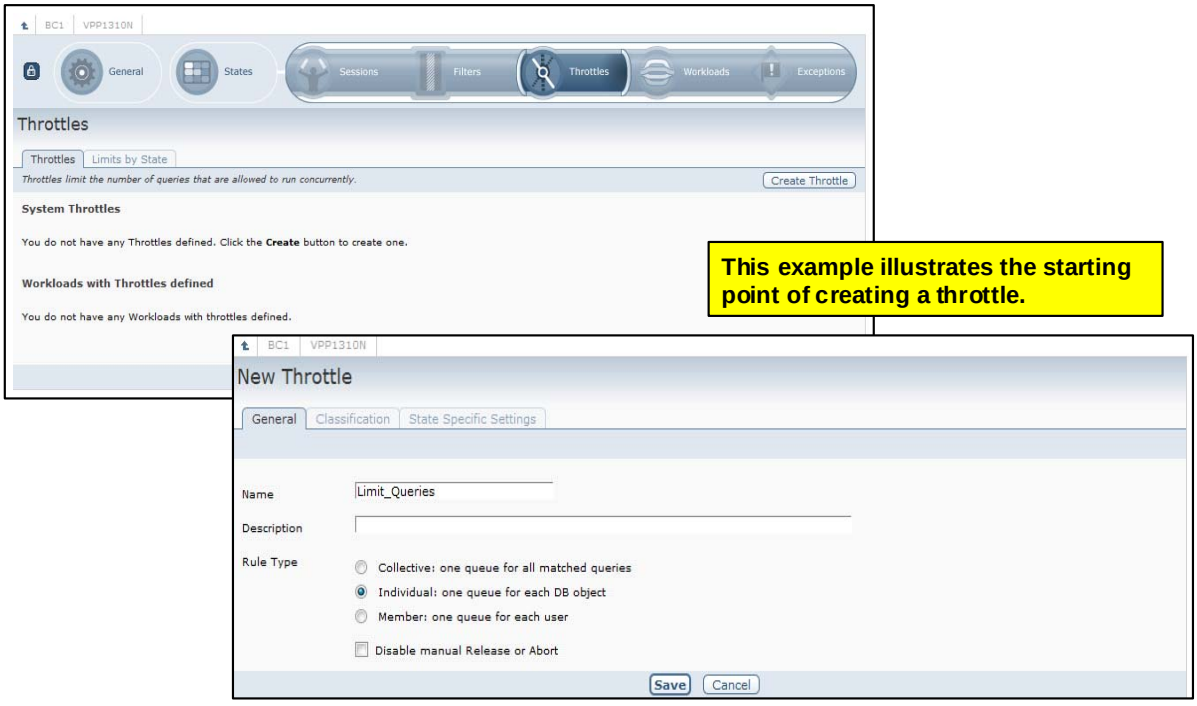


Workload Designer Example

Viewpoint's Workload Designer is required with Teradata 13.10 (and later) to manage filters, throttles, and workload definitions. Workload Designer is the replacement for Teradata Dynamic Workload Manager (TDWM).

Workload Designer Example

Viewpoint's Workload Designer is the replacement for TDWM.



Filters and Throttles for Query Management

The facing page identifies the 4 types of rules (2 filters and 2 throttles). The following is an overview of how logon and query requests are processed.

1. Request is checked for any Context objects that are currently bypassed. If present, the request is immediately executed. For logon requests, go to step 5.
2. Teradata Optimizer step plan for each statement in the query request is traversed to determine the following:
   - Type of statement and type of step
   - Objects in the request
   - Estimated answer set size (number of rows) to be returned
   - Estimated number of rows involved in each step
   - Estimated total processing time required to complete execution
   - Table join required (product or unconstrained product)
   - Full table (all-rows) scan required
   - Types of steps which are all-AMP
   - Confidence level for each step
3. Step costs (that is, the row count and processing time estimates) are used only if the confidence level of the estimates is greater than or equal to the minimum confidence level you have configured.
4. Estimated resource usage values are compared with any global Query Resource rules.
5. Referenced Context and Query objects are checked against any applicable Filter rules. If any object is currently restricted, the step values for the request are compared to any Query Resource rules.
6. Referenced Context and Query objects are checked against any applicable Object Throttle rules. If any object is currently throttled, a supplementary indicator is returned so that the request is forwarded to the Query Manager task for throttle limit checking.
7. Object limits for a query are passed to the Workload Query Manager. The Workload Query Manager determines whether a logon is processed immediately or rejected, and whether a query request is processed immediately, delayed, or rejected.

   Note: Delayed objects are held until throttle limits allow them to run at which time an OK status is returned to the Teradata Database Dispatcher.

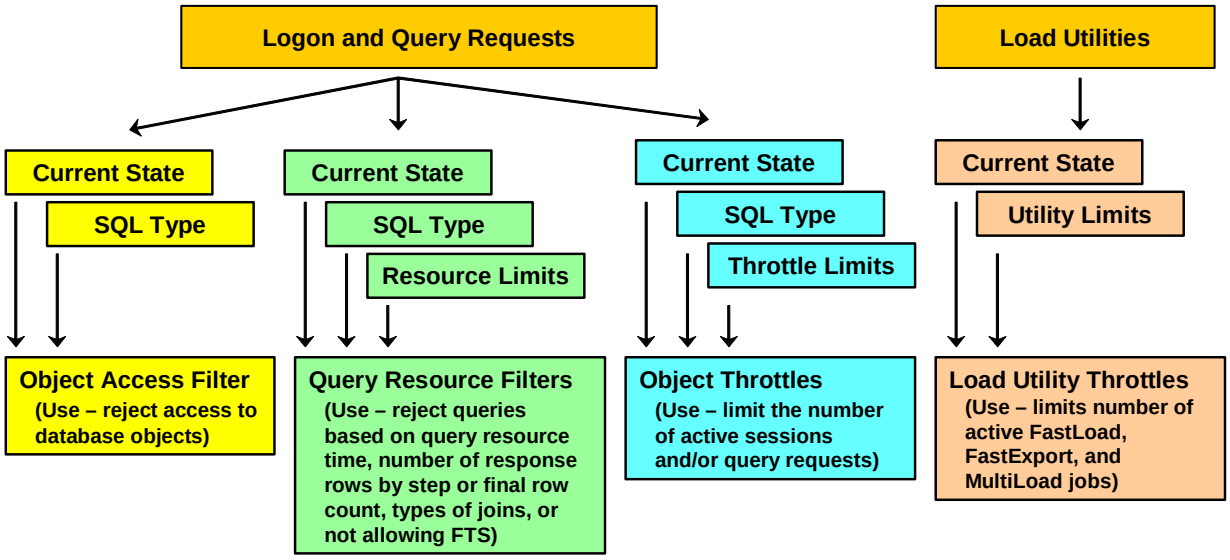
8. OK or reject status is returned to the Teradata Database Dispatcher.

   Note: TDWM examines all logon and query requests in a SQL partition before they are sent to the Teradata Database for execution.

9. Teradata Database Dispatcher takes one of the following actions depending on the applicable rules:
   - Lets the request proceed to the AMPs.
   - Rejects the request or logon.

   Note: Rejected logons and queries are stored in an exception cache. This cache is flushed based on the logging interval you define. Entries for rejected logons and queries are logged in the TDWM.EXCEPTIONLOG table so you can analyze them later.

Filters and Throttles for Query Management



Notes:

- Users and or objects are linked to filters/throttles (e.g., rules) via the Database Browser.
- DBC and Bypass Users – set of users for which filters and throttles do not apply.
- Filters and throttles can be created as Global Rules.

Object Access and Query Resource Filters

Object Access filters are used to reject queries that attempt to access to all objects associated with the filter during the time period specified.

When you define an Object Access filter, you can specify that only combinations of issuing and query objects are restricted. This lets you selectively limit access to the Teradata Database, tables, macros, and so on. For example, you could create a filter that never allows specific users access to specific database tables.

Defining Query Resource filters lets you reject queries based on database resource usage for any issuing object, query object, or object combinations associated with this type of filter. You define how resource usage is limited, as well as the dates and times the resource usage limits apply.

You can configure how Query Resource filters are enforced by ignoring EXPLAIN estimates below a specified level of confidence. For example, if the row count estimate on a query is generated with "no confidence" and the minimum explain confidence level is set to low confidence, then the row count estimate is not used.

Example of Object Access filters include:

On Saturday and Sunday, user A cannot log on to the Teradata Database.
On Weekdays between 8:00 am and 5:00 pm, table B cannot be accessed.

Examples of Query Resource filters include:

On Fridays between 9:00 am and 2:00 pm, table B cannot be involved in a Product Join AND that returns more than 1 million rows.
On Tuesdays between 12:30 pm and 4:00 pm, queries estimated to take longer than 30 minutes cannot run.

SQL Types

For Object Access filters, Query Resource filters, and Object throttles, you can specify the types of SQL requests to which the rule applies. For example, you can specify ALL, DDL, DML, or SELECT.

Global

Global filters apply to all objects, and as a result to all logon and/or query requests during the specified time period. If a filter applies to all objects, you can specify it as a global rule. Because a global rule automatically applies to all Teradata Database objects, you do not need to associate individual Teradata Database objects with the rule.

Caution: Defining a global Object Access rule causes **all** of the specified statement type requests to be rejected except those from the DBC user and any bypassed objects.

Object Access and Query Resource Filters

Object Access Filters

Object Access Filters reject any access to database objects that you associate with the restriction.

Example:

If you associate a table (T1) with an access restriction for a group of users, then TDWM rejects any query that contains T1 within the defined operating environment (may be defined time period or a defined state such as LOAD).

Query Resource Filters

Query Resource Filters reject any access to database objects based on resource usage limits, as well as the dates/times or states that the resource usage limits apply.

Example:

On weekdays between 08:00 and 17:00, queries estimated to take longer than 30 minutes are not allowed to execute for users assigned to a specific performance group.

You can configure how Query Resource filters are enforced by ignoring EXPLAIN estimates below a specified level of confidence.

Object and Load Utility Throttles

For Object Throttles, you define additional limits on logon and queries requests. For Load Utility Throttles, you choose the type of load utility to which the rule applies instead of the type of SQL request.

Object Throttles

Defining Object Throttles lets you limit the number of logon sessions and/or queries active on for particular Teradata Database objects. You can define Object Throttles that apply to most types of Teradata Database objects. You cannot associate Object Combinations with Object Throttles. You can associate Macros and Stored Procedures with Object Throttles. However, they are treated like table objects in that we do not know that they are a "Macro" or a "Stored Procedure" per se. TDWM will just know that the name in the rule matches the name on the object list.

You can set up this type of rule to reject or to delay any query that cannot be immediately processed. If more than one Object throttle applies to an object, the one with the lowest limit one is used.

> **Note:** SQL requests evaluated under this category must include an **ALL-AMP** step to be considered against throttle values. Single AMP operations (for example, prime index) are always allowed to run and are not counted against throttle limits on context objects.

Load Utility Throttles

Defining Load Utility throttles lets you control how many load utilities are simultaneously running on a Teradata Database at any given time. Using this throttle type lets you override the MaxLoadTasks value set using the DBS Control Utility. Setting a throttling rate lets you override the value without having to change it using the DBS Control Utility.

You can specify limits for all load utilities as a group, and/or specify limits for each individual load utility. Because Load Utility throttles apply only to the kind and number of load utilities running on the Teradata Database, you cannot associate Teradata Database objects with them.

Context Objects

Context objects relate to the conditions in which a request is issued. Because they relate to who issued the request, they are also called **who** objects. The types of context objects you can associate with rules are Users, Accounts, Performance Groups, and Profiles.

Object and Load Utility Throttles

Object Throttles

Object Throttles limit the number of logon sessions and/or active queries.

Example:

On Weekdays between 8:00 and 17:00, performance group $M cannot have more than 200 simultaneous sessions on the entire Teradata Database.

On Weekends, performance group "$H" cannot have more than 5 simultaneous queries on the entire Teradata Database – delay queries, do not reject.

Load Utility Throttles

Load Utility Throttles allow you to control how many load utilities are simultaneously running on a Teradata Database at any given time.

Example:

On Weekdays between 8:00 and 17:00, the maximum number of simultaneous FastLoad and/or MultiLoad jobs is 3.

Workload Definitions

A workload represents a portion of the queries that are running on a system. A Workload Definition (WD) is a workload grouping and its operating rules to assist in managing queries. The requests that belong to the same workload will share the same resource priority and exception conditions. It consists of:

Classification Criteria: criteria to determine which queries belong to the workload. This criteria defines characteristics which are detectable prior to query execution. This is also known as the *"who"*, *"where"*, and *"what"* criteria of a query. For example, *"who"* may be an account name, *"where"* is the database tables being accessed, and *"what"* may be the type of statement (UPDATE) being executed.

Exception Criteria: criteria to specify "abnormal" behavior for queries in this workload. These criteria are only detectable after a query has begun execution. If the exception criteria are met, the request is subject to the specified exception action which may be to lower the priority or abort the query.

Operating Periods: a description of hours of the day and/or days of the week (or month). Directives may be specified for exception handling and Priority Scheduler settings can be changed for each operating period.

A Workload Definition is mapped to an Allocation Group (AG) of Priority Scheduler.

Why Create Workload Definitions?

The reason to create workload definitions is to allow TASM to manage and monitor the work executing on a system.

There are three basic reasons for grouping requests into a workload definition.

Improved Control – some requests need to obtain higher priority to system resources than others. Resource priority is given on the basis of belonging to a particular workload.

Accounting Granularity – workload definitions allow you to see who is using the system and how much of the various system resources. This is useful information for performance tuning efforts.

Automatic Exception Handling – queries can be checked for exceptions while they are executing, and if an exception occurs, a user-defined action can be triggered.

Workload Definitions

What is a Workload Definition?

- It is a description of rules that represent who, where, and what of a workload. A Workload Definition is assigned to a Priority Scheduler allocation group.

Why Create Workload Definitions?

- Improved Control of Resource Allocation – resource priority is given on the basis of belonging to a particular workload.
  - Classification rules permit queries to run at the correct priority from the start.
- Improved Reporting – workload definitions allow you to see who is using the system and how much of the various system resources.
  - Service level statistics are reported for each workload.
  - Real-time and long-term trends for workloads are available.
- Automatic Exception Handling
  - After a query has started executing, a query that is running in an inappropriate manner can be automatically detected. Actions can be taken based on exception criteria that has been defined for the workload.

Example of Using Workloads

The facing page illustrates an example of creating five workload definitions to handle a mix of queries.

Recommendations Summary When Defining Workload Criteria

Lead with “Who” criteria, and add “Where”, “What” and exception criteria only when necessary.

“Who” criteria is the most exact and has the least overhead.

Keep the total number of workloads small (e.g. 10-20)

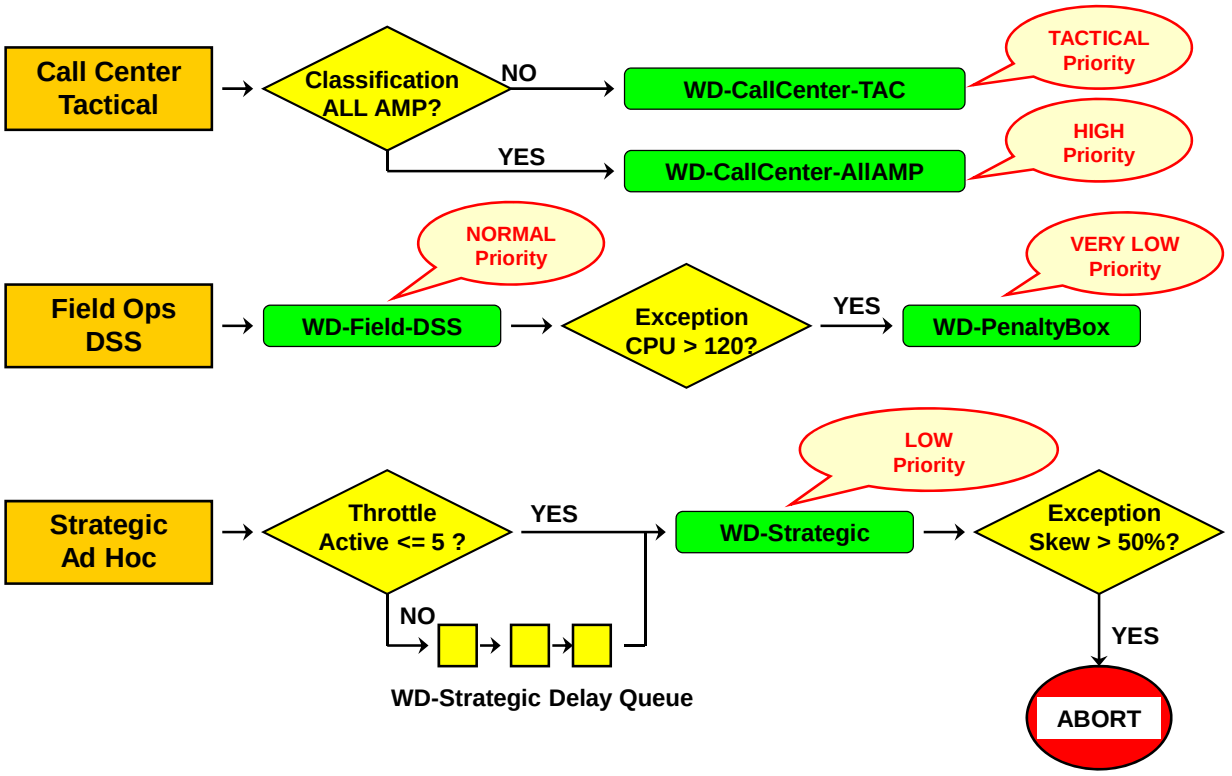
Keep the number of criteria associated with a workload as simple as possible, avoiding long confusing lists of and’d classification and exception criteria as well as include/exclude lists of “Who”/”Where” criteria.

Use order of evaluation to put more specific definitions ahead of less specific definitions, enabling the system to accurately classify a request without having to evaluate the request against the entire list of workloads and their criteria.

Avoid exception monitoring if possible, relying on classification to properly assign a request before execution even begins. This enables concurrency throttles if necessary as well as avoiding even a momentary mis-classification of the request in the unintended priority resource allocation.

If using exception monitoring, use the longest exception interval as possible to keep exception monitoring overhead lower, yet still meet your goals for detecting exceptions in a timely manner. If no exception monitoring is used, set the exception interval to the maximum 3600 seconds.

Example of Using Workloads



Creating Workloads

The facing page provides a list of the major tasks that are used in TASM. A brief description of each of the tasks is also included.

1. **Collect data to analyze** – one source of data is Priority Scheduler settings captured in PD (Priority Definition) sets. The second source is DBQL log data that represents captured queries for a period of time (e.g., three months) that represent the typical workload.

2. **Generate Recommended Workload Definitions** – use/combine the information from the two sources to generate the initial workload definitions. There are primarily two techniques in which workload definitions can be created:

   Create workload definitions from scratch. In doing so, users first collect query log information for the existing workload mix.

   Use Teradata Workload Analyzer to analyze and create workload definitions based on the two sources identified in step #1.

3. **Activate the Workload Management Rule set** – the Teradata Dynamic Workload Manager (TDWM) or Workload Designer administrator is used to optionally modify the workload definitions and activate them on the system.

Steps 4 – 6 are a reiterative process.

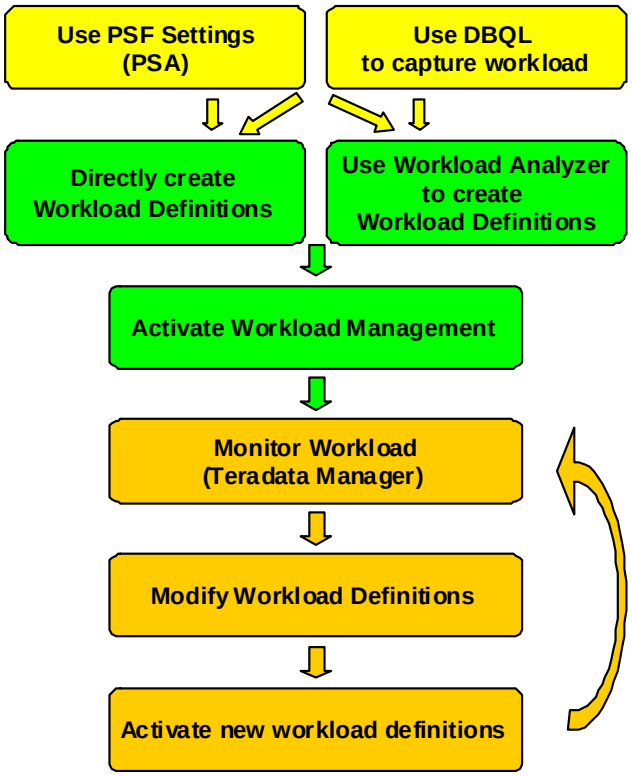
4. **Monitor the Workload** – Teradata Manager has new features which allow the administrator to monitor workloads. These enhancements are part of the Dashboard and Trend Analysis displays.

5. **Refine Workload Definitions** - based on how well the workload definitions are working, they may need to be adjusted.

6. **Activate "new" Workload Definitions** – use TDWM to activate the adjusted workload definitions.

Creating Workloads

1. Collect data to analyze.
2. Generate Workload Definitions and optionally Service Level Goals.
3. Modify Workload Definitions and activate workload rule set.
4. Monitor with Teradata Manager Dashboard and Trend Analysis.
5. Refine Workload Definitions.
6. Activate new Workload Management Rule Set and repeat process by monitoring how well the rules work.



WD – Classification Criteria

After specifying a new workload definition name and attributes and clicking on the NEXT button, it is necessary to define classification criteria for the workload.

The basic classification criteria describes the *"who"*, *"where"*, and the *"what"* of a query. This information effectively determines which queries will run in a workload.

You can specify up to six different criteria for a workload. A query is classified into a Workload Definition (WD) if it satisfies all of the Classification criteria. Normally, you will only need to specify one or two criteria for a workload definition.

The *"who"* criteria define who is executing the query. Examples include:

Account – the user's unexpanded account string (e.g., $M1$LOAD&S&D&H)
User – the Teradata username (e.g., NomarJoe, SmithRobert, DBC)
Client ID – the logon name on the network client (e.g., JN450824)
Client Address – the IP address of the network client (e.g., 141.206.28.51)
Profile – the user's Teradata profile name (e.g., Buyer)
Application – the application name on the network client (e.g., QUERYMAN)

Avoid long include/exclude lists associated with "Who" and "Where" criteria. Consider the use of accounts (that combine many users into one logical group) or profiles to minimize long "Who" include/exclude lists.

The *"where"* criteria defines which database objects are referenced by the query.

Data Objects – choices include databases, tables, views, macros, and stored procedures. Note that UDFs are not supported.

The *"what"* criteria for the query is based on optimizer estimates. Options include:

AMP Limits – is this an all-AMP request or not? Selecting this checkbox causes the workload to accept only queries that are *not* all-AMP queries.
Load Utility Type – FASTLOAD, MULTILOAD, FASTEXPORT (or all three) When selecting additional criteria, be aware that you cannot combine a load utility type with anything other than a "who" criteria.
Statement Type – the type of statement being submitted (e.g., SELECT, DDL, DML)
Row Count – minimum and/or maximum rows at each step for spool files and result set
Final Row Count – minimum and/or maximum rows for result set only
CPU Time – minimum and/or maximum estimated processing time. You can specify CPU time in hundredths of a second (using the format HHH:MM:SS.dd).

"Who" criteria has lower overhead than "where" and "what" because "who" is determined once per session logon, whereas "where" and "what" are determined once per query.

WD – Classification Criteria

- **Workload definitions have classification criteria that specify the WHO, WHERE, and WHAT.**
  - **WHO – determines who is executing the query**
    - **For example: Account string, Username, Profile, Application name, etc.**
  - **WHERE – defines which database objects that are referenced by the query**
    - **For example: databases, tables, views, macros, and stored procedures , etc.**
  - **WHAT – is based on optimizer estimates**
    - **For example: Number of AMPs used, step row count, final row count, CPU time, etc.**
- **A query will run in a specific workload based on the classification criteria.**
  - **With previous Teradata releases, queries were only classified by account string.**
- **Each of the criteria are ANDed together.**
  - **Normally, you only need to have 1 or 2 classification criteria (maximum of 6).**
  - **A query is classified into a Workload definition (WD) if it satisfies all of the Classification criteria.**

Specify Exception Criteria

After identifying the types of requests that make up the workload, you begin defining the behavior you want for those requests. Use Exception Criteria to define performance-related thresholds that trigger special handling such as aborting the requests, or continuing the requests but modifying the workload, issuing an alert, or running an external program.

Teradata checks for exception conditions at the following times.

Synchronously – at the end of each AMP step

Asynchronously – at the configurable time interval (1-3600 seconds); this value is set within TDWM using the left pane selection: *Settings → Intervals → Exception Interval*

Exception Actions specify what to do when an Exception condition is detected.

No exception monitoring – exception handling is effectively turned off and exceptions are NOT logged.

Abort – query is aborted.

Change Workload; move the query into a different workload.

Raise Alert; no change to query; send a Teradata Manager Alert

Run Program; no change to query; have Teradata Manager execute a program.

Notes:

Any exception taken on a query is automatically logged in the DBC.TDWMExceptionLog.

Skew is NOT calculated synchronously at the end of query steps. Asynchronous exception checking is the sole method used to detect skew.

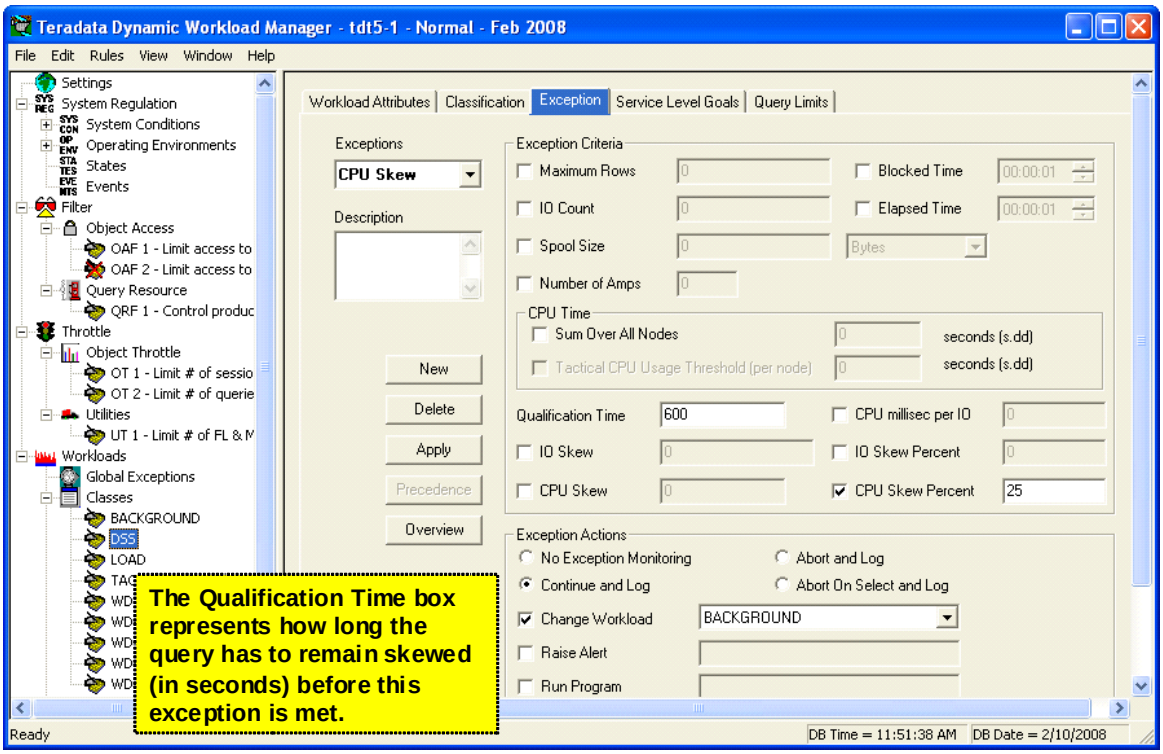
CPU Time Note

If the workload has an Enforcement Priority of tactical, initially the **Tactical CPU Usage Threshold (per node)** check box will be grayed out (not available). The **Tactical CPU Usage Threshold (per node)** check box will be enabled as soon as a positive value is specified for **Sum Over All Nodes**. This causes the Teradata Database to use PSF query milestones to move the workload to a different Allocation Group as soon as the limit is reached. Normal exception processing happens less frequently. When exception processing sees that either CPU time limit has been reached, it will perform normal exception processing.

If the workload is tactical and a positive value for **Tactical CPU Usage Threshold (per node)** is specified, then one of the Exception Actions must be **Change Workload**.

Before saving the Rule Set, TDWM checks to see that all Tactical WD's with a positive value for CPU Usage Threshold (per node) have another WD specified. The user needs to define another WD in the same Resource Partition to satisfy this condition.

Specify Exception Criteria



Example – Exception Handling

The facing page contains an example of a screen that is accessed in Teradata Manager.

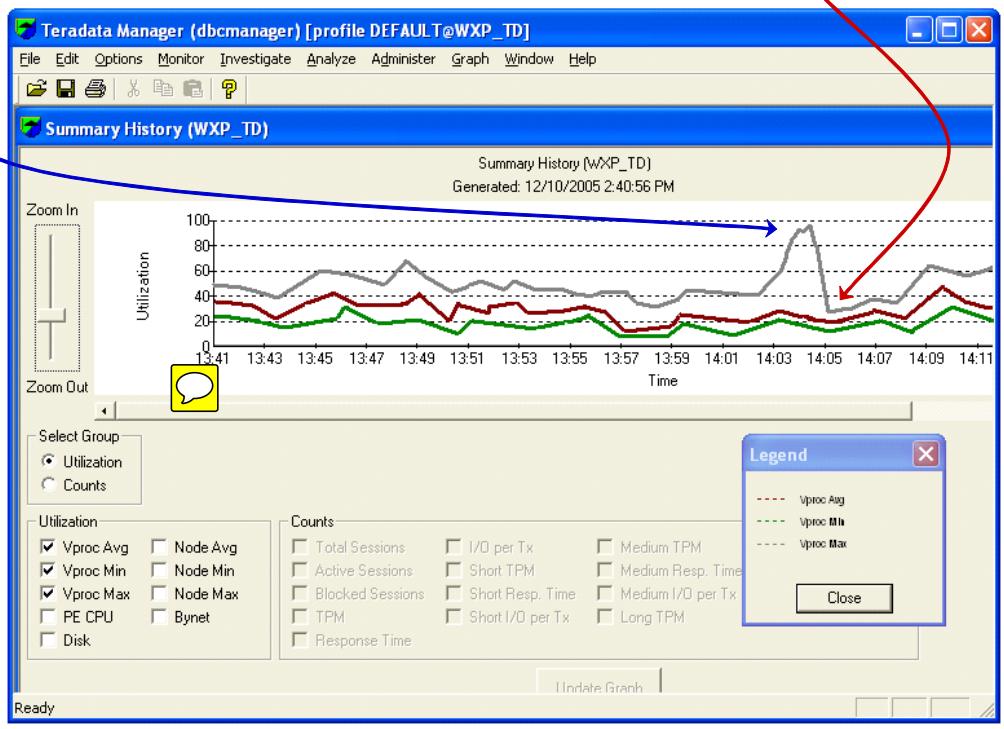
Teradata Manager → Dashboard → Virtual Utilization Summary

Several queries were submitted in the Strategic workload and these all caused AMP skewing. TASM automatically aborted these queries because of the exception criteria.

Example – Exception Handling

"Bad" Query causes skewing.

Query is automatically aborted by TASM.



Example – Exception Handling (cont.)

A user will receive an error message if a query is aborted because an exception criteria was exceeded.

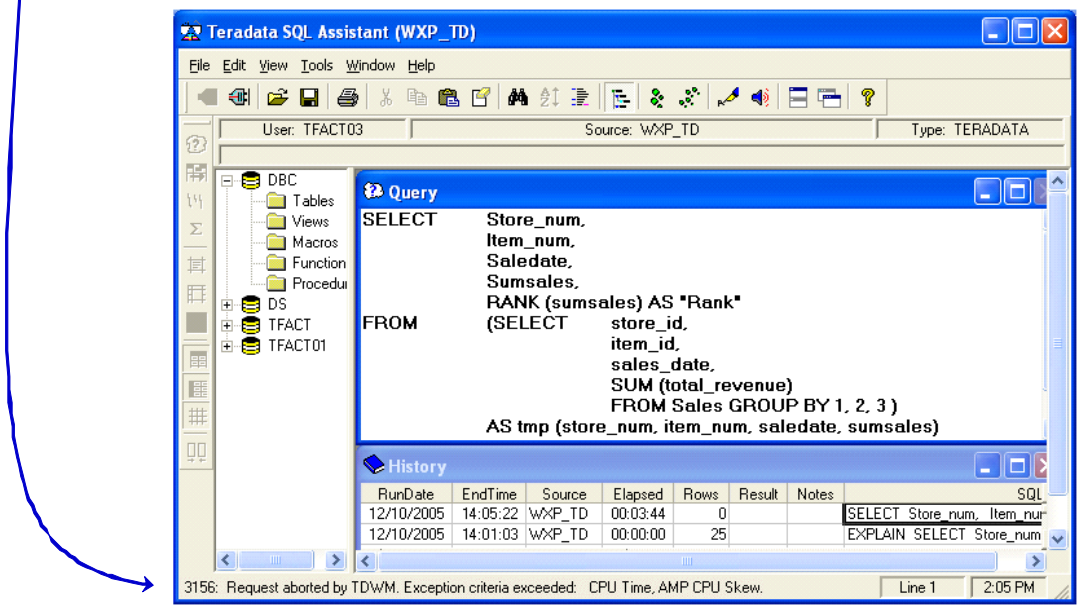
Any exception taken on a query is automatically logged in the DBC.TDWMExceptionLog.

Examples of error codes that are logged in the TDWM Exception Log are:

3149	Query request was rejected because of an Object Access Filter rule.
3150	Query request was rejected due to a Query Resource Filter rule.
3151	Query request was rejected because of an Object Throttle rule.
3152	Logon request was rejected due to an Object Access Filter rule.
3153	Logon request was rejected because of an Object Throttle rule.
3156	Request aborted by TDWM. Exception Criteria exceeded.
3158	Informational message logged by TDWM. Exception Criteria exceeded.
3162	TWM Limit for this utility type was exceeded (not returned); for load utilities.
3163	TWM Limit for all utilities has been exceeded (not returned); for load utilities.

Example – Exception Handling (cont.)

End user gets an error message describing the reason why the query was aborted.



Exception is logged in TASM Exception Log: DBC.TDWMExceptionLog

Teradata Workload Analyzer

This application provides the following capabilities:

Migrate existing PSF settings into workload definitions.
Establish workload definitions from query history or directly
Can be used “iteratively” to analyze and understand how well the existing workload definitions are working, and modify those definitions if necessary.

This tool combines data from existing Priority Scheduler settings (via Priority Definition or PD sets) and workloads (via Teradata DBQL – Database Query Log) to determine workload definitions.

This application can also apply best practice standards to workload definitions such as assistance in SLG definition and priority scheduler setting recommendations.

In addition, Workload Analyzer supports the conversion of existing Priority Scheduler Definitions (PD Sets) into new workloads. A PD set is the collection of data, including the resource partition, allocation group, period type, and other definitions that control how the Priority Scheduler manages and schedules session execution.

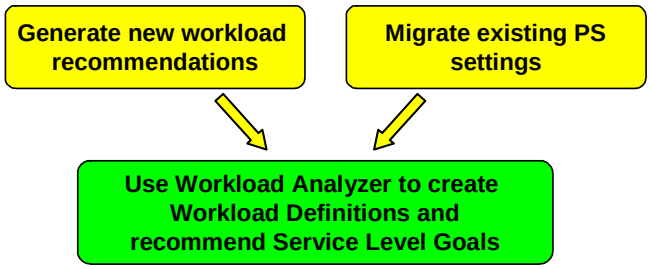
Teradata Workload Analyzer

Capabilities of Teradata Workload Analyzer include:

- Identifies classes of queries and recommends workload definitions and operating rules
- Recommends workload to allocation group mappings and Priority Scheduler weights
- Provides recommendations for appropriate workload Service Level Goals (SLG)
- Provides the ability to migrate existing Priority Scheduler Definitions (PDSets) into new workloads

Workload Analyzer provides 2 paths to creating a Workload Rule set.

1. Generate new workload recommendations using statistics from DBQL data
2. Migrate current Priority Scheduler settings



Summary

The facing page summarizes some important concepts regarding this module.

Summary

- Workload Management is accomplished by using several tools that assist in defining the rules that control the allocation of resources to workloads running on a system.
  - These rules include filters, throttles, and workload definitions.
  - Workload definitions are rules to control the allocation of resources to workloads.
- The benefit of TASM is to automate the allocation of resources to workloads.
- The key product that is used to create and manage these rules is either
  - Teradata Dynamic Workload Manager (TDWM) or
  - Viewpoint Workload Designer
- Other tools that facilitate in workload management include:
  - Teradata Workload Analyzer – helps create workload definitions by analyzing existing PSF settings and DBQL information.
  - Teradata Manager or Viewpoint– contains Workload Monitor and Trend Analysis capabilities.
  - Teradata Query Scheduler – facility to schedule query requests for Teradata

Module 51: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 51: Review Questions

1. What type of TASM filter or throttle rule is needed for the following restrictions?

   Limit the number of concurrent sessions _____
   Reject queries based on max processing time _____
   Reject queries accessing a specific DB _____
   Limit the number of FastLoad jobs _____
   Delay more the 20 queries for a specific account _____

2. What is the purpose of the default workload definition name "WD-Default"?
   a. Default workload for any queries with an enforcement policy of normal.
   b. Default workload for any queries that are not associated with a workload.
   c. Default workload for any queries assigned to Default resource partition.
   d. Default workload for any queries assigned to Standard resource partition.

3. Which query attribute is not used by the Parsing Engine software to classify a query into a Workload Definition (WD)?

   a. User name
   b. Account
   c. User Role 
   d. User Profile
   e. Optimizer estimates

Module 51: Review Questions (cont.)

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 51: Review Questions (cont.)

4. Which Teradata application can be used to initially define workload definitions?
   a. Teradata Manager
   b. Workload Analyzer
   c. Dynamic Workload Wizard
   d. Priority Scheduler Administrator

5. Place the following control options in the proper sequence from 1 to 4 as they are acted upon by Teradata software.
   ___ a. Object throttles
   ___ b. Exception criteria
   ___ c. Workload throttles
   ___ d. Object filters

Notes

Module 52

Teradata Viewpoint

After completing this module, you will be able to:

- **List 3 portlets that are available with Viewpoint.**
- **Understand the purpose of various Viewpoint portlets**
- **Use Viewpoint to monitor sessions.**

Notes

Table of Contents

What is Teradata Viewpoint? 52-4
Viewpoint Portal and Portlets 52-6
Logging onto Viewpoint 52-8
Example of Initial Session and Portlets 52-10
Viewpoint Portlet Controls 52-12
Viewpoint Rewind 52-14
Query Monitor 52-16
Viewpoint Query Monitor Detail View 52-18
My Queries 52-20
Viewpoint – Remote Console 52-22
Viewpoint Alert Viewer 52-24
Viewpoint SQL Scratchpad 52-26
Viewpoint SQL Scratchpad Object Browser 52-28
Viewpoint SQL Scratchpad Saved/History 52-30
Summary 52-32
Module 52: Review Questions 52-34

What is Teradata Viewpoint?

Teradata Viewpoint is a web portal application framework with a primary focus on Teradata Systems Management functionality that is integrated into the Teradata platform.

Teradata Viewpoint provides systems management via a web browser which is extensible to Teradata end users and management, allowing them to understand the state of the system and make intelligent decisions about their work day.

Teradata Viewpoint enables database and system administrators and business users to monitor and manage Teradata Database systems from anywhere using a standard web browser.

Teradata Viewpoint allows users to view system information, such as query progress, performance data, and system saturation and health through preconfigured portlets displayed from within the Teradata Viewpoint portal. Portlets can also be customized to suit individual user needs. User access to portlets is managed on a per-role basis.

Database administrators can use Teradata Viewpoint to determine system status, trends, and individual query status. By observing trends in system usage, system administrators are better able to plan project implementations, batch jobs, and maintenance to avoid peak periods of use. Business users can use Teradata Viewpoint to quickly access the status of reports and queries and drill down into details.

Teradata Viewpoint - Benefits

Viewpoint provides several benefits to the customer, including:

- A complete solution provided by Teradata
- A single operational view for all Teradata systems
- A leading edge interface
- Highly customizable and personalized
- Ease of accessibility; lower TCO
- Self Service extensibility to all Teradata users
- Add your own content with the PDK (Portlet Development Kit)

Teradata Viewpoint

Viewpoint is the cornerstone of Teradata Systems Management.

- Provides systems management via a web browser.
- Provides a single operational view (SOV) for multiple systems.
- Highly customizable and can be personalized.
- Teradata Management Portlets are the replacement for Teradata Manager and PMON.

Viewpoint Architecture

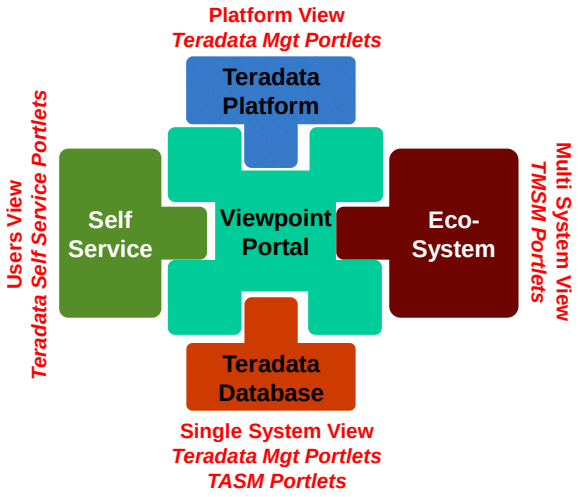
- Viewpoint Server – external server or appliance that executes the Viewpoint application
- Viewpoint Portal is an AJAX Web 2.0 application
- Data Collection (DCS) is performed by the Viewpoint server.

Browser Support Examples

- Internet Explorer 7 or later
- Firefox 3.6 or 4.0
- Chrome 10
- Safari 5

TASM Portlets

- Workload Designer support TD 12.0 to TD 14.0
- Workload Monitor and Health V2R6.2 to TD 14.0



Viewpoint Portal and Portlets

Portal Basics

To help you work efficiently, Teradata Viewpoint uses a page metaphor as the framework for displaying and updating portlets. Each portal page is a virtual work space where you decide which **portlets** to display and how to arrange them on the page. Examples of ways to organize your work include defining a page for each system being monitored, or for each type of query or user. As you work, Teradata Viewpoint continually updates the information displayed on the page that currently fills the Teradata Viewpoint portal. This page is called the *active page*.

Portlet Basics

Teradata Viewpoint system management tools are called *portlets*. Select the portlets that you want to display and monitor from submenus, or categories, under Add Content. You can also search for a specific portlet name using the filter feature. Generally, every instance of a portlet:

- Has a frame that appears when the cursor moves over any part of the portlet on the page and disappears when the cursor moves off the portlet.
- Displays the portlet name or the current settings in the upper frame, depending on the current activity.
- Has a width requirement to ensure proper display of its graphical information such as charts, spark lines, or graphs.
- Remains at a fixed size even when the browser window is re-sized. You can use the browser scroll bars to view the entire portlet view.
- Can be repositioned on the portal page.

With the exception of the Calendar portlet, multiple instances of a portlet can be added to a portal page. Each portlet instance has its own settings and controls in addition to the features shared by all portlets. The System Administrator assigns portlet privileges for each Teradata Viewpoint user.

Viewpoint Portal and Portlets

- A Viewpoint portal represents a Web location that can be accessed or logged onto.
- Each *portal page* is a virtual work space where you decide which *portlets* to display and how to arrange them on the page.
  - Viewpoint system management tools are called *portlets*. Viewpoint's Add Content allows the following content (portlets) to be added.
- Viewpoint continually updates the information displayed on the portal page. This page is called the *active* page.

Monitoring
- Alert Viewer
- Canary Response Times
- Lock Viewer
- Metrics Analysis
- My Queries
- Node Resources
- Productivity
- Query Monitor
- Query Spotlight
- Space Usage
- System Health
- TVS Monitor
- Today's Statistics
- Viewpoint Monitoring

TASM
- Workload Designer
- Workload Health
- Workload Monitor

Tools
- Calendar
- External Content
- Remote Console
- SQL Scratchpad

Trend Reporting
- Capacity Heatmap
- Metrics Graph

Logging onto Viewpoint

Logging on to the Teradata Viewpoint portal begins your session so you can begin working with the Teradata Viewpoint portal.

1. Open a browser.
2. Enter the address for your Teradata Viewpoint portal.

The Welcome page appears, with the portal version number shown at the bottom.

3. Log on to the Teradata Viewpoint portal.

If your Teradata Viewpoint system is set up to create a user profile automatically, the username and password you enter are authenticated against your company-provided username and password the first time you log on to Teradata Viewpoint. Automatic profile creation is known as *auto-provisioning*.

Logging onto Viewpoint

Logging on to the Teradata Viewpoint portal begins your session so you can begin working with the Teradata Viewpoint portal.

1. **Open a browser.**
2. **Enter the address for your Teradata Viewpoint portal.**
3. **Log on to the Teradata Viewpoint portal.**

Example of Initial Session and Portlets

The facing page illustrates an initial session with 3 portlets that have been started.

You can manage portal pages using the following guidelines:

- Add portal pages at any time during a Teradata Viewpoint session.
- Access any portal page by clicking its tab; only one page can be active at a time.
- Change the name of any tab, including the **Home** page tab; page names can be duplicated.
- Rearrange pages by dragging and dropping into a new location.
- Remove pages, along with any portlets contained on the page, with a single mouse-click.
- One page (tab) must remain, as well as the **Add Page** tab.

Adding a Portal Page

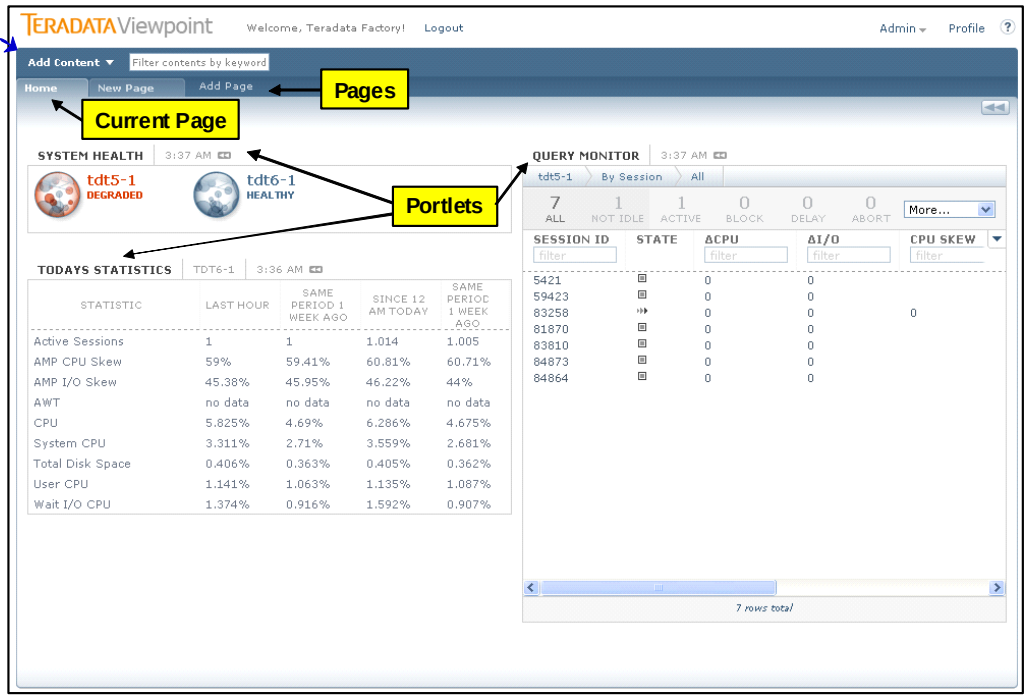
Organize your system management tools by adding pages to the Teradata Viewpoint portal. Multiple pages can be added or removed per session. The newest page is always the active page unless you click on another tab.

- In the Teradata Viewpoint portal, click Add Page.
- A New Page tab appears to the left of Add Page and becomes the active page.

Example of Initial Session and Portlets

New portlets are added via the Add Content.

This initial logon shows 3 portlets that are executing.



Viewpoint Portlet Controls

The following controls appear within each portlet frame if the control is available:

- **Rewind** – indicates that the portlet can be set to display data from a previous point in time.
- **Preferences** – accesses portlet preferences and settings. Preferences are used to specify what information is displayed, time intervals for reporting, and other features that help you customize the portlet functions.
- **Share Portlet** – captures a customized version of a portlet for use by other users. The Teradata Viewpoint Administrator must make the customized portlet available for sharing.
- **Collapse** – toggles the portlet closed. This button appears only when the portlet is open. When collapsed, only the upper and lower portlet frame sections are displayed.
- **Expand** – toggles the portlet open. This button appears only when the portlet is collapsed. When expanded, the portlet returns to its normal size and position on the portal page.
- **Maximize** – toggles the portlet to fill the portal page, covering all other portlets being displayed.
- **Remove** – removes the portlet and all its settings from the active portal page.

Viewpoint Portlet Controls

Preferences

Collapse

Share Portlet

Expand

On-Line Help
(Pop-ups must be allowed)

Maximize

Rewind

Remove

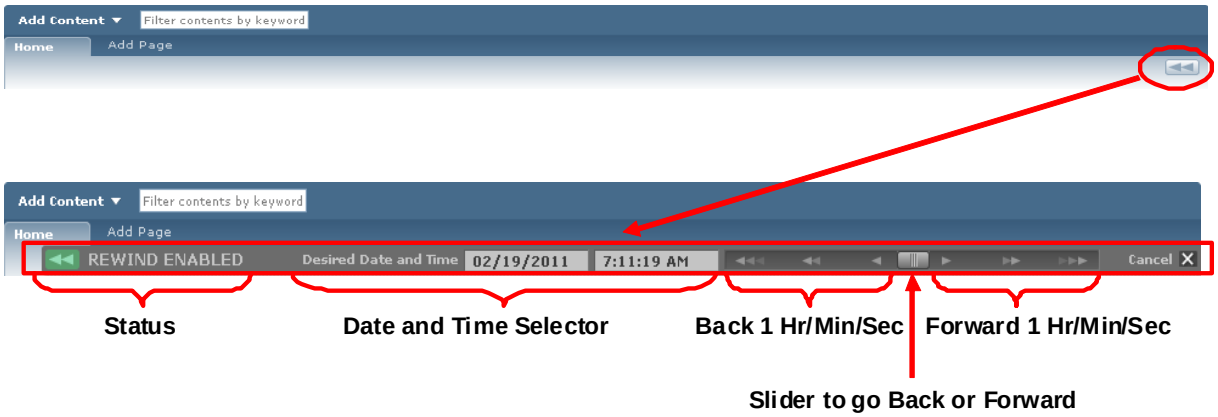
Viewpoint Rewind

The rewind feature allows you to view data that corresponds to dates and times in the past and compare it to data for a different date and time. You can rewind the data for some or all portlets on a portal page to a previous point in time, such as when a job failed. Rewinding portlet data is useful for identifying and resolving issues.

You can rewind data as far back as data is available. The rewind feature is not available for portlets that have portlet-specific methods for reviewing data over time.

Using the rewind toolbar, you can enter a specific date and time as well as scroll through the data in increments of seconds, minutes, hours, or days. All portlets on the page that are participating in rewind activities display data that corresponds to the selected rewind date and time each time a selection is made.

Viewpoint Rewind



Rewind, replay, fast forward Viewpoint portlets to review Teradata operations at past points in time

Query Monitor

The Query Monitor portlet allows you to view information about queries running in a Teradata Database system so you can spot problem queries. You can analyze and decide whether a query is important, useful, and well written. After you have identified a problem query, you can take action to correct the problem by changing the priority or workload, releasing the query, or aborting the query or session. You can take these actions for one query or session, or multiple queries or sessions at a time.

The summary view contains a table with one row allocated to each of the sessions, account strings, users, or utilities running on the database.

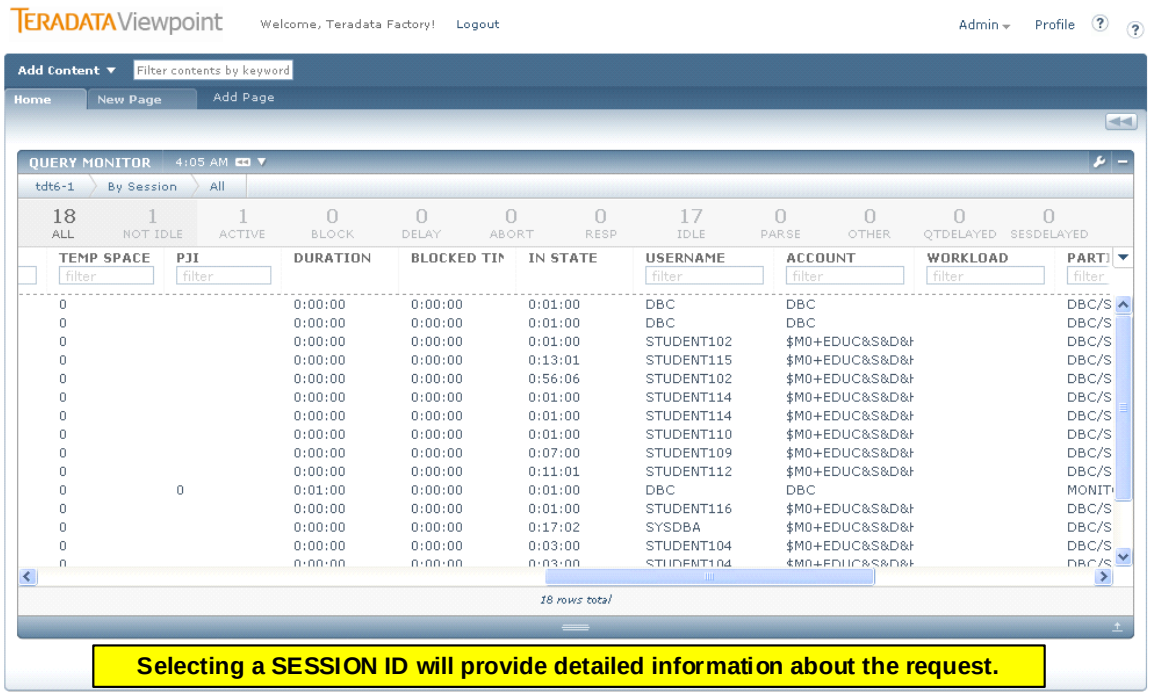
The portlet allows you to filter queries in all of the session views. You can set thresholds for any column and when the threshold is exceeded, the information is highlighted in the sessions table.

Select a row to access session and query information in the details view. Using Query Monitor, you can also determine the types of utilities that are running most frequently on the system and then set utility limits. You can spot utilities that are using a large number of partition connections and, potentially, a high number of resources.

From the PREFERENCES view, you can set the criteria values used to display sessions in the My Criteria view and customize the information displayed in the views. For example, you can set criteria values to display only those sessions currently running on the selected system that exceed the specified criteria. You can troubleshoot Teradata Database system problems to quickly explore details about queries such as the current state of a query or how long a query has been blocked.

Query Monitor

The Query Monitor portlet allows you to view information about queries running in a Teradata Database system.



Selecting a SESSION ID will provide detailed information about the request.

Viewpoint Query Monitor Detail View

The details view displays statistics and information about the selected session. This view can be accessed by clicking on a session row in the summary view.

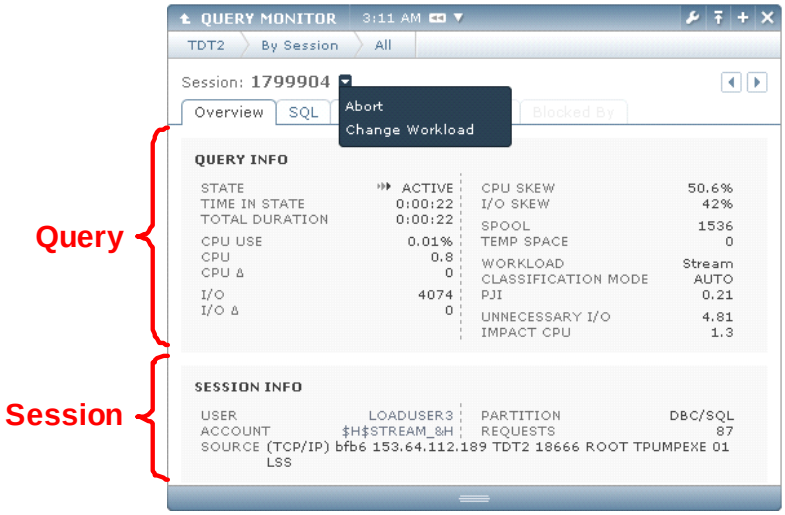
When viewing a request, you can see detailed information from the following tabs:

- **Overview** – key statistics for a session. Any value exceeding the thresholds is highlighted.
- **SQL** – SQL for the selected query.
- **Explain** – Explain steps for the query, including step statistics and explain text.
- **Blocked By** – details about other queries that are blocking this query.
- **Delay –** details about rules delaying this query.
- **Query Band** – displays the query band name and value for the selected query.

Use the **Tools** menu to change the priority or workload, release a query, or abort a query or session for one query or session at a time.

Use the **Next** and **Previous** buttons to move through sessions without returning to the summary view.

Viewpoint Query Monitor Detail View



Depending on Query State, available tabs include:

- **Overview**
- **SQL**
- **Explain**
- **Blocked By**
- **Delay**
- **Query Band**

Depending on the current state of the request, by selecting the drop down menu, the request can be:

- **Aborted**
- **Changed to another Workload**
- **Released**

My Queries

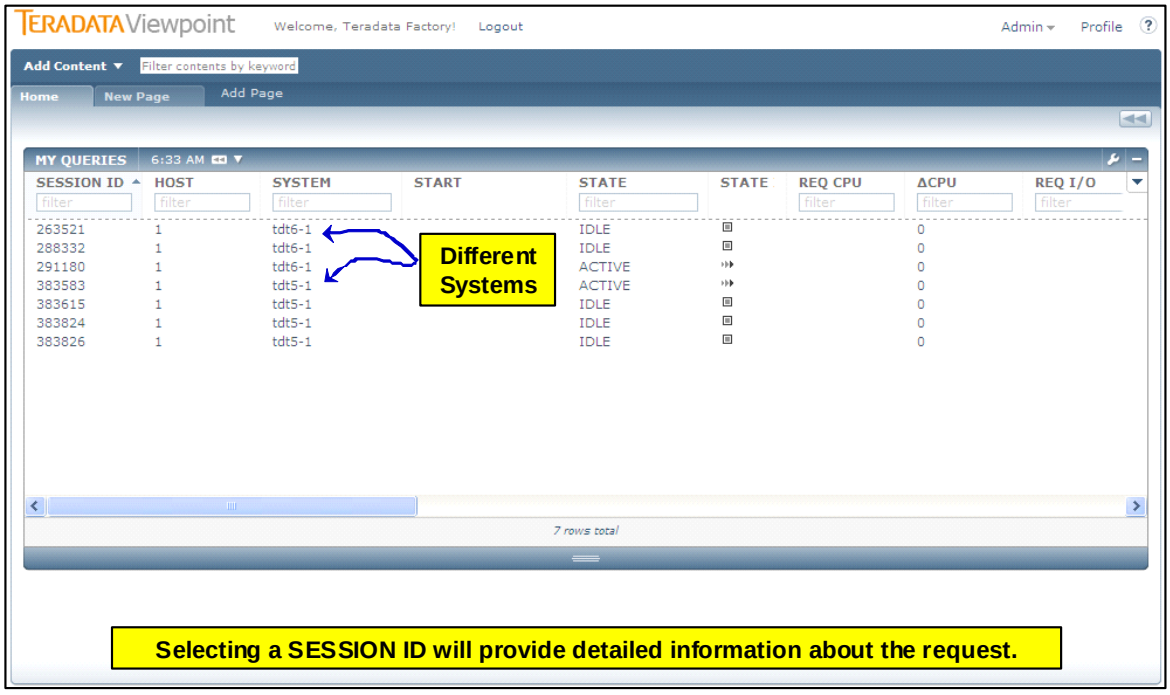
The My Queries portlet allows you to view and manage your queries across multiple Teradata Database systems. You can see if queries are queued or blocked, and you can see their impact on system resources.

Use the My Queries portlet to view information about queries in either the summary view or the details view. The summary view contains a table with one row allocated to each of the sessions logged on under one or more user names. Select a row in the summary view to see additional session and query information in the details view. Use the SQL, Explain, Blocked By, or Query Band tab in the details view to display information for the selected session.

The PREFERENCES view allows you to select one or more Teradata Database systems, and then select one or more users per system to monitor. From this view, you can also select a format for the SQL that appears in the query details view.

My Queries

My Queries allows a user to view their queries across multiple systems.



Viewpoint – Remote Console

The Remote Console portlet allows you to run many of the Teradata Database console utilities remotely from within the Teradata Viewpoint portal.

Using this portlet, you can:

- Select or search for a system.
- Select or search for a utility.
- Enter console utility commands.
- Display responses from the commands.

Teradata field engineers, Teradata Database operators, System Administrators, and System Programmers use Teradata Database utilities to administer, configure, monitor, and diagnose issues with Teradata Database.

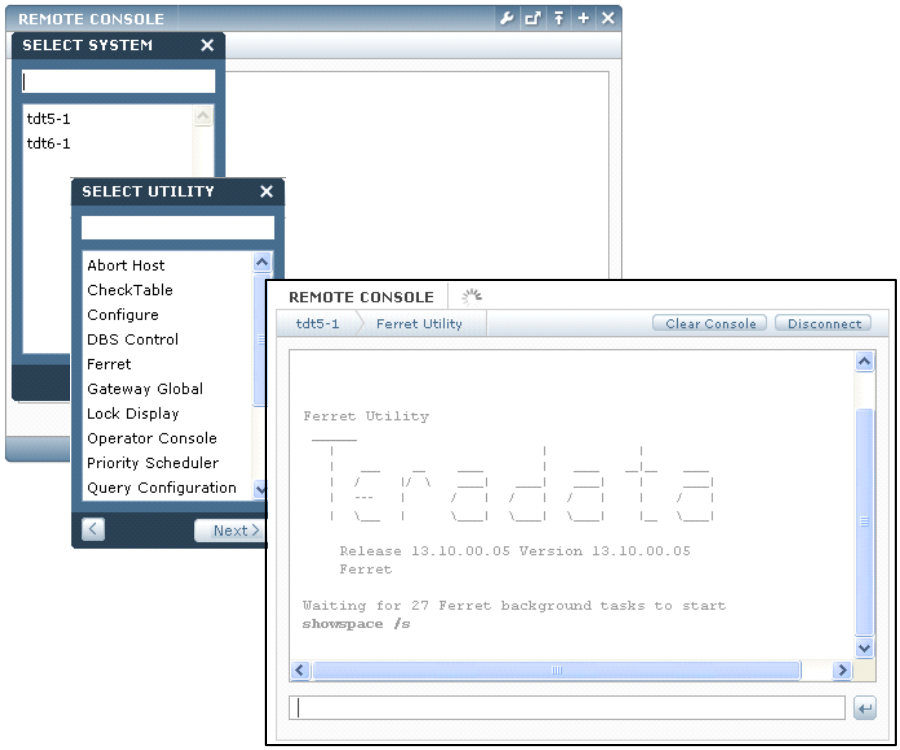
Remote Console activity requires special access rights, BUT does not require UNIX Root authority.

Remote Console

The Remote Console portlet allows you to execute system utilities

Add Content > Tools > Remote Console

- Abort Host
- Check Table
- Configure
- DBS Control
- Ferret
- Gateway Global
- Lock Display
- Operator Console
- Priority Scheduler
- Query Configuration
- Query Session
- Recovery Manager
- Show Locks
- Teradata DWM Dump
- Vproc Manager



Viewpoint Alert Viewer

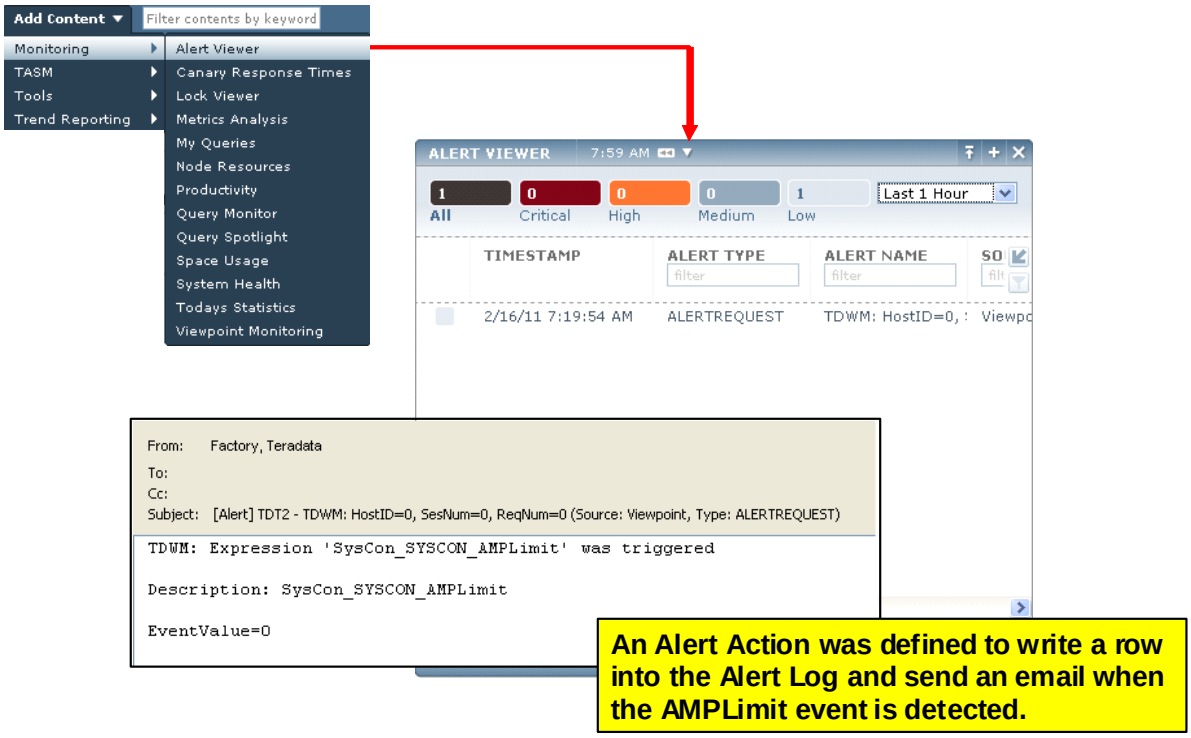
The **ALERT VIEWER** portlet allows users to view alerts defined for the system. The alert information in the summary view is updated every 30 seconds. Every alert has a timestamp, displaying the date and time at which the alert was issued.

You can filter the alerts by for example severity, time period, type, or name. You can also combine the filters to narrow the results further.

The **ALERT DETAILS** view displays detailed information about what triggered the alert, the source of the alert, and any relevant messages.

An alert is an event that the Teradata System Administrator defines as being significant. The Teradata System Administrator assigns alert severity levels to rank alerts, and can also include an explanatory message. The severity levels are: critical, high, medium, or low. The alerts displayed in the **ALERT VIEWER** portlet are specific to your system.

Viewpoint Alert Viewer



An Alert Action was defined to write a row into the Alert Log and send an email when the AMPLimit event is detected.

Viewpoint SQL Scratchpad

The **SQL SCRATCHPAD** portlet allows you to enter queries and retrieve data from a Teradata Database system. Select a system to run your query, enter a single or multi-statement query, and then view the results.

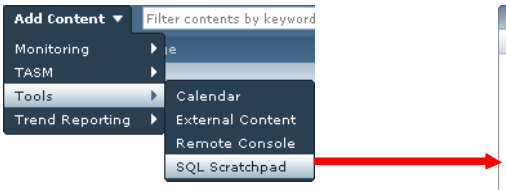
After the Teradata Database system retrieves the data, the RESULTS section expands to display four tabs. Use the **Results** tab to see the results of your query. Use the **SQL** tab to display the SQL statements used to create the query. Use the **Explain** tab to see steps listed in chronological order. The steps are refreshed every 30 seconds so you can see how longer running queries are progressing. If there are delays in processing your query, you can cancel the query. If running a query takes longer than you expected, it might be blocked. Use the **Blocked By** tab to see information about queries that are blocking the currently running query.

Use the **OBJECT BROWSER** to view a list of objects in the database. Also use the **OBJECT BROWSER** to insert an object into a query to reduce the time required to build a query and help reduce errors in object names.

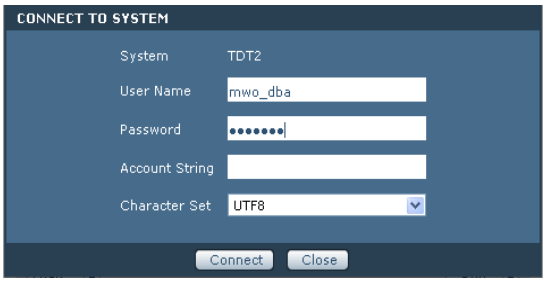
The **SQL SCRATCHPAD** portlet also allows you to save queries or export results to a file where you can sort and analyze the information. You can *pin* query results (pinning allows you to temporarily save the query results). Saving the query allows you to use the query in the current session or in future sessions. Use the **History** tab to access previously run queries from your current session.

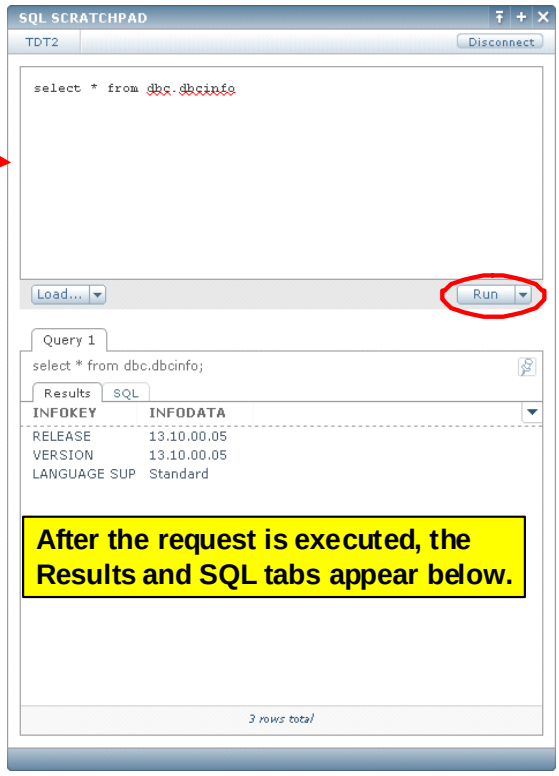
The **SQL SCRATCHPAD** portlet frame displays the system name on which the query is running.

Viewpoint SQL Scratchpad



Selecting the Run button the first time will prompt for the logon information and the query results will be displayed.





After the request is executed, the Results and SQL tabs appear below.

Viewpoint SQL Scratchpad Object Browser

Selecting the down arrow on the load button will load the Object Browser.

The **OBJECT BROWSER** allows you to view a list of objects in a Teradata Database system and insert an object into a query. Use the **OBJECT BROWSER** to reduce the time required to build a query and help reduce errors in object names. Use filtering to search for objects in the **OBJECT BROWSER**.

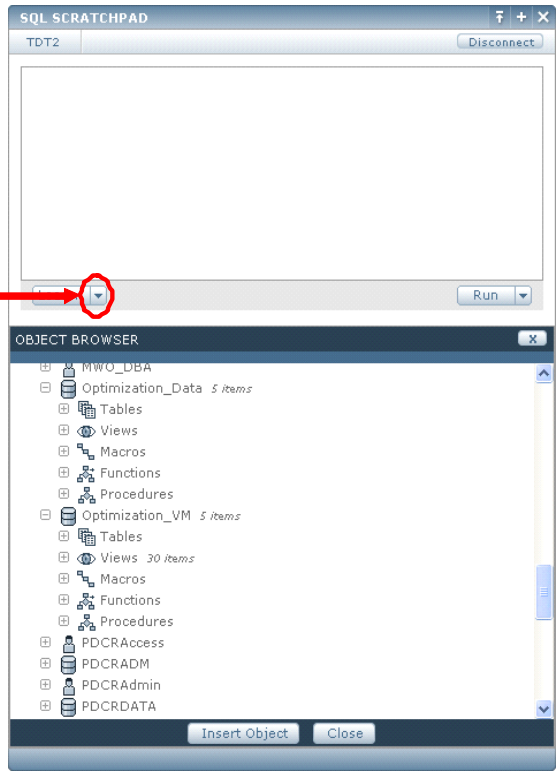
The **OBJECT BROWSER** employs a directory tree that is organized in a hierarchical structure, displaying a list of databases and users on the connected system.

Viewpoint SQL Scratchpad Object Browser

The SQL Scratchpad portlet allows you to enter queries and retrieve data.

Select the Load down arrow to load the Object Browser.

After building the SQL statement, Select the Run button to execute the SQL.



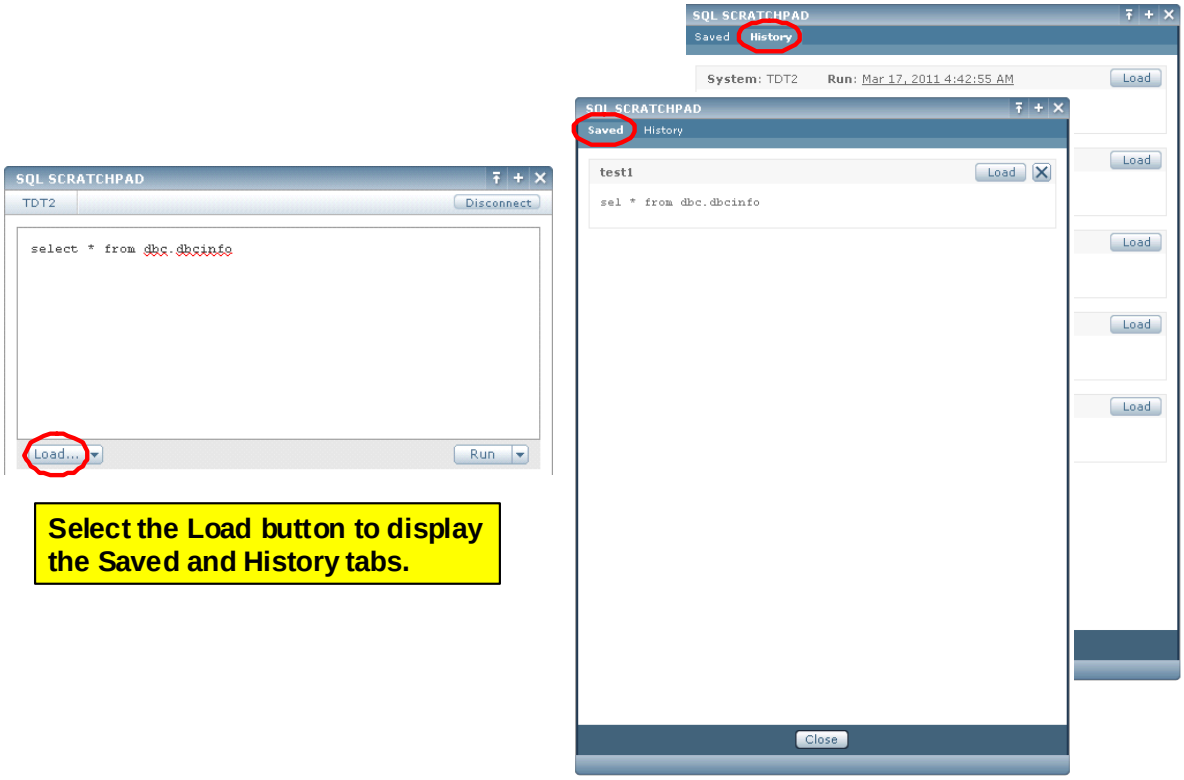
Viewpoint SQL Scratchpad Saved/History

Selecting the down arrow on the load button will load the Object Browser.

The **OBJECT BROWSER** allows you to view a list of objects in a Teradata Database system and insert an object into a query. Use the **OBJECT BROWSER** to reduce the time required to build a query and help reduce errors in object names. Use filtering to search for objects in the **OBJECT BROWSER**.

The **OBJECT BROWSER** employs a directory tree that is organized in a hierarchical structure, displaying a list of databases and users on the connected system.

Viewpoint SQL Scratchpad Saved/History



Select the Load button to display the Saved and History tabs.

Summary

The facing page summarizes some important concepts regarding this module.

Summary

Viewpoint is the cornerstone of Teradata Systems Management.

- Provides systems management via a web browser.
- Viewpoint provides a single operational view (SOV) for multiple systems.
- Highly customizable and can be personalized.

The Teradata Viewpoint Management and Self-Service portlets allow the Viewpoint user access Viewpoint and Teradata resources. Examples include:

- Alert Viewer – allows users to view alerts defined for the system
- Query Manager – allows users to view information about requests
- System Health – allows users to monitor and display the status of a selected Teradata Database system
- Node Resources – allows users to monitor physical and virtual resources
- Remote Console – allows users to run many of the Teradata Database console utilities remotely from within the Teradata Viewpoint portal
- SQL Scratchpad – allows users to enter queries and retrieve data

Module 52: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 52: Review Questions

1. **List three portlets of Viewpoint.**

_____

_____

_____

Notes

Module 53

ResUsage – Performance Monitoring

After completing this module, you will be able to:

- **Identify the purpose of various ResUsage tables.**
- **List two ways in which ResUsage data can be collected.**
- **Name the two phases of gathering resource usage data and briefly describe them.**
- **Describe the purpose of the Teradata System Emulation Tool**

Notes

Table of Contents

Performance Monitoring Tools 53-4
SAR and xperfstate 53-4
Why Collect Performance Data? 53-6
Resource Usage Data 53-8
Data Collection 53-8
Data Logging 53-8
Collection Costs 53-8
Cost 53-8
Filling the ResUsage Tables 53-10
Collection and Log Buffers 53-10
Resource Usage Logging 53-10
Real-time Performance Monitoring 53-10
Specifying ResUsage Tables and Logging Rates 53-12
Specifying Tables using the ctl Utility 53-12
Specifying Tables using Supervisor Commands 53-12
Resource Usage Tables 53-14
Setting Resource Logging from DBW 53-14
Setting Resource Logging from the Control Utility – ctl 53-14
Resource Usage Views 53-16
ResGeneralInfoView 53-16
ResCPUUsageByAMPView 53-16
ResCPUUsageByPEView 53-16
ResShstGroupView 53-16
ResSldvGroupView 53-16
ResSvdskGroupView 53-16
Resource Usage Macros 53-18
Example Output from DBC.ResNode Macro 53-20
PM/API and Viewpoint 53-22
How PM/API Collects Data 53-22
Collecting and Reporting Resource Usage Data 53-22
Collecting and Reporting Session-level Usage Data 53-22
Example 53-22
Teradata System Emulation Tool (Teradata SET) 53-24
Performance Monitoring Summary 53-26
Module 53: Review Questions 53-28

Performance Monitoring Tools

The facing page identifies the performance monitoring tools and the platforms on which they reside.

The Teradata Database provides several facilities you can use to monitor database performance.

- EXPLAIN statement
- Access logging – record the activity of specific users.
- AMPUsage – a view that provide AMP Usage information for each user and account.
- ResUsage – a set of tables, view, and macros used to record and monitor system resource usage.
- Teradata PM/API – Performance Monitor / Application Programming Interface
- Teradata Manager – Performance Monitor (formerly PMON)
- Database Console Utilities – (e.g., Showspace displays space utilization)
- Linux tools (e.g., sar)
- TDP User Transaction Monitor (TDPTMON)

For example, you may have to use several of these tools to monitor query performance. You should always have an EXPLAIN report to understand what the query is doing.

SAR and xperfstate

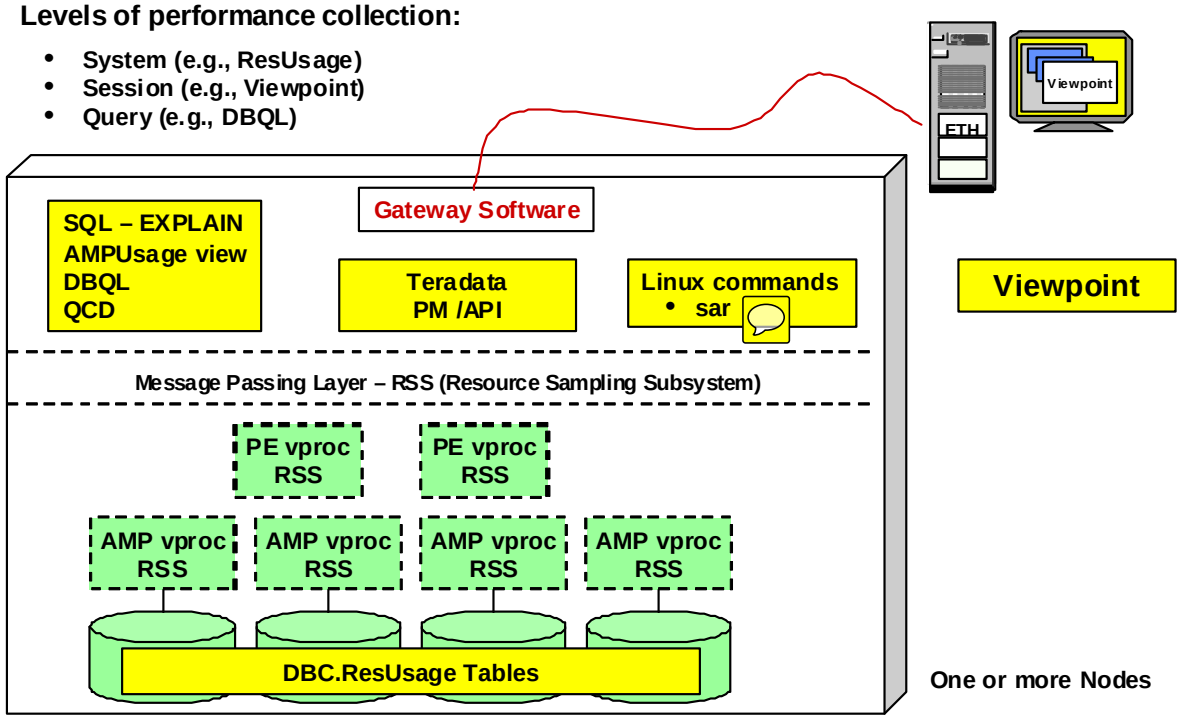
SAR (System Activity Reports) is a UNIX facility to capture performance metrics at the UNIX operating system level.

Xperfstate is a UNIX tool that provides a real-time display of system performance combining some UNIX and Teradata information.

Performance Monitoring Tools

Levels of performance collection:

- System (e.g., ResUsage)
- Session (e.g., Viewpoint)
- Query (e.g., DBQL)



Why Collect Performance Data?

Traditionally, performance data and reports have been the primary diagnostic tool available to analyze performance data on a Teradata system.

The facing page shows some uses for ResUsage reports and data.

When considering system expansion and doing capacity planning, some of the activities to consider include:

- Batch windows – how much time is available to perform batch SQL jobs. A larger system with more AMPs usually means that these jobs can run faster. However, if the network and/or channel are the bottleneck, then maybe additional network and/or channel connections may be necessary.

- Backup windows – how much time is available to perform Archive activities? If a disaster occurred and a RESTORE/RECOVERY had to be performed, how long can the system be unavailable before there is negative impact to the business?

- Maintenance Windows - how much time is available to perform batch load/unload activities? A larger system with more AMPs usually means that the load/unload functions will take less time. However, if the network and/or channel is the bottleneck, then maybe additional network and/or channel connections may be necessary.

- Ad-hoc decision support queries – how much spool is needed? When will these be executed

Why Collect Performance Data?

Performance data may be used to:

- **Measure system benchmarks.**
- **Measure component performance.**
- **Analyze performance degradation and improvement.**
- **Identify potential performance impact.**
- **Identify bottlenecks, parallel inefficiencies and other problems.**
- **Assist on-site job scheduling.**
- **Plan installations.**
- **Capacity planning – resource usage data can help determine if system expansion is necessary.**

Resource Usage Data

ResUsage data gathering is a two-phase process that encompasses data collection and data logging. The ResUsage facility consists of a set of tables, views, and macros to access system metrics.

Two Teradata subsystems work in conjunction with other subsystems to gather ResUsage data:

- Parallel Database Extension (PDE)
- Resource Sampling Subsystem (RSS)

Data Collection

During the data collection phase, both PDE and RSS gather information from the operating system and from Teradata Database. This data is temporarily stored in shared data collection buffers. Data collection continues for a user-specified period of time called the collect interval.

Data Logging

In the logging phase, RSS writes all gathered data to ResUsage tables and reinitializes the shared data collection buffers for the next log interval.

Collection Costs

Recording information in the DBC.ResUsage table requires disk space and processing time. Despite the additional resources used in performance monitoring, there are benefits to understanding how your system resources are being used.

Cost

The collection of ResUsage data incurs associated system overhead costs in three areas: I/O capacity, User DBC Perm Space and CPU utilization. The CPU has to write new rows to the ResUsage table depending on the preset logging interval. This increases CPU utilization during the collection process. In addition, the new rows added to the ResUsage tables require more perm space to hold the added data in user DBC where the table resides.

The costs for collecting ResUsage data depend on the table-logging interval, the number of active tables, and on the physical and virtual configuration of your system.

Resource Usage Data

ResUsage data provides historical system-level performance information.

- **Data is logged into various ResUsage tables at a specified logging interval.**

Two Teradata subsystems work in conjunction with other subsystems to gather ResUsage data:

- **Parallel Database Extension (PDE)**
- **Resource Sampling Subsystem (RSS)**

Data collection

- **PDE and RSS help to collect data and gather information from the operating system and the Teradata Database software.**

Data logging

- **RSS writes the collected data to ResUsage tables.**

Collections Costs

- **Disk Space**
- **Additional I/Os (minimal)**
- **Additional CPU overhead (minimal)**

Filling the ResUsage Tables

ResUsage information is gathered in three ways, depending on the nature of the data being collected.

- Counted – the number of times an event happened. The "gather or live" buffer is updated at each event.
- Time monitored – determines how much time was spent in a particular state. The "gather or live" buffer is updated at each state change.
- Tracked data – uses a snap shot of a queue length at the collect period. This information goes directly into the collect buffer.

Collection and Log Buffers

Statistical information is stored in the gather buffer and holds it there until the set collect interval. At that time, the utility moves the data to the work and collect buffers. The collected data is referred to as PM/PC (Performance Monitoring/Production Control) data.

For ResUsage data, the work buffer is moved to the log buffer, where it is held until the set log interval is reached. The data in the log buffer is written to appropriate ResUsage tables in the Teradata database.

Resource Usage Logging

When you initiate data logging, the system collects a variety of statistics for a period you specify. Teradata stores performance data in the DBC.ResUsage tables. Teradata uses nine tables to gather resource utilization data for a specified time period, and stores this information by node or vproc.

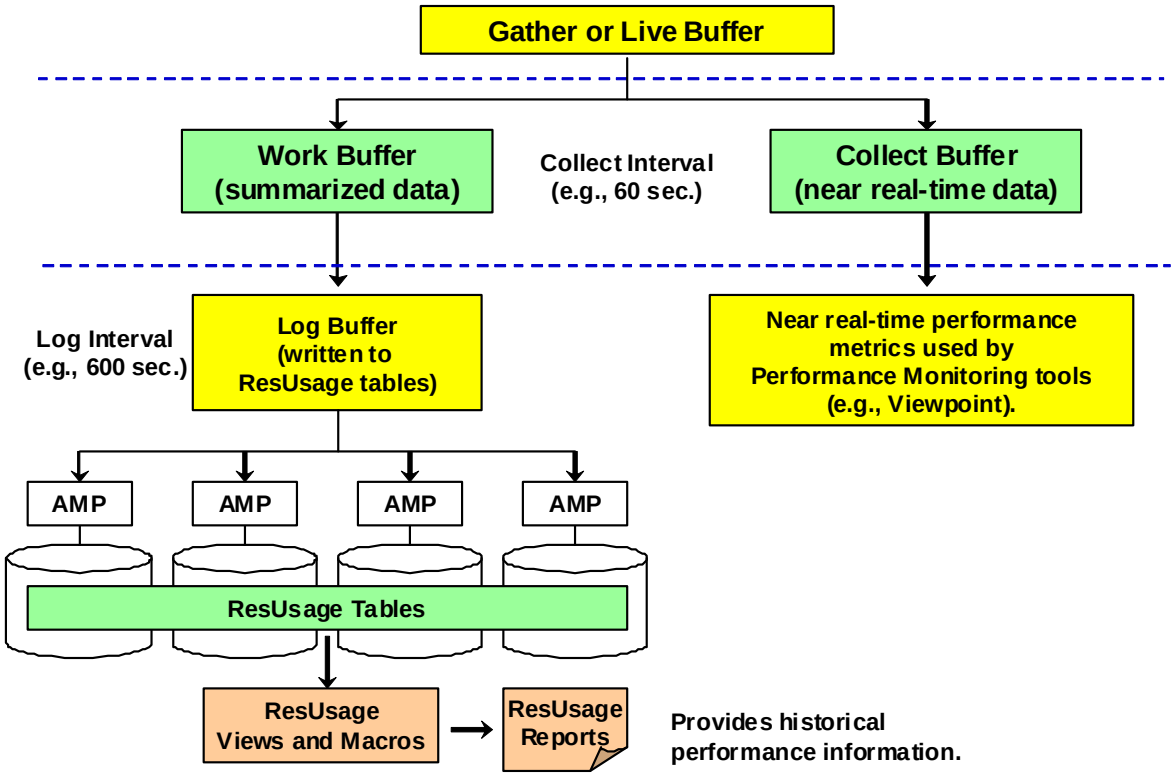
You can access the statistics stored in the DBC.ResUsage tables directly or use supplied views. In addition, you may create ResUsage reports using supplied macros that access the ResUsage views.

The system does not automatically collect resource utilization data. Consequently, you must activate resource collection and logging to gather performance data. Setting resource logging is described later in this module.

Real-time Performance Monitoring

Data in ResUsage tables is after-the-fact data, because it has been collected and stored. With Viewpoint, you can view and capture data by looking directly into the collect buffer to see what is happening real-time on the system. This can be very useful for determining what is happening as a particular transaction or request is running.

Logging ResUsage Data



Specifying ResUsage Tables and Logging Rates

Various utilities can be used to specify which of the ResUsage tables to collect and log information into. These tools include:

- ctl utility
- Supervisor commands (e.g., SET LOGTABLE …) via DB Window

  Note: Using the supervisor commands in the Database Window, you can enable tables and set collection and logging, but you cannot set Summary mode.

Teradata 13.10 Notes

- The RSS Collection Rate will not be displayed in the TD13.1 ctl RSS screen.

- The rationale for no longer displaying the Collection rate in the RSS screen is that the RSS screen is really all about logging the ResUsage data. In the past, the Collection rate had to be enabled, but in TD 13.10, the Collection and Logging are independent. The Collection rate is for PM/PC which also has its own interface for setting the Collection rate. The Collection rate is still supported in CTL since CTL should be able to set or read all the fields in the Control GDO when necessary.

- You can display the collection rate as shown below. Using ctl, …

  > print rss collection rate
  RSS Collection Rate=30
  > print vproc logging rate
  Vproc Logging Rate=0

Specifying Tables using the ctl Utility

From a Linux command line, enter the command - ***ctl*** - and then enter appropriate control utility commands as shown in detail in the reference manual. Note – the logging rate has a range from 0 – 3600 seconds (0 turns it off).

Example: The following command (from root prompt) will set the collection rate to 60 seconds and the logging rate to 600 seconds.

ctl -first "RSS Collection Rate = 60; node logging rate=600; screen rss; write; quit"

Specifying Tables using Supervisor Commands

You can also set resource logging from the Database Window Supervisor screen. The **set** and **get resource** commands may be used:

set logtable *tablename_or_ALL* ON/OFF

Specifying ResUsage Tables and Logging Rates

Tools that can be used to specify the tables to collect and log include:

- Control GDO Editor – ctl
- Supervisor commands (e.g., SET LOGTABLE ...) via DB Window

Example of specifying tables to log system data using the "ctl" utility.

```
# ctl
> screen rss
> 1=on

(0) Node Logging Rate  : 600 sec

                RSS Table Logging Enable
(1) SPMA : On       (2) IPMA : Off      (3) SCPU: Off
(4) SVPR : On       (5) IVPR : Off      (6) SLDV: Off       (7) SHST: Off
(8) SPDSK: Off      (9) SVDSK: On       (A) SAWT: On        (B) SPS : On

                RSS Summary Mode Enable
    Summarize SPMA: Off          Summarize IPMA : Off       (C) Summarize SCPU : Off
(D) Summarize SVPR: On       (E) Summarize IVPR : Off       (F) Summarize SLDV : Off
(G) Summarize SHST: Off      (H) Summarize SPDSK: Off       (I) Summarize SVDSK: Off
(J)  Summarize SAWT: On      (K) Summarize SPS  : On

> write

CTL: Control GDO successfully written.
```

Resource Usage Tables

Teradata stores ResUsage data in a set of system tables. Each ResUsage macro derives its report from one or more of these tables. You must activate logging to produce a useful ResUsage report.

The Resource Sampling Subsystem (RSS) gathers ResUsage data through shared data collection buffers. The Collect buffer gathers entries according to the collection rate intervals. During the log rate interval, the entries are moved into the Log buffer. At the end of the log period, RSS will log the gathered data to the following ResUsage Tables and re-initialize the shared data collection buffers for the next log period.

All ResUsage table names begin with ResUsage and have the following extensions.

S: System or I: Internal (of interest mainly to Teradata development personnel)

pma:	node information
cpu:	cpu-specific information
vpr:	vproc information
ldv:	logical device information
hst:	Channel and LAN host information

The following tables are new starting with Terdata 12.0.

awt:	AMP Worker Task information
ps:	Priority Scheduler performance group information
pdsk:	AMP pdisk cylinder allocation, migration, and I/O statistics
vdsk:	AMP vdisk cylinder allocation, migration, and I/O statistics

Miscellaneous notes:

- All of the ResUsage tables are located in the DBC database.
- The DBC.ResUsageSobj table exists in V2R5, but is currently no used.

Setting Resource Logging from DBW

You can also set resource logging from the Database Window Supervisor screen. The **set** and **get resource** commands may be used:

set resource coll <vproc-collect-rate> **vproc log** <vproc log rate>
set resource coll <node-collect-rate> **node log** <node log rate>
get resource

Setting Resource Logging from the Control Utility – ctl

From a Linux command line, enter the command - ***ctl*** **-** and then enter appropriate control utility commands as shown in detail in the reference manual.

Resource Usage Tables

Node Data		
ResUsageSpma	System-wide node information.	Generally enabled.
ResUsageIpma	System-wide internal node information.	Generally not needed.
CPU Data		
ResUsageScpu	Information specific to the CPUs in a node.	Enable if Spma shows no obvious bottleneck.
VProc Data		
ResUsageSvpr	Data specific to each virtual processor	Generally enabled.
ResUsageIvpr	System-wide internal vproc information.	Generally not needed.
Host and LAN Data		
ResUsageShst	Information specific to the host channels and LANs that communicate with TD.	Generally enabled.
Logical Device Data		
ResUsageSldv	Logical Device – information specific to disk I/O.	Generally not needed.
Additional tables		
ResUsageSawt	Collects and reports statistics about the AWTs.	
ResUsageSpdsk	Provides AMP-level Pdisk statistics.	Generally not needed.
ResUsageSvdsk	Provides AMP-level Vdisk statistics.	
ResUsageSps	Priority Scheduler Performance Group information.	

Resource Usage Views

Each row in the ResUsage tables represents activity during one logging period; the same is true of each row in the views. The difference between the tables and the views are the specific column values. ResUsage tables hold raw data. The views derive values from data in ResUsage tables.

ResGeneralInfoView

The ResGeneralInfoView provides an overview of system operation. Contains data from ResUsageSpma covering CPUs, disks, and BYNET information.

ResCPUUsageByAMPView

Contains data from ResUsageSvpr detailing the ways the CPUs are used by the AMPs.

ResCPUUsageByPEView

Contains data from ResUsageSvpr detailing the ways the CPUs are used by the PEs.

ResShstGroupView

The ResShstGroupView is based on the ResUsageShst table.

ResSldvGroupView

The ResSldvGroupView is based on the ResUsageSldv table.

ResSvdskGroupView

The ResSvdskGroupView is based on the ResUsageSvdsktable. This view includes resource usage detail on cylinder allocation, migration, and I/O statistics.

Resource Usage Views

ResUsage Tables

- ResUsageSpma
- ResUsageScpu
- ResUsageSvpr
- ResUsageShst
- ResUsageSldv
- ResUsageSawt
- ResUsageSps
- ResUsagePvdsk
- ResUsageSvdsk

Calculations

(raw data)

The following types of data are derived from ResUsage tables:

- Resource utilization percentages
- Event counts per second
- Average event size

ResUsage Views (Examples)

ResGeneralInfoView	View of general system information
ResCPUUsageByAMPView	View of CPU usage by AMP
ResCPUUsageByPEView	View of CPU usage by PE
ResShstGroupView	View of Host Channel and LAN activity
ResSldvGroupView	View of disk activity with Node Groups
ResVdskGroupView	View AMP virtual disk activity

Resource Usage Macros

Resource usage macros produce reports from data collected in the resource usage tables. You can use the reports to analyze key operational statistics and evaluate the performance of your system. Like other macros, resource usage macros consist of one or more Teradata SQL statements stored in the Teradata Database and executed by a single EXECUTE statement.

In addition to the name of the macro, the EXECUTE statement for resource usage macros can include optional parameters to specify the following:

- A specific single node or a group of nodes
- Starting and ending dates and times
- Starting and ending nodes of a range of nodes

The macros are installed in the DBC database by DIP. You can run these macros after logging ResUsage data on a specific job or set of jobs.

There are different macros for one node, multiple nodes, a group of nodes, or all nodes. Examples of some of the macros are listed below.

Macro	**Information Provided**
ResCPUByAmp	CPU utilization of each AMP vproc.
ResCPUByPE	CPU utilization of each PE vproc.
ResCPUByNode	CPU utilization of the node.
ResHostByLink	Host traffic for each communication link.
ResLdvByNode	Logical device traffic channeled through the node by totaling its controller links into one summarized node output line.
ResMemMgmtByNode	Memory management activity information for the node.
ResNetByNode	Net traffic for the node.

Teradata features four ResNode macros that summarize resource usage.

Macro	**Provides summary of ResUsage…**
ResNode	Averaged across all nodes
ResOneNode	For a specific node
ResNodeByNode	Node by node
ResNodeByGroup	For a node grouping

Resource Usage Macros

The ResUsage facility provides macros to report information about Teradata.

One-node Macros	Multiple-node Macros	Group-node Macros*	All-node Macros
	ResAwtByAMP		ResAwt
	ResAwtByNode	ResAmpCpuByGroup	
ResCPUByAMPOneNode	ResCPUByAMP	ResPeCpuByGroup	
ResCPUByPEOneNode	ResCPUByPE	ResCPUByGroup	
ResCPUOneNode	ResCPUByNode	ResHostByGroup	
ResHostOneNode	ResHostByLink	ResLdvByGroup	
ResLvdOneNode	ResLdvByNode	ResMemByGroup	
ResMemMgmtOneNode	ResMemMgmtByNode	ResNetByGroup	
ResNetOneNode	ResNetByNode	ResNodeByGroup	ResNode
ResOneNode	ResNodeByNode	ResPdskByGroup	
ResPdskOneNode	ResPdskByNode	ResVdskByGroup	
ResVdskOneNode	ResVdskByNode	ResPsByGroup	
	ResPsByNode		

* Group-node macros are designed for co-existence systems.

General Macro Syntax:

EXEC *Macroname* (*FromDate, ToDate, FromTime,ToTime* [, additional parameters depend on macro];

Example using ResNode macro:

```
EXEC DBC.ResNode (Date - 7, Date , , );
```

This generates data from one week ago to today using the ResNode macro.

Example Output from DBC.ResNode Macro

The facing page contains an example of general ResUsage Summary information across all nodes.

There are 23 columns (including date and time) with the ResNode report. The first 5 columns (after the date and time) represent:

CPU Bsy%	Percent of time the CPUs are busy, based on average CPU usage per node
CPU Eff%	Parallel efficiency of node CPU usage. Parallel efficiency is the total percent of time nodes are busy. It is the average for all nodes of total busy divided by the total busy time of the busiest node.
WIO %	Percent of time the CPUs are idle and waiting for completion of an I/O operation.
Ldv IOs /Sec	Average number of logical device (disk) reads and writes per second per node.
Ldv Eff %	Parallel efficiency of the logical device (disk) I/Os. It is the average number of I/Os per node divided by the number of I/Os performed by the node with the most I/Os.

If a system's resources are (CPU and I/O) are heavily utilized, it may be necessary to add system resources. The DBC.ResNode macro can be used to provide before and after results.

The example on the facing page was captured for a Teradata 5450 four-node system.

Example Output from DBC.ResNode

EXEC DBC.resnode ('2011-01-25', '2011-01-25', '08:00:00', '09:00:00');

Date	Time	CPU Bsy %	CPU Eff %	WIO %	Ldv IOs /Sec	Ldv Eff %	...
2011-01-25	08:00:00	98	100	0	1477	99	...
2011-01-25	08:10:00	99	100	0	1416	98	...
2011-01-25	08:20:00	100	100	0	1290	97	...
2011-01-25	08:30:00	100	100	0	1260	100	...
2011-01-25	08:40:00	95	99	1	1315	99	...
2011-01-25	08:50:00	97	100	0	1240	97	...

For the DBC.ResNode macro to display data, logging must be enabled on ResUsageSpma.

Notes about columns shown in this output:

- CPU Busy - % of time the CPUs are busy; based on average CPU usage per node.
- CPU Efficiency - parallel efficiency of node CPU usage; parallel efficiency is calculated by dividing average node utilization by maximum node utilization.
- WIO % - Percent of time the CPUs are idle and waiting for completion of an I/O operation.
- Ldv IOs /Sec - average number of logical device (disk) reads and writes per second per node.
- Ldv Eff % - parallel efficiency of the logical device (disk) I/Os.

PM/API and Viewpoint

The PM/API (Performance Monitor/Application Programming Interface) facility is a real-time performance-monitoring tool that allows you to collect and return performance data on a Teradata Database with low overhead.

Teradata Viewpoint is a web-based application that uses the PM/API to provide real-time performance and session information (PM/PC) which has been collected.

How PM/API Collects Data

PM/API contains monitoring commands that you issue through a logon partition called MONITOR. MONITOR collects different types of performance data, including the current system configuration; resource usage and status of individual nodes or vprocs; and of individual sessions.

PM/API collects data in memory, not in a spool file on disk. As a result, PM/API routines (except the IDENTIFY command) cannot be blocked and consequently incur low overhead. PM/API stores node and vproc resource usage data and session-level usage data in separate collection areas. The data stored in memory is updated once during each sampling period. All users share the collected data.

The MONITOR partition collects and reports resource usage data differently from session-level usage data. To interpret the information that the MONITOR returns, you must understand the difference.

Collecting and Reporting Resource Usage Data

PM/API collects and reports node and vproc usage data for a single sample period. For example, a user sets the sampling period to 120 seconds. Then she issues the MONITOR RESOURCE request. The system collects node and/or vproc usage data during the next 120 seconds. If the user does not examine the data within the next 120 seconds, the data is lost when it is overwritten by data collected during the next 120-second interval.

Collecting and Reporting Session-level Usage Data

PM/API cumulatively collects session-level usage data, such as counts and “time used.” Other data, such as locking information and “AMP State,” is not gathered cumulatively. The sampling period limits how frequently the cumulative data is updated.

Example

A user sets the sampling period to 300 seconds and issues the MONITOR SESSION request. The system collects new information every 300 seconds, and adds the information to the existing total in a cumulative fashion. Session-level data includes data for the beginning 300 seconds as well as for any subsequent intervals.

PM/API and Viewpoint

The PM/API (Performance Monitor/Application Programming Interface) ...

- is part of Teradata software and has low overhead.
- provides real-time monitoring capability and session information.
- provides the following data.
  - **Processor Data**
    - Collects/reports node/vproc usage for single period.
    - New period overwrites data from previous period.
    - Collection is not cumulative.
  - **Session-level Data**
    - Collects/reports session-level data cumulatively.
    - New sampling period increases collected data.
- Accessed via applications such as Teradata Viewpoint.

Teradata Viewpoint

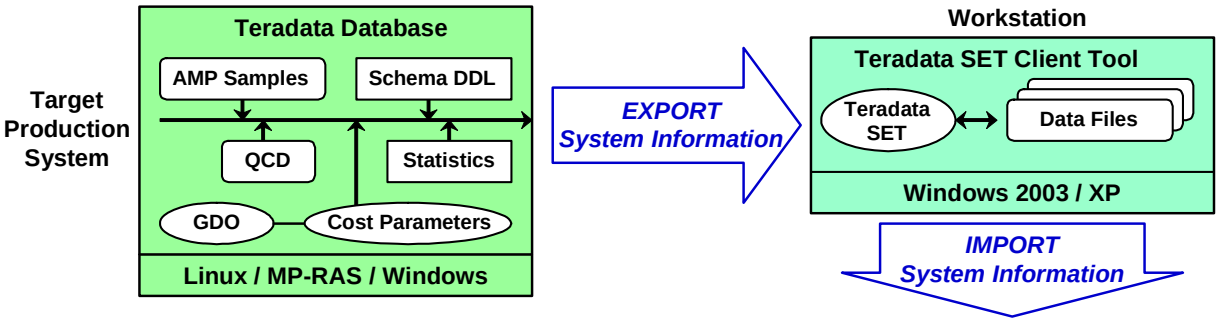
- Web-based application that utilizes portlets to ...
  - provide real-time performance monitoring.
  - show how efficiently the Teradata database is using its resources.
  - identify problem sessions and users
  - abort sessions and users having a negative impact on system performance

Teradata System Emulation Tool (Teradata SET)

The Teradata System Emulation Tool simplifies the task of emulating a target system by providing the ability to export and import all of the information necessary to fake out the optimizer in a test environment. This information can be used along with Teradata's Target Level Emulation feature to generate query plans on the test system as if they were run on the target system. This feature is useful for verifying queries and reproducing optimizer related issues in a test environment.

Teradata SET allows the user to capture system cost parameters, object definitions, random AMP samples, statistics, query execution plans and demographics by database, by query or by workload. This tool does not export user data. Upon import the user can customize or edit object definitions, random AMP samples, statistics and cost parameters. The Customize feature allows the user to perform "what-if" scenarios relating to the data demography of the tables and system performance parameters. Teradata SET also has an option to log SQL statements. The user can view the log directly from the Teradata SET window to troubleshoot any failures that occur during export or import operations.

Teradata System Emulation Tool



Features of Teradata System Emulation Tool (SET)

- Emulates a target system
- Export and import information to fool the optimizer in a test environment.
- Uses TLE (Target Level Emulation) to generate query plans on a test system.
- Verify queries and reproduce optimizer issues in a test environment.
- Capture system cost parameters, object definitions, random AMP samples, statistics, query execution plans and demographics by database, by query or by workload.
- Perform “what-if” scenarios relating to the data demography of the tables and system

Performance Monitoring Summary

The facing page summarizes some important concepts regarding this module.

Performance Monitoring Summary

- Resource usage (ResUsage) data and reports can help you to improve system performance and management.
- Various tools exist to specify the tables to collect and log as well as setting the collection and log rates.
- You can access the ResUsage statistics stored in the ResUsage tables directly or use supplied views and macros.
- The PM/API facility is an application programming interface that can be used to provides a real-time monitoring capability and session information.
- Teradata Viewpoint is a web-based interface that utilizes portlets to assist in monitoring the resources of a system.

Module 53: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 53: Review Questions

1. What are two methods of setting the ResUsage logging intervals?

   _____ _____

2. Match the following tools to its description.

____ 1. ResUsage tables	A. Set logging rates
____ 2. Teradata Viewpoint	B. Emulates a target system
____ 3. Teradata SET	C. Provides session level information
____ 4. ctl	D. Holds historical resource data

Notes

Module 54

TERADATA
Raising Intelligence

System Restarts

After completing this module, you will be able to:

- **List three different ways to restart the Teradata database.**
- **Use the RESTART command.**
- **Describe the impact of …**
  - **Disk(s) failure**
  - **Disk array controller(s) failure**
  - **BYNET(s) failure**
  - **Node failure**
  - **AWS failure**
  - **VPROC failure**

Notes

Table of Contents

Types of Restarts 54-4
Scheduled Restart/User-initiated Restart 54-4
Unscheduled Restart/Automatic Restart 54-4
Scheduled Restarts 54-6
-f option 54-6
-x option 54-6
-y option 54-6
-d option 54-6
-l delay option 54-6
-P option 54-6
-Q option 54-6
cold option 54-6
coldwait option 54-6
Restart Teradata from DB Window 54-8
RESTART Command Options 54-8
Restart using the “tpareset” Command 54-10
PDE States 54-12
Unscheduled Restarts 54-14
Disk Failure 54-14
Unscheduled Restarts (cont.) 54-16
BYNET Failure 54-16
Unscheduled Restarts (cont.) 54-18
Node Failure 54-18
VPROC Failure 54-18
AWS Failure 54-18
TPA Reset – Crashdumps 54-20
PDE DUMP 54-20
Allocating Crashdumps Space 54-22
Example 54-22
TPA Dump Maintenance 54-24
Linux Operating (Panic) Dumps 54-24
Module 54: Review Questions 54-26

Types of Restarts

There are two types of restarts on a Teradata Database:

- Scheduled restarts
- Unscheduled restarts

Scheduled Restart/User-initiated Restart

In a scheduled or user-initiated restart, use the RESTART command from either the DBW Supervisor window or from **vprocmanager** to restart the system.

Unscheduled Restart/Automatic Restart

In an unscheduled restart or automatic restart, the system reboots without user input.

The facing page provides examples of when you might need to perform scheduled restarts, and under what conditions you might encounter unscheduled restarts.

Types of Restarts

Scheduled Restarts

- Changing system parameters (e.g., DBS Control parameter is updated)
- Software upgrades
- Configuration changes (addition of new AMPs and/or PEs)

Unscheduled Restarts

- Power failure (e.g., 8/14/2003 – the North East U.S. and parts of Canada)
- Disasters (e.g., 8/29/2005 – Katrina hurricane; 10/22/2007 – Rancho Bernardo fires)
- Hardware failure
- Software failure
- Accidents

Restart Processes

1. Spool cylinders are returned to free cylinder list (unused cylinder pool).
2. Before logons are enabled, uncommitted work is rolled back.
   - 1st Tables are re-locked for background recovery.
   - 2nd Logons are enabled in cold start.

Scheduled Restarts

The facing page shows the windows and utilities from which you can restart the Teradata Database system, the necessary commands and available restart options. Additional information about options commonly used with **tpareset** or **restart** is provided below:

-f option

The -f option, used with the tpareset command, forces all TPA nodes to participate in the tpareset regardless of their current state, without rebooting Linux.

-x option

The -x option allows you to shut down the Teradata Database on the entire system without shutting down the operating system. This option does not automatically restart Teradata.

-y option

The -y option automatically answers yes to the confirmation prompt.

-d option

Specifies that a DBS dump be taken before doing the restart.

-l delay option

Specifies the delay interval in seconds to wait for other nodes to join the TPA configuration. This parameter controls how long a node will wait during the BYNET configuration phase of PDE initialization for other nodes to reach that point before continuing on without them.

-P option

Requests the node to panic after the DBS dump is saved.

-Q option

Requests tpareset to run in silent mode, i.e., user is not prompted for confirmation. This should be combined with any other desired option. This will not have any effect on the –P option.

cold option

If you use the cold option in conjunction with the restart tpa or restart commands, the system does not wait for AMP vprocs to complete recovery. Instead, the system places them in offline catchup. The system will enable logons **before** recovery is complete.

coldwait option

If you use the coldwait option, the system waits for down AMP vproc recovery to complete on all vprocs and brings all AMP vprocs online. The system will enable logons **after** recovery is complete.

Scheduled Restarts

Restart Teradata with	Use this command	Options
Command-line	tpareset <comment>	-f, -x, -y -d, -l, -Q, -P
DB Console - Supervisor	restart tpa <comment>	cold, coldwait
vprocmanager	restart	cold, coldwait
MultiTool (12.0 option)	reset (via GUI choices)	GUI menu choices

Example:

```
# tpareset -f Change of system parameters
```

To see any restarts that have occurred in the last 60 days:

```
LOGON tdpid/systemfe,service;
EXEC ALLRESTARTS (DATE - 60,);
LOGOFF;
```

The "tpatrace" command may also be used to see information about restarts.

```
# tpatrace  3
```
(shows last 3 restarts)

Restart Teradata from DB Window

In a scheduled restart, you may restart the system using the RESTART command from the DBW Supervisor screen or from vprocmanager.

RESTART Command Options

The RESTART command provides the following options:

DUMP – Default is NODUMP. This option can request that the Teradata Database restarts with or without a crashdump. DUMP=YES:NO.

COLD – A full restart, but transaction recovery will be deferred. This option allows the system to determine whether a down processor is to be kept off-line, or brought back on-line while recovery is being performed. The amount of updating to be performed on the down processor is the determining factor.

Recovery tasks are performed in the background after the system becomes available for use. Objects involved in recovery tasks are locked until the recovery is complete. All other objects are accessible to users.

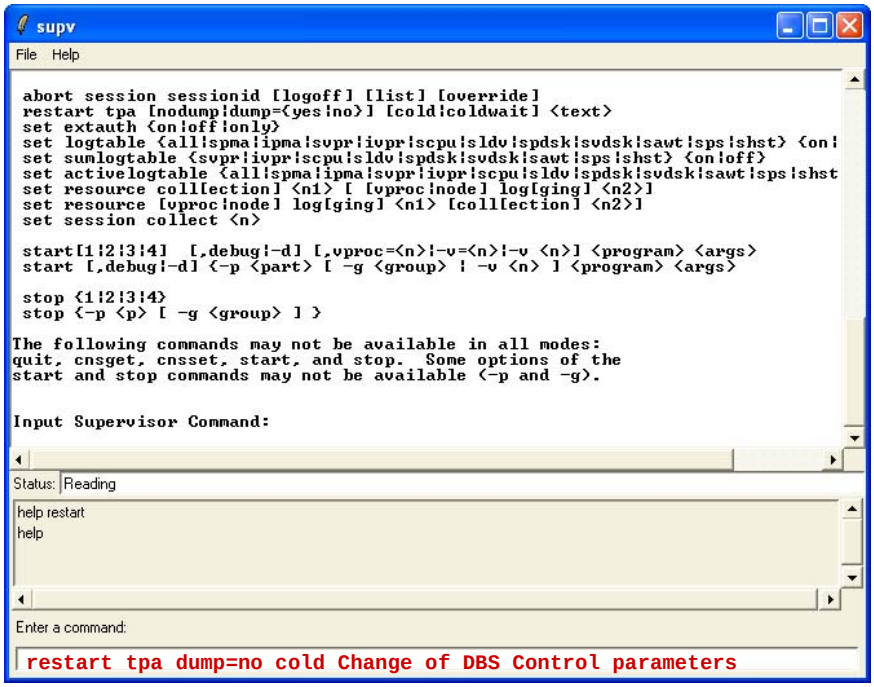
COLDWAIT – A full restart, but DBS startup will be held up until transaction recovery is complete. This option specifies that all recovery options must be completed before logons are enabled. All recovered AMP vprocs are placed on-line.

COMMENT – Enter a note explaining why the restart occurred. This entry is mandatory.

You can parse the RESTART command using commas. There is no mandatory order for the keywords (dump option and the restart kind).

Note: The SET RESTART command and Set Restart Type screen set the restart type to use during the next restart of the system.

Restart Teradata from DB Window



RESTART TPA [NODUMP:DUMP={YES:NO}] [COLD:COLDWAIT] COMMENT

Restart using the “tpareset” Command

The **tpareset** command can be used to restart the Teradata database. Common options used with **tpareset** include:

-f option

The -f option, used with the tpareset command, forces all TPA nodes to participate in the tpareset regardless of their current state, without rebooting Linux.

-x option

The -x option allows you to shut down the Teradata Database on the entire system without shutting down the operating system. This command shuts down Teradata without restarting Teradata.

Restart information is logged in numerous locations depending on the operating system. For example, with older UNIX MP-RAS systems, the following locations are utilized.

- SW_Event_Log table – view with Software_EventLog[V] view
- Console Log (e.g., /etc/.osm)
- /var/adm/streams (MP-RAS)

Restart using the "tpareset" Command

Example of using the **tpareset** command:

```
# tpareset -f Change of DBSControl parameters

        You are about to restart the database
                 on the system
                    'tdt6-1'
        Do you wish to continue (default: n) [y,n]
tpareset: TPA reset submitted.
```

Example of using the **tpatrace** command:

```
# tpatrace

TPA Initialization Trace for System tpt6-1 Node 001-01

01/28/11 11:18:34 -------------------- PDE starting   TPA Cycle = 41

01/28/11 11:18:34.16 (0/0 58a4) ---- PDE starting.
01/28/11 11:18:34.16 (0/0 58a4) State is START/BEGIN.
:
01/28/11 11:18:36.71 (0/0 58a4) State is START/VPROCSTART.
:
01/28/11 11:18:36.96 (0/0 58a4) State is RUN/STARTED.
01/28/11 11:18:39.96 (0/0 58a4) PDE started in 6 seconds.
```

Restart information is logged to Linux and to Teradata:

- **Linux messages log – /var/log/messages**
- **DBC.Sw_Event_Log table – use the following view to select Teradata restart entries**
  - **SELECT * FROM DBC.Software_Event_LogV WHERE TheFunction LIKE 'tpareset%';**

PDE States

The **/ntos/bin/pdestate** command can be used to check the current state of the PDE and Teradata software. The states have different names depending on the operating system environment.

- Open PDE (Linux/Windows) has five major operational states: DOWN, START, RUN, RESET, and STOP.

- PDE (UNIX MP-RAS) has three major operational states: NULL, NOTPA, and TPA. When PDE starts up, it transitions between them in that order. The sub-states correspond to the different phases of startup.

Examples of the major states and sub-states with MP-RAS include:

NULL/START - PDE has never started on the node(s).

NULL/STOPPED – PDE is stopped on the node(s), either explicitly or due to the start-up crash count exceeding its crash limit.

NULL/RESET – "Real" state when PDE is in reset or down state.

NULL – TPA is down on a node due to a late-joiner or other reason. The node(s) will respond to any type of reset.

NOTPA/START – the PDE is reading the local vconfig.GDO to get the TPA node list. It starts the kernel event daemons.

NOTPA/NETCONFIG – the PDE is waiting for all the node(s) to reach this state. There is a wait default of 6 min. (360 seconds) if node(s) are down. Note: Late-joiners fall out at this stage.

NOTPA/NETREADY – the PDE synchronizes the system GDO's. It selects the control and distribution nodes.

NOTPA/RECONCILE – the PDE verifies that the correct level of PDE and TPA software is installed on all the nodes.

NOTPA – the tpastartup file is processed.

TPA/START – the PDE starts all Vprocs. It enters the final FSG initialization stage by opening the Pdisks.

TPA/VPROCS – initializes and synchronizes application GDO's.

TPA/READY – the PDE start-up is complete. The DBS is not running.

TPA – the PDE start-up is complete. It also indicates that the DBS has started or is running.

PDE States

To display the DBS and PDE state without opening the vprocmanager utility.

```
# tdatcmd          (sets the path for Teradata command-line utilities)
# vprocmanager –s
DBS State: Logons are enabled - Users are logged on
PDE State: RUN/STARTED
```

The pdestate command displays the PDE state for a specific node.

```
# /usr/ntos/bin/pdestate -a
DBS State is 4: Logons are enabled – Users are logged on
PDE State is RUN/STARTED.
```

OpenPDE (Linux/Windows) has 5 primary operational states:

DOWN, START, RUN, RESET, and STOP

DOWN – no PDE services are currently running, other than those that are required to switch out of this state, i.e., startup of the database.

START – this is a transitory state between DOWN and RUN that may be seen for a brief time during database startup.

RUN – all PDE services are fully available and the TPA is started. Once this point has been reached, Teradata RDBMS can go through it's initialization.

Unscheduled Restarts

Disk Failure

When a disk fails, there may be a loss of data. Tables with fallback protection continue to be 100% available. Tables without fallback protection will only be partially available.

An AMP will attempt 5 retries to a disk array before determining that it cannot access the array or its associated Vdisk.

Unscheduled Restarts

Disk Drive Failures

Scenario 1

Failure: One disk in a drive group

Result: No TPA reset

Resolution: Replace disk – Array Controllers automatically rebuild the disk

Scenario 2

Failure: Two disks in a drive group

Result:
- TPA reset (1-5 minutes)
- AMP taken offline and marked as Fatal
- Fallback tables OK
- Non-fallback tables partially available

Resolution:
- Replace the two disks
- Reformat LUNs or Volumes in the drive group
- Perform a table rebuild
- Restore non-fallback tables

Scenario 3

Failure: Two disks in 2 different drive groups associated with AMPs in the same cluster – 2 AMPs fail in a cluster

Result: Machine halts

Resolution: Restore User DBC and tables

Unscheduled Restarts (cont.)

BYNET Failure

If a BYNET fails, processing will resume on the other BYNET. Performance will be impacted.

Unscheduled Restarts (cont.)

BYNET Failures

Scenario 1

Failure: **One BYNET fails**

Result:
- No TPA reset
- All traffic auto-switched to remaining BYNET
- Impact on system performance

Resolution: Repair BYNET

Scenario 2

Failure: **Both BYNETs fail**

Result: Teradata halts and is not available

Resolution: Repair BYNETs

Unscheduled Restarts (cont.)

Node Failure

The facing page describes a node failure.

VPROC Failure

PE VPROC This type of failure produces very little impact on system performance. It reduces the maximum number of sessions that can be active at one time, however, and logons may take longer when a PE vproc is down.

AMP VPROC If a single vproc fails in one or more clusters, the system can continue servicing users with the other AMP vprocs. However, the performance level drops causing a slow down in performance and response time.

If two or more AMP vprocs fail in a single cluster, it halts the database system. All processing stops until the administrator brings at least one of the down vprocs back on-line.

AWS Failure

The facing page describes an AWS failure.

Unscheduled Restarts (cont.)

Node Failure

Scenario

Failure: **Node Fails (e.g., O.S. hangs, 2 power supplies fail, memory fails, etc.)**

Result:
- **TPA restart (1 - 5 minutes) and vprocs migrate to other nodes in clique**
- **Possible O.S. reboot (3 - 15 minutes)**

Resolution:
- **Repair node and reboot operating system**
- **Restart Teradata to allow node to rejoin Teradata configuration**

Vproc Software Failure

Scenario

Failure: **AMP or PE Vproc fails**

Result: **TPA restart (1 - 5 minutes) and vprocs may be marked offline**

Resolution: **If necessary, run Scandisk, Checktable, and Rebuild utilities**

SWS Failure

Scenario

Failure: **SWS fails**

Result: **No restart of Teradata; SWS is not available to monitor/manage system**

Resolution: **Reboot or recover SWS**

TPA Reset – Crashdumps

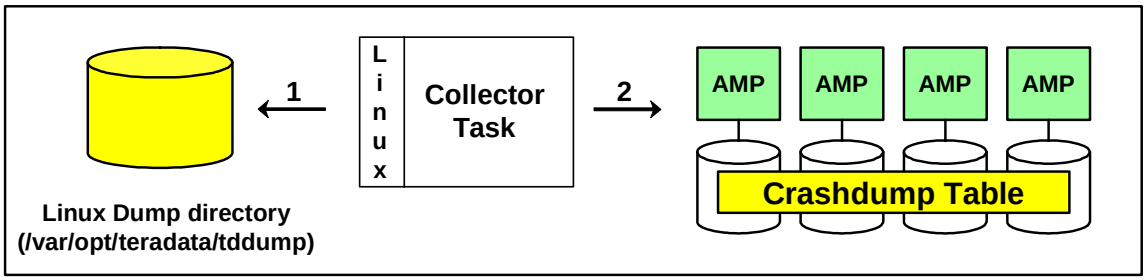
When there is an unscheduled TPA reset, a crashdump is generated (PDE dump).

PDE DUMP

A PDE dump is a selective dump of system memory; including only information that might be needed to analyze a problem within PDE or TPA (the only one currently is the Teradata Database). It can also contain pages read in from swap space that are not in memory at the time of the dump. Exact contents vary depending on the cause of the dump.

PDE dumps, being selective, vary in size depending on the system configuration and the cause of the crash. PDE dumps are much smaller than memory. PDE dumps are taken in parallel on all nodes.

TPA Reset – Crashdumps



1. Selective memory and swapped pages are written to “pdedump” space.
2. As part of Teradata restart, a background collector task reads “pdedump” and writes dump information to a Crashdump table in Crashdumps database.
   - If the Crashdumps database is out of perm space, the collector task outputs a warning message and retries every 60 minutes to create a crashdump table.

Allocating Crashdumps Space

During installation of a Teradata system, a user called CRASHDUMPS is created as a child of user DBC. Crashdumps is allocated 1GB of permanent space. Teradata recommends you allocate enough space to this database to hold three crash dumps.

Dump size is approximately 150 - 250 MB per node. Dump size can vary depending on the number of vprocs running, how busy the system was at the time of the crash, and a number of other factors. The use of fallback approximately doubles the total space requirement.

If a site needs additional space for the Crashdumps database, increase the MaxPerm space by submitting the MODIFY USER statement.

Example

The diagram on the facing page illustrates how to calculate the appropriate amount of permanent space for the Crashdumps database.

A site has a four-node system. The administrator needs to allocate enough space for three crash dumps. The formula is:

(4 x 96 GB x 3) = 1152 GB without fallback
(4 x 96 GB x 3) x 2 = 2304 GB with fallback

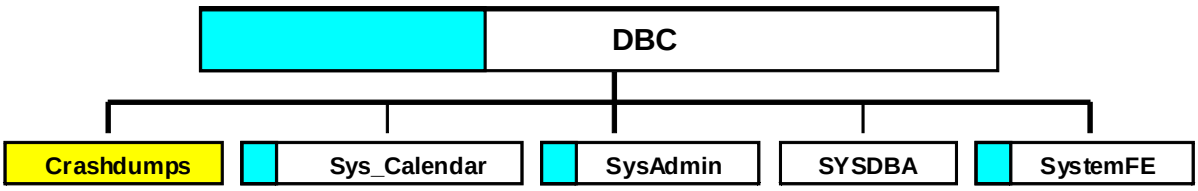
Since a crash dump is normally about 50% of actual memory, the above numbers can be divided in half. For initial sizing, use 1152 GB which will probably be enough for 3 crash dumps with fallback in the future.

You should monitor the actual Crashdumps space usage and adjust it up or down as appropriate, depending on the size of the dumps a site typically gets.

To modify Crashdumps space, you need to log on to the system as user DBC and submit the following SQL statement:

```
MODIFY USER Crashdumps AS PERM = 1152E9;
```

Allocating Crashdumps Space



Use the following formula to estimate the amount of space to allocate:

Example: Four-node system, 96 GB memory/node, and space for 3 Crashdumps.

(4 x 96 GB) x 3) = 1152 GB without fallback – double for fallback if needed

MODIFY USER Crashdumps AS PERM = 1152E9;

Note: Normally, a crashdump uses only 50% of node memory; for initial sizing purposes, allocate enough space for an entire memory image per node without fallback.

Example of Crashdump name: **Crash_20110930_143249_20**

(Date) (Time) (Segment #)

Help USER Crashdumps;

Table/View/Macro name	Kind	Comment
Crash_20110930_143249_20	T	PDE:13.10.00.05,TDBMS:13.10.00.05,TGTW:13.10.00.05;

TPA Dump Maintenance

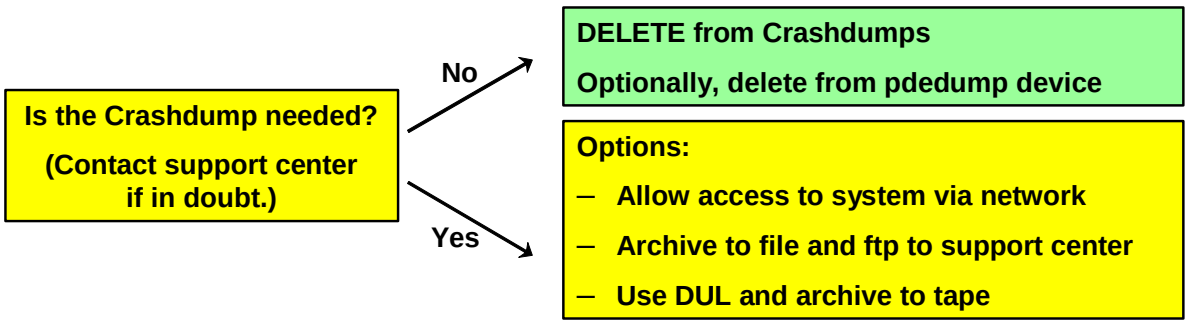
The facing page describes tasks for maintaining TPA dumps.

Linux Operating (Panic) Dumps

A Linux operating system dump is a complete dump of system memory including PDE and Kernel information, but only for the node(s) that panicked. If Linux panics on multiple nodes, you get a separate dump for each one.

Since this is a complete dump of memory, dump size is equal to the memory size on your system.

TPA Dump Maintenance



Module 54: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 54: Review Questions

1. What is the operating system command to restart Teradata? _____
2. What is the DB Window supervisor command to restart Teradata? _____
3. Which of the following choices will cause a Teradata restart? _____

    a. SWS hard drive failure
    b. Single drive failure in RAID 1 drive group
    c. Two drive failures in same RAID 1 drive group
    d. Single power supply failure in a TPA node
    e. TPA node CPU failure
    f. One of BYNETs fails
    g. LAN connection to SMP is lost

Notes

Module 55

System and Maintenance Utilities

After completing this module, you will be able to:

- **List two ways in which a system utility can be started.**
- **Explain how to use the following utilities to maintain a Teradata database:**
  - **DBSControl**
  - **Ferret Packdisk**
  - **Ferret Defragment**
  - **Ferret Scandisk**
  - **Checktable**
  - **Table Rebuild**
  - **Recovery Manager**
  - **Showlocks**
  - **UpdateSpace**
  - **Vprocmanager**
- **List the order in which to execute the Checktable and Scandisk utilities.**

Notes

Table of Contents

Starting Teradata System Utilities 55-4
SMP and Database Window Utilities 55-6
Teradata Database Window 55-8
DBW Supervisor Window 55-10
DBS Control Utility 55-12
DBS Control Record – General Fields 55-14
DBS Control Record – General Fields 55-16
DBS Control Record – File System Fields 55-18
DBS Control Record – Performance Fields 55-20
Modifying DBS Control Parameters 55-22
Ferret – Defragment and Packdisk 55-24
Checking Data Integrity 55-26
Ferret – Scandisk Utility 55-28
Checktable Utility 55-30
Checktable – Levels of Checking 55-32
Checktable – Example 55-34
Table Rebuild Utility 55-36
Recovery Manager Utility 55-38
Recovery Manager Commands 55-40
Rcvmanager – List Status 55-42
Rcvmanager – List Locks 55-44
Rcvmanager – List Status (2nd Example) 55-46
Rcvmanager – List Rollback Tables 55-48
Rcvmanager – Cancel Rollback on Table 55-50
Showlocks Utility 55-52
Orphan or Phantom Spool Issues 55-54
Update Space Utility 55-56
Vprocmanager 55-58
Summary 55-60
Module 55: Review Questions 55-62

Starting Teradata System Utilities

Teradata Database offers several user interfaces from which the utilities may be started as shown on the facing page. Note that not all utilities support all the user interfaces.

Once a utility is started, some utilities present their own interactive command-line or graphical user interfaces. These utilities let you browse and enter information until you explicitly exit the utility.

Other utilities present their own interactive command-line or graphical user interfaces. These utilities continue running until they are explicitly stopped by the user. Many utilities that present their own command environment are stopped by entering the QUIT command.

Utilities that present a graphical user interface are usually stopped by clicking the Exit or Close command from the graphical menu.

Teradata Console Task

The Teradata Console (CNS) task is responsible for managing the Teradata DB Window.

There are numerous ways in which the CNS task can be invoked.

- xdbw or the Teradata Database window
- Command-line utilities
- cnsterm (command-line interface to access supervisor)
- cnstool (command-line interface to access supervisor)

Examples of HUTCNS utilities (not covered in this class)

Examples:	
Session Status	(QRYSESSN) enter SES
Configuration Display	(QRYCONFIG) enter CON
Locks Display	(SHOWLOCKS) enter LOC
Recovery Manager	(RCVMANAGER) enter RCV
Showlocks	Displays host utility (HUT) locks that are held on databases and tables during archive and restore operations.
Rcvmanager	Displays the number of unprocessed transactions and down-AMP-recovery journal rows, as well as locks held by transaction recovery.

Starting Teradata System Utilities

Typical interfaces to start Teradata System Utilities are:

• Command line	Linux and Windows
• Database Window	Linux (X Windows) and Windows (Database Window)
• Remote Console	Viewpoint
• Host Utility Console (HUTCNS)	Mainframe operating system

Linux

Command Line	Execute "tdatcmd" – adds the directories to the path to execute command-line utilities.
Database Window	This Database Window is an X client program that requires an X server to be running on the local machine. example: xdbw -display 192.0.2.24:0.0 &

Windows

Command Line	Start the Teradata Command Prompt in Windows – adds the Teradata directories to the PATH environment variable Start>Programs>Teradata Database>Teradata Command Prompt
Database Window	Open Database Window (DBW) from the Windows Start menu: Start>Programs>Teradata Database>Database Window

SMP and Database Window Utilities

The facing page lists various Teradata control and support utilities and from where they are executed. These functions can be secured through the SMP **cnsperms** and **rhosts** files. This **cnsperms** file contains 4 fields separated by colons. The format is:

A:B:C:D where A = linux_userid_name@hostname
B = ALL or a list of allowed supervisor commands
C = Utilities the user is allowed to start
D = Utilities the user is NOT allowed to start

List of commands or utilities is space separated. Field D is ignored if field C is present.

This file is maintained by DB Window Supervisor commands.

- GET PERMISSIONS
- GRANT
- REVOKE

This file is automatically copied to all nodes by PDE software. Only execute these commands when all SMP nodes are available so that all copies are updated. Note that the HELP and GET commands require no permissions (these are always accepted).

cnstool and cnsterm commands

cnstool and **cnsterm** are command-line interfaces to the Teradata DB Console functions.

Starting cnstool

After executing **cnstool**, commands directed to the Supervisor or any application area have to be preceded with the appropriate number. Window numbers 1 through 4 are the console utility windows, 5 is the Database I/O window, and 6 is the Supervisor Window.

Only the root user is allowed to use this command: **# cnstool**

To start a utility such as qrysconfig, enter the following:
6: start qryconfig (assume qryconfig is started in area 1)
1:offline; (offline is a qryconfig option)

To exit from cnstool, enter either **Del** or **Control C.**

Starting cnsterm

When you start cnsterm, the only command line option is the window number.

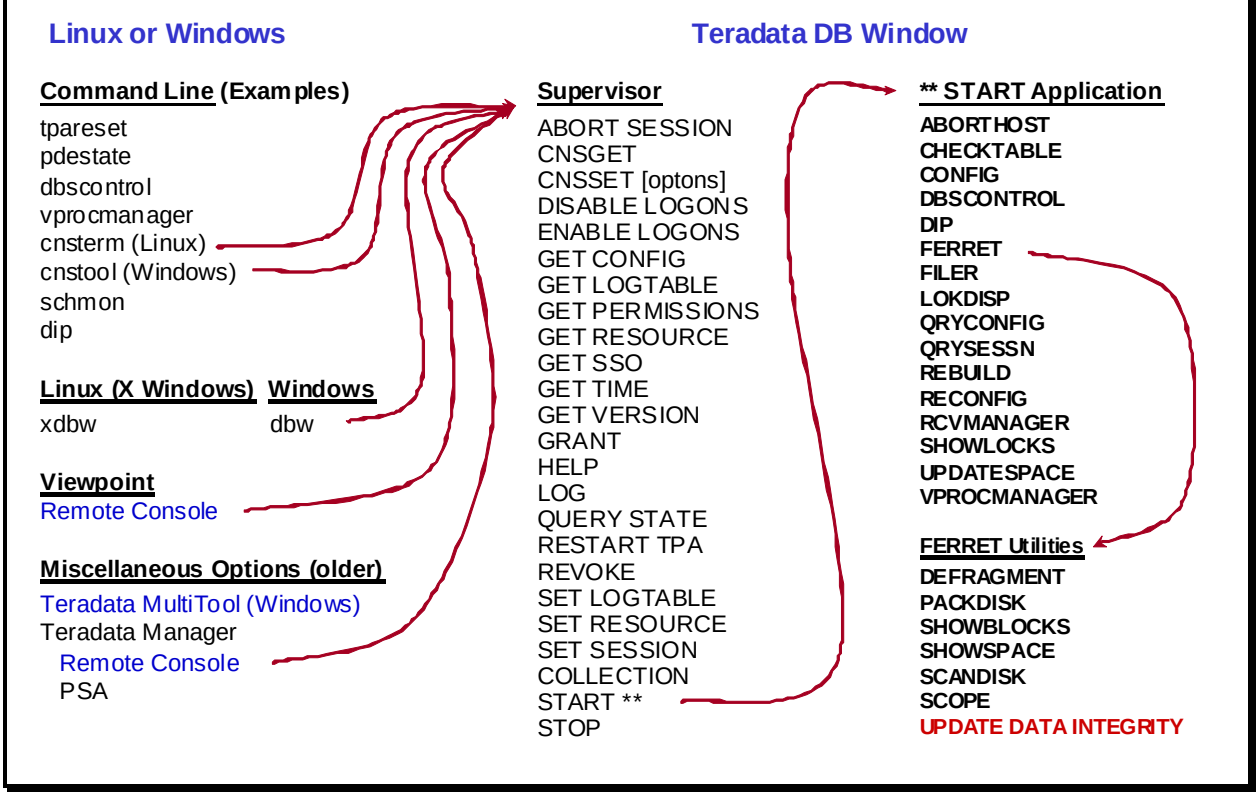
Only the root user is allowed to use this command: **# cnsterm *n***

where ***n*** is the window number.

For example, to display the supervisor: **# cnsterm 6**

To exit from any screen, press the keyboard's "break" or interrupt key (**Del** or **Control C).**

Using the Interfaces to Access Utilities



Teradata Database Window

The Database Window (DBW) is the console software for the Teradata Database. The DBW software provides a dimension of flexibility to a database administrator since you can start the console from virtually any workstation.

Database Window Icons

The DBW contains an icon labeled “Supvr” that opens the Supervisor window. You can start Teradata AMP-based utilities from the Supervisor window.

Once a Teradata utility is running, the DBW displays a new application icon. This icon opens a window where the Teradata utility is running. You can have a number of application icons present in the DBW, with each representing a different Teradata utility. You may move back and forth from one utility to another by returning to the DBW.

You can have multiple instances of the DBW window running at the same time. While you can have up to nine DBW windows open, you probably should not have more than 7 open. Two windows should be reserved for remote support, if necessary.

There are four application utility partitions available with the DBW window. You can run up to four utilities at one time, as well as any commands you execute from the Supervisor window.

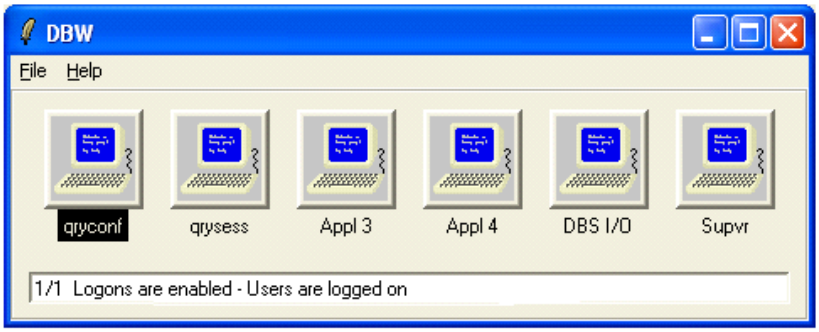
To start DBW in Linux, execute the following command from the Linux command line:

xdbw -display hostname:0.0 &

You must enter the command from the PDN node, or you must specify the PDN node using the machine option with the command as shown:

i -machine l7544 -display hostname:0.0 &

Teradata Database Window



Database Window (DBW)

- **The Database Window provides a windows interface to the Supervisor and the application areas.**

Database Window Icons

- **The Database Window contains a “Supvr” icon that opens the Supervisor window.**
- **Once a Teradata utility is running, the DBW displays a new icon which opens a window for the Teradata utility.**

DBW Supervisor Window

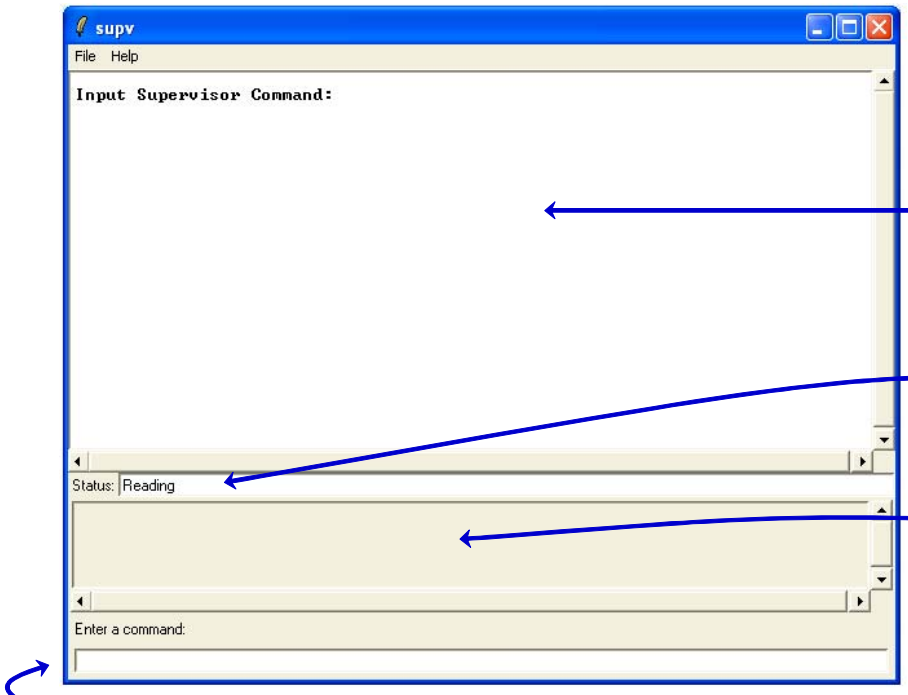
You must start the DBW before you can start the Supervisor program. To open the Supervisor window, click the “Supvr” icon in the DBW.

Sub-windows

The Supervisor window contains the following four sub-windows:

Output	Displays the results of user commands. It displays *Input Supervisor Command* when the Supervisor window first opens. Use the scroll bars to review results from previous commands not currently visible in the Output sub-window.
Status	Displays the current status message. The word *Status:* appears to the left of the sub-window and is used by CNS to indicate the state of the application running in this window. Current states include: Blank, Running: and Reading:
Command History	Displays a list of commands you previously entered. Use the scroll bars to review commands previously entered that are not currently visible in the Command History sub-window.
Input	Area where you type commands. The phrase *Enter a command:* appears just above this sub-window.

DBW Supervisor Window



Output Display Area

Status Area

Command History Area

Input Command Area

DBS Control Utility

The DBS Control utility is used to view/modify the DBS Control record fields which:

- Establish system values
- Tune performance
- Debug/diagnose problems

DBSControl Record fields are logically grouped based on how the Teradata Database uses them. The physical position of the field in the record is not significant. The group names are defined as follows:

This group...	**Contains fields used...**
General	For various purposes by the Teradata Database
File System	By Teradata Database V2 File System
Performance	For tuning performance
CheckSum	Set of parameters for CHECKSUM feature

There are multiple ways to access the DBSControl utility:

1. DB Window – **START DBSCONTROL**
2. Command line – **/tpasw/bin/dbscontrol**
3. Teradata Manager – Remote Database Console (pre 13.10 option)

Because modifying the DBSControl Record fields may have system wide ramifications, only trained personnel should use it.

For more information about DBSControl and all fields displayed by the utility, see the *Teradata RDBMS User Utilities*

DBS Control Utility

The DBS Control utility is used to view/modify the DBS Control Record fields which:

- **Establish system values**
- **Tune performance**
- **Debug/diagnose problems**

There are multiple ways to access the DBS Control utility.

1. **Command line – dbscontrol**
2. **DB Window (Supervisor) – START DBSCONTROL**

Parameters are divided into categories. Key categories include:

- **General**
- **File System**
- **Performance**

DISPLAY or HELP commands

- **DISPLAY GENERAL | FILESYS | PERFORMANCE | CHECKSUM**
- **HELP GENERAL | FILESYS | PERFORMANCE | CHECKSUM**

DBS Control Record – General Fields

This page only lists the first 30 general parameters.

Field	Purpose
1. Version	Indicates the DBS Control Record version number set by SysInit. Cannot be changed with DBSControl.
2. SysInit	Indicates whether or not the last SysInit was successful – cannot be changed with DBSControl. - set by SysInit.
3. DeadLockTimeout	Used for deadlock time-out detection cycles – time is in seconds.
4. Reserved	Reserved for future use.
5. HashFuncDBC	Defines the DBS hashing function. Cannot be changed with DBSControl - set by SysInit.
6. Reserved	Reserved for future use.
7. Reserved	Reserved for future use.
8. SessionMode	This field defines the system default transaction mode, case sensitivity, and character truncation rule for a session. The setting is either 0 (Teradata) or 1 (ANSI).
9. LockLogger	Enables logging of lock delays due to database locks.
10. Rollback Priority	Defines the priority for rollback operations – RUSH or user prioity. If RUSH is wanted, set to TRUE.
11. MaxLoadTasks	Controls the combined number of FastLoad, MultiLoad, and FastExport tasks allowed (max is 15).
12. RollForwardLock	Defines the system default for the RollForward Using Row Hash Locks option. This allows the DBA to specify that row hash locks should be used to lock the target table rows during a RollForward. To enable this feature set the field to TRUE. To disable the feature set the field to FALSE.
13. MaxDecimal	Defines max number of decimal digits used in expression typing (valid values are 15 and 18).
14. CenturyBreak	Defines which Teradata dates are specific to the 21st Century. Valid values range from 0 to 100.
15. DateForm	Defines the standared date format – IntegerDate is 0, ANSIDate is 1.
16. System TimeZone Hr.	This field defines the System TimeZone Hour offset from UTC. Permitted range: -12 to 13.
17. System TimeZone Min.	This field defines the System Time Zone Minute offset from UTC. Permitted range: -59 to 59.
18. RollbackRSTransaction	Used when a subscriber replicated transaction and a user transaction are involved in a deadlock. TRUE means rollback the subscriber replicated transaction. FALSE means rollback the user transaction.
19. RSDeadLockInterval	Deadlock checking between a subscriber-replicated transaction and a user transaction.
20. RoundHalfwayMagUp	This field indicates how rounding should be performed when computing values of DECIMAL type.
21. DefaultDateFormat	Default date format for the system. If a date format is defined then the format is used as the default for the system. Date formats are defined between enclosing single quotes ‘ ’. The following is an example, m g 21 = ‘yyyy/mm/dd’.
22. Target Level Emulation	Used when a user wants to emulate a Target Level Machine. TRUE means the user wants to run Target Level Emulation. FALSE means the user does NOT. The default is FALSE
23. Export Width Table ID	For Expected Defaults, enter 0. For Compatibility Defaults, enter 1. For Maximum Defaults, enter 2.
24. EnableStepText	When TRUE, dispatcher step text will include names and costs. When FALSE no name and cost information will be available. The default value is FALSE.
25. EnableDBQM	When TRUE, validation of all SQL through the DBQM rule will be enforced. When FALSE no validation through DBQM will be done. This toggle has a dependency on the EnableStepText toggle (#24). The default value is FALSE.
26. External Authentication	26. - This field indicates whether external authentication is enabled. The valid values are 0, 1, 2.
27. IdColBatchSize	Indicates the size of the pool of numbers to be reserved for generating numbers for a batch of rows to be bulk-inserted into a table with an identity column. The valid range of value is 1 .. 1000000. The default value is 100000.
28. LockLogger Delay Filter	Indicate whether locking logger delay filter is ON. If it is ON, the delay filter time in field 29 will take effect. The default is OFF.
29. LockLogger Delay Filter Time	Indicates the delay filter time used by locking logger. Only blocked lock request with delay time greater than this value will be logger. The valid range is 0 .. 1000000 seconds, default is 0 second.
30. Object Use Count Collect Rate	Determines the default amount of time which the Data Dictionary columns of database object AccessCount and LastAccess Time Stamp are reset automatically. The default is 0, which disables the database object use count feature. To enable, the recommended value is 10 minutes.

DBS Control Record – General Fields

General Fields:

1. Version = 6
2. SysInit = TRUE
3. DeadLockTimeOut = 240 (seconds)
4. (Reserved for future use)
5. HashFuncDBC = 5 (Universal)
6. (Reserved for future use)
7. (Reserved for future use)
8. SessionMode = 0 (Teradata)
9. LockLogger = FALSE
10. RollbackPriority = FALSE
11. MaxLoadTasks = 5
12. RollForwardLock = FALSE
13. MaxDecimal = 15 (18, or 38)
14. CenturyBreak = 0
15. DateForm = 0 (IntegerDate)
16. System TimeZone Hour = 0
17. System TimeZone Minute = 0
18. RollbackRSTransaction = FALSE
19. RSDeadLockInterval = 0 (240)
20. RoundHalfwayMagUp = FALSE
21. (Reserved for future use) =
22. Target Level Emulation = FALSE
23. Export Width Table ID = 0
24. (Reserved for future use)
25. DBQL Options (spec. options) = 0
26. External Authentication = 0 (On)
27. IdCol Batch Size = 100000
28. LockLogger Delay Filter* = FALSE
29. LockLogger Delay Filter Time* = 0
30. ObjectUseCountCollectRate* = 0 minutes (Disabled)

DBS Control Record – General Fields

This page only lists the additional general parameters.

Field	Purpose
31. LockLogSegmentSize	Indicates the size of the locking logger segment used by the lock manager in kilo bytes. The minimum value is 64 KB and maximum value is 1024 KB.
32. System Default Cost Profile Id.	This value is used to select the cost profile from DBC.CostProfiles that will be used to create cost prediction method parameter values.
33. DBQLFlushRate	Determines the frequency for writing DBQL cache entries to DBQL dictionary tables. Default is 600 seconds (10 minutes). Valid range is 1 to 3600 seconds. The recommended value is 600 seconds or more.
34. Memory Limit Per Transaction	Specifies the maximum amount (number of pages) of in-memory temporary storage that can be used by the RSG to store the records for one transaction. If the transaction exhausts this amount, then it is moved to a disk file (specified by general field number 37). Default - 2 pages; minimum - 0 pages and maximum value is 127 pages.
35. Client Reset Timeout	Specifies how long the RSG should wait for an intermediary to reconnect after a communication failure, an intermediary reset, or a server reset before taking the needed actions. Default - 300 seconds; minimum - 0 sec and maximum value is 65535 sec.
36. Temporary Storage Page Size	Specifies the size (in KB) of a memory page that is used to store data rows of a replicated transaction. Default - 4 KB; minimum – 1 KB and maximum is 1024 KB.
37. Spill File Path	Specifies a directory that will be used by the RSG for spill files. The default path is C:\Program Files\Teradata\Tdat\tdconfig\tdrsg
38. MDS Is Enabled	Assumed that the MDS packages are installed, the MDS will be running if this flag is TRUE. The default is FALSE.
39. Checktable table lock retry limit	(Default = 0 is retry forever). When a table is locked, checktable will retry until it gets the table. If the retry limit is set, checktable will retry within the specified limit. Retry limit range = 0 to 32767 minutes
40. EnableCostProfileTLE	This boolean determines whether new Optimizer Cost Profile System (OCES) diagnostics are enabled in combination with Target Level Emulation (TLE).
41. EnableSetCostProfile	This controls usage of DIAGNOSTIC SET PROFILE ... statements to dynamically change cost profiles used for query planning. Meaningful values are 0,1,2,20,21, and 22.
42. UseVirtualSysDefault	If this == 0, then the system default cost profile is always SysDefault. If this > 0, then the system default cost profile is chosen based on run time envirnoment characteristics.
43. DisableUDTImplCastForSysFuncOp	This field disables/enables implicit cast/conversion of UDT expressions passed to system operators/functions. Conversions are from UDTs to compatible predefined types. A value of TRUE disables implicit conversions. A value of FALSE (default) enables.
44. CurHashBucketSize	The number of bits currently used for the hash bucket size - 16 or 20.
45. NewHashBucketSize	The number of bits used for the hash bucket size for next reconfig or sysinit – 16 or 20.
46. MaxLoadAWT	This field defines the maximum number of AMP worker tasks (AWT) that concurrent FastLoad and MultiLoad can use altogether. The valid range is 0 .. (60% of maxampworkertasks).
47. MaxRowHashBlocksPercent	This field defines the maximum percentage of an AMP Lock Table's control blocks that a transaction can utilize for its rowhash locks.
48. MonSesCPUNormalization	This field disables/enables normalization of CPU values in Monitor Session response.
49. TempLargePageSize	This field specifies another method of RSG storage memory pool in addition to the existing temporary storage page size.
50. RepCacheSegSize	This field specifies the size of a cache segment (in kilobytes) that is allocated in each AMP to store EVLobjects,
51. MaxDownRegions	Sets the maximum allowable number of down regions for a subtable. Default is 6.
52. MPS_IncludePEOnlyNodes	Include PE-only nodes in Monitor Physical Summary PM/API response calculation.
53. PrimaryIndexDefault	Enables/disables the behavior of whether PI tables or NoPI tables are created when a CREATE TABLE DDL is used without the PRIMARY INDEX, NO PRIMARY INDEX or PRIMARY KEY or UNIQUE constraints. In this case, a value of D : results in tables created as per Teradata Default (Default Setting). P : results in tables with the first column as the non-unique Primary Index. N : results in tables created without a primary index(as NoPI tables). A value of 'D' is equivalent to a value of 'P'.
54. AccessLockForUncomRead	This field, when enabled imposes access lock on source table(s) of DML statement when transaction isolation level is Read Uncommitted (RU). The default value is FALSE.

DBS Control Record – General Fields

General Fields (cont.)

31. LockLogSegmentSize = 64 KB
32. CostProfileId = 0
33. DBQLFlushRate = 600 (sec.)
34. Memory Limit Per Transaction = 2 pages
35. Client Reset Timeout = 300 (sec.)
36. Temporary Storage Page Size = 4K bytes
37. Spill File Path = /var/tdrsg
38. MDS Is Enabled = FALSE
40. EnableCostProfileTLE = FALSE
41. EnableSetCostProfile = 0
42. UseVirtualSysDefault = 0
43. DisableUDTImplCastForSysFuncOp = FALSE
44. CurHashBucketSize = 20 bits
45. NewHashBucketSize = 20 bits
46. MaxLoadAWT = 0
47. MaxRowHashBlocksPercent = 50%
48. MonSesCPUNormalization = FALSE
49. TempLargePageSize = 64K Bytes
50. RepCacheSegSize = 512K bytes
51. MaxDownRegions = 6 *def*
52. MPS_IncludePEOnlyNodes = FALSE
53. PrimaryIndexDefault = D *def*
54. AccessLockForUncomRead = FALSE

General Fields (New with Teradata 13.10)

55. EnabNonTempoOp = FALSE
56. InclNTforGrntOrRevokAll = FALSE
57. TimeDateWZControl =
58. (Reserved for future use)
59. SysInitRelease = 13100000
60. DefaultCaseSpec = FALSE
61. PMPC_TimeoutSecs = 60 (Sec.)
62. ExportOrderBySegmentLimit = 500
63. MLoadDiscardDupRowUPI = FALSE
64. DBQL CPU/IO Collection = 0
65. No13dot0Backdown = TRUE

General Fields (New with Teradata 14.0)

66. SnapBypassAggrCache = FALSE
67. RedriveProtection = 0
68. RedriveFallbackResponse = FALSE
69. PMPC_SessionRateThreshold = 60 (Sec.)
70. RoundNumberAsDec = FALSE
71. DMLStatementShipping = 0 (Disabled)

DBS Control Record – File System Fields

Field	Purpose
1. FreeSpacePercent	This field is used by the DBS and the File System to determine the percentage of free space to leave on cylinders during data load operations. The valid range of values is 0 .. 75. The default value is 0 (percent). A table definition (CREATE or ALTER) overrides this value.
2. MiniCylPackLowCylProd	Determines the number of free cylinders below which it will start to perform the "MiniCylPacks" operation in anticipation of their need. Setting this field to 0 indicates that "MiniCylPacks" will only be performed when no free cylinders exist. The default value is 10.
3. PermDBSize	Determines the maximum size of a Permanent Table's multi-row Data Blocks in 512-byte sectors. The valid range of values is 14 .. 255. The default value is 127 (sectors). A table definition (CREATE or ALTER) overrides this value.
4. JournalDBSize	Determines the maximum size of Transient Journal and Permanent Journal Table multi-row Data Blocks in 512-byte sectors. The valid range of values is 1 .. 127. The default value is 12 (sectors).
5. DefragLowCylProd	Determines the number of free cylinders below which it will start to perform the "Cylinder defragmentation" operation. Setting this field to 0 disables "Cylinder defragmentation". The default value is 100.
6. PermDBAllocUnit	Determines the allocation unit for Permanent Table's multi-row Data Blocks in units of 512-byte sectors. If a Permanent Table Data Block contains multiple rows, the size of the Data Block will be a multiple of the PermDBAllocUnit. The valid range of values is 1 .. 63. The default value is 1 (sector).
7. Cylinders Saved for PERM	Used to save X number of cylinders for Perm Data only – cannot be used by Spool. Range is 1 to 100; default is 10. If the number of available cylinders falls below this value, spool files will not be allocated space.
8-14. WAL parameters	Parameters to setup options for Write-Aheada Logging. (V2R6.2 feature)
15. Free Cylinder Cache Size	This field is used by the File System to determine how many cylinders are to be managed in File System cache for use as spool cylinders. NOTE - A value of 1 will disable use of the cache. Valid range is 1 - 1000. Default is 100.
16. Bkgrnd Age Cycle Interval -	This field is used by the File System to determine the amount of time that elapses between background cycles to write a subset of modified segments in the cache to disk. This background activity serves to reduce the size of the WAL log and promotes improved disk space utilization in WAL modes. Valid range is 1-240. The default value is 60 (seconds).

DBS Control Record – File System Fields

File System Fields:

1. FreeSpacePercent = 0%
2. MiniCylPackLowCylProd = 10 (free cylinders)
3. PermDBSize = 127 (sectors)
4. JournalDBSize = 12 (sectors)
5. DefragLowCylProd = 100 (free cylinders)
6. PermDBAllocUnit = 1 (sectors)
7. Cylinders Saved for PERM = 10 (cylinders)
8. DisableWALforDBs = FALSE
9. DisableWAL = FALSE
10. WAL Buffers = 20 (WAL log buffers)
11. MaxSyncWALWrites = 5 (MaxSyncWalWrites)
12. SmallDepotCylsPerPdisk = 2 (cylinders)
13. LargeDepotCylsPerPdisk = 1 (cylinders)
14. WAL Checkpoint Interval = 60 (seconds)
15. Free Cylinder Cache Size = 100 (number of cylinders)
16. Bkgrnd Age Cycle Interval = 60 (seconds)
17. DisableAutoCylPack = FALSE (Enabled)
18. AutoCylPackColddata = FALSE (Disabled)
19. AutoCylPackFSP = 10 (Percent; Default value)
20. AutoCylPackStyle = 2 (PACK / UNPACK)
21. AutoCylPackThresh = 5 (Percent; Default value)
22. AutoCylPackFreeCylThresh = 20 (Cylinders; Default value)
23. AutoCylPackInterval = 60 (Seconds; Default value)
24. AutoCylPackIOThresh = 1 (Number; Default value)
25. (Reserved for future use)
26. (Reserved for future use)
27. MergeBlockRatio = 60 (Percent; Default value)
28. DisableMergeBlocks = FALSE (Enabled)

* Parameters 8 to 14 apply to WAL
* Parameters 15-16 apply to TD 13.0
* Parameters 17-28 apply to TD 13.10

DBS Control Record – Performance Fields

Field	Purpose
1. DictionaryCacheSize	This field defines the size of the dictionary cache for each PE Vproc on the system. The valid range of values is 64 .. 16384. 64-bit system default is 3072 (kilobytes).
2. DBSCacheCtrl	Enable or disable the performance enhancements associated with Cache Control Page-Release Interface. FALSE cause old caching rules to be used. With old cache rules, data blocks added to spool tables or used in sort operations are NOT cached. With old cache rules, data blocks for user tables are always cached.
3. DBSCacheThr	Specifies the percentage value to use to calculate the cache threshold when the DBSCacheCtrl field is enabled. The valid range of values is 0 .. 100. The default value is 10.
4. MaxParseTreeSegs	This field defines the maximum number of 64KB tree segments that the parser will allocate while parsing a request. The valid range of values is 12 .. 12000. 64-bit sys default is 2000.
5. ReadAhead	Enable or disable the performance enhancements associated Read-Ahead Sequential File Access Workload. The default value is TRUE.
6. StepsSegmentSize	Defines the maximum size of the plastic steps segment. Range of values is 64 – 1024 Kbytes. Default is 1024.
7. RedistBufSize	This field determines the size in units of kilobytes of hashed row redistribution buffers, subject to certain adjustments.On systems with few virtual AMP's, a larger buffer size will usually have a positive effect on performance. However, on systems with many virtual AMP's, making the buffer size too large will cause excessive memory consumption, especially if many queries involving hashed row redistribution are run concurrently. The range of valid values is 1 to 63. The default value is 4.
8. DisableSyncScan	Enables or disables the performance enhancements associated with synchronized full table scans.
9. SyncScanCacheThr	SyncScanCacheThr - This field specifies the percentage of file system (FSG) cache that the Teradata file system can assume is available for synchronized scans of tables. It does not reserve cache for this purpose. Instead, this value specifies the amount of permanent data the Teradata file system should try to retain in memory at any one time for all tables involved in synchronized scans. Whether that much memory is truly available depends on actual workload. The valid range of values is 0 .. 100, where 0 indicates that the default value should be used. The default value is 10.
10. HTMemAlloc	Specifies the percentage of memory to be allocated to a hash table for a hash join. A value of 0 turns off hash joins. Valid range is 0 to 10. The default value is 0.
11. SkewAllowance	Specifies a percentage factor used by the optimizer in deciding on the size of each hash join partition. It makes allowance for data skew in the build relation. Valid range is 20 to 80. The default value is 75.
12. Read Ahead Count	ReadAhead Count - If the ReadAhead field is set TRUE, ReadAhead Count is used to specify the number of data blocks that will be preloaded in advance of the current file position while performing sequential scans. The valid range of values is 1 .. 100. The default value is 1.
13. PPICacheThrP	This field specifies the percentage value to be used for calculating the cache threshold used in operations dealing with multiple partitions. Valid range is 0 to 500. The default is 10.
14. ReadLockOnly	ReadLockOnly – This field is used to disable or enable the special read-or-access lock protocol on DBC.AccessRights table during access rights validation and on dictionary tables accessed by read-only queries during request parsing. The default value is FALSE. FALSE enables this feature.
15. IAMaxWorkloadCache	This tunable field defines the maximum size in megabytes of the workload cache, which is used during index analysis. Valid range is 32 to 128. The default value is 32 (megabytes).
16. MaxRequestsSaved	This field governs the number of request cache entries allowed on a PE. The default number of entires that can be saved in the cache is 600. The user can modify this field to a fixed value that ranges from 300 to 2000 and, is a multiple of 10.
17. UtilityReadAheadCount	Specifies the number of data blocks the Teradata utilities will preload at a time while it performs its sequential scan. The utilities uses this field instead of the ReadAhead and ReadAhead Count fields. The valid range of values is 1 .. 100. Default is 10 blocks.
18. StandAloneReadAheadCount	Specifies the number of data blocks to preload when File System startup or a utility runs as a standalone task. The valid range of values is 1 .. 100. Default is 20 blocks.
19. DisablePeekUsing	This field enables or disables the performance enhancements associated with exposed using values. A value of F enables the feature. A value of T disables the feature. The default is F.
20. IVMaxWorkloadCache	This tunable field defines the maximum size in megabytes of the workload cache used by Index Wizard Validation. Valid range is 1 to 32. The default value is 1 (megabytes).

DBS Control Record – Performance Fields

Performance Fields:		Notes
1. DictionaryCacheSize	= 3072	*Size of data dictionary cache for each PE*
2. DBSCacheCtrl	= TRUE	
3. DBSCacheThr	= 10%	
4. MaxParseTreeSeg	= 2000	*Raise for more complex SQL requests*
5. ReadAhead	= TRUE	
6. StepsSegmentSize	= 1024 (kilobytes)	*Maximum size of the plastic steps segment in KB*
7. RedistBufSize	= 4 (kilobytes)	
8. DisableSyncScan	= FALSE	
9. SyncScanCacheThr	= 10%	
10. HTMemAlloc	= 10%	
11. SkewAllowance	= 75%	
12. Read Ahead Count	= 1	
13. PPICacheThrP	= 10	
14. ReadLockOnly	= FALSE	*(FALSE enables this feature)*
15. IAMaxWorkloadCache	= 32 (megabytes)	
16. MaxRequestsSaved	= 600 (default)	
17. UtilityReadAheadCount	= 10	
18. StandAloneReadAheadCount	= 20	
19. DisablePeekUsing	= FALSE	*(new in 12.0)*
20. IVMaxWorkloadCache	= 1 (megabytes)	*(new in 12.0)*
21. RevertJoinPlanning	= FALSE	*(new in 13.0)*
22. MaxJoinTables	= 128	*(new in 13.0)*
23. DBQLXMLPlanMemLimit	= 8192 (KB; Def.)	*(new in 13.10)*
24. LimitInlistCVal	= 1048576	*(new in 13.10)*
25. NumStatisticsCacheSegs	= 4 (SizeofSegment is 1024 KB)	*(new in 14.0)*

Modifying DBS Control Parameters

The **modify** command is used to change DBS Control parameters. After making a modification, it is necessary to **write** the update to disk.

Example: Set the Century Break value

Another example (different than the facing page) follows. To change the Century Break value, you need to use modify and write commands of the DBSControl utility.

To change the Century Break value:

1. From the command-line prompt (or the Supervisor window of the Database Window):

   dbscontrol

2. From the DBS Control window:

   display general

3. Use the modify command to change flag 14:

   modify general 14 = 50

4. Write changes to the GDO:

   write

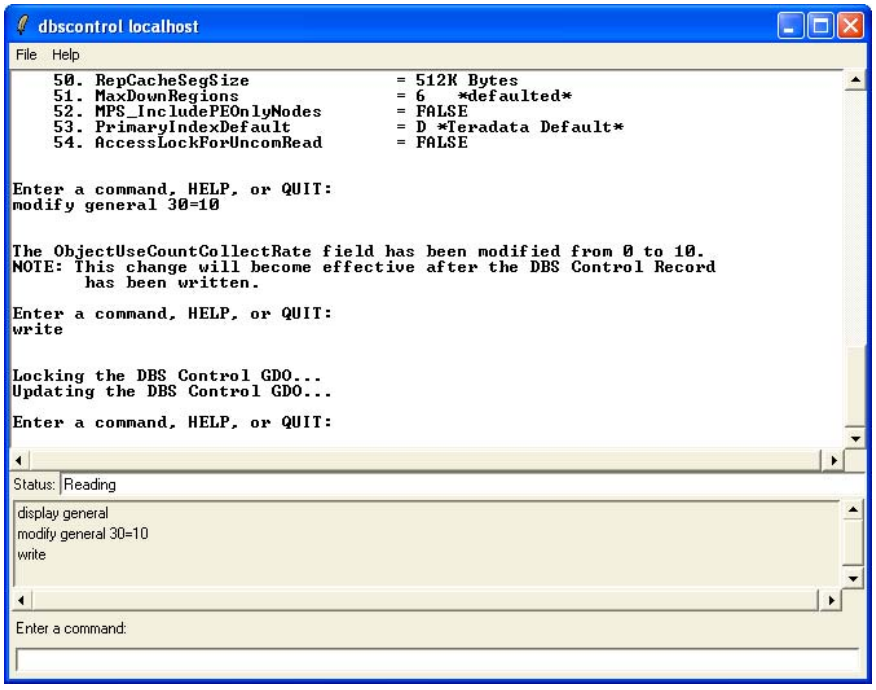
Note: The change does not take place until the next database restart, even though the flag shows the change right away. If you DISPLAY GENERAL, you'll see the new setting, but the setting is not really available until the system is restarted.

Modifying DBS Control Parameters

To modify a DBS Control parameter, use the **Modify** and **Write** commands.

This example changes parameter 30 to a value of 10 minutes.

To save the changes, use the Write command.



Ferret – Defragment and Packdisk

SCOPE Command

The SCOPE command defines the range of tables and/or vprocs to display or reconfigure with the Defragment and Packdisk commands.

The facing page has an example of the SCOPE and PACKDISK commands.

DEFRAGMENT Command

Over time, it is possible that INSERTs and DELETEs can cause cylinders to become fragmented. If this is the case, the DEFRAGMENT command may be used to defragment the cylinders on an AMP (or the system) depending on SCOPE options.

PACKDISK Command

The PACKDISK command alters a disk to reconfigure the cylinders within the scope defined by the SCOPE command. PACKDISK uses the default Free Space Percent or a new percentage specified as part of the command to pack the entire disk or a single table.

The allowable scope for PACKDISK is vprocs or tables, but not both.

The system will automatically perform mini-cylpacks when the number of cylinders falls below a certain internal threshold value. The PACKDISK command can be used to force this situation.

Starting PACKDISK

PACKDISK is a command within the Ferret utility. To start PACKDISK, enter **packdisk fsp = nnn** (where fsp = free space percent and nnn equals the percentage of cylinder free space) in the command window of the Ferret partition. Key the command in uppercase, lowercase or a combination of both. Note the interactive area where the utility has been started.

Stopping PACKDISK

To terminate the PACKDISK command, enter **ABORT**.

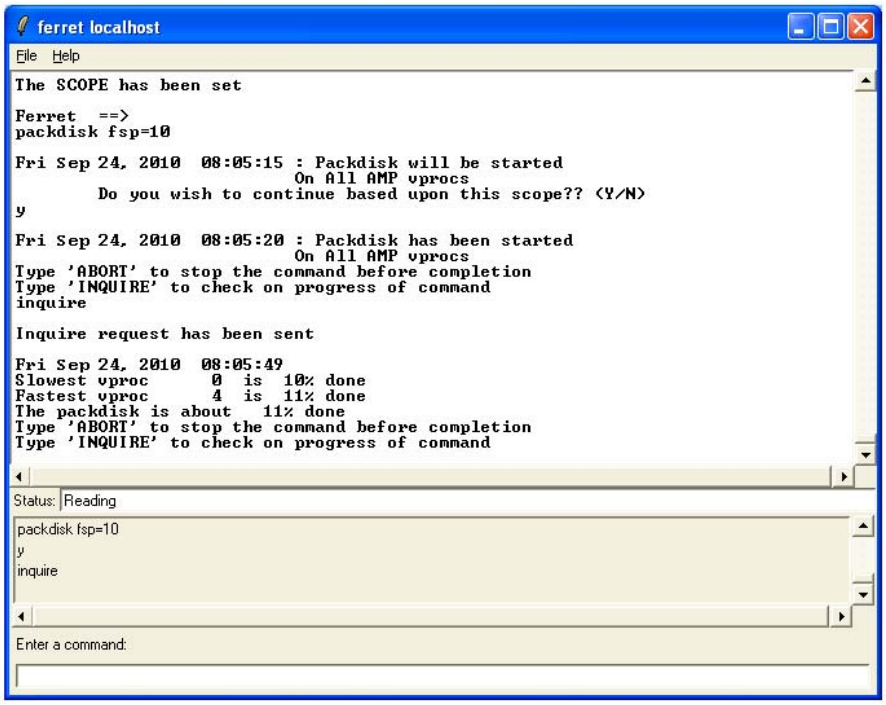
Ferret – Defragment and Packdisk

DEFRAGMENT

combines free sectors and moves them to the end of a cylinder.

PACKDISK

fill (or packs) cylinders up to the FSP (Free Space Percentage).



Checking Data Integrity

There are two utilities that are used to check data consistency.

- SCANDISK - checks the AMP's file system structures (CIs and DBs for consistency)
- CHECKTABLE - checks for consistency in internal data structures such as table headers, SI subtables, row identifiers, etc.

Checking Data Integrity

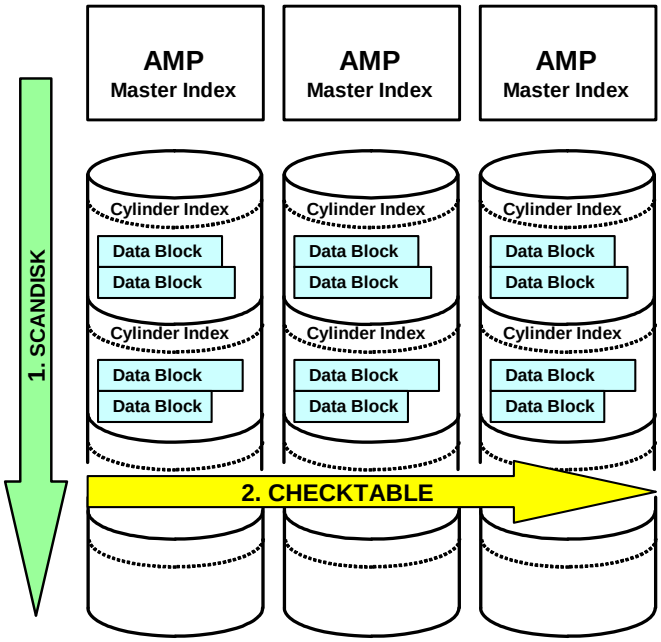
SCANDISK

Checks the AMP's file system structures (CIs and DBs for consistency)

CHECKTABLE

Checks for consistency in internal data structures such as table headers, SI subtables, row identifiers, etc.

Typically, first execute SCANDISK, then CHECKTABLE.



Ferret – Scandisk Utility

The SCANDISK utility/command enables you to determine if there is a problem with the AMP file system and assess its extent. SCANDISK is a diagnostic tool designed to check for inconsistencies between key file system data structures such as the master index, cylinder index, and data blocks.

As an administrator, you can perform this procedure as preventative maintenance to validate the file system, as part of other maintenance procedures, or when users report that there are file system problems.

Execute the SCANDISK command in the Ferret utility while the system is operational.

The SCANDISK command:

- Verifies data block content matches the data descriptor.
- Checks that all sectors are allocated to one and only one of the following:
  - Bad sector list
  - Free list
  - A data block
- Ensures that continuation bits are flagged correctly.

If Scandisk discovers a problem with a disk, you must use the Table Rebuild utility to rebuild any tables it reports as having bad data for the particular AMP vproc. (The Table Rebuild utility is discussed later in this lesson.) The output of the SCANDISK command is displayed on the screen directly after the command completes.

To avoid potential TPA resets, run SCANDISK prior to:

- Running the Checktable utility
- Rebuilding database tables using the Table Rebuild utility

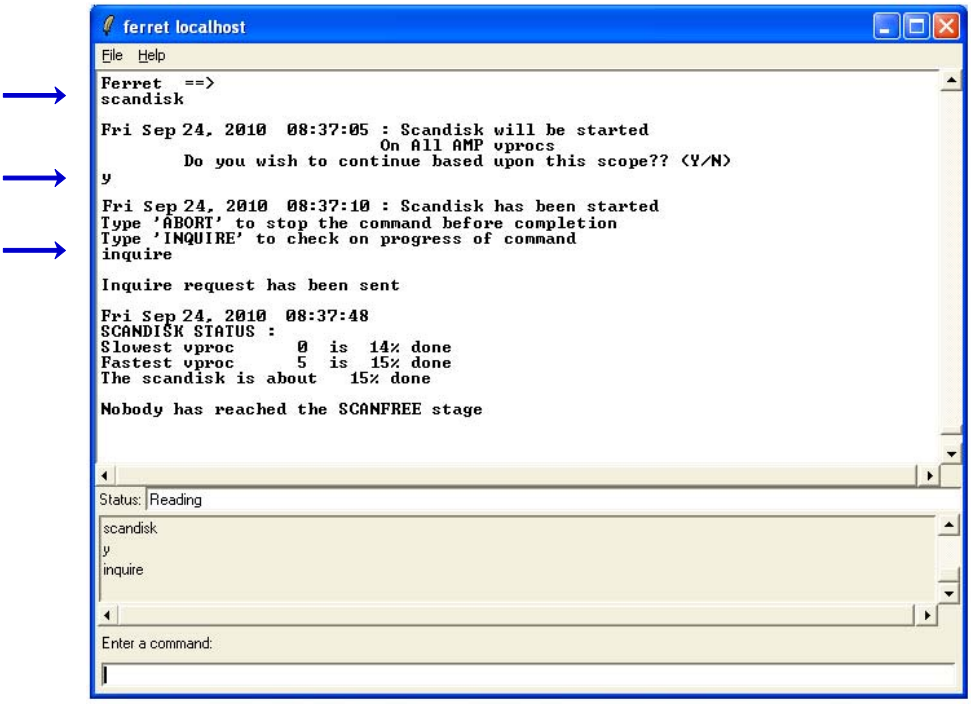
Starting Scandisk

Enter **start Ferret** and from within the Ferret utility window, enter the command **Scandisk**. The SCANDISK command may be limited by the SCOPE command to scan only one table, a range of tables, or the whole vproc.

Stopping Scandisk

Scandisk terminates itself after performing the scan.

Ferret – Scandisk Utility



If Scandisk encounters an error for a specific AMP, execute (table) REBUILD for that AMP.

Checktable Utility

Checktable is a diagnostic tool designed to check for inconsistencies in internal data structures such as table headers, row identifiers, and secondary indexes. Checktable can help determine if there is corruption in your system. Normally, Checktable is executed on a system that is quiescent.

Use the Checktable utility as both a diagnostic and validation tool. As a diagnostic tool, you can identify problems with data integrity. As a validation tool, you can verify data integrity prior to a reconfiguration or archive. Checktable only identifies inconsistencies; it does not correct them.

Always run Scandisk before you run Checktable. Checktable assumes the underlying structure of the file system is intact. If there are structural errors, Checktable could cause a tpa reset on the database. Scandisk is located within the Ferret utility.

The estimated run time for a Checktable varies depending on the characteristics of the data. The more non-unique secondary indexes defined on the tables, the longer it takes to run Checktable. If you invoke Checktable when users are logged on, the time it takes to process the Checktable will depend on the activity on the system and the amount of resource contentions that it encounters (for example, object locks).

Starting Checktable

To start the utility, enter **start Checktable** from the DBW supervisor.

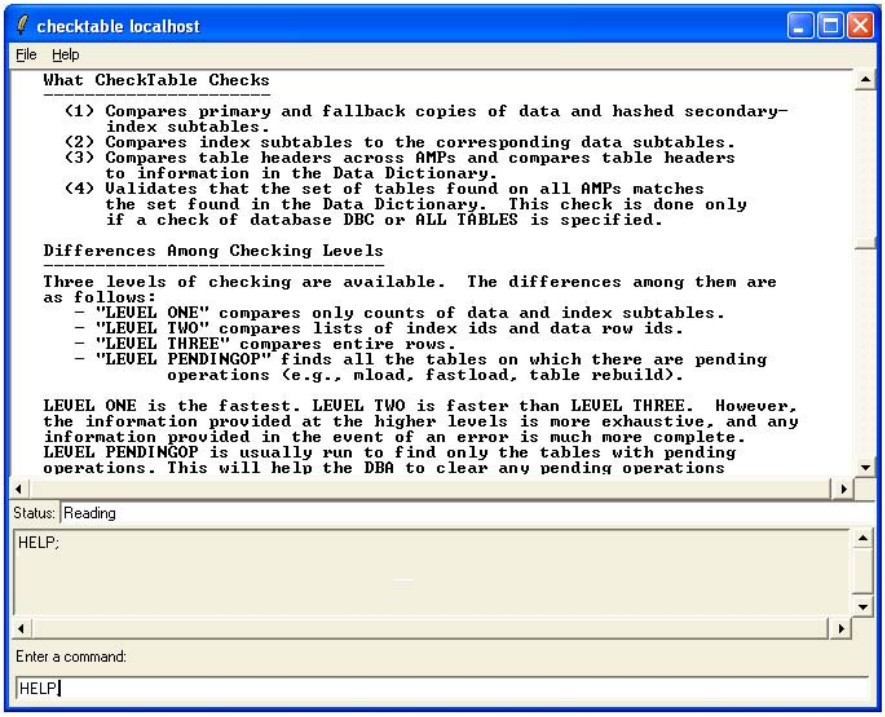
Stopping Checktable

To stop Checktable, enter **QUIT;**

Checktable Utility

Three levels of typical checking.

Checktable Command



From supervisor of DB Window, START CHECKTABLE

Checktable – Levels of Checking

The Checktable utility provides more than three levels of checking. The first three levels of checking are typically used by customers. Each level is a superset of the lower levels and runs all previous level checks.

Level One Level one checking compares the counts of data and index subtables. Use level-one checking to identify specific tables that contain errors. If errors are detected, perform a more detailed check using level-two or level-three checking.

Level Two Level two compares lists of index and row IDs as well as primary to fallback checksums. This level also verifies that hash codes reflect correct row distribution in any given subtable. Level two checking requires significantly more system resources than level one. You can use spool space to check tables that have secondary indexes.

Level Three Use level three checking to obtain detailed diagnostic information. This level provides the most detailed check and requires the most system resources. Level three compares entire rows, byte by byte.

Teradata Recommendations

Teradata recommends that you perform the following maintenance routine once a month:

1. Run a Scandisk diagnostic for all Vdisks. Scandisk performs intra-disk integrity checks by determining that the underlying file system is intact. Users may want their field support representative to start this task.

2. Follow Step 1 with a Checktable run at Level 2. The Checktable utility completes the diagnostic analysis with inter-disk integrity checks, according to the rules of the database system.

Checktable – Levels of Checking

Checktable provides three levels of checking that are typically used:

Level 1 Compares only the counts of data and index subtables.

Level 2 Includes level 1 checking; also compares lists of index and data row ids/keys and primary to fallback checksums.

Level 3 Includes levels 1 and 2 checking; also compares entire rows.

While Checktable is running, you may use the following function keys:

F2 Displays current status.

F3 Aborts the current table check and continues with the next.

F4 Aborts the current Checktable command.

F7 Help

It is recommended that you periodically (e.g., monthly) schedule the following maintenance routines:

- **A SCANDISK diagnostic run for all AMPs. This function performs intra-disk integrity checks. Your Customer Engineer can start this diagnostic tool.**
- **A CHECKTABLE run at level 2. Checktable completes the diagnostic analysis with inter-disk integrity checks.**

Checktable – Example

The facing page shows the output from executing Checktable against several databases. Additional options in Checktable are:

Error Limit for a Check Command

The error limit is the maximum number of errors that can be found during checking a table. If the number of found errors exceeds the error limit, Checktable stops checking on the current table but continues to check the next table.

To by-pass tables which are locked.

The SKIPLOCKS option is intended to help the user by-pass any contention on tables. Without this option Checktable will block indefinitely on the table to be checked until it has been unlocked. When this option is specified Checktable will automatically skip the in-use (locked) tables.

To check database(s) or table(s) in serial or parallel.

SERIAL/PARALLEL mode allows the user to specify whether the Checktable utility should check the specified databases/tables in SERIAL mode or in PARALLEL mode. Default mode <checkmode> is SERIAL mode.

In SERIAL mode, the Checktable utility checks a single table at a time.

In PARALLEL mode, the Checktable utility checks the specified database(s)/table(s) in parallel. The number of tables that can be checked simultaneously in parallel depends on the resource availability. A status command can be used to determine the number of parallel checks being performed at any given point of time.

To control resource consumption

The CheckTable utility runs with MEDIUM priority by default. The user has an option to specify a performance group name with the command. Checktable will then run in the priority for the performance group name. The user can also specify LOW, MEDIUM, HIGH, or RUSH priority explicitly.

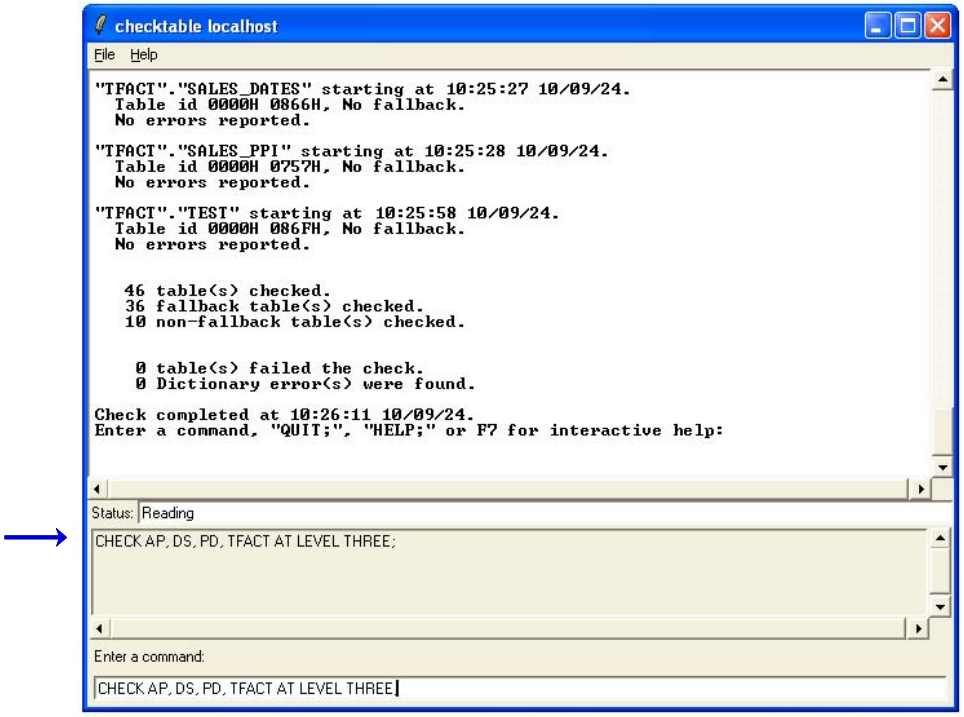
Specify the first character L (=LOW), M (=MEDIUM), H (=HIGH) or R (RUSH) when using PRIORITY option.

This option can be used to control the resource consumption by the Checktable utility.

To Specify Indexes or Large Objects

Use BUT ONLY or BUT NOT to include or exclude data subtables in the check.

Checktable – Example



CheckTable Command: CHECK AP, DS, PD, TFACT AT LEVEL THREE;

Table Rebuild Utility

Table Rebuild is a utility that repairs data corruption. It does so by rebuilding tables on a specific AMP vproc based on data located on the other AMP vprocs in the fallback cluster.

Table Rebuild can rebuild data in the following subsets:

- The primary or fallback portion of a table
- An entire table (both primary and fallback portions)
- All tables in a database
- All tables that reside on an AMP vproc

Table Rebuild performs differently based on the type of table (e.g., fallback or not, etc.)

Type of Table	Action
Fallback tables	Delete the table header (one row table which defines a user data table) Delete specified portion of the table being rebuilt Rebuild the table header Rebuild the specified portion of the table
Non-fallback tables	Delete the table header Rebuild the table header
Permanent journal tables	Delete data for the table being rebuilt Rebuild the table header Locks the table (“pending rebuild”) **Note:** You must restore non-fallback data.

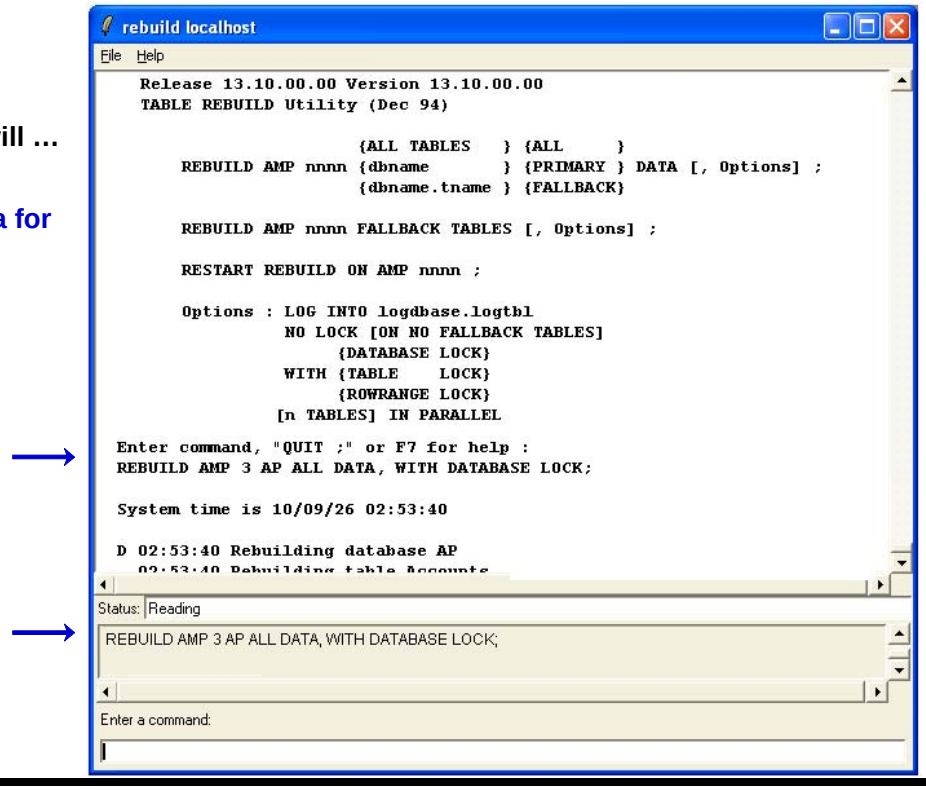
Note: You can start REBUILD when the failed AMP has been fixed and brought back to the OFFLINE state.

The command syntax is shown on the facing page.

Table Rebuild Utility

For an AMP that has failed (e.g., a Vdisk failure) and has been repaired, REBUILD will …

- **Rebuild table headers and data for fallback tables.**
- **Only build table headers for non-fallback tables.**



Recovery Manager Utility

The Rcvmanager (Recovery Manager) utility lets you monitor the backing out of incomplete transactions on tables that may be locked for access. The resulting journal is called the Transaction Journal. Rcvmanager also shows the count of records of data presently in the Down AMP Recovery Journal. This journal represents the data rows an AMP vproc must recover from the other AMP vprocs in the cluster before coming back online to the database.

The Recovery Manager utility runs only when the system is in one of the following states:

- Logon
- Logon/Quiet
- Logoff
- Logoff/Quiet
- Startup (if system has completed voting for transaction recovery).

If the system is not in one of the above states, Recovery Manager will terminate immediately after you start it.

Starting Rcvmanager

To start Recovery Manager, enter **start rcvmanager** in the Supervisor interactive area.

Stopping Rcvmanager

After you start Recovery Manager, you cannot stop it with the Supervisor program STOP command.

You must use the Rcvmanager **QUIT;** command to stop the program.

Note: All Rcvmanager commands end with a semicolon (;).

Recovery Manager Utility

RCVMANAGER provides a means for the user to interact with the recovery subsystem.

Following a Teradata restart, you can ...

- **View the number of rows being rolled back via the Transient Journal (TJ).**
- **View the number of rows being updated on an AMP via the Recovery Journal (Fallback tables and rows that were updated while the AMP was out of service).**
- **List the tables that are locked until the rollback completes.**
- **Change the priority of Rollback and/or Recovery operations.**

Other options …

- **View which tables are being rolled back.**
- **Set the rollback priority for a specific session to a specific performance group.**
- **For a table or tables, cancel rollback processing for an online user requested abort or following a system restart.**
  - **WARNING: The target table will be unusable after this command is issued.**
- **View the tables for which rollback processing is not pending cancellation during the online transaction recovery.**
  - **The table is removed from the list when no more TJ rows exist for the table.**

Recovery Manager Commands

The **LIST STATUS** command displays information about recovery operations in progress. The processor id option provides additional detailed information about a specific down AMP.

The **LIST LOCKS** command displays a list of all locks currently held by online transaction recovery.

The **LIST ROLLBACK TABLES** command displays the list of tables which are currently being rolled back in the system. Separate listings are generated to distinguish between online transaction recovery and system recovery. Table ids can be selected from this listing for executing the CANCEL ROLLBACK ON TABLE command. In case a '*' character follows the table names then they cannot be specified in the CANCEL ROLLBACK ON TABLE command.

The **LIST CANCEL ROLLBACK TABLES** command displays the list of tables for which rollback processing is pending cancellation during the online transaction recovery. These tables are removed from the list when all the journal rows corresponding to the tables have been skipped on all the AMPs.

The **PRIORITY** command can be used to either display or set the current priorities for rebuild or recovery. For example:

RECOVERY PRIORITY; displays the current recovery priority setting.
RECOVERY PRIORITY LOW; sets the recovery priority to Low.
DEFAULT PRIORITY; sets the recovery priority to Low and the rebuild priority to Medium.

The **CANCEL ROLLBACK ON TABLE** command is used to specify a table for which rollback processing is to be cancelled for an online user requested abort or during system recovery. The DBC password is required to execute this command. Multiple tables can be specified by separating their table ids with commas. WARNING: The target table will be unusable after this command is issued, and will become usable only when the table is dropped and created again, or when the table is restored from an archived backup, or when a DELETE ALL operation is performed on that table. The CANCEL ROLLBACK command should only be used in cases where the rollback will take longer than the restore of the table, or in cases where the table is unimportant (i.e., a temporary table). A single table retrieve operation can be performed on the target table by using the READ OVERRIDE locking modifier on it.

The **ROLLBACK SESSION... PERFORMANCE GROUP** command can be used to either display or set the current performance group of the rollback for a particular session. The priority associated with the specified performance group is used to change the priority of the rollback for the specified host-id and session number.

Recovery Manager Commands

From supervisor: START RCVMANAGER

Commands are:

- **LIST STATUS [<proc-id>] ; shows status of transaction and/or down AMP recovery**
- **LIST LOCKS; displays all locks currently held by online transaction recovery**
- **REBUILD PRIORITY [Low | Medium | High] ; sets the table rebuild priority**
- **RECOVERY PRIORITY [Low | Medium | High] ; sets the AMP recovery priority**
- **DEFAULT PRIORITY; sets both priorities back to their default**
- **HELP;**
- **QUIT;**

Other commands include:

- **LIST ROLLBACK TABLES; – view the tables being rolled back for which rollback processing is not pending cancellation**
- **LIST CANCEL ROLLBACK TABLES; view the tables that are pending cancellation**
- **CANCEL ROLLBACK ON TABLE <table-id> [{, <table-id>} ...] ;**
- **ROLLBACK SESSION <host>, <session> PERFORMANCE GROUP [<Perf Group Name>];**

Rcvmanager – List Status

The Recovery Manager utility uses two basic commands: the LIST STATUS and LIST LOCKS commands. These commands display information about online transaction recovery and offline AMP recovery.

The LIST STATUS command generates two reports:

- ONLINE TRANSACTION RECOVERY JOURNAL
- DOWN AMP CATCHUP JOURNAL

ONLINE TRANSACTION RECOVERY JOURNAL

This report pertains to online transaction recovery and displays a list of all active recovery sessions as well as the maximum number of transaction journal rows remaining to be processed for the AMP that has this maximum count. Since all AMPs must complete the processing of a given recovery session before the processing of the next session begins, this information is sufficient to compute the worst-case count of transaction journal entries to be scanned.

The online transaction recovery journal counts are updated by each AMP every time a checkpoint is taken. Thus, every time an AMP checkpoints, the system will decrement its online transaction recovery journal count by 1000 and a later LIST STATUS command may display different results. If there are no recovery sessions active, then the report displays only the title.

The entries in this report are:

Recovery Session	ID of active recovery session
Count	Maximum number of transaction journal rows remaining to be processed for a specific AMP
AMP W/Count	The AMP to which the corresponding count applies

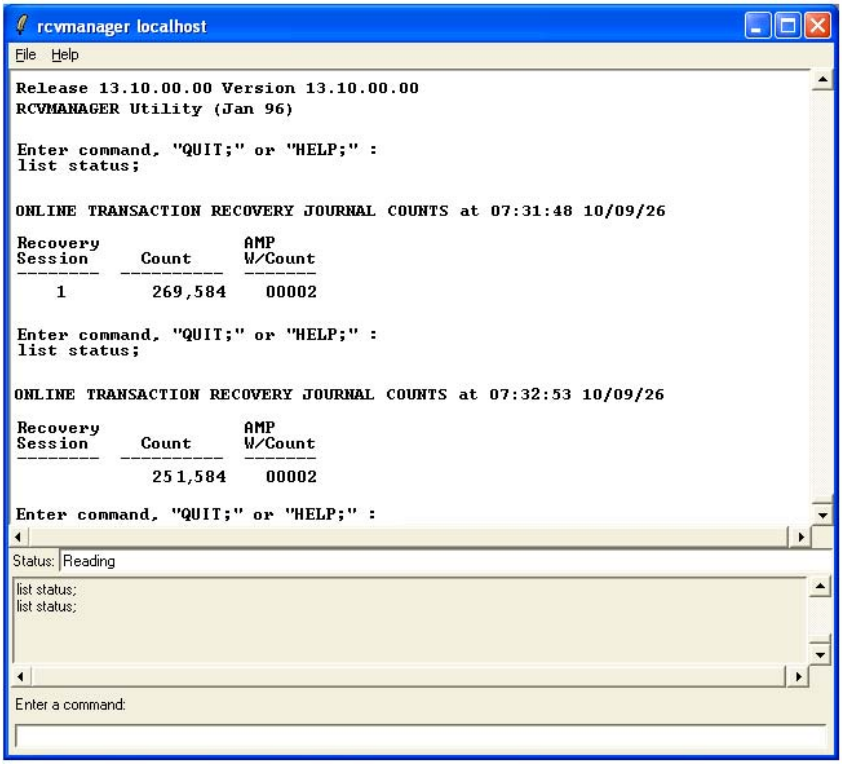
Rcvmanager – List Status

After a Teradata restart, the list status command is used.

In this example, there are almost 270K TJ rows remaining to be rolled back.

The list status command is used a second time and the count has decreased.

Note: The AMP with the highest count of TJ rows to roll back is listed.



Rcvmanager – List Locks

The screen display on the facing page continues the same example and has an example of using the List Locks command.

The LIST LOCKS command displays all locks currently held by transaction recovery. The command generates a singe report called, *Locks Held By Online Transaction Recovery.*

The report is sorted alphabetically by object name. The report does not display information for row range and row hash locks, but does display the table within which the row resides. If Recovery Manager is unable to determine the database name associated with an object, then it displays the database ID in decimal and hexadecimal. The same is true if the table name cannot be determined.

Note: LIST LOCKS displays only those locks currently held by online transaction recovery. Currently, there is no way to display locks held by offline catchup.

Rebuild & Recover Priority

The Rcvmanager utility also includes the PRIORITY command that can used to specify priorities for:

- Table rebuild operations
- System recovery operations

Both operations are independent of each other. For either operation, if you do not explicitly set a recovery priority, the system uses the default priority. If you do not enter a new priority, the current priority setting displays. The system saves the priority settings for both operations in the Recovery Status system table.

The REBUILD PRIORITY command applies to any Table Rebuild started from the console, automatic table rebuild due to disk error recovery and MLOAD rebuild of target tables for non-participant online AMPs.

The RECOVERY PRIORITY command enables you to set a priority for the system recovery operation.

The syntax is:

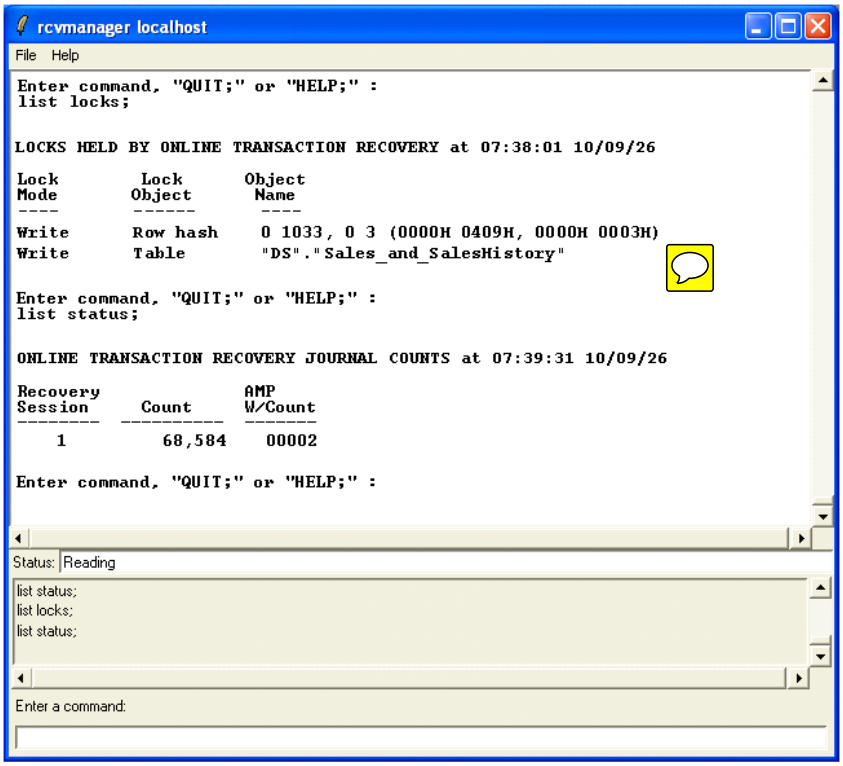
REBUILD PRIORITY HIGH | MEDIUM | LOW ;
RECOVERY PRIORITY HIGH | MEDIUM | LOW ;
DEFAULT PRIORITY ;

Sets REBUILD PRIORITY to MEDIUM and RECOVERY PRIORITY to LOW

Rcvmanager – List Locks

To determine which tables are locked as part of "Online Transaction Recovery", the list locks command is used.

The list status command is used a third time and the count has decreased even more.



Rcvmanager – List Status (2nd Example)

AMP CATCHUP JOURNAL COUNTS

This report pertains to offline AMP recovery and displays an entry for every offline AMP. Entries include the following:

AMP to be caught up

Designates which AMP needs to be recovered.

Asterisk (*) indicates that the AMP would be brought into **online catchup** if a restart were to occur (CJ < 3000 and the OJ/TJ = 0).

AMP Status

Describes the recovery mode now in progress for the recovering AMP:

NOT IN RECOVERY — AMP is still down, and has not come up into recovery yet.

OFFLINE CATCHUP -- AMP is offline, and is in catchup mode
.
ONLINE CATCHUP — The AMP is still catching up, but it is online and will accept new transactions.

Changed Row Journal Count

The number of rows updated in the cluster while an AMP was down. Usually DML changes that impact a row on a single AMP.

Ordered System Change Journal

The number of system or table level changes in the cluster. Usually DDL changes that impact row(s) on all of the AMPs.

Transient Journal Count

Number of transaction journal entries (rows) remaining in all recovery sessions on the recovering AMP:

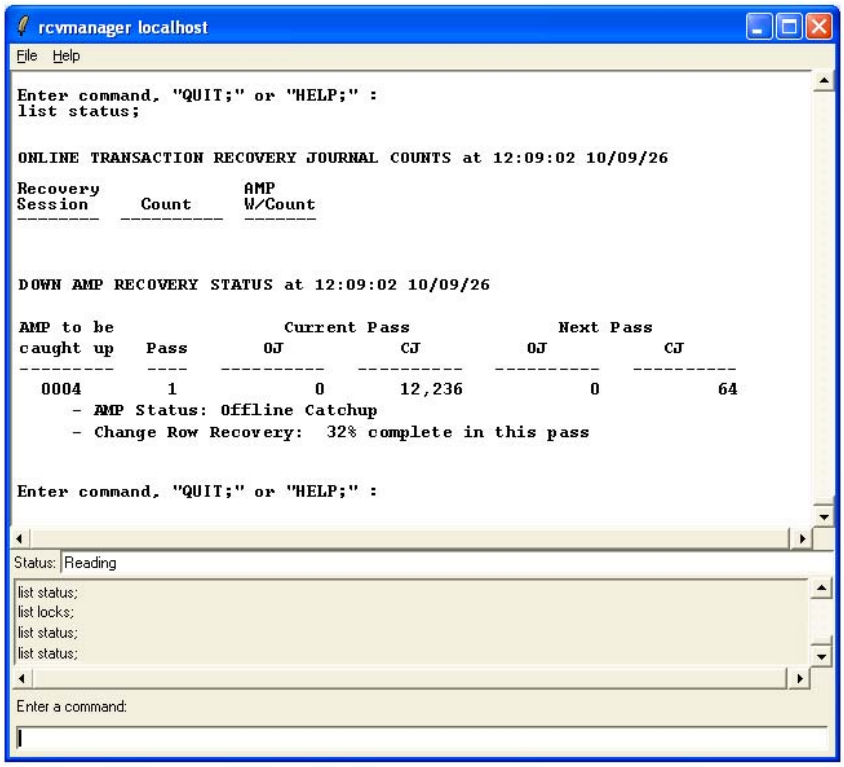
N/A - AMP is not available
0 - Local transaction recovery completed

Rcvmanager – List Status (2nd Example)

If an AMP has been offline, use the **list status** command to determine if fallback table rows are being updated from the down AMP recovery journal.

OJ (Ordered System Change Journal) – DDL changes to fallback tables.

CJ (Changed Row Journal) – DML changes to fallback tables.



Rcvmanager – List Rollback Tables

The **LIST ROLLBACK TABLES** command displays the list of tables which are currently being rolled back in the system. Separate listings are generated to distinguish between online transaction recovery and system recovery.

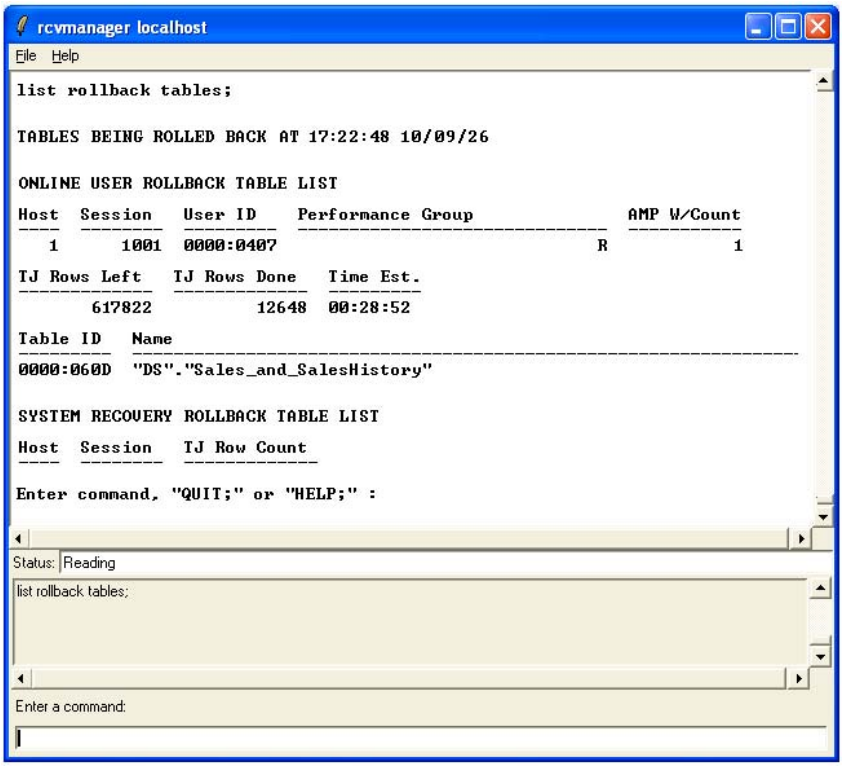
Table Ids can be selected from this listing for executing the CANCEL ROLLBACK ON TABLE command. In the situation where a '*' character follows the table names, then they cannot be specified in the CANCEL ROLLBACK ON TABLE command.

Rcvmanager – List Rollback Tables

In this example, the List Rollback Tables lists a table that is being rolled back because of a user aborted transaction.

The number of rows remaining to be rolled back and the estimated time for rollback is provided.

The Table ID and table name are also provided.



Rcvmanager – Cancel Rollback on Table

The **CANCEL ROLLBACK ON TABLE** command is used to specify a table for which rollback processing is to be cancelled for an online user requested abort or during system recovery. The DBC password is required to execute this command. Multiple tables can be specified by separating their table ids with commas.

The CANCEL ROLLBACK command should only be used in cases where the rollback will take longer than the restore of the table, or in cases where the table is unimportant (i.e., a temporary table). A single table retrieve operation can be performed on the target table by using the READ OVERRIDE locking modifier on it.

WARNING: The target table will be unusable after this command is issued, and will become usable only when the table is dropped and created again, or when the table is restored from an archived backup, or when a DELETE ALL operation is performed on that table.

For example,

SELECT * FROM Sales_and_SalesHistory;

Error 7562: Invalid operation on table Sales_andSalesHistory

DELETE Sales_and_SalesHistory ALL:

Completed: 483, 350 rows processed (command executes successfully)

Rcvmanager – Cancel Rollback on Table

CANCEL ROLLBACK should only be used when …

- rollback will take longer than a restore of the table
- or the table is unimportant (i.e., a temporary table)

Is only executed by the Support Center!!

You will prompted for the DBC password.

After a cancelled rollback, the table is unusable.

SELECT * FROM …;

- **Error 7562**

You can …

- **DROP TABLE tbname;**
- **DELETE tbname ALL;**

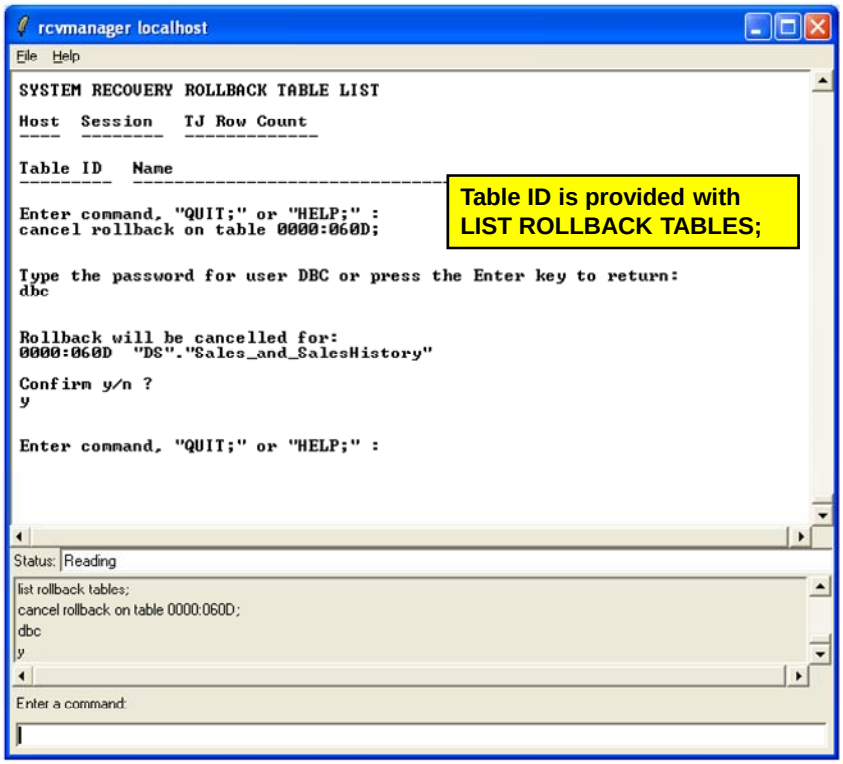


Table ID is provided with LIST ROLLBACK TABLES;

Showlocks Utility

The Showlocks utility enables you to retrieve information about host utility locks the ARC utility places on databases and tables during backup or restore activities. This utility also displays information about locks placed during a system failure.

Host utility locks may interfere with application processing and are normally released after the utility process is complete.

If locks interfere with application processing, you can remove them by invoking the RELEASE LOCK statement. It is available through the ARC utility or as an SQL statement. An individual session may be in a "blocked" state due to one of the following situations:

- An ARC operation failed and a database cannot be accessed
- Locks were not implicitly or explicitly released after an ARC operation
- A lock was not released by the user after a system failure occurred

Showlocks can be started from DB Window Supervisor screen or HUTCNS console.

Supervisor – **start showlocks**
HUTCNS – **LOCksdisplay** (entering LOC is all that is needed)

Report Contents

The utility displays the following information for each utility lock:

- Database name that contains lock
- Table name that contains lock (if applicable)
- User name of user who placed the utility lock
- Lock mode (read, write, exclusive, access)
- Read = Dump
- Write = Roll
- Exclusive = Restore/Copy
- Access = Group read lock or Checkpoint
- ID of vproc (all AMPs when lock resides on all AMPs)

If an object has more than one utility lock, Showlocks provides information for the most restrictive lock placed on the object.

Showlocks Utility

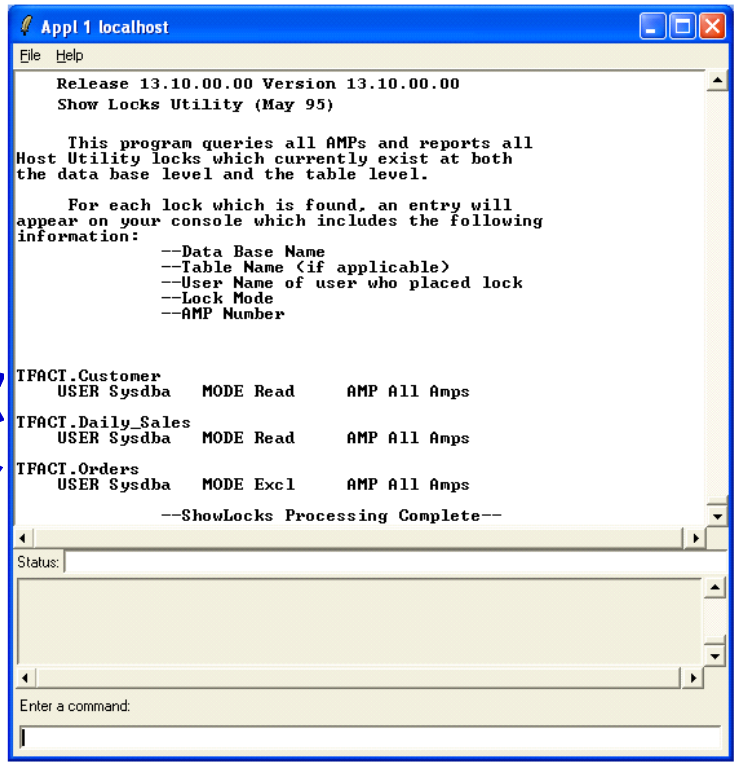
SHOWLOCKS displays "host utility" (HUT) locks that ARC has placed on databases and tables during an archive, restore, or recovery operation.

From supervisor: START SHOWLOCKS

The ARCHIVE command places a READ lock on database/table.

The RESTORE/COPY command places an EXCLUSIVE lock on database/table.

This utility displays the host utility locks that are present and terminates. There are no commands to exit this utility.



Orphan or Phantom Spool Issues

Like all tables, spool tables require a Table ID. There is a range of tableids exclusively reserved for spool tables (C000 0001 thru FFFF FFFF) and the system cycles through them. If a spool file (table) is incorrectly not dropped, it remains in existence. Eventually, the system will cycle through all the table ids and reassign the tableid which is in use by our left over spool table. Usually, the presence of this table is detected, the query which was going to use the tableid is aborted – even though it is innocent of any wrongdoing – and the following message is returned to the user and put in the error log:

*** FAILURE 2667 Left-over spool table found: transaction aborted.

In rare cases, the leftover spool file (orphan spool) is not detected and the leftover spool is used. Since it was not created by the current transaction, its format is incorrect and the system will crash in an unpredictable way.

A more subtle condition is when a spool table is dropped, but the steps which reduce the tally of spool space currently in use are not. This is phantom spool. The tallies say the table exists, but it does not. Phantom spool does not cause restarts. Unless the space involved is a significant percentage of the total spool reserve, it is just an annoyance.

The query on the facing page can be run to flag the presence of either variety (real or phantom) of leftover spool.

Should this query return rows, the next step is to run the utility **updatespace**. This can be done while the system is in operation, but naturally; it is best done during periods of lower usage. **Updatespace** will correct the phantom spool problem. If the above query still returns rows after **updatespace** is run, then there are actual leftover spool tables. The way to get rid of them is to perform a database reset. Do not be concerned about how to avoid a restart. Do a restart and be operational faster and a whole lot surer and safer.

An important caveat about the query: notice that the user who caused the leftover spool table to be created must not be logged on when the query is run. If he/she is, then spool is considered legitimate by the query. Now, it is not uncommon for some sites to have a user which is nearly always logged on. If the leftover spool was created by such a user, this query will be not report it unless it is run when the system is quiescent.

Filer can also be used by qualified personnel in the GSC to detect real leftover spool on a quiescent system. This often has an advantage of sometimes giving important clues about the root cause.

So what do you do when you discover leftover spool? Institute a procedure to detect leftover spool tables on a regular cycle. You want to discover and eliminate them before the system attempts to reuse the id. How quickly the spool table ids are reused is very site dependent, but something near 2 weeks is the median value.

Orphan or Phantom Spool Issues

In rare cases, when a query completes (or is aborted), the spool file is not dropped.

Orphan Spool – spool file (tableid) incorrectly remains in existence
Phantom Spool – spool file tableid is gone, but the spool space remains allocated.

The following query can be executed to determine the presence of leftover spool (orphan or phantom).

```
SELECT      Databasename, Vproc, CurrentSpool
FROM        DBC.Diskspace
WHERE       Databasename NOT IN (SELECT UserName FROM DBC.SessionInfo)
AND         CurrentSpool > 0
ORDER BY    1, 2
WITH        SUM (CurrentSpool);
```

Should this query return rows, the next step is to run the utility "updatespace".

The UpdateSpace utility is also used to correct inconsistencies in the DBC.DatabaseSpace table, which might occur as the result of unusual (rare) types of system failures.

From supervisor: START UPDATESPACE

Update Space Utility

The Update Space utility (**updatespace)** recalculates the permanent, temporary, or spool space used by either of the following:

- A single database and its individual tables
- All databases in a system and their individual tables

The Update Space utility accomplishes this by performing the following:

- Examining storage descriptors and adding up space for each table.
- Setting values in CurrentPermSpace, CurrentTempSpace, or CurrentSpoolSpace in the DBC.DatabaseSpace table for each table and for the containing database as a whole.

A different utility, Update DBC (**updatedbc**), recalculates the maximum allowed values for permanent, temporary, and spool space in the DBC.Dbase and DBC.DatabaseSpace tables.

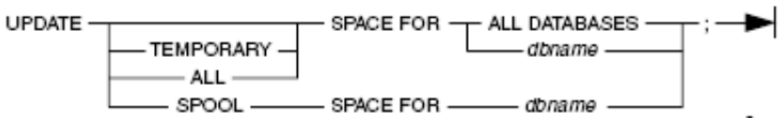
The following table lists the difference between the Update DBC and Update Space utilities.

- Update DBC recalculates <u>maximum</u> allowed values for permanent, temporary, and spool space.

- Update Space recalculates <u>current</u> usage for permanent, temporary, and spool space.

The only reason to use Update Space is to correct inconsistencies in the DBC.DatabaseSpace table, which might occur as the result of rare types of system failures.

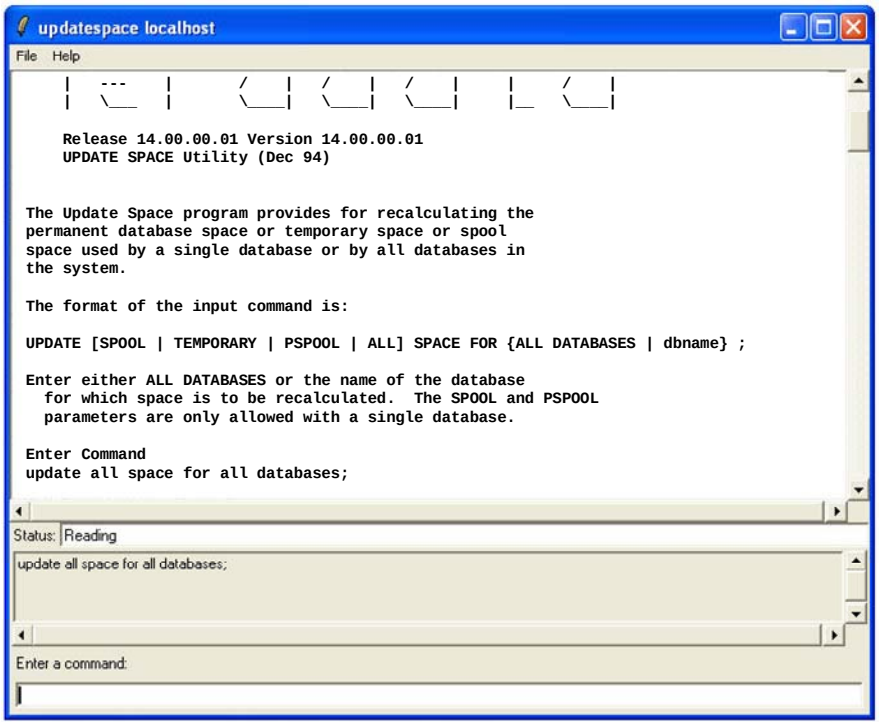
The format of the command to use with the “Update Space” utility is:



Update Space Utility

The Update Space utility, updatespace, recalculates the permanent, temporary, or spool space in the DBC.DatabaseSpace table.

Command →



Vprocmanager

The Vprocmanager utility provides a means to manage/manipulate various vproc attributes. Vprocmanager can be started from the Supervisor (**start vprocmanager**) or command-line (**vprocmanager**).

Valid commands are: STATUS, INITVDISK, RESTART, BOOT, SET, HELP, and QUIT,.

STATUS

The simple format of this command (i.e., STATUS without any options) returns the DBS and PDE status tables in their entirety.

INITVDISK <VprocId>

This command initializes the DBS File System on the Virtual Disk (Vdisk) associated with the specified AMP vproc. It is only applicable to NEWPROC or FATAL AMP vprocs.

Valid VprocIds are decimal numbers in the range of 0 .. 16383. Hex number may be used with a trailing “x”.

RESTART [TPA] [NODUMP | DUMP = {YES, NO}][<RestartKind] [<comment>]

This command is used to force different flavors of DBS restarts.

- A system dump will not be taken when the NODUMP option is specified. This is the default action.
- The DUMP = YES option causes a system dump to be taken.
- The DUMP = NO option is equivalent to NODUMP.
- Valid RestartKinds are COLD or COLDWAIT.
- The comment specifies the reason for the restart.

BOOT <VprocId>

This command will reinitialize the AMP's disk in anticipation of all-tables table rebuild and start the DBS partitions on the specified AMP. It is only applicable to vprocs with a VprocState of FATAL and a ConfigStatus of Down. A confirmation input is needed to process the initialization.

Valid VprocIds are decimal numbers in the range of 0 .. 16383. Hex number may be used with a trailing “x”.

SET

Sets the state of a vproc to either ONLINE, OFFLINE, or FATAL.

Vprocmanager

Commands:

STATUS – provides status information about vprocs

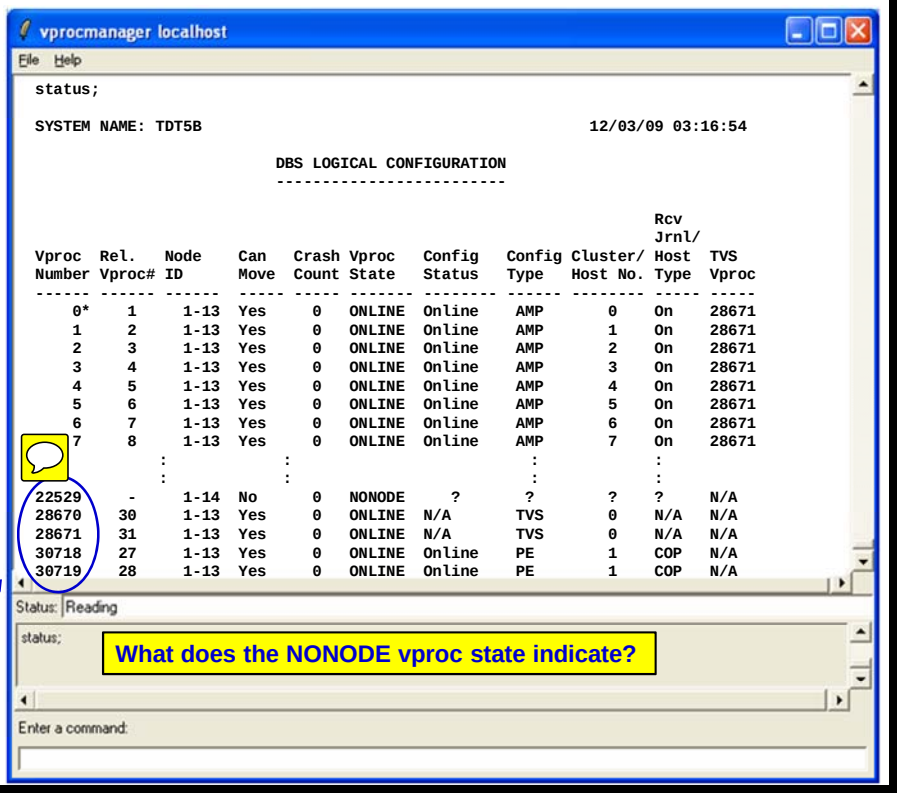
SET – sets the state (e.g., ONLINE, OFFLINE) of a vproc.

RESTART – restarts Teradata

INITVDISK – initializes the DBS File System on a Vdisk.

BOOT – initializes the File System and boots a specific vproc

Note the large vproc numbers that are available in 14.0.



Summary

The facing page summarizes some important concepts regarding this module.

Summary

- **DBSControl** – used to view/modify the DBS Control Record fields to establish system and default values.
- **Packdisk** – fill (packs) cylinders up to the Free Space Percentage (FSP) with the purpose of freeing up cylinders.
- **Defragment** – combines free sectors and moves them to the end of a cylinder.
- **Scandisk** – identifies and determines the extent of any problems with the AMP file system.
- **Checktable** – checks for inconsistencies in internal structures such as table headers, row identifiers and secondary indexes.
- **Table Rebuild** – repairs data corruption.
- **Recovery Manager** (RCVManager) – enables you to view information about online transaction recovery and AMP recovery.
- **Showlocks** – displays host utility locks.

Module 55: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 55: Review Questions

1. What are two ways that you initiate a Teradata system utility (e.g., dbscontrol)?

   _____ _____

2. Identify the purpose of the following DBS Control utility parameters.

   CenturyBreak  _____

   DateForm _____

   MaxLoadTasks _____

   SessionMode _____

3. True or False. The Checktable utility has only two levels of internal table checking.

4. True or False. The Table Rebuild utility rebuilds tables differently depending on whether the table is a fallback, non-fallback or permanent journal table.

5. The _____ utility does a consistency check within an AMP's file system.

6. The _____ utility does a consistency check for a table across all AMPs.

7. The _____ utility can be used to set an offline AMP to online.

Notes

Module 56

Permanent Journals

After completing this module, you will be able to:

- **Describe journaling options and the type of recovery each option provides.**
- **Determine when to use permanent journals instead of (or in addition to) fallback to provide data integrity.**
- **Create, modify and delete permanent journals for databases and tables.**
- **Use the DBC.Journals view to associate tables with specific permanent journals.**

Notes

Table of Contents

Automatic Data Protection Mechanisms (Review) 56-4
- Transient Journal 56-4
- Fallback Protection 56-4
- Down AMP Recovery Journal 56-4
- RAID 1 and RAID 5 56-4
- Cliques 56-4

Permanent Journals 56-6
Location of Change Images 56-8
- AMP Definitions 56-8
- After-Image Journals Save Storage Space 56-8

Assigning Tables to a Permanent Journal 56-10
- Journaling Functions 56-10

Creating a Permanent Journal 56-12
- Deleting a Permanent Journal 56-12
  - Syntax 56-12

Assigning a Permanent Journal 56-14
- Rules and Limitations 56-14
- DBC.Tables 56-14
- Syntax 56-14

Before-Image Journals 56-16
- Before Images 56-16

After-Image Journals 56-18
- Site Disaster 56-18

Journal Subtables 56-20
- Current Journal 56-20
- Restored Journal 56-20

Permanent Journal Statements 56-22
- Backing up tables on a Teradata System 56-22

Recovery with Permanent Journals 56-24
Journals View 56-26
- Columns Defined 56-26

Summary 56-28
Module 56: Review Questions 56-30

Automatic Data Protection Mechanisms (Review)

The Teradata system offers a variety of methods to protect data. Some methods are automatically activated when particular events occur in the system. Other data protection methods require that you set options. Each data protection technique offers different types of advantages under different circumstances.

Transient Journal

The Transient journal maintains snapshots of rows in tables before you or other users make changes to them. If the transaction fails or if you abort the request, the Transient Journal copies its snapshot into the existing table which rolls back any changes the failed transaction may have made to the table.

Fallback Protection

Fallback protection is an optional data protection feature that you activate with the CREATE or MODIFY commands. Fallback provides data level protection by automatically creating a copy of each row on a fallback AMP. If the primary AMP fails, the system can access the fallback copy. The fallback feature allows automatic recovery using the Down AMP Recovery Journal once the down AMP comes back on-line. Fallback protected tables occupy twice the space in your system as non-fallback tables.

Down AMP Recovery Journal

The Down AMP Recovery Journal supports fallback protection. If a primary AMP fails, the fallback feature allows automatic data recovery using the Down AMP Recovery Journal. This feature consists of these two journals: DBC.ChangedRowJournal and DBC.OrdSysChngTable.

RAID 1 and RAID 5

RAID 1 provides data redundancy through disk mirroring which means that data on one disk is identical to the information on another disk. If one disk fails, the alternate disk takes over. RAID 5 (or RAID S) protects data with a technique called “data parity protection”. Data is striped across multiple disks while the parity of each piece of data is preserved to allow array controllers to determine what the missing data is. The user experiences no downtime

Cliques

Group of SMP nodes sharing a common set of disk arrays. If a node fails, vprocs can migrate to other nodes within the clique. Although Teradata will restart, allows Teradata to continue running in the event of a node failure.

Automatic Data Protection Mechanisms (Review)

- **Transient Journal**
  - Takes before-image (snapshot) of row before change is made
  - Copies before-image of row back to table if transaction fails
  - Maintained within the WAL Log
- **Fallback Protection**
  - Optional data protection feature for a table
  - Creates copy of each row on fallback AMP
- **Down AMP Recovery Journal**
  - Automatically used for fallback tables when an AMP is down
  - Other AMPs in the cluster identify rows that have changed for a down AMP
- **RAID 1 or RAID 5**
  - Data redundancy through disk mirroring (RAID 1) or data parity protection (RAID 5)
  - Provides protection from physical disk failure
- **Cliques**
  - Group of nodes where vproc migration can occur
  - Provides protection from node failure(s)

Permanent Journals

The Teradata system offers a manual method called permanent journals that you can use to protect data. The purpose of a permanent journal is to maintain a sequential history of all changes made to the rows of one or more tables. Permanent Journals help protect user data when users commit, rollback, or abort transactions. A permanent journal can capture a snapshot of rows before a change, after a change, or both. Each database or user space can contain only one journal table.

Existing data tables can write to a journal table defined in its parent or to a journal table located in another database or user. Journal tables require permanent space.

You can create permanent journal tables with the CREATE USER/CREATE DATABASE statement or the MODIFY USER/MODIFY DATABASE statement.

Permanent journal tables exist within a database or user space. Only one permanent journal can be assigned to that user or database. The journal may be located in the same database or user as the tables that use the journal or in a different database.

Permanent Journals

Permanent journals:

- Optional features that can provide protection for software and hardware failures.
- Store committed, uncommitted and aborted changes.
- Users manage journal tables.

Permanent journal options:

- Single before change image: BEFORE
  - Captures images before a change is made
  - Protects against software failures
  - Allows rollback to a checkpoint
- Single after-change image: AFTER
  - Captures images after a change is made
  - Protects against hardware failures
  - Allows rollforward to a checkpoint
- Dual image: DUAL BEFORE or DUAL AFTER
  - Maintains two images copies
  - Protects against loss of journals

- Keyword JOURNAL with no other keywords capture single before and after images.

Location of Change Images

Tables that include fallback and journaling options automatically receive dual image journal protection. Tables with no-fallback protection can request either single or dual permanent journals.

The chart on the facing page illustrates the location of change-image journals. The placement of permanent journals depends on: requested image type (either before or after) and the protection type (either fallback or non-fallback).

AMP Definitions

The following definitions are used to describe how AMPs are used to store before and/or after images.

Primary AMP — Holds before- and/or after-images for any table with fallback protection. Holds single before images and dual after-images for non-fallback protected tables.

Fallback AMP — Contains before- and/or after-images for tables with fallback protection. The DBC distributes duplicate data rows to fallback processors by assigning the row's hash code to a different AMP in the cluster.

Backup AMP — With most systems, clusters consist of 2 AMPs. Therefore, the Fallback and Backup AMP is the same AMP.

For older systems with clusters > 2 AMPS, the Backup AMP holds three types of images for non-fallback tables: single or dual after images, and dual before images. The hashing algorithm is not used for the row distribution. All images for one AMP go to a single backup, which is always in the same cluster. For example, if AMPs 1, 2, 3, and 4 are in the same cluster, 1 backs up 2, 2 backs up 3, 3 backs up 4, and 4 backs up 1. There is no way to predict the backup AMP.

After-Image Journals Save Storage Space

If fallback protection is too costly in terms of storage space, after-image journals offer alternative data protection with minimal space usage. After-image journals write changes to the backup AMP. Since the system only duplicates changed rows rather than all of the rows, storage space is minimized.

Since changes are written to the backup AMP, a primary AMP failure does not cause a loss of data. You can recover all table data by restoring the appropriate archive tape and rolling forward the rows stored in the after-image journal.

Location of Change Images

The location of journal rows depends on the image type requested (before or after) and the protection type of the journaled tables.

Fallback Tables

Journal Option	Change Image Location
After images	Primary AMP and fallback AMP
Before images	Primary AMP and fallback AMP

- **Dual images are always maintained for fallback tables.**
- **To determine the fallback AMP for a journal row, the fallback hash map is used.**

Non-fallback Tables

Journal Option	Change Image Location
After images	Backup AMP
Before images	Primary AMP
Dual after images	Backup AMP and primary AMP
Dual before images	Primary AMP and backup AMP

- **For no fallback tables, you may request either single or dual journal images.**
- **Since most systems (all new systems) have 2-AMP clusters, the fallback and backup AMP is the same AMP in a cluster.**
  - **With clusters > 2 AMPs, a backup AMP is another AMP in the same cluster as the primary AMP assigned to journal rows.**

Assigning Tables to a Permanent Journal

When you create a new journal table, there are options you can use to control the type of information the table captures.

A permanent journal provides four basic options:

Option	Description
Single Image	Captures/stores one copy of the data.
Dual Image	Captures/stores two separate copies of data: one copy on the primary AMP and one on the fallback AMP or backup AMP.
Before Image	Captures/stores row values before a change occurs.
After Image	Captures/stores row values after a change occurs.

Unlike transient and recovery journals, permanent journal options capture and store all changes whether, committed, uncommitted, or aborted. In addition, journal maintenance and activity are under user control.

Journaling Functions

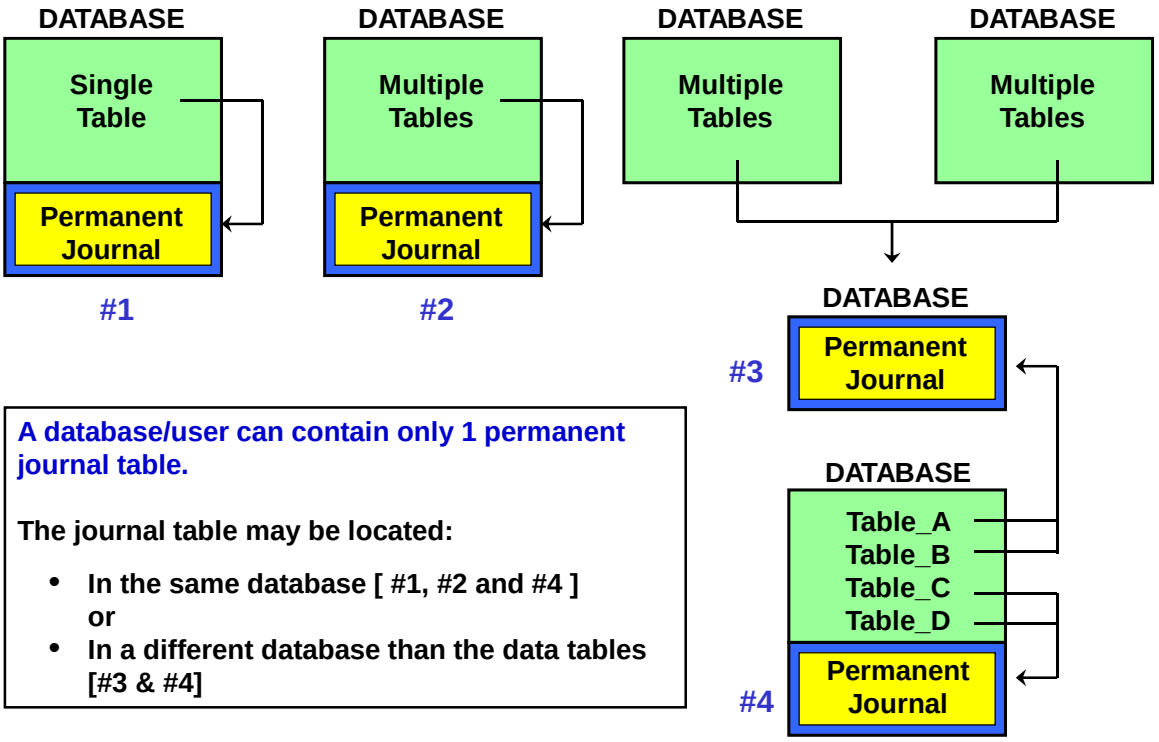
Journal tables use rollback operations for software failure recoveries. To restore data tables to the state they were in before a software failure; configure a permanent journal to capture before-change information.

Permanent journaling is not a substitute for RAID technology or fallback protection. Both options provide duplicate images of all rows in a table. The journal tables only maintain images for changed rows.

Journal tables can protect against:

- Loss of data caused by a disk failure in a table that is not fallback or RAID protected.
- Loss of data if two or more AMP vprocs fail in the same cluster. (This would mean the loss of two disks in a rank per failed AMP vproc.)
- Incorrect operation of a batch or application program.
- Disaster recovery of an entire system.
- Loss of changes made after a data table is archived.
- Loss of one copy of the journal table (Dual journal).

Assigning Tables to a Permanent Journal



A database/user can contain only 1 permanent journal table.

The journal table may be located:

- **In the same database [#1, #2 and #4] or**
- **In a different database than the data tables [#3 & #4]**

Creating a Permanent Journal

You create permanent journals when you create a user or database. To create permanent journals within an existing user or database, use the MODIFY statement. The facing page shows examples of using these statements.

The following restrictions apply to the use of permanent journals:

- If a journal table in another user/database is specified as the default, that other journal table must already exist.
- You can change a DEFAULT JOURNAL for a user or database only if no tables or other databases journal into it.
- Permanent journals are not supported across an AMP configuration change. Rollforward or Rollback operations terminate if there is a change in the hash maps for primary, fallback, or backup rows.
- Permanent journals are not supported across certain Data Definition (DDL) statements. Statements that may prevent a rollforward or rollback operation from passing that point in the journal include:
- ALTER TABLE
- RENAME TABLE
- MODIFY USER or MODIFY DATABASE
- COMMENT

Deleting a Permanent Journal

Use the MODIFY USER or MODIFY DATABASE statement to delete a permanent journal. Before you delete the journal, you must use the ALTER TABLE statement to stop the journaling being done to that journal.

Syntax

```
ALTER [TABLE NAME]
    ,WITH [JOURNAL TABLE=JOURNAL TABLE NAME];
    ,NO BEFORE JOURNAL
    ,NO AFTER JOURNAL;

MODIFY DATABASE [DATABASE NAME AS]
    DROP DEFAULT JOURNAL TABLE=[JOURNAL TABLE NAME];
```

Creating a Permanent Journal

You create permanent journals at the user/database level when you define a new user or database:

```
CREATE  DATABASE Payroll_Tab AS PERM = 100E6
        DEFAULT JOURNAL TABLE = Payroll_Jrnl;
```

Or you can create them in an existing user or database:

```
MODIFY  DATABASE HR_Tab AS
        DEFAULT JOURNAL TABLE = HR_Jrnl;
```

They are identified in the DD/D as TableKind 'J':

```
SELECT   DatabaseName, TableName, TableKind
FROM     DBC.TablesV
WHERE    TableKind = 'J' ;
```

Response:

DatabaseName	TableName	TableKind
Payroll_Tab	Payroll_Jrnl	J
HR_Tab	HR_Jrnl	J

Assigning a Permanent Journal

Permanent journals are optional. You can specify journal options at the database/user level or at the individual table level. The journal options you can define are:

JOURNAL	DUAL AFTER JOURNAL
BEFORE JOURNAL	NO JOURNAL
AFTER JOURNAL	NO AFTER JOURNAL
DUAL JOURNAL	NO BEFORE JOURNAL
DUAL BEFORE JOURNAL	

You can define a DEFAULT JOURNAL TABLE associated with a user or database. You can associate an individual table within the database with the DEFAULT JOURNAL (by default) or another journal table by specifying that on the CREATE or ALTER TABLE statement.

Users activate permanent journaling by including the JOURNAL option in the CREATE or MODIFY statements for users or databases. The following page illustrates CREATE USER and CREATE TABLE statements that create and assign permanent journals.

If you create a database/user and specify a default journal table, but do not specify any journaling options, the default at the database level is NO BEFORE and NO AFTER journaling for tables created in the database. When creating a table in this database/user and if you want journaling, you must specify the journaling options you want as part of the CREATE TABLE.

Rules and Limitations

You must allocate sufficient permanent space to a database or user that will contain permanent journals. If a database or user that contains a permanent journal runs out of space, all table updates that write to that journal abort.

DBC.Tables

The DBC.Tables view can display the names of existing journal tables. The TableKind field displays the letter J for any table set up as a permanent journal. The query statement on the next page displays a list of journal table names.

Syntax

```
[NO]     [BEFORE] [[,][NO] [AFTER JOURNAL]]
[DUAL]  [AFTER]  [[,][DUAL ]   [BEFORE JOURNAL]]
DEFAULT JOURNAL TABLE = [dbname.] tname
```

Assigning a Permanent Journal

```
{ CREATE USER | CREATE DATABASE } . . .
   [ [ [ NO | DUAL ] [ AFTER | BEFORE ] JOURNAL ] . . . ]
   [ DEFAULT JOURNAL TABLE = journal_name ] ;
```

Default journal values at the database levels are:

Journal Option	Default
NONE SPECIFIED	NO JOURNAL MAINTAINED
NEITHER AFTER NOR BEFORE	BOTH TYPES IMPLIED
NEITHER DUAL NOR NO	FALLBACK – DUAL IMAGES; NO FALLBACK – SINGLE IMAGES

At the table level, you can indicate journal options with the CREATE statement:

```
CREATE TABLE . . .
   [ [ [ NO | DUAL ] [ AFTER | BEFORE ] JOURNAL ] ... ]
   [ WITH JOURNAL TABLE = journal_name ] ;
```

Default journal values at the table levels are :

Journal Option	Default
NONE SPECIFIED	Defaults to USER/DATABASE
AFTER IMAGE ONLY	Defaults FOR BEFORE IMAGE
BEFORE IMAGE ONLY	Defaults FOR AFTER IMAGE
NEITHER DUAL NOR NO	Defaults to PROTECTION TYPE

Note: If a database or user that contains a permanent journal runs out of space, all table updates that write to that journal abort.

Before-Image Journals

You can define permanent journals to record:

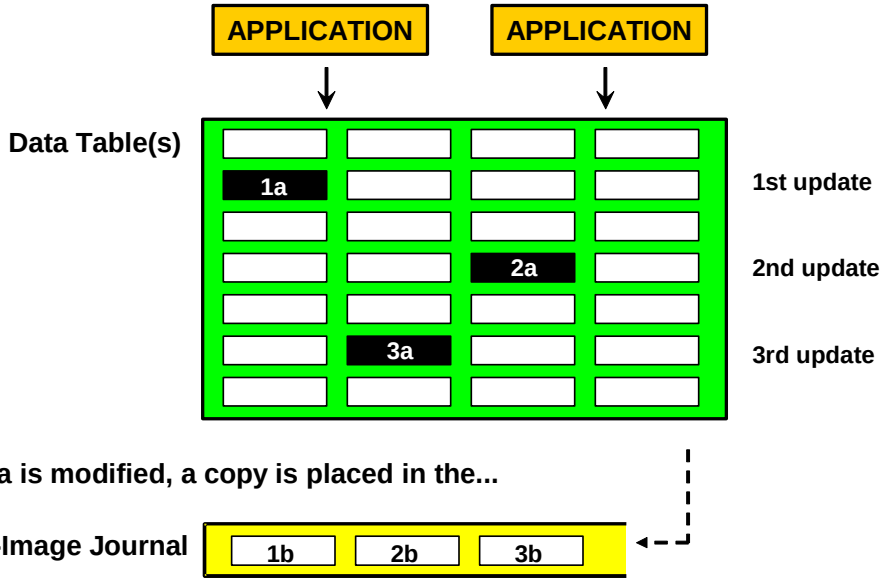
After Change Images	The data in a row after a change has occurred is recorded in the permanent journal.
Before Change Images	The data in a row prior to its change is recorded in the permanent journal.
Both (Before Change Images and After Change Images)	The data in the permanent journal is maintained in the internal DBC format and is not accessible to the user through any SQL statements. Users delete permanent journals.

Before Images

Before Images are used for ROLLBACK recovery as shown on the following page. Once a before-image journal is created, a snapshot of an existing row is stored in the journal table before any data is modified. In the event of a software failure, the before-image journal can roll back any unwanted changes. Permanent journals roll back all transactions from a table to a checkpoint. They may not be used to roll back specific transactions.

Before-Image Journals

Before-images are used to roll back users' changes to one or more tables by returning the data to a previous consistent state.



After-Image Journals

After you create an after-image journal, a snapshot of a row value is stored in the permanent journal after a change is committed. If a hardware failure occurs, the after-image journal can roll forward any changes made to data tables since the last full system backup.

Site Disaster

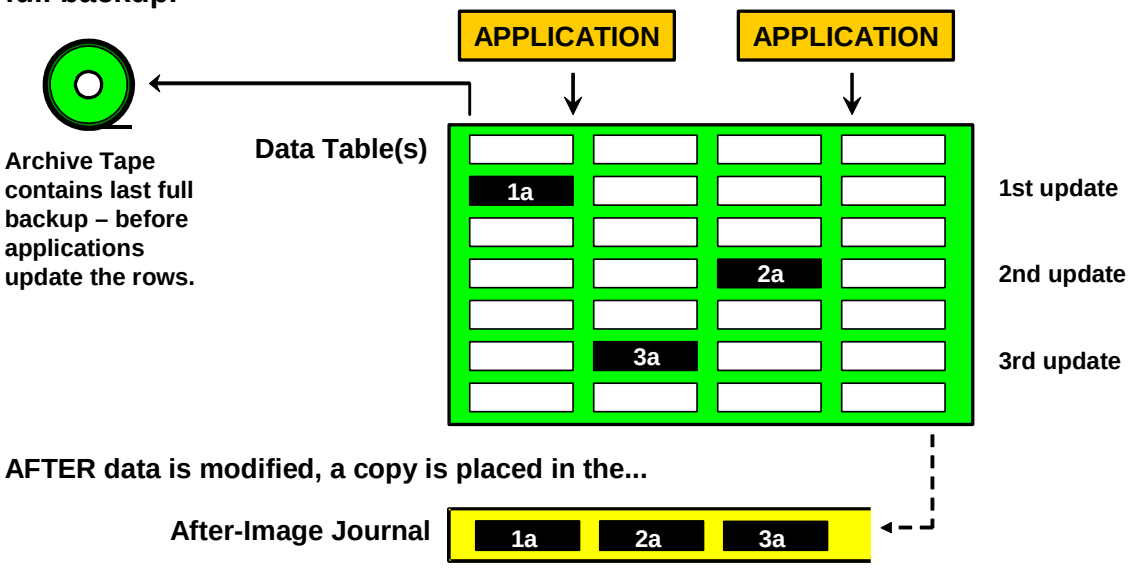
To protect against the loss of data in the event of a site disaster, many applications require that data archives be kept off-site at all times. Ideally, users dump the database to magnetic tape daily and store the tape off-site.

Daily archives may not be practical for very large databases. To solve this problem, you can activate after-change journals and take a daily archive of the journal itself that provides archived copies of all changes made since the last full database archive. The full backup tapes along with the journal backup tapes could restore the entire database.

The facing page shows after images in the permanent journal are used for ROLLFORWARD recovery.

After-Image Journals

After-images can be used to apply changes users have made since the last full backup.



To recover data that must be restored, use the after-images in the permanent journal to rollforward users' changes since the restored backup was taken.

Journal Subtables

Each journal table consists of three areas:

- Active area (part of current journal subtable)
- Saved area (part of current journal subtable)
- Restored area (part of restored journal subtable)

The active and saved areas together are referred to as the Current Journal. The restored subtable is called the Restored Journal. The contents and purpose of each subtable are discussed below:

Current Journal

Each time you update a data table that has an associated journal table; a change image is appended to the active subtable. You cannot archive journal tables while the change images are in the active subtable. Instead, you must move the images to the saved subtable.

To move images from active to saved areas, you must submit the Checkpoint With Save statement. A checkpoint places a marker at the chronological end of the active subtable. The database assigns an event number any time a user submits the checkpoint statement. The With Save option of the checkpoint statement inserts a checkpoint in the active subtable and then appends the contents of the active subtable to the end of the saved subtable.

After the database appends the contents, it initiates a new active subtable automatically. You can now submit an ARCHIVE JOURNAL TABLE statement. Archiving the journal saves it to tape.

Restored Journal

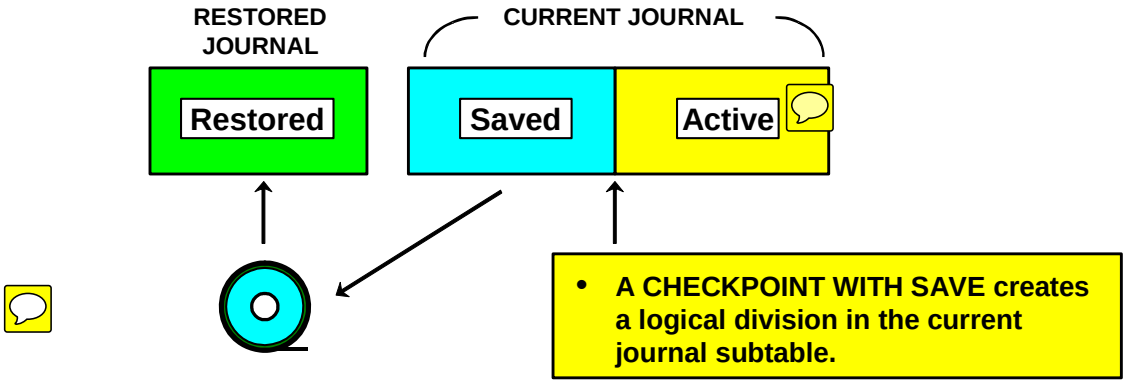
To restore a journal, move the journal table contents from the portable storage media back to the restored subtable. The information stays there until you invoke roll operations.

Permanent journals are maintained in an internal Teradata database format. They are not accessible by SQL statements and cannot be used for audit trail purposes.

Journal Subtables

A permanent journal table consists of three areas:

- **Active area** – part of Current journal
- **Saved area** – part of Current journal
- **Restored area** – part of Restored journal



- A CHECKPOINT WITH SAVE creates a logical division in the current journal subtable.

Restored journals replace the contents of the restored subtable.

- Subsequent journal images append to the active area.
- You can ARCHIVE and delete rows only from the Saved Journal area.

Permanent Journal Statements

Use the ARC (Archive and Recovery) utility to perform backup and recovery functions associated with permanent journals. The archive and recovery functions include:

ROLLFORWARD	Replaces a data row by its after-image from the beginning of the journal, to either a checkpoint or to the end of the journal.
ROLLBACK	Replaces a data row by its before change image from the end of the journal, to a checkpoint or to the beginning of the journal.
DELETE	Deletes the contents of either the saved or restored journal areas.

Backing up tables on a Teradata System

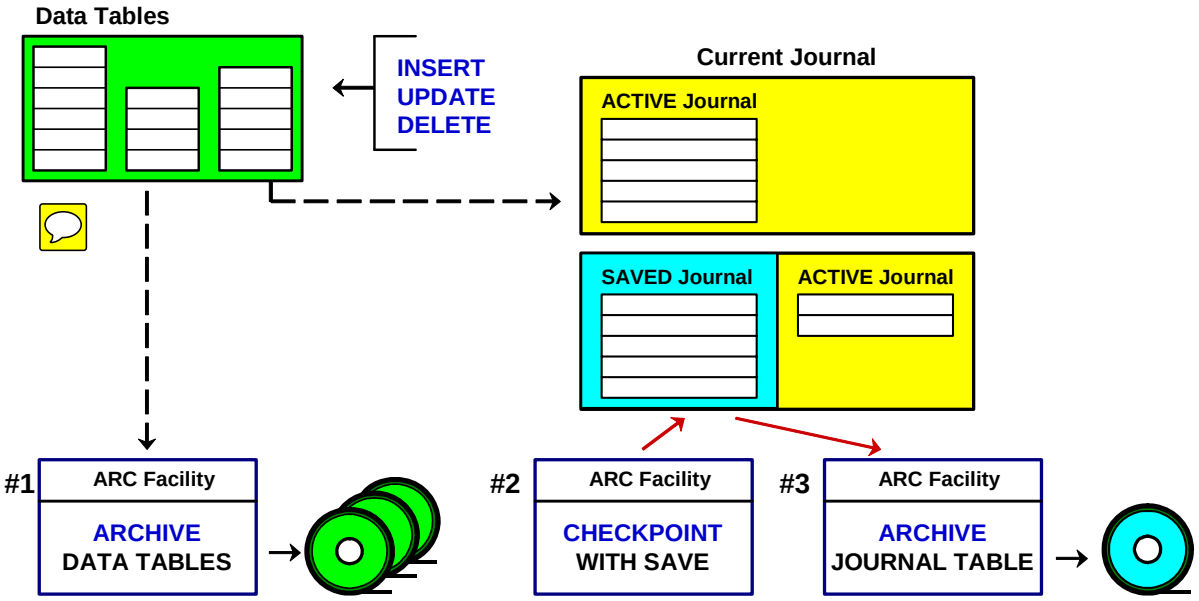
- Archive the data tables onto portable storage media.
- Submit a checkpoint with a SAVE statement to move change images from the active journal to the saved journal.
- Archive the journal tables onto portable storage media.
- Submit the DELETE JOURNAL statement to erase the saved journal rows.

Permanent Journal Statements

1. First, ARCHIVE data tables.

After users and/or applications have modified tables, save the journal images.

2. The CHECKPOINT WITH SAVE command creates a saved journal.
3. You can ARCHIVE and delete saved journal rows.



Recovery with Permanent Journals

An example of how to use some of these ARC statements is shown when a batch program is run:

1. Submit an SQL Checkpoint statement as the first statement of the batch job, with or without a Checkpoint name.

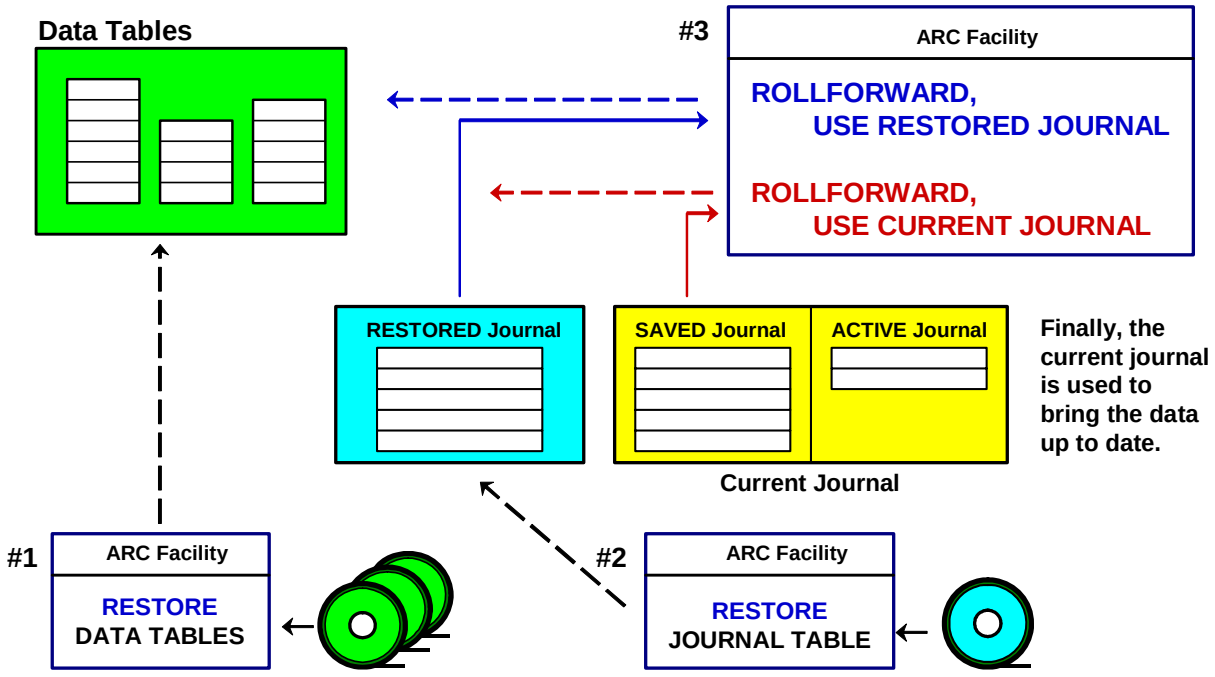
2. If required, ROLLBACK to the Checkpoint using either the checkpoint name or the event number supplied by the DBC when you executed the Checkpoint command. Later changes are also backed out.

3. The data table is now in its *original* condition.

A permanent journal is time-oriented, not transaction-oriented.

Recovery with Permanent Journals

Tables or databases are restored first. Next, archived journals are restored, one at a time, and then the restored journal is rolled forward.



Journals View

The Teradata system provides a system view called DBC.Journals[V][X], that displays links between journal tables and the data tables that journal into them. The DBC.JournalsX View is a restricted view. The restricted version of the view displays only those objects that you own or to which you hold access rights.

The example on the next page uses the SELECT statement to list all of the tables in the system that uses a permanent journal. In addition, it requests to see a list of the journal names.

The response displays the table names first followed by the journal names.

Columns Defined

The Journals view has four different columns. Each one is described below:

Tables_DB	Displays the name of a database where a data table resides that has the journal option activated.
TableName	Displays the name of a data table that records changed images in a journal table.
Journals_DB	Displays the name of a database where a journal table resides.
JournalName	Displays the name of a journal table associated with a listed data table.

Journals View

Associates journals with tables when the user owns or holds rights on the objects referenced.

DBC.Journals[V][X]

Tables_DB	TableName	Journals_DB	JournalName

Example: List all tables in the system that use a journal and list the names of the journals.

```
SELECT    TRIM (Tables_DB)      || '.' || TableName
                AS "Table Name"              (CHAR(30))
          ,TRIM (Journals_DB)   || '.' || JournalName
                AS "Assigned to Journal"     (CHAR(30))
FROM      DBC.JournalsV
ORDER BY  1 ;
```

Example Results:

Table Name	Assigned to Journal
HR_Tab.Employee	HR_Tab.HR_Jrnl
HR_Tab.Department	HR_Tab.HR_Jrnl
HR_Tab.Job	HR_Tab.HR.Jrnl
Payroll_Tab.Paycheck	Payroll_Tab.Payroll_Jrnl

Summary

The facing page summarizes some important concepts regarding this module.

Summary

- Permanent journals maintain a sequential history of all changes made to the rows of one or more tables.
- You create a permanent journal when you CREATE/MODIFY a user/database.
- Permanent journal image options:
  - Single before-change images
    - Capture images before a change is made and allows rollback to a checkpoint. Protects against software failures.
  - Single after-change images
    - Capture images after a change is made and allows rollforward to a checkpoint. Protects against hardware failures.
  - Dual images
    - Maintain two copies of before or after images. Protects against loss of journals.
- Use ARC facility to perform backup and recovery operations associated with permanent journals.
- The Journals[X] view provides information about links between journal tables and the tables that journal to them.

Module 56: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 56: Review Questions

1. True or False. A permanent journal stores committed, uncommitted, and aborted changes to a row in a table.
2. True or False. A database or user can have many permanent journals.
3. True or False. Separate Permanent Journals are required for before and after images.
4. True or False. The Saved and Active areas are both part of the Current Journal.
5. True or False. The CREATE JOURNAL statement may be used to create a permanent journal.
6. True or False. Tables that use a Permanent Journal must be in the same database as the Permanent Journal.

Notes

Module 57

TERADATA
Raising Intelligence

A Tale of Three Tables

After completing this module, you will be able to:

- **Analyze the efficiency of backup procedures after both drives in a mirrored pair have failed.**

Notes

Table of Contents

Permanent Journal Scenario 57-4
Table X 57-6
Table Y 57-8
Table Z 57-10
Permanent Journals 57-12
Archive Policy 57-14
  Daily Archive Procedures 57-14
  Weekly Archive Procedure ..... **Error! Bookmark not defined.**
Archive Scenario 57-16
After Restart Processing Completes 57-18
After REBUILD and Restart of Teradata 57-20
Table X Recovery 57-22
Table Y Recovery 57-24
  Recovery Action 57-24
Table Z Recovery 57-26
  Recovery Action 57-26
After Recovery 57-28
Summary 57-30

Permanent Journal Scenario

In the following scenario, assume that a user has three tables in a four AMP system. Each table has its own data protection features stored on all four AMPs. The illustrations on the following pages illustrate data protection features in effect for each table.

In this example, AMPs 1 and 2 are clustered together and AMPs 3 and 4 are clustered together.

Permanent Journal Scenario

A Tale of Three Tables

A user has three data tables:

Table X Fallback
- **Before and**
- **After Image Journals**

Table Y No Fallback
- **No Before and**
- **Dual After Image Journals**

Table Z No Fallback
- **Single Before and**
- **Single After Image Journals**

Table X

Table X is defined as having fallback, dual before images, and dual after images.

Table X

- ✓ Fallback protected data
- ✓ Fallback protected before images
- ✓ Fallback protected after images

Assume AMPs 1 & 2 are in a cluster and AMPs 3 & 4 are in a cluster.

	AMP 1	AMP 2	AMP 3	AMP 4
Data	Table X Primary 1	Table X Primary 2	Table X Primary 3	Table X Primary 4
	Table X Fallback 2	Table X Fallback 1	Table X Fallback 4	Table X Fallback 3
PJ	After 1	After 2	After 3	After 4
	After 2	After 1	After 4	After 3
	Before 1	Before 2	Before 3	Before 4
	Before 2	Before 1	Before 4	Before 3

Table Y

Table Y has no fallback protection, but has dual after image journaling defined.

Table Y

- ✓ No fallback
- ✓ Dual after image

Assume AMPs 1 & 2 are in a cluster and AMPs 3 & 4 are in a cluster.

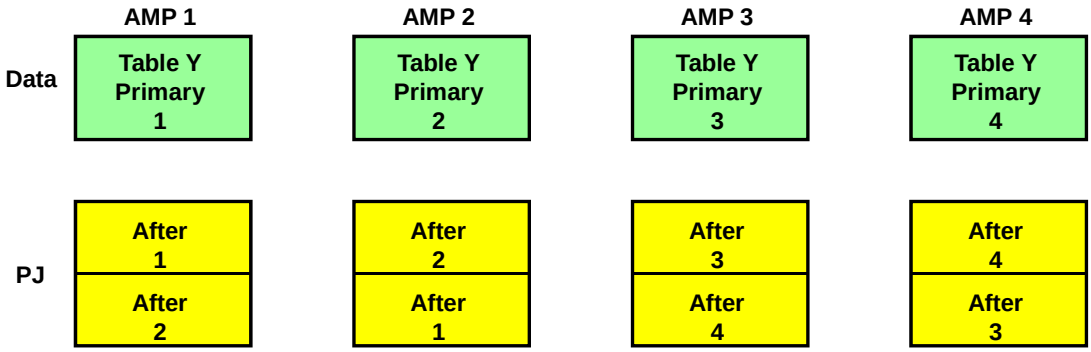


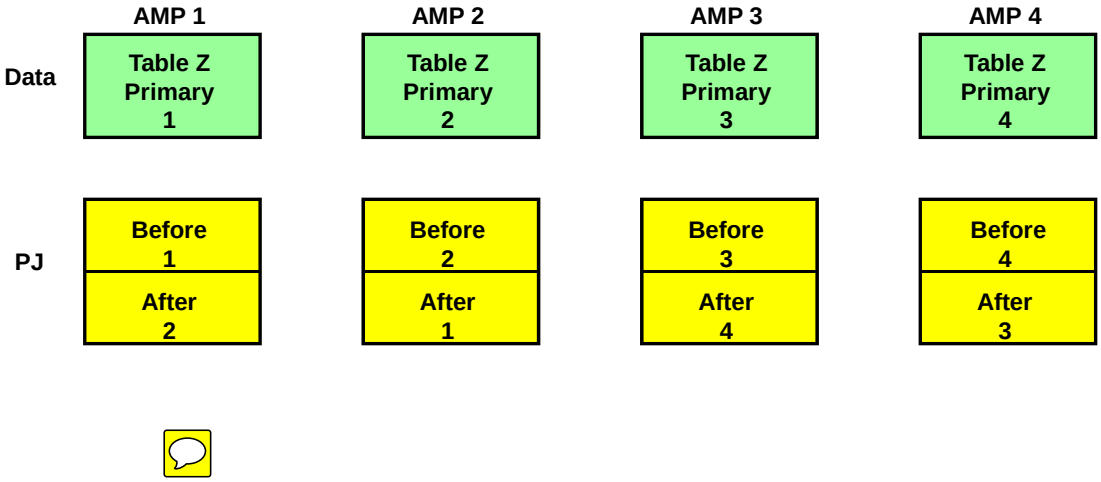
Table Z

Table Z has no fallback protection. This table has single before and after-images.

Table Z

- ✓ No fallback
- ✓ Single before images
- ✓ Single after images

Assume AMPs 1 & 2 are in a cluster and AMPs 3 & 4 are in a cluster.

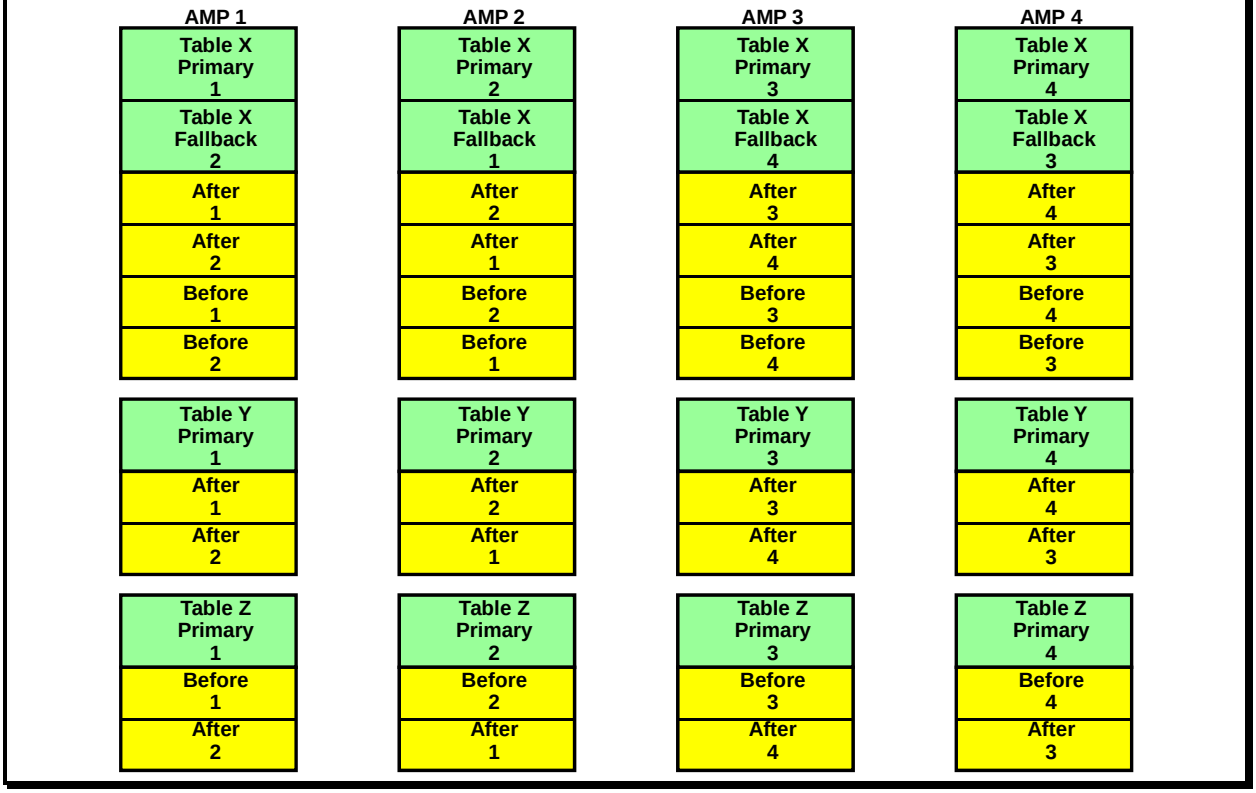


Permanent Journals

The facing page shows the three tables and all their data protection options.

Permanent Journals

(Putting all tables three together)



Archive Policy

The company established an archive policy to cover any data loss in the event of a site disaster. The archive policy has two components:

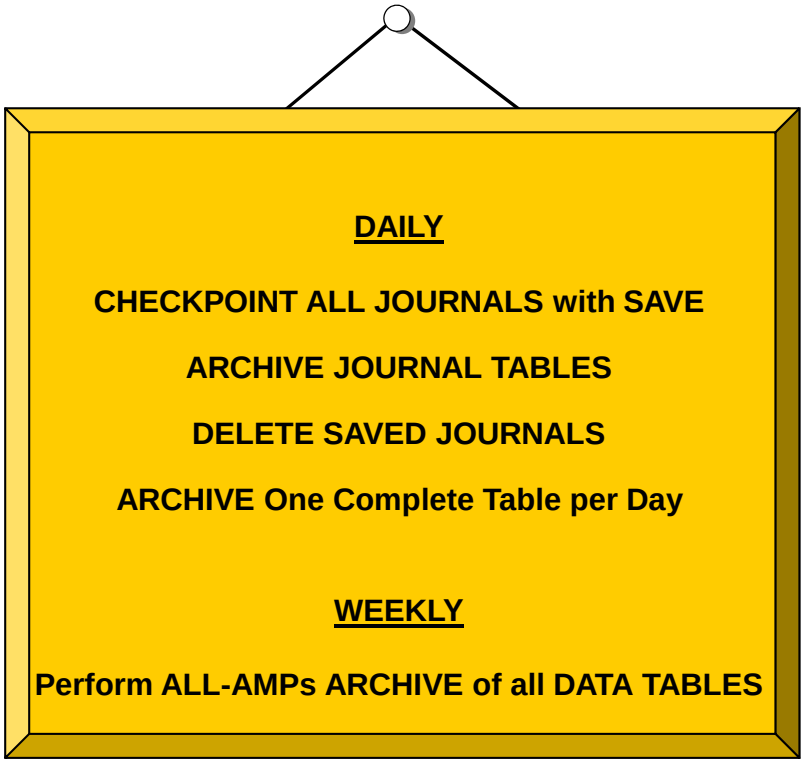
- Daily archive procedures
- Weekly archive procedures

Daily Archive Procedures

Each day the administrator submits a CHECKPOINT WITH SAVE command for each journal table which appends any changes stored in the active journal subtable to the saved journal subtable. In addition, it initiates a new active journal subtable. Second, the administrator archives each current journal, and then deletes the saved journal subtable from the saved journal.

Only one table is archived each day. By the end of the week, each table has been archived once.

Archive Policy



Archive Scenario

The company activated its archive policy and implemented daily and weekly backup procedures as scheduled. Each day the administrator archives journals X, Y, and Z.

On Monday, the administrator archived data table X, and on Tuesday archived table Y. On Wednesday, the administrator archived data table Z. On Thursday, two drives failed in a drive group.

Archive Scenario

Monday:

Archive journals X, Y and Z
Archive table X

Tuesday:

Archive journals X, Y and Z
Archive table Y

Wednesday:

Archive journals X, Y and Z
Archive table Z

Thursday:

AMP 3: Two drives fail in a drive group

After Restart Processing Completes

The administrator utilized restart procedures to replace the down AMP. The diagram on the facing page outlines each restart step. Each restart procedure is explained below:

1. Replace the 2 drives.

2. Initialize the rank.

3. Format the array (RAID 1 or 5).

4. Initialize and rebuild the AMP's Vdisk using Vprocmanager functions (e.g., BOOT).

5. Use the table REBUILD utility to rebuild AMP 3 - the VprocState will be UTILITY during this phase.

6. REBUILD will set the VprocState to ONLINE when finished.

7. Restart Teradata.

After Restart Processing Completes

AMP 1	AMP 2	AMP 3	AMP 4
Table X Primary 1	Table X Primary 2	1. Replace the 2 drives. 2. Initialize the rank. 3. Format the array (RAID 1 or 5). 4. Initialize and rebuild the AMP's Vdisk using Vprocmanager functions (e.g., BOOT). 5. Use the table REBUILD utility to rebuild AMP 3 - the VprocState will be UTILITY during this phase. 6. REBUILD will set the VprocState to ONLINE when finished. 7. Restart Teradata.	Table X Primary 4
Table X Fallback 2	Table X Fallback 1		Table X Fallback 3
After 1	After 2		After 4
After 2	After 1		After 3
Before 1	Before 2		Before 4
Before 2	Before 1		Before 3
Table Y Primary 1	Table Y Primary 2		Table Y Primary 4
After 1	After 2		After 4
After 2	After 1		After 3
Table Z Primary 1	Table Z Primary 2		Table Z Primary 4
Before 1	Before 2		Before 4
After 2	After 1		After 3

After REBUILD and Restart of Teradata

The diagram on the facing page shows the row information that the administrator recovered after she executed the REBUILD and RESTART commands.

Table X is fully recovered. All primary and fallback rows are restored. In addition, all before and after-journal images are recovered as well. The administrator needs to perform additional recovery measures on table Y and table Z.

After REBUILD and Restart of Teradata

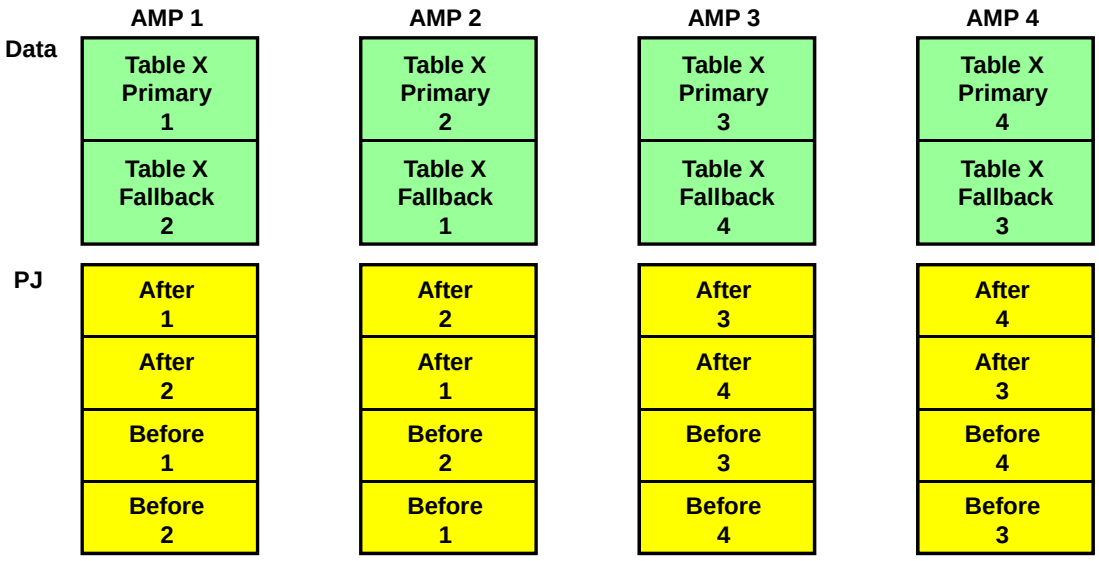
AMP 1	AMP 2	AMP 3	AMP 4
Table X Primary 1	Table X Primary 2	Table X Primary 3	Table X Primary 4
Table X Fallback 2	Table X Fallback 1	Table X Fallback 4	Table X Fallback 3
After 1	After 2	After 3	After 4
After 2	After 1	After 4	After 3
Before 1	Before 2	Before 3	Before 4
Before 2	Before 1	Before 4	Before 3
Table Y Primary 1	Table Y Primary 2	Table Y (Header)	Table Y Primary 4
After 1	After 2	After 3	After 4
After 2	After 1	After 4	After 3
Table Z Primary 1	Table Z Primary 2	Table Z (Header)	Table Z Primary 4
Before 1	Before 2		Before 4
After 2	After 1		After 3

Table X Recovery

Table X has primary and fallback rows restored. All journal images are also recovered.

Table X Recovery

- ✓ Fallback protected data
- ✓ Fallback protected before images
- ✓ Fallback protected after images



Fully Recovered

Table Y Recovery

The diagram on the facing page illustrates table Y after REBUILD and RESTART procedures.

The administrator used tables stored on AMP 2 and AMP 4 to restore the two permanent journal tables stored on AMP. The primary table is still missing. The administrator needs to perform some interactive recovery procedures to fully recover missing data for table Y.

The administrator will be unsuccessful if he/she attempts to access the row information from table Y. The following message may appear in response to an attempted SQL statement:

2642 AMP Down: The request against non-fallback Table_Y cannot be done.

Recovery Action

The administrator must perform the following steps to fully recover table Y:

1. Perform an all-AMP RESTORE using Tuesday's ARCHIVE of table Y to restore all data rows stored in the archive file from table Y.

2. Do NOT release the utility locks.

3. Restore Wednesday's ARCHIVE of journal Y.

4. Perform a all-AMP ROLLFORWARD using the RESTORED journal from table Y. Doing so replaces the existing rows in table Y with any after-change images made since the last backup on Tuesday.

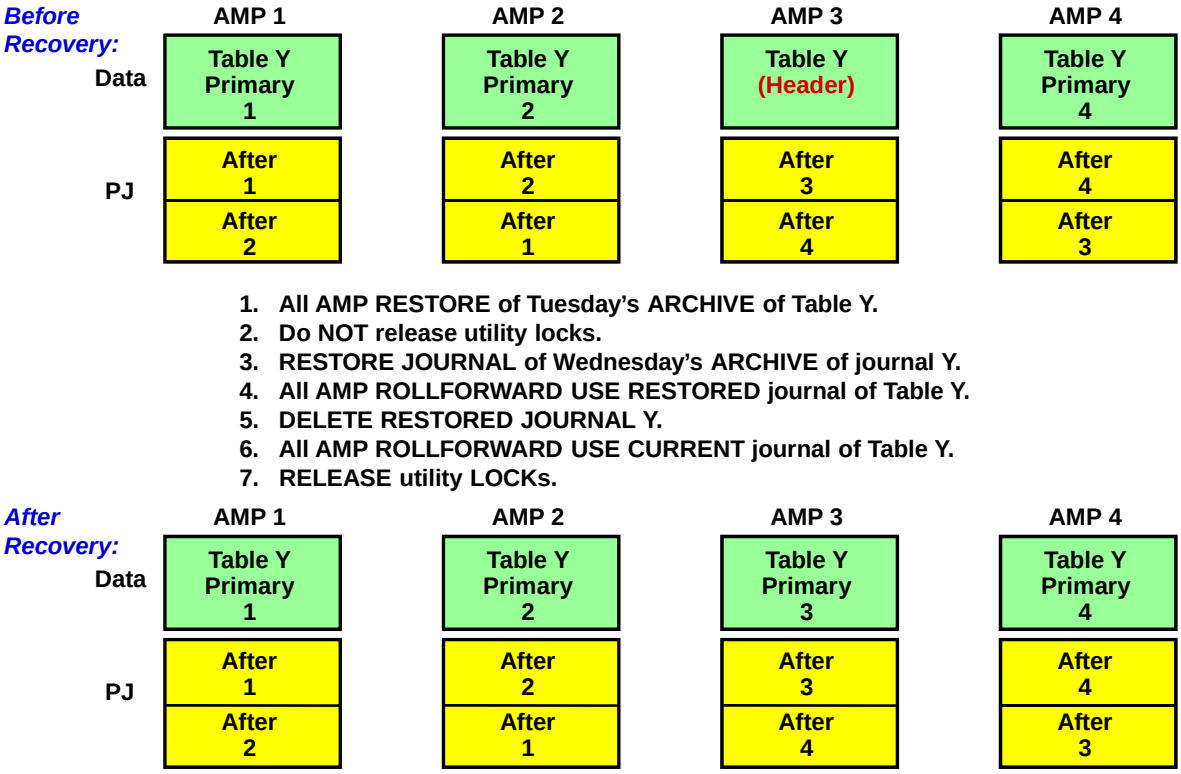
5. Use the DELETE JOURNAL command to delete restored journal Y. This action deletes all stored images from the restored journal.

6. Perform an all-AMP ROLLFORWARD using the CURRENT journal from table Y. This step replaces existing table rows with any after-change images stored in the active and/or saved subtables of the permanent journal.

7. RELEASE all utility locks.

Table Y is now fully recovered. All its contents are now available to users.

Table Y Recovery



1. All AMP RESTORE of Tuesday's ARCHIVE of Table Y.
2. Do NOT release utility locks.
3. RESTORE JOURNAL of Wednesday's ARCHIVE of journal Y.
4. All AMP ROLLFORWARD USE RESTORED journal of Table Y.
5. DELETE RESTORED JOURNAL Y.
6. All AMP ROLLFORWARD USE CURRENT journal of Table Y.
7. RELEASE utility LOCKs.

Table Z Recovery

The first diagram on the facing pace illustrates table Z after REBUILD and RESTART procedures.

Neither permanent journal tables stored on AMP 3 were restored. In addition, the primary table information is still missing. The administrator needs to perform some interactive recovery procedures to fully recover the missing data for table Z.

Recovery Action

The administrator must perform the following steps to fully recover table Z:

1. Perform an all-AMP RESTORE using Wednesday's ARCHIVE of table Z to restore all data rows stored in the archive file from table Z. The administrator does not restore the journal tables for table Z since a complete backup of the table was performed on the same day as the journal archive. All changes through Wednesday would be in the archive of the entire table.

2. The administrator does NOT release the utility locks.

3. Perform an all-AMP ROLLFORWARD using the CURRENT journal from table Z. This action replaces existing table rows with any after-change images stored in the active and/or saved subtables of the permanent journal. Any changes in the current journal would have occurred on Thursday before the disk failure.

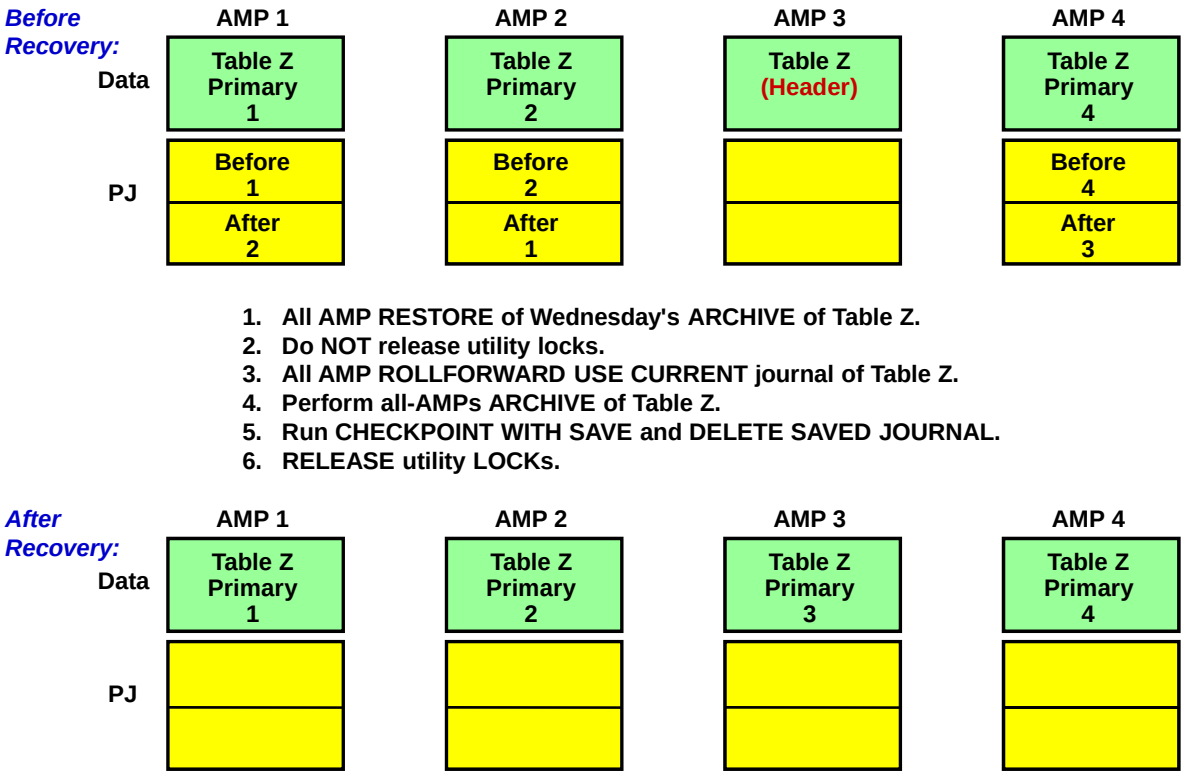
4. Perform an all-AMPs archive of table Z to protect against a second disk failure in the same cluster. The administrator is unable to restore the journal for AMP 3 because he/she did not elect dual images. Another disk failure in this cluster leaves data unrecoverable. To correct this, the administrator deletes the saved journal and starts a new journal.

5. Perform a CHECKPOINT WITH SAVE and DELETE SAVED JOURNAL. The CHECKPOINT step moves any stored images from the active subtable to the saved subtable of the current journal and initiates the active subtable. The DELETE step erases the contents of the saved subtable since they are no longer needed.

6. RELEASE all utility locks.

Table Z is now fully recovered. All its contents are now available to users. Notice that the table is recovered but the journals are not.

Table Z Recovery



1. All AMP RESTORE of Wednesday's ARCHIVE of Table Z.
2. Do NOT release utility locks.
3. All AMP ROLLFORWARD USE CURRENT journal of Table Z.
4. Perform all-AMPs ARCHIVE of Table Z.
5. Run CHECKPOINT WITH SAVE and DELETE SAVED JOURNAL.
6. RELEASE utility LOCKs.

After Recovery

The diagram on the facing page shows the three tables after recovery. The following summary outlines the effects of permanent journals on recovery from a single disk failure.

Fallback Tables, Dual Image Tables (Table X)

- Processing continues
- Journals play no part in recovery

No Fallback Tables, Dual Image Journals (Table Y)

- Limited processing continues
- Data and journal tables are fully recovered

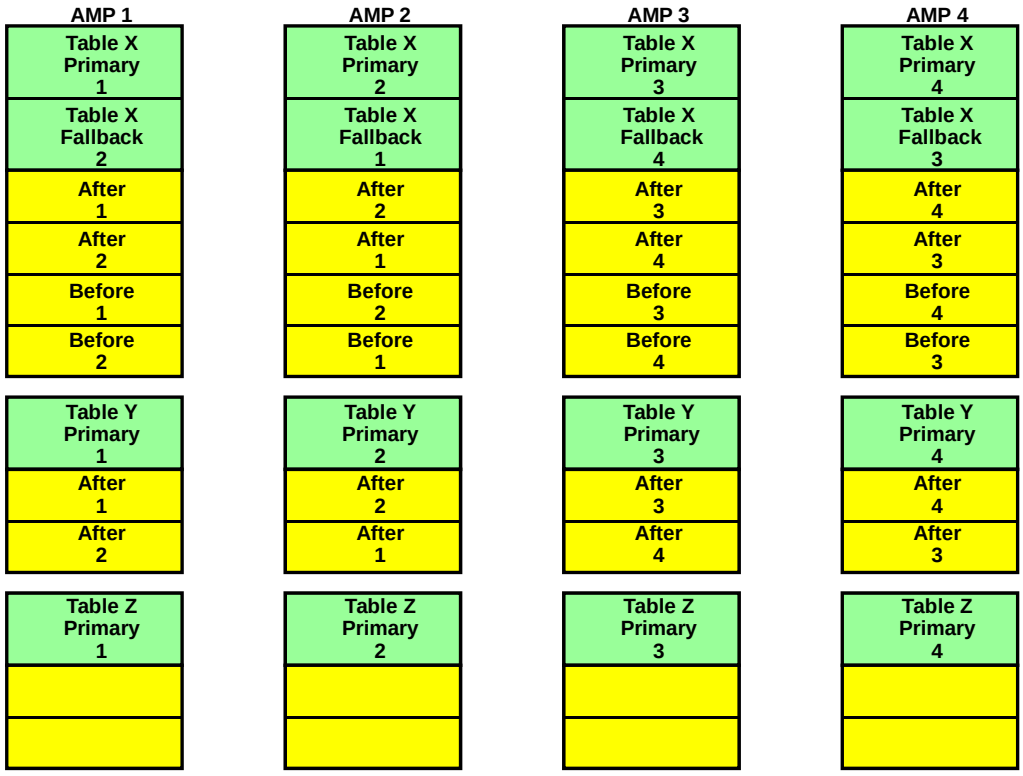
No Fallback Tables, Single Image Journals (Table Z)

- Limited processing continues
- Data is fully recovered
- Journals are lost

No Fallback Tables, No Journals

- Limited processing continues
- The administrator can only recover data to the point of the last archive

After Recovery



Summary

The facing page contains some useful concepts on how permanent journals operate during recovery.

Summary

Fallback Tables	**Data is fully recoverable.**
Dual Image Journals	**Journals play no part in recovery.**
No Fallback Tables	**Data is partially available.**
Dual Image Journals	**Data and journals are fully recoverable.**
No Fallback Tables	**Data is partially available.**
Single Image Journals	**Data is recoverable, but journals are lost.**
No Fallback Tables	**Data is partially available.**
No Journals	**Data can be recovered only to the point of the last archive.**

Notes

Module 58

TERADATA
Raising Intelligence

Archiving Data

After completing this module, you will be able to:

- **Understand how to use the ARC facility to back up data on external media.**
- **State the access privileges needed to execute Archive and Recovery statements.**
- **Identify the kind of utility locks placed during archive and recovery procedures, and use statements to release the locks when appropriate to do so.**
- **Understand the syntax and the restrictions when archiving selected partitions of a PPI table.**

Notes

Table of Contents

Archive and Recovery Utility (ARC) 58-4
Common Uses for ARC 58-4
Archive and Recovery Phases 58-6
Restore versus FastLoad 58-8
ARC 58-10
NetBackup 58-10
Session Control 58-12
Multiple Sessions 58-14
ARC Statements 58-16
ARCHIVE Statement 58-18
ARCHIVE Examples 58-20
ARCHIVE Examples (cont.) 58-22
Archiving Selected Partitions of PPI Table 58-24
Considerations 58-24
ARCHIVE Partition Example 58-26
ANALYZE Statement 58-28
ANALYZE Output 58-30
Archive Objects 58-32
Archive Objects (cont.) 58-34
Single Database Archive 58-34
Database ALL Archive 58-34
Single or Multiple Table Archives 58-34
EXCLUDE Option 58-34
Archive Levels 58-36
Multi-Stream Archives 58-36
Cluster Archives 58-36
Archive Options 58-38
ONLINE Archive Option 58-40
BakBone NetVault Example 58-42
Database DBC Archive 58-44
RESTORE Considerations 58-44
Summary 58-46
Module 58: Review Questions 58-48

Archive and Recovery Utility (ARC)

The basic function of the Archive and Recovery (ARC) utility is to back up and optionally restore databases and database objects (e.g., tables, views, macros, stored procedures, etc.). The ARC utility performs four major tasks:

- Archive
- Restore
- Copy
- Recovery

The archive task dumps information from the Teradata system onto some type of portable storage media. The restore function reverses the archive process and moves the data from the storage media back to the database. The copy feature allows you to copy data from one system onto another system. The recovery feature utilizes information stored in permanent journals to rollback or rollforward row information.

Common Uses for ARC

The Teradata system provides a number of automatic data protection features. However, these features do not cover all types of data loss. The ARC utility provides additional data protection for the situations listed below:

- Loss of an AMP's Vdisk for no fallback tables
- Loss of multiple AMPs in the same cluster
- Failed batch processes
- Accidentally dropped tables, views, or macros
- Miscellaneous user errors
- Disaster recovery

You can use Teradata ARC to do the following:

- Archive a database, individual table, or selected partitions of a PPI table from a Teradata Database to a client resident file.

- Restore a database, individual table, or selected partitions of a PPI table back to a Teradata Database from a client resident archive file.

- Copy an archived database, table, or selected partitions of a PPI table to a Teradata Database on a different hardware platform than the one from which the database or table was archived.

- Place a checkpoint entry in a journal table.

- Recover a database to an arbitrary checkpoint by rolling it back or rolling it forward, using change images from a journal table.

- Delete change image rows from a journal table.

Archive and Recovery Utility (ARC)

Major tasks or functions of the ARC facility include:

Archive – captures user data on portable storage media.
Restore – restores data from portable storage media.
Copy – transfer archived data to another system or optionally back to same system
Recovery – recovers changes to data from permanent journal tables.

ARC provides additional data protection for these situations:

- Loss of an AMP's Vdisk for no fallback tables
- Loss of multiple Vdisks (AMPs) in the same cluster
- Failed batch processes
- Accidentally dropped tables, views or macros
- Miscellaneous user errors
- Disaster recovery

Common uses for ARC:

- Archive a database, individual table, or selected partitions of a PPI table.
- Restore a database, individual table, or selected partitions of a PPI table.
- Copy an archived database, table, or selected partitions of a PPI table to a Teradata Database on a different system.

Archive and Recovery Phases

Archive or recovery jobs always operate in two phases. The steps of each phase are described on the facing page.

The archive process is intensive. You may want to create a user just for archive activities so that you can use your user ID to perform other actions while archive is running.

Teradata ARC creates files when you archive databases, individual data tables, selected partitions of primary partition index (PPI) tables, or permanent journal tables from the Teradata Database. You provide Teradata ARC with such files when you restore databases, individual data tables, partitions of tables, or permanent journal tables back to the Teradata Database.

Teradata ARC also includes recovery with rollback and rollforward functions for data tables defined with a journal option. Moreover, you can checkpoint these journals with a synchronization point across all AMPs, and you can delete selected portions of the journals.

Archive, Restoration, and Recovery Phases

Phase 1 — Dictionary Phase

1. Allocate an event number (from DBC.Next).
2. Issue a BEGIN TRANSACTION statement.
3. Resolve object name.
4. Check access rights.
5. Place locks:
   - Utility locks on data dictionary rows.
   - Utility locks on data rows.

   Note: READ locks on ARCHIVE; EXCLUSIVE locks on RESTORE.
6. Delete existing tables prior to RESTORE.
7. Issue an END TRANSACTION statement.

Phase 2 — Data Phase

1. Issue a BEGIN TRANSACTION statement.
2. Insert rows into RCEVENT and RCCONFIGURATION.
3. Perform the operation.
4. Update RCEVENT.
5. Release locks (if user specified).
6. Issue an END TRANSACTION statement.

Restore versus FastLoad

You could consider running a FastLoad utility job to restore the information to disk. This would mean that instead of archiving to tape, you have used BTEQ EXPORT or some other means to put the information into a host file for the FastLoad utility. FastLoad requires an empty table.

FastLoad Steps

Steps involved with FastLoad include.

- FastLoad uses a single session to send the INSERT statement to the PE and AMP vprocs.
- Multiple sessions are then used to facilitate sending rows to the AMP vprocs.
- Upon receipt, each AMP vproc hashes each record and redistributes it over the BYNET. This is done in parallel.
- The receiving AMP vproc then writes these rows directly to the target table as unsorted blocks.
- When loading completes, each AMP vproc sorts the target table, puts the rows into blocks, and writes the blocks to disk.
- Then, fallback rows are generated if required. FastLoad operates only on tables with no secondary indexes.
- You have to create any required indexes when the FastLoad is complete.

Restore Steps

Restoring to the same configuration includes:

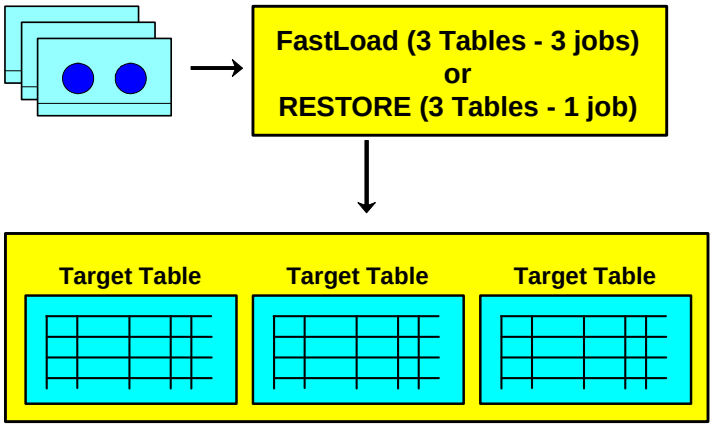
- Recovery of data blocks to the AMP vproc.
- The blocks are already in the appropriate format.

Restoring to a different configuration includes:

- The block is first sent to the AMP vproc in the old configuration.
- Then, it strips off its own rows and forwards (redistributes) the remainder of the block to the AMP vproc for the new configuration. Since the original rows were sorted in data blocks by RowID, the result is usually much faster than a normal redistribution.

ARC is the easiest and fastest to restore a very large number of objects. FastLoad operates on a table-by-table basis, while ARC can restore an entire machine with one simple command.

Restore versus FastLoad



FastLoad is a very fast loader, but not as fast as RESTORE in rebuilding a table. Why?

- FastLoad has to hash each row, redistribute every row, collect and write to disk, then read, sort, and write back to disk.
- RESTORE copies blocks to the appropriate AMPs.

Which is easier?

- FastLoad operates on a table by table basis (one at a time).
- RESTORE can restore all of the tables for one or more databases with a single job.

Even if rebuilding 1 table, RESTORE is faster than FastLoad.

ARC

Teradata Database offers a wide variety of utilities, management tools, and peripherals. Some of these reside on Teradata Database and others are part of the Teradata Tools and Utilities management suite available for installation in client environments. With database management tools, you can back up and restore important data, save dumps, and investigate and control Teradata Database configuration, user sessions, and various aspects of its operation and performance. Management and analysis tools help keep the database running at optimum performance levels.

Teradata Backup, Archive, and Restore (BAR) supports third party software products that provide data archiving, backup, and restore functions. Teradata utilizes software extensions called TARA (Tiered Archive Restore Architecture) and plug-ins that help connect BAR software to the Teradata Database.

There are several ways to invoke the Archive facility.

- NetBackup – TARA
- Tivoli Storage Manager – TARA
- NetVault
- Command Line (arcmain)
- Host or Mainframe

Teradata Archive/Recovery utility (ARC), working with BAR application software, writes and reads sequential files on a Teradata client system to archive, restore, recover, and copy Teradata Database table data. Through its associated script language, it also provides an interface between Teradata Backup Application Software solutions and Teradata Database.

NetBackup

Symantec™ NetBackup and NetBackup Extension for Teradata NetBackup Extension for Teradata is an access module that enables NetBackup to work with Teradata Database. Administrators can schedule automatic, unattended backups for client systems across a network. It supports parallel backups and restores coordinated across multiple hosts in a single Teradata Database. Teradata TARA in the NetBackup framework and is comprised of three components: TARA Server, TARA GUI, and NetBackup Extension for Teradata.

ARC

The ARC facility is required to archive/restore/copy the Teradata Database.

- **ARCMAIN is the program name of the Teradata ARC utility.**
- **ARCMAIN is normally executed in batch mode, but it can be run interactively.**
- **Required dependency of the BAR (Backup and Recovery) backup products.**
- **ARC version must match the Teradata DBS version.**

Teradata Backup, Archive, and Restore (BAR) supports third party software products that provide data archiving, backup, and restore functions. Teradata utilizes software extensions called TARA (Tiered Archive Restore Architecture) and plug-ins that help connect BAR software to the Teradata Database.

The BAR application software offering includes:

- **NetBackup (from VERITAS software) – utilizes TARA**
- **Tivoli Storage Manager – utilizes TARA**
- **NetVault (from BakBone software)**
- **Command Line (execute arcmain)**
- **Host or Mainframe**

Utilities such as NetBackup allow you to create scripts, schedule jobs, and provide various tape management capabilities.

Session Control

To use the ARC utility, you must use the LOGON statement to logon to the Teradata system before you can execute other ARC statements. The user ID with which you log on has to have access rights for the ARC statements that you want to use.

The facing page shows the LOGON and LOGOFF statements.

Since the archive process can be intensive, you may want to create a user just for archiving to free your user ID for other processes while archive is running.

In general, the amount of system resources (that is, memory and processing power) that are required to support the archive or recovery increases with the number of sessions. The impact on a particular system depends on the specific configuration.

Teradata ARC uses two control sessions to control archive and recovery operations. The LOGON statement always connects these sessions no matter what type of operation being performed. Teradata ARC connects additional data sessions based on the number indicated in the SESSIONS parameter. These sessions are required for the parallel processing that occurs in archive and restore or copy operations.

If additional data sessions are required, Teradata ARC connects them at one time. Teradata ARC calculates the number of parallel sessions it can use, with maximum available being the number of sessions indicated with this parameter. Any connected sessions that are not actually used in the operation result in wasted system resources.

To request a specific number of sessions, the "SESSIONS=x*nn*" runtime parameter can be used. If not specified, the number of sessions defaults to 4 plus 2 control sessions for a total of 6 sessions.

The SESSIONS parameter specifies the number of Teradata Database sessions that are available for archive and recovery operations. This number does not include any additional sessions that might be required to control archive and recovery operations.

Session Control

The LOGON statement:

1. Causes two sessions to be logged on: one for SQL statements, and one for control requests.

   When it encounters an ARCHIVE or RESTORE command, ARC starts additional data sessions requested in the ***SESSIONS=nnn*** runtime parameter.

2. Identifies the user and account to charge for used resources.

3. Identifies the user to the Teradata database system so that the software may verify ownership or check access rights. The system verifies access rights as it executes each statement.

CHECKPOINT	Permits you to execute both the SQL and ARC utility checkpoint statements.
DUMP	Permits you to execute the ARC Archive statement
RESTORE	Permits you to execute the following ARC statements:
	Restore Delete Journal Rollforward Release Lock* Rollback Build

The LOGOFF statement:

1. Ends all Teradata sessions logged on by the task, and
2. Terminates the utility.

* To release a lock held by another User, you must specify Override and hold DROP privileges on the underlying objects.

Multiple Sessions

You can specify the number of archive and/or recover sessions with which to work, or use the default. To set the number, use the SESSIONS runtime parameter.

For small systems (e.g., less than 40 AMPs), the recommended number of sessions is:

- One per AMP vproc for archive.
- Two per AMP vproc for recovery.

The number of sessions to use can vary based on a number of factors. Several are described below.

The description on the facing page tells more about how the vprocs use the sessions.

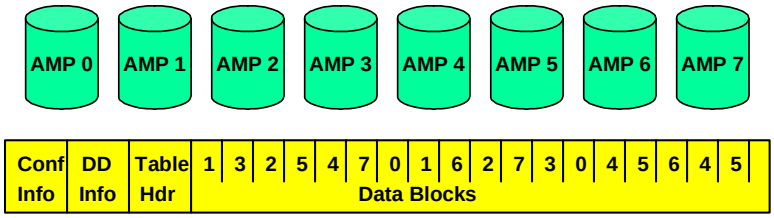
If fewer than one session per vproc is specified for the archive:

- For vproc groups, archive/recovery will archive blocks from each group with each vproc completed before the next starts.
- In this case, a large number of sessions allocated to recovery will not help recovery performance.

For larger configurations, say over 100 AMP vprocs, specifying one session per AMP will not increase performance because of other limiting component(s).

In this case, for maximum throughput, cluster level operation is recommended with one session per AMP for involved AMPs. For example, if the system has 50 clusters with 4 AMPs each, you can partition it into two jobs with 25 clusters each and 100 sessions per job provided that your site has two (or more) tape drives available and enough host resources to run two jobs in parallel.

Multiple Sessions



The appropriate number of sessions depends on …

- number of AMPs
- number of channel or LAN connections
- speed and type of tape subsystem

For small systems, 1 session per AMP ensures all data blocks from all AMPs are evenly distributed.

- Teradata assigns each session to a vproc. All sessions stay with that vproc until all required data is archived. Then will it be moved to another vproc if necessary.
- Archive attempts to build blocks from each vproc in turn. The blocks are composed of complete database blocks.
- Data blocks from different vprocs are never mixed within the same archive block.

ARC Statements

The ARC utility contains a number of commands to perform archive, restore, and recovery tasks. Some of the commands are shown on the facing page.

Additional ARC options can be set via runtime parameters. The following environment variables are used with ARC.

ARCDFLT – this is the environment variable that points to the file containing the system-wide default parameters values.

Example: SET ARCDFLT=C:\TESTARC\CONFIG.ARC

The file CONFIG.ARC would include valid runtime parameters. For example:

```
SESSIONS=8
RESTARTLOG=C:\TEMP\arcrlog1
```

ARCENV – this is the environment variable that specifies any valid Teradata ARC runtime parameters.

Example: SET ARCENV=RESTARTLOG=C:\TEMP\arcrlog2

ARCENVX – same as ARCENV, except that ARCENVX has the highest override priority. Any runtime parameter set in ARCENVX is guaranteed to be used.

Examples of typical runtime parameters that can be used include:

RESTARTLOG (or RLOG) = *filename*

The RESTARTLOG runtime option is available only on Windows and MP-RAS platforms.

Teradata ARC adds the extension type RLG to the name of the file specified in RESTARTLOG. Therefore, do not use this extension in the name of the restart log.

The restart log is created under the current directory or the working directory, if defined, unless the full path is specified.

Teradata ARC does not automatically remove the restart log files after the successful completion; therefore you may need to clean the files periodically.

SESSIONS = *nnn*

Two additional control sessions are automatically added.

ARC Statements

LOGON	Begins a session.
LOGOFF	Ends a session.
ARCHIVE	Archives a copy of a database or table to a host-resident data set/file.
ANALYZE	Reads an archive tape to display information about its content.
RESTORE	Restores a database or table from a archive file to specified AMPs.
COPY	Restores a copy of an archived file to a specified Teradata database system.
BUILD	Builds indexes and fallback data.
RELEASE LOCK	Releases host utility locks on databases or tables.
DELETE DATABASE	Deletes a database.
CHECKPOINT	Marks a journal for later archive or recovery activities.
ROLLBACK	Recovers a database and tables to a state that existed *before* some change.
ROLLFORWARD	Recovers a database or table to a state that existed *after* some change.
DELETE JOURNAL	Deletes SAVED or RESTORED Journal rows.
REVALIDATE REFERENCES	Revalidate referential integrity; a housekeeping or cleanup function.

ARCHIVE Statement

The ARCHIVE statement allows you to backup database objects to host media (usually magnetic tape). The format for this statement is shown on the following page.

> **Note:** ARCHIVE is the preferred term as the DUMP command is supported only for backward compatibility.

The ACHIVE control statement allows you to specify the archive:

- Type
- Objects
- Levels
- Options

The EXCLUDE option allows you to specify an alphabetical listing of database/user names that you want excluded. The values do NOT have to match existing database/user names. For example, you could EXCLUDE (A) TO (D).

Referential Integrity

Tables with *unresolved* referential integrity constraints cannot be archived. An unresolved constraint occurs when a CREATE TABLE (child) statement references a table (parent) that does not exist. Create the parent table (use the SQL command CREATE TABLE). Effectively, create the referenced table to resolve these constraints.

ARC and HASH/JOIN Indexes

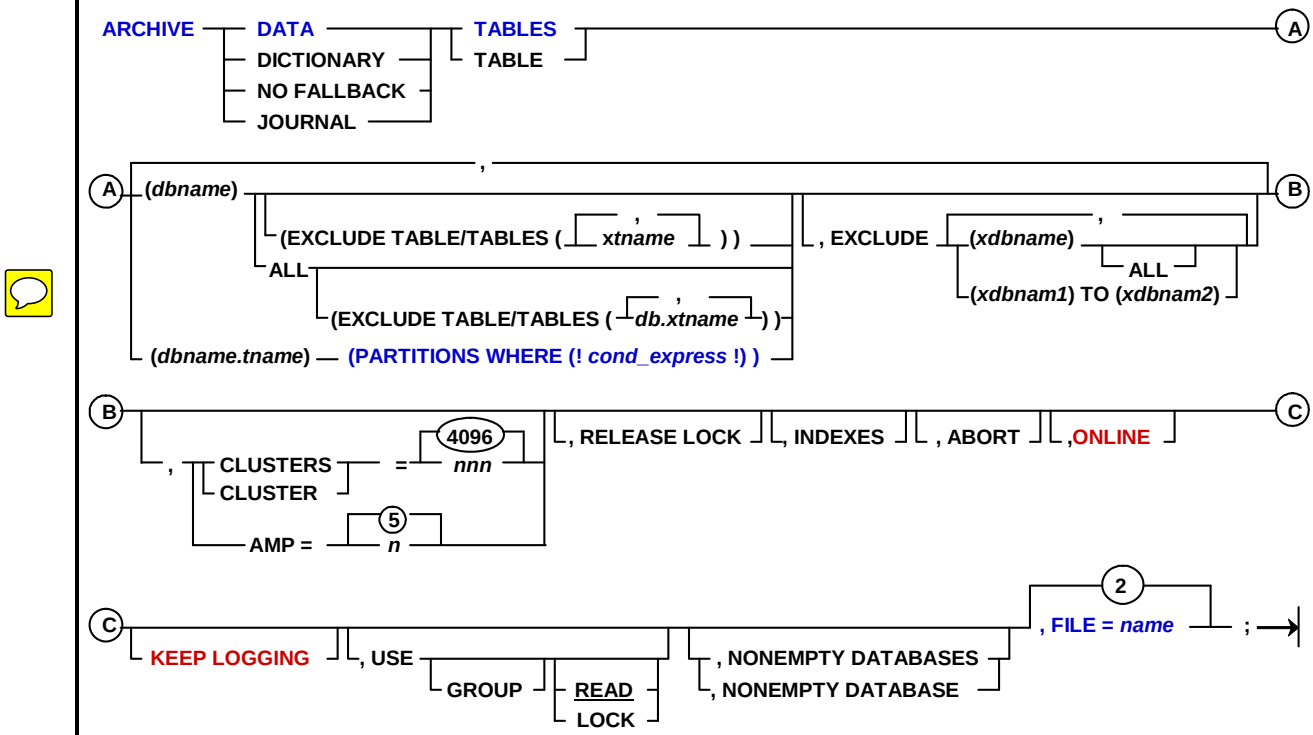
Prior to Teradata 13, ARC cannot be used to archive or restore Hash and/or Join Indexes. However, prior to Teradata 13.0, you are permitted to archive a base table or database that has an associated Hash or Join Index defined.

The output of the SHOW HASH INDEX or the SHOW JOIN INDEX statement includes a special status message if a Join Index has been marked invalid.

PARTITIONS WHERE

This option specifies the conditional expression for selecting partitions. If the condition selects a partial partition, the entire partition is archived.

ARCHIVE Statement



ARCHIVE Examples

Examples of the Archive scripts are shown on the facing page.

EXCLUDE TABLE OPTION Details

This option is only accepted in a database level object in a DATA TABLES of all amps or cluster operations. A database level object that has one or more excluded tables is a partial database. An archive of a partial database contains the dictionary info and the table header row of the excluded table but actual data rows are excluded, i.e., not archived.

On the restore side, if a partial database archive is restored, no data rows will be restored for the excluded tables. If the table is excluded state, ARC restores the dictionary info and the table header row, but leaves the table in restore state. This protects the table from other application's attempt to access it before the table level restore is performed. Table level restore for the excluded tables is expected to follow the partial database restore to fully restore a partial database. If the intention of the user is to really exclude the table, the user has an option to run an explicit BUILD statement for the excluded tables. The excluded tables become accessible and are empty; they can then be dropped.

If ALL keyword is specified after the object name, then only fully qualified table names in the form of databasename.tablename is accepted in the list of EXCLUDE TABLES. If ALL is not specified then a fully qualified table name cannot be entered in the list of EXCLUDE TABLES, i.e., database names cannot be prefixed.

EXCLUDE TABLES cannot be used with the following options. ARC0215 error message will be issued if any of these conditions are detected.

Table level object: (databasename.tablename)
DICTIONARY, JOURNAL, NO FALLBACK
AMP= or PN=

ARC0106: "User excluded table(s) (%s) does/do not belong to database %s"
This error is issued when a table specified in EXCLUDE TABLE list is not part of the database object. The object will be aborted, the database will be skipped and the next database will be processed.

By default, Teradata ARC counts output sectors and output rows. The row count is the number of primary data rows archived when you specify the DATA or NO FALLBACK option. Both counts are included in the output listing.

ARCHIVE Examples

archive1_pd.arc

```
LOGON dbc/sysdba,dbapass;
ARCHIVE DATA TABLES (PD)
  , ABORT
  , RELEASE LOCK
  , FILE = arc1_PD;
LOGOFF;
```

archive2_pd.arc

```
LOGON dbc/sysdba,dbapass;
ARCHIVE DATA TABLES (PD)
    (EXCLUDE TABLES (dept_summary, phone_summary))
  , ABORT
  , RELEASE LOCK
  , FILE = arc2_PD;
LOGOFF;
```

arcmain < archive2_pd.arc

Portion of output from executing above script

```
ARCHIVING DATABASE "PD"
TABLE "Department" -   3,446 BYTES, 60 ROWS ARCHIVED
TABLE "Dept_Summary" - EXCLUDED BY USER
TABLE "Employee" -  65,077 BYTES,   1,000 ROWS ARCHIVED
TABLE "Emp_Phone" -  52,504 BYTES,   2,000 ROWS ARCHIVED
TABLE "GT_Deptsalary" - 548 BYTES, 0 ROWS ARCHIVED
TABLE "Job" -   2,898 BYTES, 66 ROWS ARCHIVED
VIEW  "LargeTableSpaceTotal" - ARCHIVED
TABLE "Phone_Summary" - EXCLUDED BY USER
MACRO "SetAnsiDate_OFF" - ARCHIVED
MACRO "SetAnsiDate_ON" - ARCHIVED
"PD" - LOCK RELEASED
DUMP COMPLETED
```

ARCHIVE Examples (cont.)

Additional examples of Archive scripts are shown on the facing page.

In the second example, Demo, Guest_Users, and Sandbox are databases or users that will be excluded from the archive.

EXCLUDE TABLE Caution

When you do a full database-level restore of an archive with excluded tables, the data dictionaries and the table headers of *all* tables, including excluded tables, are replaced.

As a result, *all* of the existing rows in the excluded tables are deleted.

You can restore individual tables from a database-level archive with excluded tables. In the RESTORE statement, you must individually specify all the tables you want to restore, except the excluded tables. By omitting the excluded tables, you preserve the data dictionaries and table headers of the excluded tables. That way you can restore the database from the archive without altering the excluded tables.

However, you cannot name macros, views, or stored procedures as objects in your RESTORE statement. So if you create an archive with excluded tables and you want to preserve the excluded tables, you cannot recover the macros, views, or stored procedures from the archive.

Archiving Large Objects (LOBs) Notes

Teradata ARC also supports the archive operation for tables that contain large object columns as long as the database systems are enabled for large object support. However, large object columns cannot be restored on a system that uses a hash function that is different than the one used for the archive.

An archive of selected partitions with LOBs is supported, but the restore is not. To restore selected partitions of LOBs, perform a full-table restore.

ARCHIVE Examples (cont.)

archive3_ds.arc

```
LOGON dbc/sysdba,dbapass;
ARCHIVE DATA TABLES (DS)
  , ABORT , RELEASE LOCK, ONLINE
  , FILE = arc3_DS;
LOGOFF;
```

This script archives DS and allows writes to tables in the database by using the ONLINE Archive option.

archive4_sysdba.arc

```
LOGON dbc/sysdba,dbapass;
ARCHIVE DATA TABLES
    (Sysdba) ALL
         (EXCLUDE TABLES  (PD.dept_summary, PD.phone_summary))
        , EXCLUDE              (Demo), (Guest_Users) ALL, (Sandbox)
  , ABORT , RELEASE LOCK
  , FILE = arc4_Sys;
LOGOFF;
```

This script archives Sysdba and all of its child databases/users and excludes some tables and databases.

archive5_DBC.arc

```
LOGON dbc/dbc,dbcpass;
ARCHIVE DATA TABLES (DBC) ALL
  , RELEASE LOCK
  , FILE = arc5_DBC;
LOGOFF;
```

This script archives DBC and all of its child databases/users.

(ABORT is not a valid option when archiving data dictionary tables.)

Archiving Selected Partitions of PPI Table

Starting with Teradata Database V2R6.0, you can perform an all-AMPs archive on one or more partitions of a table rather than performing a full-table backup. The ability to select partitions from PPI tables is limited to all-AMP archives. Dictionary, cluster, and journal archives are not supported.

Use archive partitioning to accomplish the following tasks:

- Archive only a subset of data (this can minimize the size of the archive and improve performance).
- Restore data in a table that is partially damaged.
- Copy a limited set of data to a disaster recovery machine or to a test system.

Considerations

Consider the following when archiving selected partitions in PPI tables:

- Archiving selected partitions operates on complete partitions within tables, meaning that the selection of a partial partition implies the entire partition.
- A restore operation always deletes the selected partitions of the target table before restoring the rows that are stored in the archive.
- PPI and non-PPI tables are allowed in a single command. This allows you to manage both table types in a single database with the EXCLUDE TABLES option.
- Partitioning is based on one or more columns specified in the table definition.
- Partition elimination restricts a query to operating only in the set of partitions that are required for the query.
- Incremental archives are possible by using a partition expression that is based on date fields, which indicate when a row is inserted or updated.
- An archive or restore of selected partitions only places full-table locks. Locks on individual partitions are not supported.
- It is recommended that you re-collect table statistics after a restore of selected partitions because statistics are part of the table dictionary rows, which are not restored during a partition-level restore.
- If a table has a partitioning expression that is different from the partitioning expression used in the PPI archive, a PPI restore is possible as long as no other significant DDL changes are made to the table.

Archiving Selected Partitions of PPI Table

You can perform an all-AMPs archive on one or more partitions of a table rather than performing a full-table backup and restore.

- Archiving selected partitions is limited to all-AMP archives.
- Dictionary, cluster, and journal archives are not supported.

Considerations:

- Archiving selected partitions operates on complete partitions within tables.
  - Defining a partial partition means that the entire partition will be archived.
  - A restore operation always deletes the selected partitions of the target table before restoring the rows that are stored in the archive.
- An archive or restore of selected partitions only places full-table locks. Locks on individual partitions are not supported.
- Re-collect table statistics after a restore of selected partitions because statistics are part of the table dictionary rows, which are not restored during a partition-level restore.

ARCHIVE Partition Example

An example of an Archive script that archives partitions of a PPI table is shown on the facing page.

The table definition for the Sales_PPI table is:

```
CREATE SET TABLE TFACT.Sales_PPI
    (store_id          INTEGER NOT NULL,
     item_id           INTEGER NOT NULL,
     sales_date        DATE FORMAT 'YYYY-MM-DD',
     total_revenue     DECIMAL(9,2),
     total_sold        INTEGER,
     note              VARCHAR(256) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX (store_id, item_id)
PARTITION BY RANGE_N (sales_date BETWEEN DATE '2002-01-01' AND DATE
'2011-12-31' EACH INTERVAL '1' MONTH );
```

Additional Notes when Archiving Partitions

Bounding condition – is well-defined if the PARTITION BY expression on the source table consists of a single RANGE_N function, and if the specified range does not include NO RANGE or UNKNOWN.

Use Correct Specifications – the incorrect use of specifications may cause the following problem. An incorrect PARTITIONS WHERE specification during backup can result in an incomplete archive or difficulties during a restore operation.

Restrict Updates to Active Partitions – it is not possible for the ARC facility to determine which partitions have been modified since the last backup. If changed partitions are not re-archived, the changes are lost when restored.

For example, if, for a given table, the backup strategy is to only backup the active (latest) partition of the table, and a change is made to a non-active partition (to fix an incorrect update), the change is not archived unless you run a separate archive of the changed partitions.

The remedy for this situation is either to restrict updates to the active partitions only (by using views to control which rows/partitions are updated) or to re-archive all modified partitions.

ARCHIVE Partition Example

archive6_ppi.arc

```
LOGON dbc/sysdba,dbapass;
ARCHIVE DATA TABLES
   (TFACT.Sales_PPI) (PARTITIONS WHERE (!Sales_Date BETWEEN '2011-10-01' AND '2011-12-31'!))
  , ABORT, RELEASE LOCK, FILE = arc6_PPI;
LOGOFF;
```

arcmain < archive6_ppi.arc

Portion of output from executing above script

```
ARCHIVE DATA TABLES (TFACT.Sales_PPI) (PARTITIONS WHERE (!Sales_Date BETWEEN '2011-10-01' AND '2011-12-
31'!))
,ABORT, RELEASE LOCK, FILE = arc6_PPI;
UTILITY EVENT NUMBER  - 36
LOGGED ON    4 SESSIONS
ARCHIVING DATABASE "TFACT“

Archive Bounding Condition:
RANGE_N("TFACT"."SALES_PPI"."sales_date"  BETWEEN DATE '2002-01-01' AND DATE '2011-12-31' EACH INTERVAL
'1' MONTH ) IN (118 TO 120 )

[Bounding condition is well-defined]
TABLE "Sales_PPI" -   2,391,241 BYTES,  40,500 ROWS ARCHIVED
"TFACT"."SALES_PPI" - LOCK RELEASED
DUMP COMPLETED
```

ANALYZE Statement

The ANALYZE statement reads data from an archive tape and displays information about tape contents. When you invoke the statement, you can choose a specific database or a range of databases from which to display information. This information will help you if you are trying to restore a specific database instead of the entire archive set. This statement does not require a prior logon.

The ANALYZE statement provides the following information about the database(s) you specify:

- Time and date of the archive operation
- The archive level: all-AMPs; clusters of AMPs; or specific AMPs
- The name of each database, data table, journal table, view, and macro in each database and the fallback status of the tables. Information appears only if you use the keyword LONG with the DISPLAY option.
- If an archive file contains a selected partition archive of a table, the bounding condition used to select the archived partitions is displayed with an indication as to whether the bounding condition is well-defined.

The CATALOG option (not shown on facing page) generates/rebuilds the CATALOG table in the CATALOG database.

DISPLAY Option

If no option is listed, display is the default. It shows the time, date and level of the archive. If you use the LONG option, the display includes the names of all tables, views, macros, triggers, or stored procedures.

VALIDATE Option

This option reads each archive record in the specified database. It checks that each data block in the file can be read but does not check whether the data block read has valid rows or not, i.e., it does not check anything inside the data block record. It only checks whether or not the data block record can be read.

You can specify both the DISPLAY and VALIDATE options on a single ANALYZE statement.

ANALYZE Statement

Format:

```
ANALYZE    [ * | ALL | [ (Databasename) | (Dbname1) TO (Dbname2) ] [, ...]
           [ , DISPLAY [ LONG] | , VALIDATE ]
           , FILE = name ;
```

Notes:

- The ANALYZE statement instructs the ARC utility to read an archive file and display information about its content.
- The LONG option displays all table, view, macro, trigger, and stored procedure names.
  - If an archive file contains a selected partition archive of a table, the bounding condition used to select the archived partitions is displayed.
- The VALIDATE option reads each record to check that each block on the archive file is readable.
- ANALYZE doesn't require a LOGON or LOGOFF statement.

Example: analyze1_pd.arc (script name)

```
ANALYZE (PD), DISPLAY LONG, FILE = arc1_PD;
```

To execute: arcmain < analyze1_pd.arc

ANALYZE Output

An example of the output from the ANALYZE command is shown on the facing page.

ANALYZE Output

Output from ...

```
ANALYZE (PD),
    DISPLAY LONG,
    FILE = arc1_PD;
```

```
   :
01/13/2011 11:30:02  CHARACTER SET IN USE: ASCII
01/13/2011 11:30:02  ANALYZE (PD),
01/13/2011 11:30:02  DISPLAY LONG,
   :
01/13/2011 11:30:02  ARC VERSION  13
01/13/2011 11:30:02  ARCHIVED AT    01-13-11 10:25:22
01/13/2011 11:30:02  ARCHIVE CHARACTER SET: ASCII
01/13/2011 11:30:02  ARCHIVED FROM  ALL AMP DOMAINS
   :
01/13/2011 11:30:02  UTILITY EVENT NUMBER  -  15
01/13/2011 11:30:02
01/13/2011 11:30:02  DATABASE "PD"
01/13/2011 11:30:02    TABLE "Department"
01/13/2011 11:30:02    TABLE "Dept_Summary"
01/13/2011 11:30:02    TABLE "Employee"
01/13/2011 11:30:02    TABLE "Emp_Phone"
01/13/2011 11:30:02    TABLE "GT_Deptsalary"
01/13/2011 11:30:02    TABLE "Job"
01/13/2011 11:30:02    VIEW  "LargeTableSpaceTotal"
01/13/2011 11:30:02    TABLE "Phone_Summary"
01/13/2011 11:30:02    MACRO "SetAnsiDate_OFF"
01/13/2011 11:30:02    MACRO "SetAnsiDate_ON"
01/13/2011 11:30:02
01/13/2011 11:30:02  ANALYZE COMPLETED
   :
```

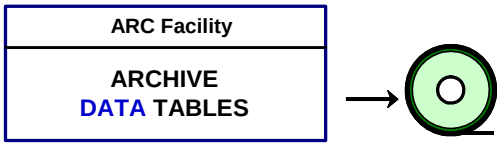
Archive Objects

The archive statement can only back up one table type at a time: data; dictionary; no fallback; or journal. Users must submit separate archive statements in order to archive each.

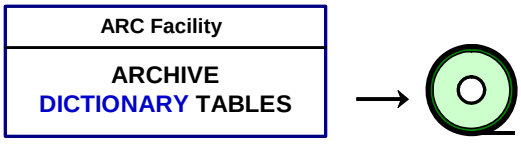
Below is a description of each archive type:

DATA TABLES	Archives fallback and non-fallback tables, views, triggers, and macros when you archive from ALL AMPs or clusters of AMPs.
DICTIONARY TABLES	Backs up DD/D rows that describe the databases or tables archived during a cluster- level archive. If you archive a database, the archive includes table, view, trigger, and macro definitions. If you archive a table, back up only includes table definition rows. DD/D information for permanent journals is not included.
NO FALLBACK TABLES	Run this archive type only to back up no fallback tables on an AMP that was down during a DATA TABLE archive. It completes the previous ALL AMP or cluster archive.
JOURNAL TABLES	Archives the dictionary rows and selected contents of the journal tables.

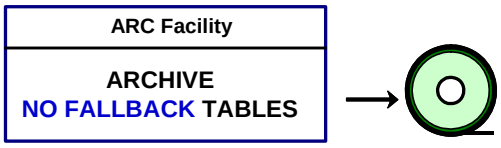
Archive Objects



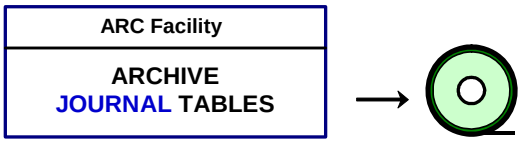
- Tables, Views, Macros, Triggers, Stored Procedures, UDFs, and Join/Hash Indexes in 13.0.
- Does NOT archive Journal Tables
- All AMP or cluster archive



- DD/D rows to complement cluster-level archive



- Non-fallback tables
- Archives AMP data missed during previous all AMP or cluster-level archive



- Used to archive Journal Tables

Archive Objects (cont.)

The information backed up in an archive operation varies depending upon the type of object you select:

- Single database or table
- Multiple databases or tables
- All databases

Single Database Archive

An ALL AMP database archive backs up a wide range of DD/D information. It archives all objects that belong to the database including views, macros and the data tables themselves. The information archived for the data tables includes table, column, and index information as well as table headers and data rows. A table header is a row of information about the table that is kept in the first block of the table.

Database ALL Archive

A Database ALL archive archives the parent and all children. The backed up objects are identical to those archived in a single database archive.

Single or Multiple Table Archives

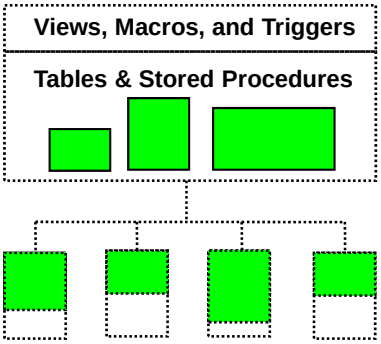
For each table specified in the archive statement, the ARC utility backs up table, column, and index information along with table headers and the actual data rows.

EXCLUDE Option

This option directly affects which databases are backed up. The exclude option changes the range of objects that the ARC utility archives. Users can leave out a single database, a database and all of its children, or a range of alphabetically sorted databases.

Archive Objects (cont.)

An ALL AMP archive that identifies a database that contains data tables, etc.



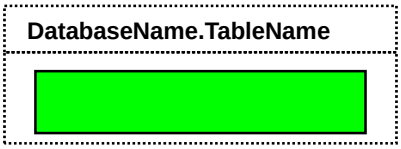
Archives all DD/D information for the identified database, including views, macros, triggers, stored procedures, and UDFs. Join and hash indexes are also archived in TD 13.

The archive also includes all table, column and index information, as well as table headers and data rows.

The ALL option archives all items listed above for the specified database, as well as all its descendants.

Note:
The EXCLUDE option allows you to exclude a single database, a database and all its descendants, a range of alphabetically sorted databases, or specific tables.

An ALL AMP archive that identifies a single table or partition(s) of a table.



An ALL AMPs table archive that identifies a table archives table, column and index information, as well as table headers and data rows.

Archive Levels

The default archive level for any archive operation is all AMPs.

Normally, you do not specify an archive level in your ARCHIVE statement since ALL is the default. When an AMP is off-line during an all-AMP archive, non-fallback tables may only be partially archived.

You need to perform a single-AMP back up of NO FALLBACK TABLES to obtain a complete back up. Fallback tables are always completely archived even if a vproc is down, because there is either a primary or fallback copy of the data on another AMP vproc.

Multi-Stream Archives

Characteristics of Multi-stream Archives include:

- ARC enhancement to use local (same-node) sessions to archive data for each AMP
- Provides substantial (up to 2x) archive and restore performance compared to non-local AMP
- Eliminates most BYNET traffic during BAR operations
- Must be configured properly to get full benefit
  - Advocated solutions will use this automatically
  - Enterprise-fit solutions must properly balance node connectivity and session count to get maximum performance
  - ARC user must have MONITOR privilege to get necessary AMP-level information

Cluster Archives

As an alternative to archiving data tables from all AMPs into a single archive, you can break the archive into a set of archive files called a cluster archive. A cluster archive archives data tables by groups of AMP clusters. You can run a cluster archive in parallel, or schedule it to run over several days. It may be faster to restore a single vproc since the system has fewer tapes to scan to recover lost data.

In general, cluster archiving improves the archive and recovery performance of very large tables. In addition, it can simplify the restore process of non-fallback tables for a specific AMP vproc.

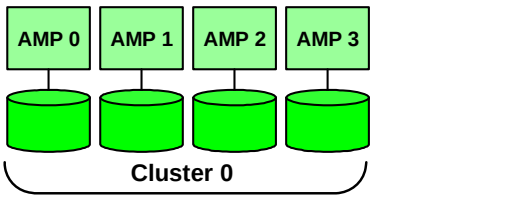
A cluster archive does not contain any dictionary information. You must perform a DICTIONARY TABLE archive before you run a cluster archive for the first time, because Database DBC is automatically excluded for this kind of archive operation. You must run the dictionary table archive again any time there is a change in the structure of the tables in the cluster archive.

Cluster archives have two restrictions:

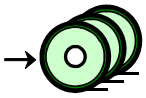
- You cannot create a cluster archive of journal tables.
- You cannot setup cluster archives when you are archiving DBC database.

Archive Levels

- By default, Teradata performs ALL AMP level archives.
- BAR solutions (e.g., NetBackup) also provide Multi-Stream backups.
  - Effectively multiple ARC jobs.
  - Data is divided automatically by ARC at the AMP level.
  - Alternative to cluster backup and usually provides the better performance.
- ARC also provides options to archive at the cluster or the individual AMP level.



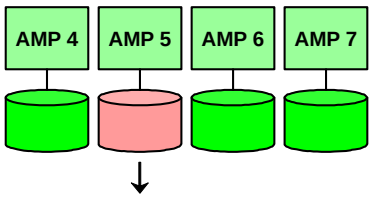

```
ARCHIVE DATA TABLES
(Sysdba) ALL ,
CLUSTER = 0,
FILE = cluster0;
```



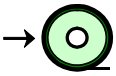
Cluster level archives group data from one or more clusters into separate archive data sets.

A single AMP can be recovered in less time.

Dictionary information is archived separately.




```
ARCHIVE NO FALLBACK
TABLES (Sysdba) ALL ,
AMP = 5,
FILE = amp5only;
```



Single AMP archives are only used to complete the archive of no fallback tables after the AMP is returned to service.

Archive Options

The archive statement includes a number of options. Each option is described below:

RELEASE LOCK	Automatically releases Utility Locks if the operation completes successfully.
INDEXES	For all-AMP archives only, this option specifies to include secondary indexes with the archive. You will need more time and media to archive objects with their secondary indexes. Neither the GROUP READ LOCK nor the ONLINE Archive feature supports the INDEXES keyword of ARC, to back up secondary index data. Index data will be rebuilt from the primary data when the table is restored.
ABORT	Causes all AMP or cluster archives to fail with error messages if an AMP is off-line and the objects to be archived includes: • No fallback tables • Single image journals
NONEMPTY DATABASES	Instructs the ARC utility to exclude users/databases without tables, views, macros, or triggers from the archive.
ONLINE	Online archive allows backups to be performed without placing read locks on the tables; this means that write availability is maintained while the backup is running.
USE GROUP READ LOCK	Permits you to archive as transactions update locked rows. You must define after image journaling for the table during the time the archive is taking place.

Archive Options

- **Release Lock**
  - Utility locks automatically released upon successful operation completion
- **Indexes**
  - Restricted to all-AMP dumps
  - Includes secondary indexes with archive; requires more time and media
- **Abort**
  - Fails All-AMP or cluster archives AND provides error messages if:
    - AMP is off-line AND,
    - Archived objects include no fallback tables OR single-image journals
- **Nonempty Database(s)**
  - Excludes users/databases without tables, views, macros, triggers, or stored procedures from archive operation

- **Online (12.0),**
  - Permits concurrent table archiving and transaction updates on databases and/or tables
  - Preferred online archive technique – captures before-images of changes along with archive
  - Point-in-time archive; if a table is restored, it will reflect the table when the archive was started
- **Use Group Read Lock**
  - Permits concurrent table archiving and transaction updates on locked rows
  - Requires after-image journaling of table
  - After restoring a table and applying after-image journal images, the table will have rows that reflect the state of the table when the archive was completed.

ONLINE Archive Option

The facing page identifies key characteristics of ONLINE Archive.

Some additional benefits of Online archive include:

- Removes the restore restrictions associated with journaling (GROUP READ LOCK)
  - Logged (changed) data is stored in the same archive as the base data.
- For an all-AMP backup, the feature can be used by adding a single keyword, 'ONLINE', to the ARC 'ARCHIVE' statement.
  - No ARC syntax changes are needed when restoring an online archive.
- Cluster archives require a single new statement to activate the feature; this is specified in the dictionary archive step.
- Cross-release and cross-platform restores are allowed.

The following archive/restore options are not supported with the TD12 Online Archive feature:

- Archives of the following objects are not supported with Online Archive:
  - Database DBC, Permanent Journal tables, Temporary tables, tables in Fastload state, MultiLoad aborted tables, MultiLoad work tables
- Archive of Selected Partitions.
- INDEXES option.
- RESTORE FALLBACK option in RESTORE statement.
- USE [GROUP] READ LOCK option.
- Archive of non-fallback tables with a down AMP.
- The following operations cannot be performed on a table during an online archive of that table:
  - DDL operations (except COLLECT STATISTICS, ACCESS LOGGING, and COMMENT).
  - FastLoad.

ONLINE Archive Option

Online Archive Description

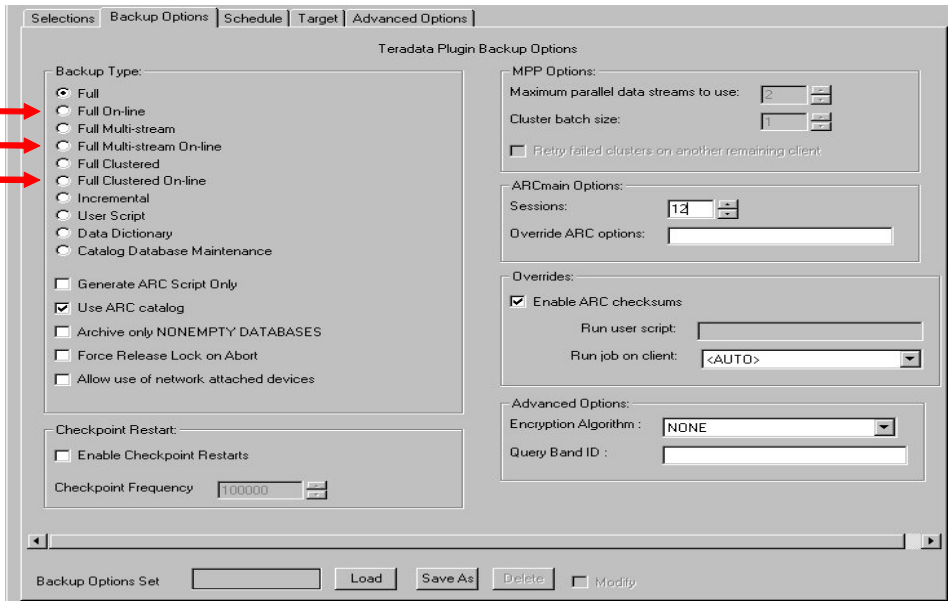
- **Simplifies the backup and restore process for backups in an active Data Warehouse environment. Perform backups without affecting write availability of tables.**
  - **Use when table writes are needed at all times**
- **Table-level online archives.**
  - **A short read lock is used to establish a consistency point. Effectively a point-in-time archive.**
  - **Limited DDL is allowed (e.g., COLLECT STATISTICS, COMMENT)**
  - **FastLoad is not allowed.**
- **Database-level online archives.**
  - **An access lock will be placed on the database; this will prevent table create, table drop, and DDL changes.**
- **Logged (changed) data is automatically stored in the same archive as the base data.**
  - **Removes the restore restrictions associated with journaling (GROUP READ LOCK).**
  - **ONLINE Archive captures the before-images (i.e., Transient Journal type image) in a subtable associated with the data table. This subtable is automatically archived with the data table and the before-images are used to rollback any changes after the archive was started.**
- **For an all-AMP backup, the feature can be used by adding a single keyword (ONLINE) to the ARCHIVE statement.**
  - **No ARC syntax changes are needed when restoring an online archive.**

BakBone NetVault Example

The facing page contains a screenshot of the ONLINE options available with BakBone NetVault.

- Fully supported starting with NetVault Teradata plug-in version 11.0 or later.
- NetVault must be configured to use Teradata Database 12.0 features.
  –
  – Otherwise the NetVault ‘Online’ options will use the original GROUP READ LOCK method to perform the online archive.
- Online archives can be performed by selecting the ‘Full Online’, `Full Multi-Stream On-line’ or ‘Full Clustered Online’ backup types.
- The Online Archive feature can be used with prior NetVault releases by manually creating an ARC script and submitting it as a ‘User Script’ backup.

BakBone NetVault Example



NetVault must be configured to use Teradata Database 12.0 features;
Otherwise the NetVault 'Online' options will use the GROUP READ LOCK method for online archive.

Database DBC Archive

An archive of the information in DBC should be done every time DDL makes changes to the definitions stored in the database. Examples of the types of commands that make these changes are:

- CREATE DATABASE/USER
- MODIFY DATABASE/USER
- CREATE/ALTER TABLE
- CREATE/REPLACE VIEW
- CREATE/REPLACE MACRO
- CREATE INDEX
- DROP TABLE/VIEW/MACRO
- DROP INDEX
- GRANT
- REVOKE

Database SYSUDTLIB is linked with database DBC and is only archived if DBC is archived. SYSUDTLIB cannot be specified as an individual object in an **ARCHIVE** statement.

If database DBC is involved in an archive operation, it is always archived first, followed by database SYSUDTLIB. If additional databases are being archived, they will follow SYSUDTLIB in alphabetical order.

RESTORE Considerations

If you drop a table in a database, you cannot restore a dropped table unless you restore the entire database.

Furthermore, you cannot restore a dropped database unless you restore database DBC first.

If you need to restore all of a user database or database DBC (that is, all of the Teradata Database) because of a catastrophic event, you can restore the dictionary information for the database at the database level before you restore the individual tables. Restoring the dictionary first restores the table definitions, so you are able to successfully restore the tables.

You can only restore Database DBC to an initialized Teradata Database – usually following a SYSINIT. An initialized Teradata system can only have the user DBC and the default users of ALL, Default, and Public in order to RESTORE DBC (ALL).

Database DBC Archive

An archive of database DBC causes the system to copy the following tables to the archive.

AccessRights	**Specification of all GRANTed rights**
AccLogRuleTbl	**Stores access logging specifications**
Accounts	**Lists all authorized account numbers**
CollationTbl	**Defines MULTINATIONAL collation**
DBase	**Definition of each DATABASE and USER**
DBQLRuleTbl	**DBQL Rule table (13.10)**
Hosts	**Character set default override rules**
LogonRuleTbl	**User, host, password requirements**
Next	**Internal table for generating TABLE and DATABASE identifiers**
OldPasswords	**Lists passwords that are no longer in use,**
Owners	**Defines all databases owned by another**
Parents	**Defines the parent/child relationship between databases**
Profiles	**Defines Profiles**
RCConfiguration	**Records the configuration for the RCEvents rows**
RCEvent	**Records all archive and recovery events**
RCMedia	**Records all removable media used in archive activities**
RepGroup	**Defines each replication group in the server.**
Roles	**Defines Roles**
RoleGrants	**Contains Users and Roles granted to Roles**
Translation	**National character support tables**
UDTCast	**Contains source & target data types used in casting operations for UDTs.**
UDTInfo	**Captures the specifics contained within the CREATE TYPE statement.**
UDTTransform	**Contains the transform group name and the routine identifiers.**

Database DBC can only be restored to an initialized Teradata Database system.

Summary

The facing page summarizes some important concepts regarding this module.

Summary

- Archive and Recovery (ARC) is a command-line utility that performs three operations: archive, restore and recovery.
- For small systems, the optimum number of sessions for archive and recovery operations is:
  - One per AMP vproc for archive
  - Two per AMP vproc for recovery
- An archive operation can back up a single database or table, multiple databases or tables, or all databases.
- Available archive levels are all-AMP, specific AMP and cluster archives.

Module 58: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 58: Review Questions

1. True or False. The Archive and Recovery utility protects against more types of potential data loss than automatic data protection features.

2. True or False. Recovery and FastLoad are about the same in ease and speed to recover data.

3. True or False. An All-AMPs archive of a database archives all of the objects in the database.

4. True or False. Archiving a partition of a PPI table places a partition-level lock on the partition being archived.

Notes

Module 59

Restoring Data

After completing this module, you will be able to:

- **Use the ARC facility to replace existing data on a Teradata system with information stored on portable storage media.**
- **Understand the RESTORE, COPY, BUILD, REVALIDATE REFERENCES FOR, and RELEASE LOCK statements.**
- **Use Recovery Control views to obtain ARC event information.**

Notes

Table of Contents

Understanding Restore Operations 59-4
Considerations before Restoring Data 59-4
Restore-Related Statements 59-6
The Restore Statement 59-8
Restoring Examples 59-10
RESTORE Example and Output 59-12
EXCLUDE TABLE Caution: 59-12
RESTORE Considerations if the Configuration has Changed 59-12
Restoring Selected Partitions of PPI Table 59-14
RESTORE Partition Example 59-16
COPY Statement 59-18
Copying Partitioned Data 59-18
Copying Objects 59-20
Keyword Options with COPY 59-20
Copying 59-22
BUILD Statement 59-24
RELEASE LOCK Statement 59-26
Revalidate References 59-28
Revalidate References Output 59-30
Recovery Control Data Dictionary Views 59-32
DBC.Association[V][X] Views 59-32
DBC.Events[V][X] Views 59-32
DBC.Events_Configuration[V][X] Views 59-32
DBC.Events_Media[V][X] Views 59-32
Association View 59-34
Events View 59-36
Restoring Data Summary 59-38
Module 59: Review Questions 59-40

Understanding Restore Operations

A restore operation transfers database information from archive files backed up on portable storage media to all AMP vprocs, clusters of AMPs, or specified AMP vprocs.

Considerations before Restoring Data

Before performing a restore operation, consider the following items.

Dropped Database and Users – a restore of a database DBC drops all new databases or users created since the time the archive was created.

Dropped Tables, Views, and Macros – a restore of a user database drops any new tables, views, macros, stored procedures, and triggers created since the archive of the database.

Restoring Undefined Tables with COPY – because of potentially conflicting database and table internal identifiers, you cannot restore a database or table to another system that does not contain an equivalent definition of the entity (for example, the same name and internal identifier). To restore data tables that are not already defined in the data dictionary, use the COPY statement.

Insufficient Memory for Large Tables – Teradata ARC uses the same methodology as the Teradata SQL CREATE INDEX function to rebuild secondary table indexes. If there is insufficient available disk space, it may not be possible to restore a very large table because of the amount of temporary disk space that is required to recreate a secondary index.

Join Indexes – Prior to Teradata 13, ARC did not archive or restore join indexes. If a database containing a join index is restored, then the join index will no longer exist when the restore operation is complete. If a partial database restore is done where a table is restored, any join indexes that reference that table will be marked as invalid.

Matching Hash Functions for Large Objects – Teradata ARC supports the restore operation for tables that contain large object columns as long as the database system is enabled for large object support. However, large object columns cannot be restored on a system that uses a hash function that is different than the one used for the archive.

Certain Statements Force the Restoration of the Entire Database – a Teradata SQL DROP or RENAME statement cause the definition of an entity to be removed from the dictionary, and this same definition cannot be re-created using a Teradata SQL CREATE statement because a create operation is a new logical definition.

Understanding Restore Operations

Restore operations transfer information from archive files to AMP vprocs.

Data Definitions

- **Database archives contain dictionary definitions.**
- **Dictionary table archives contain dictionary definitions.**

Replacing Objects

- **ALL AMP vproc archives contain data and dictionary definitions.**
- **Restore operations replace both.**

Notes:

- **You can only RESTORE an entity if the Data Dictionary has an equivalent definition of the entity being restored (same name and internal ID).**
- **The COPY operation can be used if the object doesn't exist. To COPY the object in a database/user, the database/user must exist on the target system.**

Restore-Related Statements

The Archive and Recovery utility provides several recovery control statements you use during restore-related operations. Each command is described below:

ANALYZE	Reads the contents of an archive tape and displays the information.
BUILD	Builds indexes for fallback and non-fallback tables. It also builds fallback rows for fallback tables, and can build journal tables by sorting the change images.
COPY	Copies a database or table from an archived file to the same or different Teradata system than the one from which it was archived.
DELETE DATABASE	Deletes all data tables, views and macros from the database. This command does not remove journal tables.
RELEASE LOCK	Removes a utility lock from a specific database or table.
RESTORE	Moves data from archive files back to the same Teradata system from which it was archived. You can also restore data to another system. For example, migrating from V1 to V2 required a fresh system (Sysinit).
REVALIDATE REFERENCES FOR	Validates inconsistent restraints against a target table thereby allowing users to execute UPDATE, INSERT and DELETE statements on the tables.

You may invoke the Archive and Recovery utility from a channel-attached MVS or VM host system.

Restore-Related Statements

LOGON	Begins a session.
LOGOFF	Ends a session and terminates the utility.
ANALYZE	Reads an archive tape to display information about its contents.
RESTORE	Restores a database or table from an archive file to specified AMP Vprocs.
COPY	Restores a copy of an archived file to a specified Teradata database System.
BUILD	Builds Indexes and fallback data.
RELEASE LOCK	Releases host utility locks on databases or tables.
DELETE DATABASE	Deletes data tables, views and macros from a database.
REVALIDATE REFERENCES	Validates inconsistent restraints against a target table thereby allowing users to execute UPDATE, INSERT and DELETE statements on the tables.

The Restore Statement

The RESTORE statement allows you to replace database objects from an archive tape to the same system or to another system. The ARC facility has four types of restore or recover operations described below:

Data Tables

The DATA option restores fallback, non-fallback, or both types of data tables to all AMP vprocs or clusters of AMP vprocs. If you restore a database, the data dictionary definitions (for table, view, macro, and triggers) are restored automatically. If you restore a table, only table definition rows are included.

Dictionary Tables

The DICTIONARY option restores data dictionary rows that describe the databases or tables dumped (necessary with a cluster-level restore). A RESTORE DICTIONARY TABLES only restores the definitions of all the entities in the dictionary for the selected databases.

No Fallback Tables

Use the no fallback option to restore a single processor.

Journal Tables

This option restores an archived journal for subsequent use in a roll operation.

Restore Fallback

This option applies only to data table restores of fallback tables. This option restores the fallback copy of primary and unique secondary indexes while restoring the data.

No Build

NO BUILD prevents secondary indexes on non-fallback tables from being restored or built. On fallback tables, it prevents the creation of secondary indexes and fallback table rows.

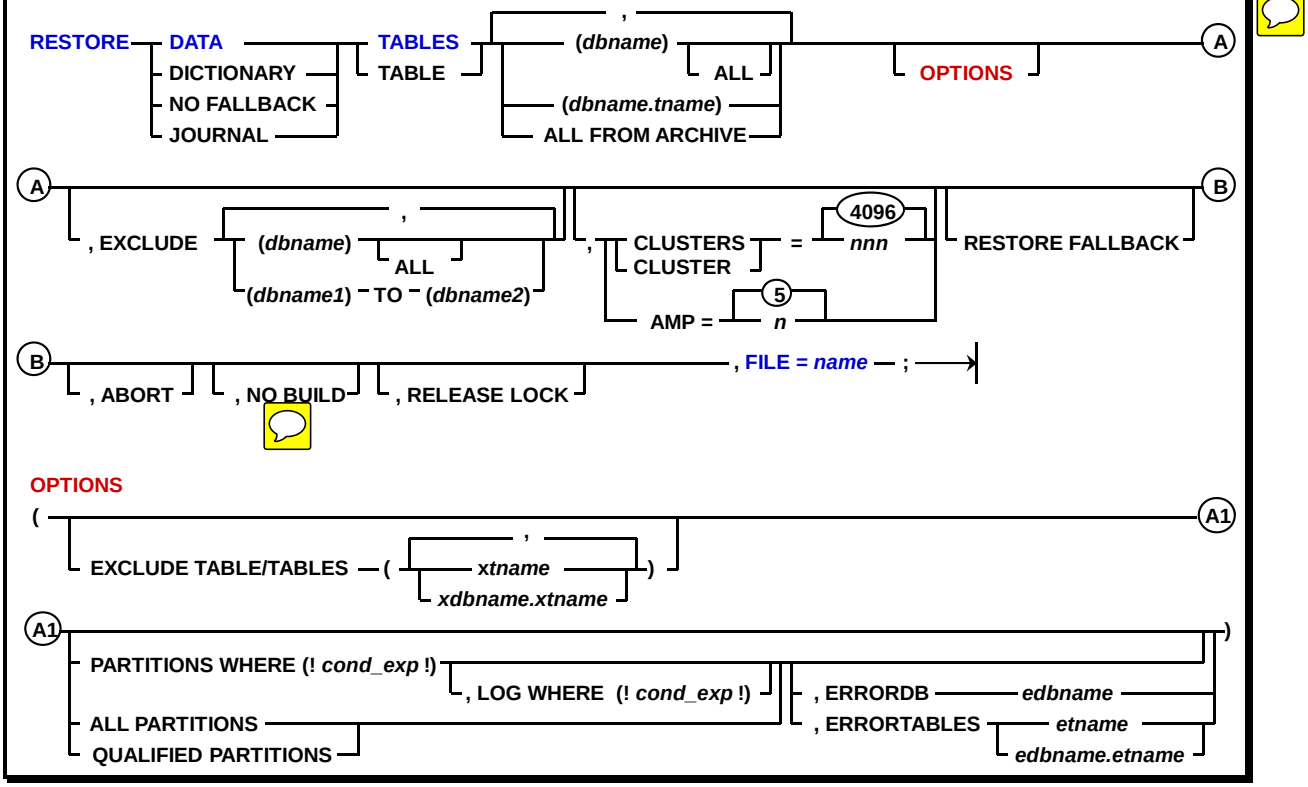
Release Lock

Automatically release of the utility locks when a restore completes successfully.

Abort

This option causes an all-AMP restore to abort with error messages if an AMP is offline and the restore includes a non-fallback table. It does not affect a specific-AMP restore.

RESTORE Statement



Restoring Examples

Three RESTORE examples are shown on the facing page.

As mentioned before, you can restore archived data tables to the Teradata Database if the data dictionary contains a definition of the entity (same name and same internal ID) you want to restore.

For example, if the entity is a database, that database must be defined in the dictionary. Or, if the entity is a table, that table must be defined in the dictionary. You cannot restore entities not defined in the data dictionary.

A dictionary table archive contains all table, view, and macro definitions in the database. A restore of a dictionary archive restores the definitions of all data tables, views and macros. However, it does not restore any data.

ALL FROM ARCHIVE

In the first example, the ALL FROM ARCHIVE keywords take the place of the database and/or table names that are normally specified after the DATA, DICTIONARY, JOURNAL, or NO FALLBACK TABLES keywords.

You are not allowed to specify any other database or table names to be restored when using ALL FROM ARCHIVE. All databases and tables in the given archive file will be restored, and any existing databases or tables will be overwritten.

ALL FROM ARCHIVE can not be used to restore database DBC. The user must exclude DBC if it is present in the archive being restored.

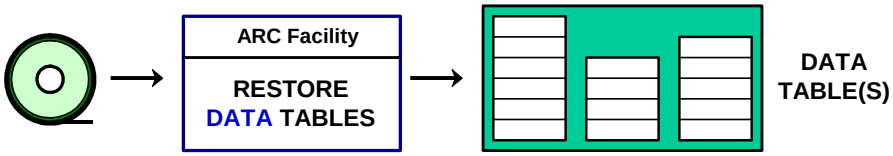
Specified Database

The second example is restoring the DS database when all AMP vprocs are online. The restore type is data and the restore object is all databases belonging to user DS. Since there is no mention of any restore levels, such as a specific AMP number, the system assumes all AMPs. The release lock option removes the utility lock after completing the restore operation. The name of the archive file is “arch3_DS”.

The third example has a narrower scope. This statement is only restoring non- fallback tables on AMP 1. The administrator has already performed an all-AMPs restore on the rest of the system. The release lock option removes the utility lock after completion of the restore operation. The archive filename is “arch3_Sys”.

Any databases or users created since the Archive of the dictionary or any table, view, or macro created since the archive of a database will be dropped when you restore the DBC database or a user database.

RESTORE Examples



restore1_pd.arc

```
LOGON dbc/sysdba,dbapass;
RESTORE DATA TABLES  ALL FROM ARCHIVE
  , RELEASE LOCK, ABORT, FILE = arc1_PD;
LOGOFF;
```

Restore all tables and databases from archive (ALL AMPs restore with all AMPs online).

restore3_ds.arc

```
LOGON dbc/sysdba,dbapass;
RESTORE DATA TABLES (DS)
  , RELEASE LOCK, ABORT, FILE = arc3_DS;
LOGOFF;
```

Restores all objects for DS and restores all rows to the point-in-time of archive – ONLINE option is not used on RESTORE.

restore_amp1.arc

```
LOGON dbc/sysdba,dbapass;
RESTORE NO FALLBACK TABLES (SYSDBA) ALL
  , AMP=1, RELEASE LOCK, FILE = arc4_Sys;
LOGOFF;
```

Restores non-Fallback tables for Sysdba and ALL child databases/users of Sysdba (single AMP restore - AMP 1).

RESTORE Example and Output

An example of the output from a RESTORE command is shown on the facing page.

EXCLUDE TABLE Caution:

When you do a full database-level restore of an archive with excluded tables, the data dictionaries and the table headers of *all* tables, including excluded tables, are replaced.

As a result, *all* of the existing rows in the excluded tables are deleted.

You can restore individual tables from a database-level archive with excluded tables. In the RESTORE statement, you must individually specify all the tables you want to restore, except the excluded tables. By omitting the excluded tables, you preserve the data dictionaries and table headers of the excluded tables. That way you can restore the database from the archive without altering the excluded tables.

RESTORE Considerations if the Configuration has Changed

When restoring or copying to the same configuration that the archive came from, then the first AMP takes all the rows in the block and there is no redistribution.

However, if the configuration is different, then the data rows have to be redistributed. Prior to Teradata 12, the redistribution occurred during the restore phase. As arcmain restored or copied blocks of data to a system, each block was sent to the AMP that owned the first row. That AMP took rows from the block until a row was detected that belonged to a different AMP, the remainder of the block was sent on to that AMP. This continued until the block was empty. This approach worked well enough until PPI came along. With PPI tables, blocks began to bounce between AMPs more than a non-PPI table.

Starting with Teradata 12, the restore was modified so when an AMP receives a block the entire block is placed in a temporary subtable. When the build occurs the data is sort merged to place the data in row order and redistributed to the correct AMP. This made the restore blazing fast, but the build can be very slow depending on the size of the system and the type of table. More memory helps, but when the data (an AMP has to sort merge) is larger than it can buffer, then there is additional I/O that will slow you down more.

With Teradata 13, the sort merge is moved to the end of the restore phase. The blocks are still stored in temporary sub tables, but the sort merge will usually be faster in Teradata 13. With Teradata13, the redistribution occurs at the end of the restore phase. As blocks of data are sent to the AMPs, the AMPs break the data into buffers. There is 1 buffer for each AMP. As the buffers fill they are sent to the correct AMP. At the end of the restore phase each AMP sorts and merges the data.

The build phase under TD13 is like V2R6, secondary indexes and fallback are recreated and the restore flag is reset.

The old redistribution algorithm can still be used by adding NOSORT as a command line parameter. There are times when NOSORT will be faster and times when SORT will be faster. Only testing will determine the whether this option is useful or not.

RESTORE Example and Output

restore1_pd.arc

```
LOGON dbc/sysdba,dbapass;
RESTORE DATA TABLES
   ALL FROM ARCHIVE
  , ABORT
  , RELEASE LOCK
  , FILE = arc1_PD;
LOGOFF;
```

Output from this restore script ...

```
...    :
...    RESTORE DATA TABLES ALL FROM ARCHIVE
...    , ABORT
...    , RELEASE LOCK
...    , FILE = arc1_PD;
...    UTILITY EVENT NUMBER  - 17
...    LOGGED ON    4 SESSIONS
...    STARTING TO RESTORE DATABASE "PD"
...    "LargeTableSpaceTotal" - VIEW RESTORED
...    "SetAnsiDate_OFF" - MACRO RESTORED
...    "SetAnsiDate_ON" - MACRO RESTORED
...    DICTIONARY RESTORE COMPLETED
...    "Department" -   3,446 BYTES, 60 ROWS RESTORED
...    "Dept_Summary" -   2,966 BYTES, 50 ROWS RESTORED
...    "Employee" -  65,077 BYTES,   1,000 ROWS RESTORED
...    "Emp_Phone" -  52,504 BYTES,   2,000 ROWS RESTORED
...    "GT_Deptsalary" - 548 BYTES, 0 ROWS RESTORED
...    "Job" -   2,898 BYTES, 66 ROWS RESTORED
...    "Phone_Summary" -  51,464 BYTES,   1,960 ROWS RESTORED
...    "PD" - LOCK RELEASED

...    STATEMENT COMPLETED
```

Restoring Selected Partitions of PPI Table

Selected partitions can be directly backed up and restored with a PARTITIONS WHERE option that restricts the list of rows processed. The PARTITIONS WHERE option operates on complete table partitions. A RESTORE (or COPY) completely wipes out selected partitions (specified by the PARTITIONS WHERE option) of an existing target table before recovering the rows stored on the backup tape.

A restore of selected partitions is impacted by the various maintenance activities that can occur on a table. For example, a user may not be able to perform a full-table backup every time a secondary index is added or dropped, or the partitioning expression is changed. A restore of selected partitions is able to restore data into a target table with different characteristics than the source stored on tape.

To RESTORE selected partitions, a table must already exist on the target system.

PARTITIONS WHERE Keyword

Use the PARTITIONS WHERE option to specify the conditional expression, which contains the definition of the partitions that you want to restore. The following restrictions apply to the use of PARTITIONS WHERE option:

- The object must be an individual table name (not a database).
- The source and target tables must have a PARTITIONS BY expression defined.
- The restore is an all-AMP restore (not a dictionary, cluster, or journal restore).
- If the table belongs to a database that is specified in the RESTORE statement, the table is excluded from the database-level object (with EXCLUDE TABLES) and is individually specified.
- Any name specified in the conditional expression is within the table specified.
- It is recommended that the only referenced columns in the conditional expression be the partitioning columns or system-derived column PARTITION of the table. References to other columns do not contribute to partition elimination, and might accidentally qualify more partitions than intended.

LOG WHERE Keyword

You might find that the PARTITIONS WHERE option does not capture all the rows that need to be restored. In this case, use the LOG WHERE option to insert into a Teradata-generated error table archived rows that both fall outside the partitions specified by the PARTITIONS WHERE conditional expression and match the LOG WHERE conditional expression.

Use the option only if PARTITIONS WHERE is also specified for the object. If LOG WHERE is omitted, the default is to log to the error table only the rows in the partitions being restored that have errors.

Restoring Selected Partitions of PPI Table

PARTITION Options with RESTORE and COPY commands.

PARTITIONS WHERE (! conditional expression !)

This option specifies a conditional expression that contains the definition of the partitions that you want to restore/copy.

LOG WHERE (! conditional expression !) **– the conditional expression specifies rows to log to the error table when restoring selected partitions.**

ERRORDB / ERRORTABLES **– specifies the location of the error log for partition-level operations.**

ALL PARTITIONS

Use this option to restore/copy all of the archived partitions for a PPI table.

QUALIFIED PARTITIONS ***(may not be used very often)***

Use this option only to restore/copy a specific-AMP archive after restoring selected partitions from an all-AMP archive done while an AMP is down.

Note: For partition archives, specify PARTITIONS WHERE or ALL PARTITIONS.

If PARTITIONS WHERE or ALL PARTITIONS options are not specified for a RESTORE or COPY operation, the default action is to overwrite the entire table with the archived table definition and data. Essentially, this is the same as a full-table restore.

RESTORE Partition Example

An example of a Restore script that restores all partitions in a partition archive for a PPI table is shown on the facing page.

The partitioning expression for the Sales_PPI table is:

```
PARTITION BY RANGE_N (sales_date  BETWEEN DATE '2002-01-01' AND DATE '2011-12-31'
EACH INTERVAL '1' MONTH );
```

Additional Notes when Restoring Partitions

Always Specify PARTITIONS WHERE or ALL PARTITIONS – If the PARTITIONS WHERE or ALL PARTITIONS options are not specified for a RESTORE or COPY operation, the default action is to overwrite the entire table with the archived table definition and data. Essentially, this is the same as a full-table restore.

For example, if you forget to use PARTITIONS WHERE when you try to restore a single partition backup, data is dropped from the table and the single partition stored on the archive is restored.

Know What Partitions are Being Deleted – with a RESTORE or COPY operation, all partitions that match the PARTITIONS WHERE condition are deleted, even if they are not stored in the archive.

For example, if you restore an archive that only contains the data for November 2012, but mistakenly enter a PARTITIONS WHERE condition that matches both October and November 2012, the data for both October and November 2012 are deleted, and only November 2012 is restored.

The remedy for this situation is to be very careful about using the PARTITONS WHERE condition. If there is any doubt about which partitions are affected, COPY the selected partition backup to a staging table, and manually copy the desired partition(s) into the target table using INSERT/SELECT and/or DELETE.

Avoid Restoring From a Previous Partitioning Scheme – when changing the partitioning expression for a table, it is possible to change the boundaries of existing partitions. If these partitions are restored, Teradata might either drop more data than expected or restore less data than expected, if the archive does not include data for all of the selected partitions.

For example, if an archive is done on a table partitioned by month with the archive data corresponding to October 2011, and the table is re-partitioned by week, then a PPI restore of the October backup (using ALL PARTITIONS) overwrites the data for all weeks that contain at least one day in October. As a result, the last few days of September and the first few days of October might be deleted and not restored.

The remedy for this situation is to avoid restoring partition backups from a previous partitioning scheme to an updated table. Or, use LOG WHERE for the weeks that contain days in both October and September/November, and manually copy the rows into the table.

RESTORE Partition Example

restore6_ppi.arc

```
LOGON dbc/sysdba,dbapass;
RESTORE DATA TABLES
  (TFACT.Sales_PPI) (ALL PARTITIONS), ABORT, RELEASE LOCK, FILE = arc6_PPI;
LOGOFF;
```

arcmain < restore6_ppi.arc

Portion of output from executing above script

```
RESTORE DATA TABLES
  (TFACT.Sales_PPI) (ALL PARTITIONS), ABORT, RELEASE LOCK, FILE = arc6_PPI;
UTILITY EVENT NUMBER  - 38
LOGGED ON   4 SESSIONS
STARTING TO RESTORE TABLE "TFACT"."Sales_PPI"
Archive Bounding Condition:
RANGE_N("TFACT"."SALES_PPI"."sales_date"  BETWEEN DATE '2002-01-01' AND DATE '2011-12-31' EACH INTERVAL
'1' MONTH ) IN (118 TO 120 )
        [Bounding condition is well-defined]

Restore Bounding Condition:
 RANGE_N("TFACT"."SALES_PPI"."sales_date"  BETWEEN DATE '2002-01-01' AND DATE '2011-12-31' EACH INTERVAL
'1' MONTH ) IN (118 TO 120 )
         [Bounding condition is well-defined]
 DICTIONARY RESTORE COMPLETED
 "Sales_PPI" -   2,391,241 BYTES,  40,500 ROWS RESTORED
 "TFACT"."SALES_PPI" - LOCK RELEASED
```

COPY Statement

Use the COPY statement to recreate tables and/or databases that have been dropped or to restore them to the same system or to a different system.

Some of the options for the COPY statement are:

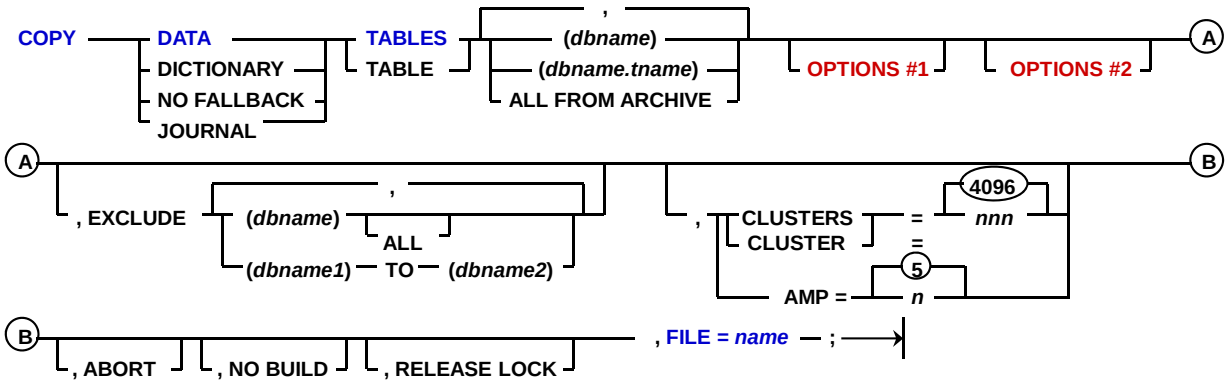
NO FALLBACK	Copies fallback tables into non-fallback tables. This option applies during the COPY of an all-AMP or DICTIONARY archive.
NO JOURNAL	Copies all tables with journaling disabled. This option applies during the COPY of an all-AMP or DICTIONARY archive.
WITH JOURNAL TABLE =	Overrides the default journal table of the receiving database for tables that had journaling enabled. This option applies during the COPY of an all-AMP or DICTIONARY archive.
APPLY TO	Specifies to which tables in the receiving system change images apply. This option is required when copying journal images.
NO BUILD	Prevents secondary indexes on non-fallback tables from being copied or built. On fallback tables, it prevents the creation of secondary indexes as well as fallback table rows. There is no rehashing of V1 to V2 data.
ABORT	Aborts ALL AMP copies with error messages if an AMP is offline and the restore includes a non-fallback table.
RELEASE LOCK	Causes ARC to release utility locks when a copy completes successfully.

Copying Partitioned Data

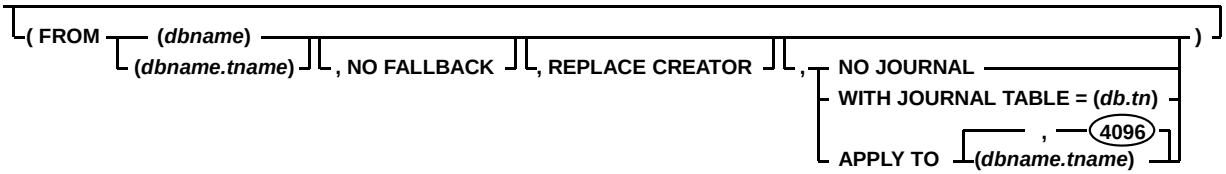
You can copy selected partitions of PPI tables, meaning that you can backup of one or more partitions of a table so you can archive, restore, and copy only a subset of data in a table.

To COPY selected partitions, a table must already exist on the target system. For a COPY operation, the existing table must have been created by a full-table COPY from the source machine.

COPY Statement



OPTIONS #1



OPTIONS #2

Same options with RESTORE – shown previously.

Copying Objects

The COPY statement has two uses:

- It uses an archived file to recreate tables and/or databases that have been dropped.
- It copies archived files to a different system.

The COPY statement can perform one of the following tasks:

- Copy an object that has been dropped back into the original system.
- Copy an object from one system to another.
- Copy an object back to the same system.

Keyword Options with COPY

NO FALLBACK Keywords

This option applies only during a copy of a dictionary archive or an all-AMPs archive. If a fallback table has permanent journaling on the archive, the table has dual journaling of its non-fallback rows after the copy when Teradata ARC applies the NO FALLBACK option (unless NO JOURNAL is specified).

FROM Keyword

The object specified in the FROM keyword identifies the archive object. This option applies only during a copy of a dictionary archive or an all-AMPs archive.

Journal enabled tables in the original archive carry their journaling forward to the copy unless you specify the NO JOURNAL keywords.

The NO JOURNAL keywords apply to all tables in a database when you copy a database. This option has no effect on a receiving database's journaling defaults.

If the object you specify in the FROM option is a table, ARC copies only that table.

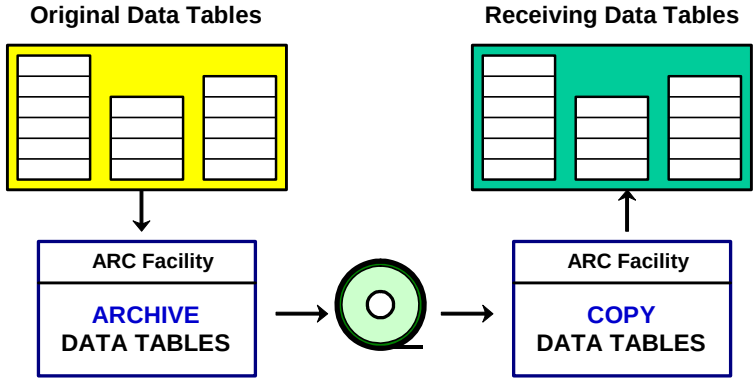
WITH JOURNAL TABLE Keywords

This option only applies during a copy of a dictionary archive or an all-AMPs archive. To use this option, you must have INSERT access rights to the referenced journal table. The source table must have a journal or this option has no effect.

If you are copying a database, the journaling you specify with this option applies only to those tables that had journaling enabled in the original database. This option has no effect on a receiving database's default journaling.

If the database has default journaling specified, then ARC uses those options. This option only overrides the journal table in the *receiving* database, and is only valid if the originating table had journaling enabled.

Copying Objects



copy1_demo.arc

```
LOGON dbc/sysdba,dbapass;
COPY DATA TABLES (SANDBOX)
   (FROM (DEMO))
  , ABORT , RELEASE LOCK
  , FILE = arc_demo;
LOGOFF;
```

Copy objects from archive of Demo database to Sandbox database.

copy2_demo.arc

```
LOGON dbc/sysdba,dbapass;
COPY DATA TABLE (SANDBOX.Employee)
    (FROM (DEMO),
              NO FALLBACK, REPLACE CREATOR)
, ABORT , RELEASE LOCK
, FILE = arc_demo;
LOGOFF;
```

Copy Demo.Employee from archive to Sandbox.Employee and change some attributes.

Copying

Examples of the output from the previous COPY operations are shown on the facing page.

Views, Macros, Triggers, and Stored Procedures

Copying a full database archive is the same as a restore operation. ARC deletes or drops all existing tables, views, macros, stored procedures, and triggers in the receiving system. Copying a full database archive copies all views, macros, and stored procedures in the archive to the receiving database.

But triggers cannot be copied with the COPY statement. If a trigger is defined in a database, then <trigger> NOT COPIED is displayed when the COPY statement is executed. This is not an error or warning. Triggers must be manually recreated via SQL.

You cannot copy one or more stored procedures from one database to another using the COPY statement. They can only be copied as part of a full database.

If your views, stored procedures, and macros have embedded references to databases and objects that are not in the receiving environment, those views, stored procedures, and macros will not work. To make any such views, stored procedures, and macros work, recreate or copy the references to which they refer into the receiving database.

If your views, stored procedures, and macros have embedded references to databases and objects that *are* in the receiving environment, they will work correctly.

Note: Make sure you fully qualify all table, stored procedure, and view names in a macro and all table names in a view. If you do not, you may receive an error. When you execute a COPY statement, partial names are fully qualified to the default database name. In some cases, this may be the name of the old database.

Referential Integrity

After an all-AMPs copy, copied tables do not have referential constraints. First, referential constraints are not copied into the dictionary definition tables, database DBC.ReferencedTbls and database DBC.ReferencingTbls, for either a referenced (parent) or referencing (child) table copied into a Teradata Database. Moreover, all referential index descriptors are deleted from the archived table header before it is inserted into the copied table.

For tables that already exist (same name) in the target system, reference constraints remain. However, on any table for which the copied table is a referenced table (a parent table) or a referencing table (a child table), the RI constraint will be marked in the dictionary definition tables as inconsistent.

Output of Copying Objects

Output from 1st copy example

```
...  COPY DATA TABLES (SANDBOX) (FROM (DEMO))
...  , ABORT
...  , RELEASE LOCK
...  , FILE = arc_demo;
...  UTILITY EVENT NUMBER  - 19
...  LOGGED ON    4 SESSIONS
...  "SANDBOX"."Department" CREATED
...  "SANDBOX"."Employee" CREATED
...  "SANDBOX"."Emp_Phone" CREATED
...  "SANDBOX"."Job" CREATED
...  "SANDBOX"."Salary_Log" CREATED
...  STARTING TO COPY DATABASE "SANDBOX"
...  "Raise_Trig" - TRIGGER NOT COPIED
...  DICTIONARY COPY COMPLETED
...  "Department" -   3,446 BYTES, 60 ROWS COPIED
...  "Employee" -  65,077 BYTES,   1,000 ROWS COPIED
...  "Emp_Phone" -  52,504 BYTES,   2,000 ROWS COPIED
...  "Job" -   2,898 BYTES, 66 ROWS COPIED
...  "Salary_Log" - 626 BYTES, 1 ROWS COPIED
...  "SANDBOX" - LOCK RELEASED
```

Output from 2nd copy example

```
...  COPY DATA TABLE (SANDBOX.Employee)
         (FROM (DEMO), NO FALLBACK,  REPLACE CREATOR)
...  , ABORT
...  , RELEASE LOCK
...  , FILE = arc_demo;
...  LOGGED ON    4 SESSIONS
...  UTILITY EVENT NUMBER  - 20
...  *** Warning 3803:Table 'Employee' already exists.
...  STARTING TO COPY TABLE "SANDBOX"."Employee"
...  DICTIONARY COPY COMPLETED
...  "EMPLOYEE" -  65,077 BYTES,   1,000 ROWS COPIED
...  "SANDBOX"."EMPLOYEE" - LOCK RELEASED
...
```

BUILD Statement

The BUILD statement recreates unique and non-unique secondary indexes on non-fallback and fallback tables. This statement also builds fallback rows for fallback tables when the restore statement was performed with the NO BUILD option and generates journal tables by sorting the change images.

You must rebuild indexes for non-fallback tables after a restore operation if any of the following situations occur:

- An AMP vproc is offline during a dump or restore.
- The restore operation is not an all-AMP vproc restore.
- The archive did not include the INDEXES option.
- The restore included the NO BUILD option.

Format

DATA TABLES
JOURNAL TABLES
NO FALLBACK TABLES or NO FALLBACK TABLE

- This identifies the type of table to build.

- The default is NO FALLBACK TABLE.

- Specify DATA TABLES when building fallback, non-fallback, or both types of tables from all AMPs. This option normally follows the restore of a cluster archive.

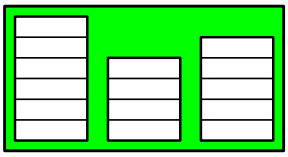
- Specify NO FALLBACK TABLE only when building indexes for non-fallback tables.

The format of the BUILD statement is shown on the facing page. The following example builds unique and non-unique secondary indexes for all tables in Sysdba and any child user/databases. The release lock option removes the utility lock after successful completion of the build operation.

BUILD DATA TABLES (Sysdba) ALL, RELEASE LOCK;

BUILD Statement

NO FALLBACK Data Tables

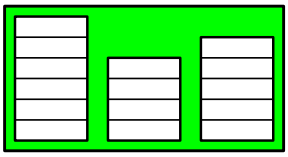


ARC Facility

BUILD
DATA TABLES

- Recreates unique secondary indexes
- Recreates non-unique secondary indexes

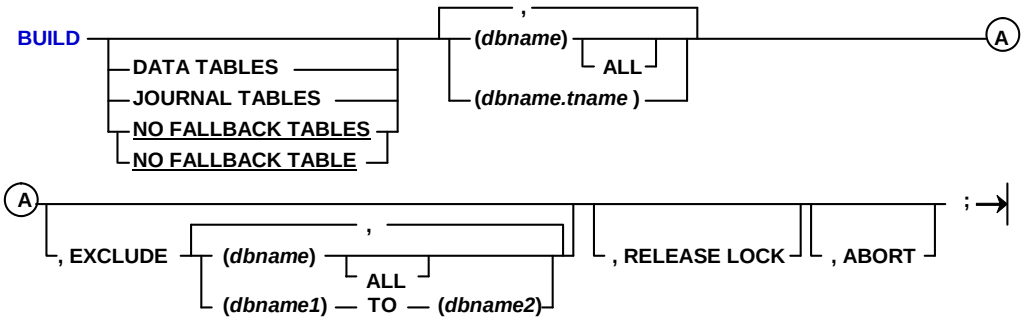
FALLBACK Data Tables



ARC Facility

BUILD
DATA TABLES

- Recreates unique secondary indexes
- Recreates non-unique secondary indexes
- Builds fallback rows



RELEASE LOCK Statement

The ARC utility places locks on database objects while it performs archive and restore activities. These locks are referred to as utility-level locks.

The ARC utility does not automatically release these locks upon successful completion of an ARC command. In fact, these locks remain intact even when an AMP vproc goes down and comes back online. You must submit the RELEASE LOCK statement to remove the locks.

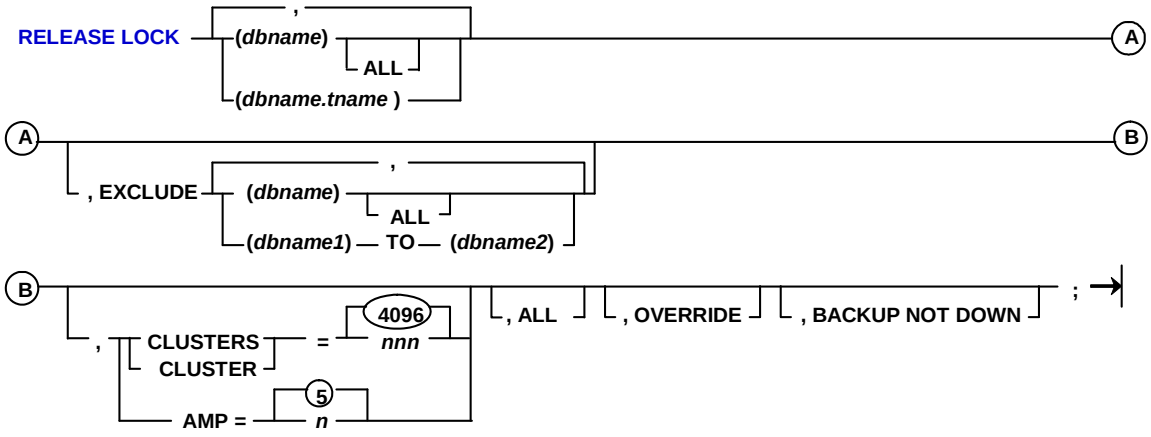
Not everyone can issue the release lock statement. You must have either the DUMP or the RESTORE privilege on the locked object. You can also release a utility-level lock if you are the owner of the locked object.

You may submit the RELEASE LOCK option at the same time you issue ARCHIVE, ROLLBACK, ROLLFORWARD, RESTORE, COPY, and BUILD commands. This accomplishes the same purpose as issuing the RELEASE LOCK statement.

The release lock syntax is shown on the facing page. Options are described below:

ALL	Releases locks on AMP vprocs that are offline when the RELEASE LOCK statement is issued. The utility releases locks when the AMP vprocs return to online operation.
OVERRIDE	Allows locks to be released by a user other than the one who set them. This option requires that the User has the DROP DATABASE privilege on the object or is an owner.
BACKUP NOT DOWN	Allows locks to remain on non-fallback tables (with single after-image journaling) for those AMP vprocs where the permanent journal backup AMP vproc is offline. The utility releases all other locks requested.

RELEASE LOCK Statement



RELEASE LOCK	**You must have ARCHIVE or RESTORE privilege on the object or be the owner.**
OVERRIDE	**You must have DROP DATABASE privilege on the object or be an owner.**
ALL	**Also releases locks on offline AMPs. (Locks released when vproc is returned to service.)**
BACKUP NOT DOWN	**Allows locks to remain on no fallback table (with single after image journals) on vproc whose permanent journal backup vproc is down.**

Revalidate References

When either referenced (parent) or referencing (child) table is restored, the reference is marked inconsistent in the database dictionary definitions. As a result, the system does not allow application users to execute UPDATE, INSERT or DELETE statements on such tables.

The REVALIDATE REFERENCES FOR statement validates the inconsistencies thereby allowing users to execute UPDATE, INSERT and DELETE statements on the tables.

The REVALIDATE REFERENCES FOR statement:

- Validates the inconsistent reference index on the parent table and the child table.
- Creates an error table.
- Inserts rows that fail the referential constraint specified by the reference index into the error table.

If inconsistent restraints remain after you execute the statement, you can use the statement, ALTER TABLE DROP INCONSISTENT REFERENCES, to remove them.

Required Privileges

To use the REVALIDATE REFERENCES FOR statement, the username you have specified in the LOGON statement must have one of the following privileges:

- RESTORE privileges on the table you are revalidating
- Ownership privileges on the database or table

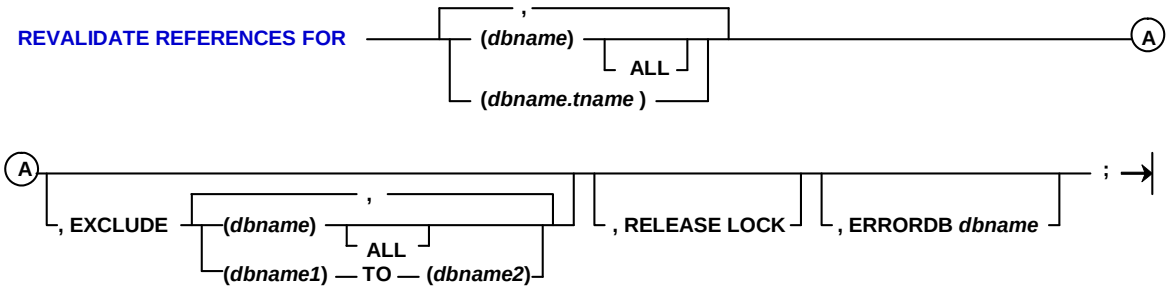
Example

The facing page shows the syntax for the REVALIDATE REFERENCES FOR statement.

Revalidate References

The REVALIDATE REFERENCES FOR statement validates the inconsistencies between foreign and parent keys.

This allows users to execute UPDATE, INSERT and DELETE statements on the tables.



Example:

revalidate_ref_pd.arc

```
LOGON dbc/sysdba,dbapass;
REVALIDATE REFERENCES FOR (PD) , RELEASE LOCK;
LOGOFF;
```

Revalidate references for the PD database.

Revalidate References Output

The facing page illustrates the output from using the REVALIDATE REFERENCES FOR statement in the previous example.

Note: Error tables were **not** created for Department and Emp_Phone because all of the foreign key values are parent key values. The Revalidate References command will only create "error tables" if there are invalid foreign key values.

Revalidate References Output

revalidate_ref_pd.arc

```
LOGON dbc/sysdba,dbapass;
REVALIDATE REFERENCES FOR (PD)
, RELEASE LOCK;
LOGOFF;
```

Output from this restore script

```
 :
...  REVALIDATE REFERENCES FOR (PD)
...  , RELEASE LOCK;

...  VALIDATING REFERENCE INDEX '0' FOR TABLE PD.DEPARTMENT
...  REFERENCES VALIDATED FOR INDEX 0.
...  NO ERRORS FOUND.

...  VALIDATING REFERENCE INDEX '0' FOR TABLE PD.EMPLOYEE
...  REFERENCES VALIDATED FOR INDEX 0.
...  1 ERRORS FOUND AND LOGGED IN TABLE PD.EMPLOYEE_0

...  VALIDATING REFERENCE INDEX '4' FOR TABLE PD.EMPLOYEE
 ...  REFERENCES VALIDATED FOR INDEX 4.
...  1 ERRORS FOUND AND LOGGED IN TABLE PD.EMPLOYEE_4

...  VALIDATING REFERENCE INDEX '8' FOR TABLE PD.EMPLOYEE
...  REFERENCES VALIDATED FOR INDEX 8.
...  1 ERRORS FOUND AND LOGGED IN TABLE PD.EMPLOYEE_8

...  VALIDATING REFERENCE INDEX '0' FOR TABLE PD.EMP_PHONE
...  REFERENCES VALIDATED FOR INDEX 0.
...  NO ERRORS FOUND.

...  PD.EMPLOYEE - LOCK RELEASED
...  PD.DEPARTMENT - LOCK RELEASED
...  PD.JOB - LOCK RELEASED
...  PD.EMP_PHONE - LOCK RELEASED
...  STATEMENT COMPLETED
 :
```

Recovery Control Data Dictionary Views

The system views that contain information about ARC utility events are listed below. The name, purpose, and dictionary table name of each view is also included.

DBC.Association[V][X] Views

These views provide information about objects that have been imported from another Teradata Database system or otherwise created using the ARC COPY statement. The associated table name is DBC.RCEvent.

DBC.Events[V][X] Views

These views provide a row for each archive and recovery activity. The associated table name is DBC.RCEvent.

DBC.Events_Configuration[V][X] Views

These views provide information about archive and recovery activities that do NOT affect all AMP vprocs. The associated table name is DBC.RCConfiguration.

DBC.Events_Media[V][X] Views

These views provide information about archive and recovery activities that involve removable media. The associated table name is DBC.RCMedia.

Recovery Control Data Dictionary Views

View Name	Description
DBC.Association[V][X]	Provides information about about objects you import from another database system (an example is provided).
DBC.Events[V][X]	Provides an audit trail of all archive and recovery activity (an example is provided).
DBC.Events_Configuration[V][X]	Provides information about archive and recovery activities that did not affect ALL AMPs.
DBC.Events_Media[V][X]	Provides information about archive and recovery activities that involve removable media.

Association View

The DBC.Association[V][X] views allow you to retrieve information about an object imported from another Teradata Database.

An existing object created with the ARC utility COPY statement also displays in the Association view. If you later drop a copied object from its new destination, the information is deleted from the Association table and is no longer available.

Example

The example on the facing page uses the Association view to list all tables, views, or macros that were copied into the Sandbox database. The result of the query displays imported table names. The object column displays the current name of each table. The “From_Source” column provides the name of the original table. The event column shows the event number assigned to the copy operation.

Association View

Provides information about COPY operations.

Enables you to retrieve information about an object imported from another Teradata database.

DBC.Association[V][X]

DatabaseName*	TableName	EventNum
Original_DatabaseName	Original_TableName	
Original_TableKind	Original_Version	
Original_ProtectionType	Original_JournalFlag	
Original_CreatorName	Original_CommentString	

* DatabaseName: The name of the database or user where the imported object now resides.

Example: List all objects copied into the Sandbox database.

```
SELECT    TRIM (DatabaseName)             || '.' || TableName             (FORMAT 'X(25)') AS Object
         ,TRIM (Original_DatabaseName) || '.' || Original_TableName (FORMAT 'X(25)') AS From_Source
         ,EventNum                                                  (FORMAT 'Z(4)9') AS Event
FROM      DBC.AssociationV
WHERE     DatabaseName LIKE '%Sandbox%'
ORDER BY  Event_Num, Object;
```

Object	From_Source	Event
Sandbox.Department	DEMO.Department	19
Sandbox.Emp_Phone	DEMO.Emp_Phone	19
Sandbox.Job	DEMO.Job	19
Sandbox.Salary_Log	DEMO.Salary_Log	19
Sandbox.Employee	DEMO.Employee	20

Events View

The DBC.Events[X] views track ARC activity. The ARC utility inserts a new row in the Events system table each time another ARC activity begins. The Events views return a row for each activity tracked. Each event type is listed below:

Checkpoint Event Row	Created for each journal checkpoint
Copy Event Row	Created for each database or table copied
Delete Event Row	Created for each journal deleted
Dump Event Row	Created for each database or table dumped
Restore Event Row	Created for each database or table restored
Rollback Event Row	Created for each database or table rolled back
Rollforward Event Row	Created for each database or table rolled forward

Example

The SQL statement on the next page requests a list of all ARC activity that took place on January 13, 2011. The results display five ARC activities.

Events View

Provides an audit trail of all archive and recovery activities for objects visible to you.

DBC.Events[V][X]

CreateDate	AllAMPsFlag	LockMode	CreateTime
RestartSeqNum	JournalUsed	EventNum	OperationInProcess
JournalSaved	EventType	TableName	IndexPresent
UserName	CheckpointName	DupeDumpSet	DatabaseName
LinkingEventNum*	ObjectType	DataSetName	

Example:

List all ARC activity that occurred on Jan 13, 2010.

```
SELECT    CreateDate
          ,EventNum        (FORMAT 'Z(4)9') AS Event
          ,UserName        (FORMAT 'X(12)')
          ,EventType       (FORMAT 'X(12)')
          ,DatabaseName    (FORMAT 'X(12)') AS DBName
FROM      DBC.EventsV
WHERE     CreateDate = '2011-01-13'
ORDER BY  EventNum ;
```

Example Results:

CreateDate	Event	UserName	EventType	DBName
2011-01-13	15	SYSDBA	Dump	PD
2011-01-13	16	SYSDBA	Dump	PD
2011-01-13	17	SYSDBA	Restore	PD
2011-01-13	18	SYSDBA	Restore	PD
2011-01-13	19	SYSDBA	Copy	SANDBOX
2011-01-13	20	SYSDBA	Copy	SANDBOX

Restoring Data Summary

The facing page summarizes some important concepts regarding this module.

Summary

- Restore operations transfer database information from archive files stored on portable media to all AMP vprocs, AMP clusters or specified AMP vprocs.
- You can restore archived data tables to the database if the data dictionary contains a definition of the entity you wish to restore.
- The primary statements that you use in recovery operations are:
  - ANALYZE
  - REVALIDATE REFERENCES FOR
  - RESTORE
  - RELEASE LOCK
  - COPY
  - BUILD
- Teradata features several recovery control system views that contain information about ARC utility events.

Module 59: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 59: Review Questions

1. True or False. You can use the RESTORE command to restore entities that are not defined in the data dictionary.

2. True or False. When you execute a RESTORE of a database, any tables or views created since the archive of the database are dropped when you restore the database.

3. True or False. You can use the COPY operation to copy tables, views, macros, and triggers from one system to another system.

4. The REVALIDATE REFERENCES FOR statement is used to validate Referential Integrity between tables that are identified as _____.

   a. Invalid
   b. Missing
   c. Unresolved
   d. Inconsistent

Notes

Module 60

TERADATA
Raising Intelligence

Data Recovery Operations

After completing this module, you will be able to:

- **Describe how to use the following statements to recover archived data back to the Teradata Database:**
  - **CHECKPOINT**
  - **DELETE JOURNAL**
  - **ROLLBACK**
  - **ROLLFORWARD**

Notes

Table of Contents

Data Recovery Using Roll Operations 60-4
The CHECKPOINT Statement 60-6
CHECKPOINT WITH SAVE Statement 60-8
    Example 60-8
    Checkpoint Lock Mechanisms 60-8
    Checkpoint with Offline AMPs 60-8
Using the ROLLBACK Command 60-10
    Example 60-10
    NO DELETE Option 60-10
The ROLLBACK Statement 60-12
ROLLFORWARD Statement 60-14
    Example 60-14
    PRIMARY DATA Option 60-14
ROLLFORWARD Restrictions 60-16
  AMP-Specific Restore 60-16
  All-AMP Restore 60-16
    Example 60-16
The ROLLFORWARD Statement 60-18
DELETE JOURNAL Statement 60-20
    Access Privileges 60-20
    Restrictions 60-20
Summary 60-22
Module 60: Review Questions 60-24

Data Recovery Using Roll Operations

The restore statement allows you to move information from archive files back to the Teradata database. The restore operation can restore data or journal tables.

After you execute a RESTORE statement, data tables are ready to use.

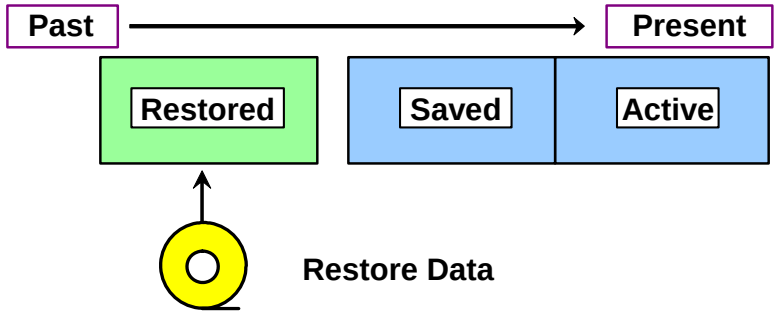
When you restore a journal table, the system restores the information to a permanent journal subtable. Before you can use the tables, you must perform a rollback or rollforward operation to move the journal tables back to the data tables.

Roll operations can use either the current journal or the restored journal. If you specify the current journal, then the ARC utility uses information stored in both the active and saved subtables.

A permanent journal is checkpoint-oriented rather than transaction-oriented. The goal of the journals is to return existing data tables to some previous or subsequent checkpoint. For example, if a batch program corrupted existing data, the rollback operation would return the data to a checkpoint prior to the running of the batch job.

A rollforward operation might occur after an all-AMP restore. After you move the data and journal archive files back to the database, the data tables would only include changes committed since the last full backup. Any intermediate changes would reside in the journal tables. The rollforward operation would replace the existing data with changes from the journal table.

Data Recovery Using Roll Operations



- The RESTORE function copies journal archive files to the restored subtable of the permanent journal.
- ROLLBACK and ROLLFORWARD statements apply journal table contents to data tables.
- Roll operations can use:
  - Current journal (active and saved subtable)
  - Restored journal (restored subtable)

The CHECKPOINT Statement

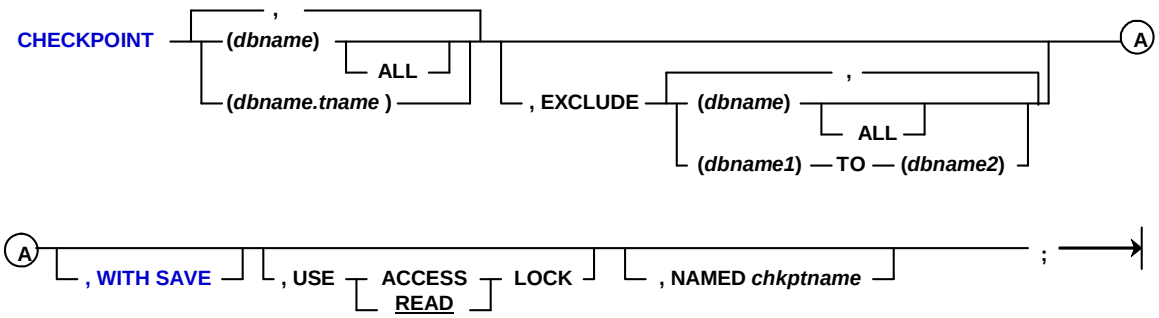
Use the CHECKPOINT statement to indicate a recovery point in the Journal.

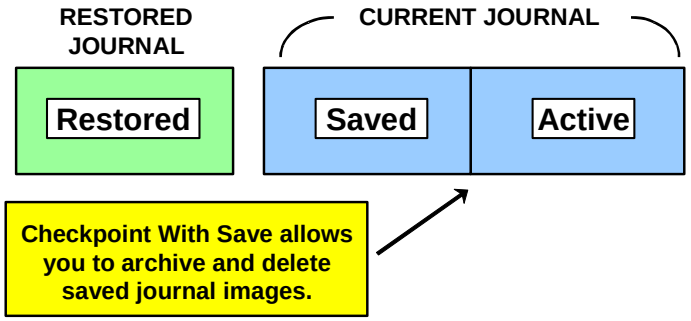
The CHECKPOINT statement places a marker row after the most recent change image row in the active subtable of a permanent journal. The database assigns an event number to the marker row and returns the number in response. You may assign a name to the CHECKPOINT command rather than use the event number in subsequent ARC activities.

Use the following options with the CHECKPOINT statement:

WITH SAVE	Use this option before you archive saved journal images to a host media to append the active journal subtable to the saved journal subtable. After you archive the saved area of the journal, you can delete this section of the current journal to make space for subsequent saved journal images. The saved journal subtable has no fixed size and can grow to the limit of the database.
USE ACCESS LOCK	A checkpoint with save may optionally use an access lock. Without this option, the system must acquire a read lock on all tables assigned to the journal being checkpointed. Updates to tables may continue when you use the access lock option. Updates in progress complete after the checkpoint.
NAMED checkpointname	Checkpoint names may be up to 30 characters long and are not case-specific. Teradata software always supplies an event number for each checkpoint. Use the number to reference a checkpoint if a name is not supplied. If there are duplicate checkpoint names in the journal and an event number is not specified: • Rollforward uses the first (oldest) occurrence. • Rollback uses the last (latest) occurrence.

The CHECKPOINT Statement





CHECKPOINT WITH SAVE Statement

The CHECKPOINT WITH SAVE option inserts a marker row and appends any stored images preceding the marker row from the active to the saved subtable. The database automatically initiates a new active subtable. You can dump the contents of the saved subtable to an archive file.

Example

The facing page shows two different current journals, before and after a checkpoint operation. The active subtable before checkpoint contains five change image rows. After checkpoint with save, the active subtable is empty, and the saved subtable contains the five change rows and a marker row.

Checkpoint Lock Mechanisms

The default lock mechanism for the checkpoint command is read lock. The read lock suspends update activity for all data tables that might write changes to the journal table during checkpoint. This lock provides a clean point on the journal.

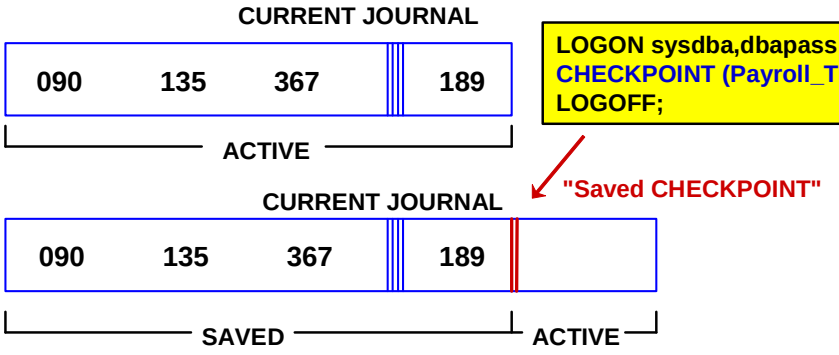
The USE LOCK option permits users to assign an access lock rather than a read lock. The access lock accepts all transactions that insert change images to the journal, but it treats them as though they were submitted after the checkpoint was written. The access lock option requires that you also use the WITH SAVE option.

Since users cannot know which side of a checkpoint a particular transaction will fall on, restoring to a checkpoint created under an access lock cannot guarantee that transactions in progress at the time of the checkpoint will be included in that restore.

Checkpoint with Offline AMPs

An individual AMP may be off-line when you issue the checkpoint command. In this case, the utility automatically generates a system log entry that marks the checkpoint as soon as the AMP comes back on-line. The system startup process generates the checkpoint and requires no user input.

CHECKPOINT WITH SAVE Statement



To create a saved checkpoint, archive the saved journal, and delete the images in the permanent journal, you could execute the following script.

```
LOGON sysdba,dbapass;
CHECKPOINT (Payroll_Tab.Salaries_Jnl), WITH SAVE;
ARCHIVE JOURNAL TABLE (Payroll_Tab.Salaries_Jnl), ABORT, RELEASE LOCK, FILE = pay_jrl;
DELETE SAVED JOURNAL (Payroll_Tab.Salaries_Jnl);
LOGOFF;
```

Using the ROLLBACK Command

The ROLLBACK command helps you recover from one or more transaction errors. It reverses changes made to a database or table. To accomplish this reversal, it replaces existing data table rows with before-change images stored in a permanent journal. The before-change images must reside in either the restored or current subtables of a permanent journal. If you choose the current subtable for rollback procedures, the database uses the contents of both the active and saved subtables.

When you use the restored subtable for rollback procedures, you need to verify it contains the desired journal table. If it does not, submit the RESTORE JOURNAL TABLE command with the appropriate removable storage media. This process ensures that you restore the correct subtable contents. The Teradata database does not have any simple tools for looking at journal subtables to determine that they contain the desired data.

Example

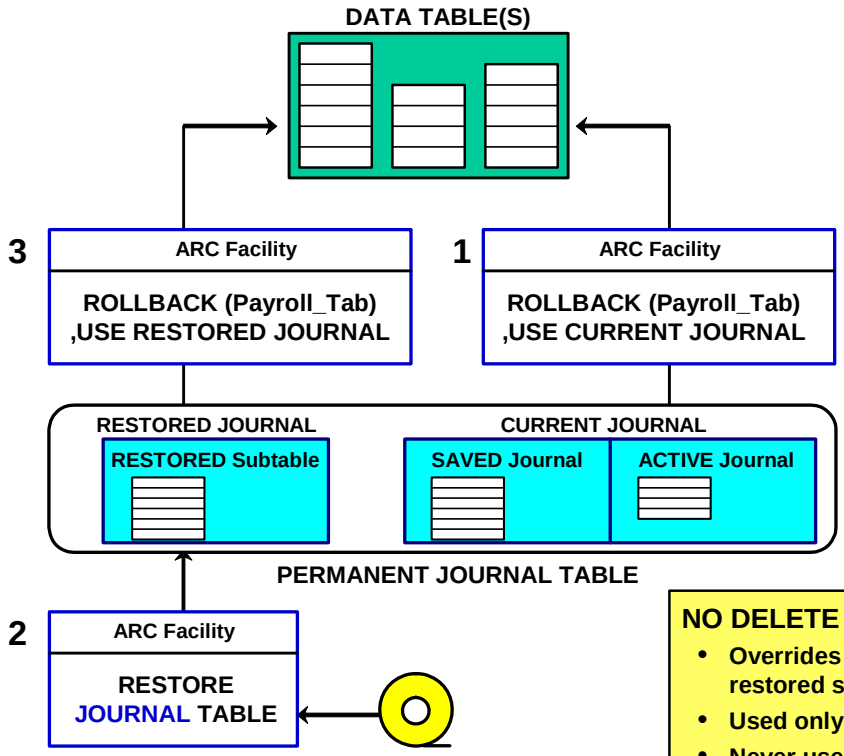
The example on the facing page illustrates a rollback procedure. First, (step 1), activate the ROLLBACK CURRENT JOURNAL statement to rollback any changes made since the journal table was archived. This statement rolls back the current subtable. Next (step 2), run the RESTORE JOURNAL TABLE command to load the appropriate archive file into the restored subtable of the permanent journal.

Finally (step 3), submit the ROLLBACK RESTORED JOURNAL command to reverse the changes by replacing any changed rows with their before-image rows stored in the restored journal. Repeat Steps 2 and 3 as necessary

NO DELETE Option

By default, the rollback procedure automatically deletes the contents of the restored subtable after successfully completing the command. The NO DELETE option overrides the default, enables you to recover selected tables first, and then later recovers other tables that may have changes in the journal.

Using the ROLLBACK Command



NO DELETE Option

- Overrides automatic deletion of restored subtable
- Used only for restored subtables
- Never used with current subtables

The ROLLBACK Statement

To recover from one or more transaction errors, use the ROLLBACK statement. To use this statement, you must define the table with a before-image journal table. The ROLLBACK is performed to a checkpoint or to the beginning of the current or restored journal.

The system uses the before images to replace any changes made to the table or database since a particular checkpoint was taken.

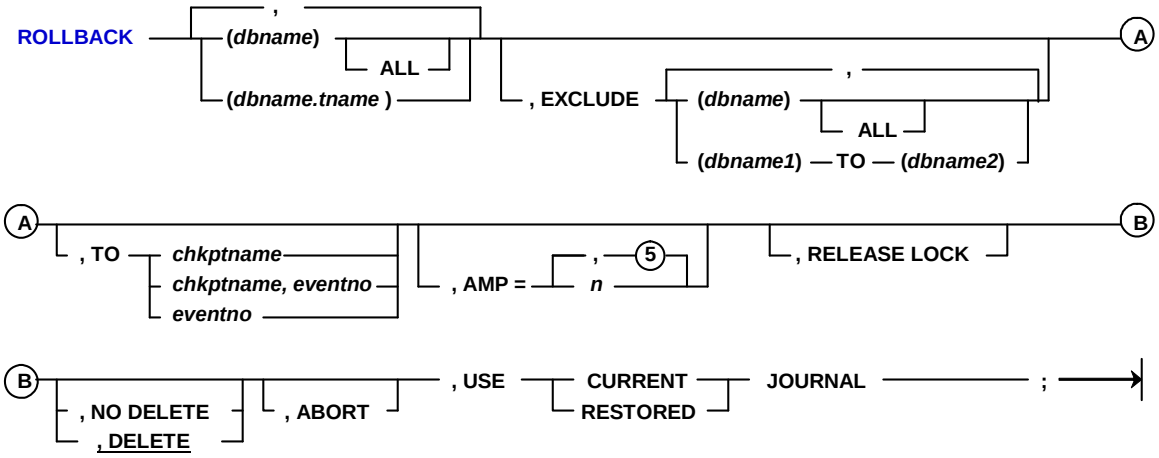
The facing page shows the format of the ROLLBACK statement. A description of the "TO CHECKPOINT" option follows:

TO checkpointname, eventno

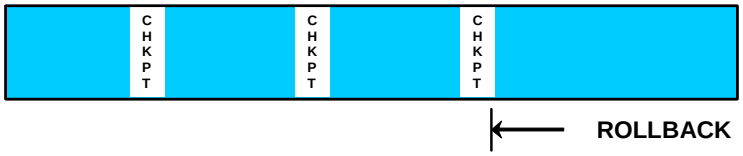
Checkpoint names need to match existing names used with a previous CHECKPOINT statement. An eventno is the software-supplied event number of a previous checkpoint. You can supply either one of these or both. To find the checkpoint names or event numbers, select information about the checkpoint from the DBC.Events view.

If there are duplicate checkpoint names in the journal and an event number is not supplied, rollback stops at the last chronological entry made with a matching name.

The ROLLBACK Statement



Use ROLLBACK to recover from a transaction error.



ROLLFORWARD Statement

The ROLLFORWARD command helps you recover from a hardware error and changes existing rows in data tables by replacing them with after-change images stored in a permanent journal. The after-change images must reside in either the restored or current subtables of a permanent journal.

When you use the restored subtable for rollforward procedures, you need to verify that it contains the desired journal table. If it does not, submit the RESTORE JOURNAL TABLE command with the appropriate portable storage media. This process ensures that you restore the correct subtable.

Example

The example on the facing page illustrates a rollforward procedure. First, the administrator runs the RESTORE DATA TABLE command. Then, he/she runs the RESTORE JOURNAL TABLE command to load the appropriate archive files into the restored permanent journal subtable. Next, he/she submits the ROLLFORWARD RESTORED JOURNAL command to replace existing data table rows with their after-image rows stored in the restored journal.

Lastly, he/she activates the ROLLFORWARD CURRENT JOURNAL statement to rollforward any changes made since the journal table was archived. This statement rolled forward the saved subtable first followed by the active subtable.

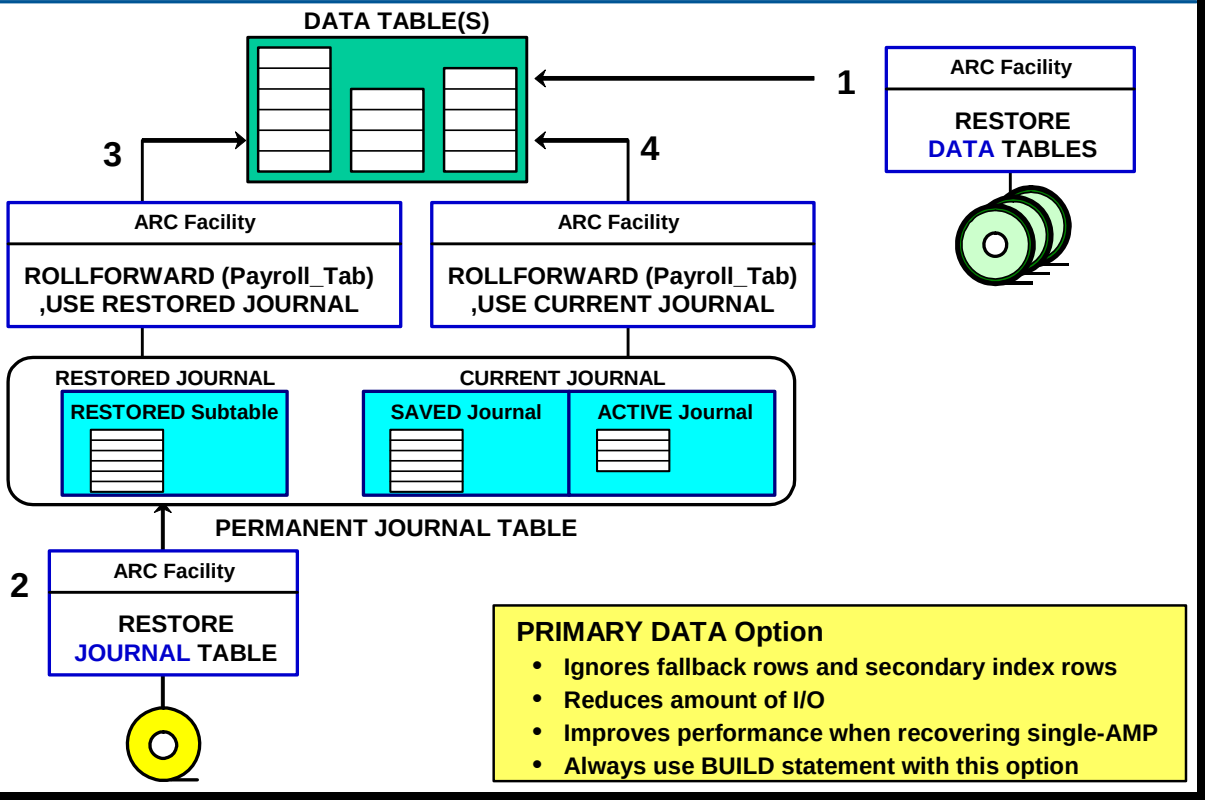
PRIMARY DATA Option

This option replaces only primary row images during the rollforward process. It ignores secondary index and fallback rows.

If you use this option with a rollforward operation, you can reduce the amount of I/O. It also improves the rollforward performance when recovering a specific AMP from disk failure.

Unique indexes are invalid when recovering a specific AMP. Always submit a BUILD statement when the rollforward command includes the PRIMARY DATA option.

Using the ROLLFORWARD Command



PRIMARY DATA Option

- Ignores fallback rows and secondary index rows
- Reduces amount of I/O
- Improves performance when recovering single-AMP
- Always use BUILD statement with this option

ROLLFORWARD Restrictions

The diagrams on the facing page illustrate several important restrictions in using the ROLLFORWARD statement.

AMP-Specific Restore

If you perform a restore operation on a specific AMP rather than on all AMPs, the ROLLFORWARD command does not permit you to use the TO CHECKPOINT NAME option. Following an AMP-specific restore, the system permits a rollforward only to the end of the journal. You must follow up the restore process with a rollforward of the entire journal table.

All-AMP Restore

When you perform an all-AMP restore, you choose whether to submit the ROLLFORWARD command with the TO CHECKPOINT NAME option, or to the end of the journal.

The PRIMARY DATA option of the ROLLFORWARD statement indicates that the operation should ignore secondary index and fallback rows that will reduce the amount of I/O during rollforward. If you use this option, follow up with the BUILD statement.

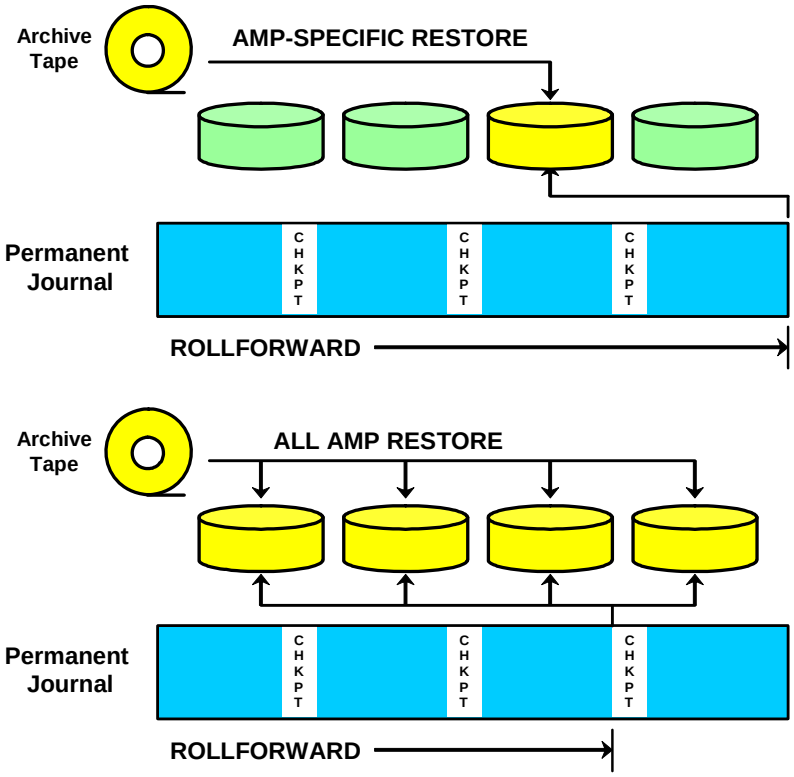
Note: Use the DBC.Events view to determine event numbers and/or checkpoint names.

Example

```
SELECT EventNum FROM DBC.Events WHERE CreateDate = '2011-02-15';

SELECT CheckPointName FROM DBC.Events WHERE CreateDate = '2011-02-15';
```

ROLLFORWARD Restrictions



Following an AMP-specific restore, a ROLLFORWARD is permitted only to the end of the journal.

After an ALL-AMP restore, a ROLLFORWARD may be done to a checkpoint, or to the end of the journal.

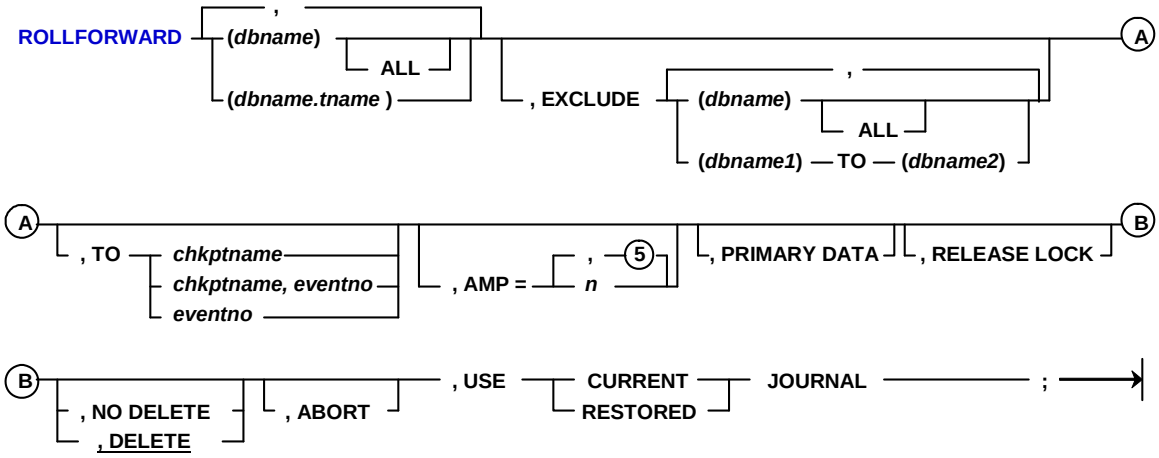
The ROLLFORWARD Statement

Use the ROLLFORWARD statement to recover from a hardware error. Before you can rollforward, you must have a backup copy of the table rows and AFTER Image journal rows since the last backup.

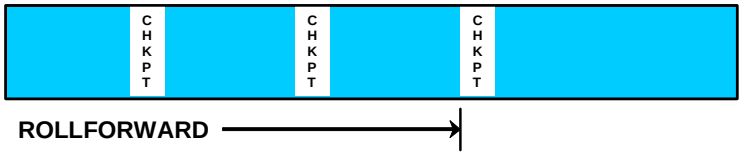
The format of the ROLLFORWARD statement is shown on the next page. A description of some of the options follows:

PRIMARY DATA	During a rollforward operation, this option instructs the software to ignore secondary index and fallback row updates. A BUILD operation will rebuild the invalidated fallback copy and indexes.
TO checkpointname, eventno	Checkpoint names need to match existing names used with a previous CHECKPOINT statement. An event number is the software-supplied event number of a previous checkpoint. You can supply either one or both of these. To find the checkpoint names or event numbers, select information about the checkpoint from the DBC.Events view. If there are duplicate checkpoint names in the journal and an event number is not supplied, rollforward stops when it encounters the first chronological entry made with a matching name.

The ROLLFORWARD Statement



Use ROLLFORWARD to recover from a hardware failure.



DELETE JOURNAL Statement

The DELETE JOURNAL command enables you to erase the contents of either the restored subtable or the saved subtable of a permanent journal. You cannot delete the contents of the active subtable. You must have the RESTORE privilege to execute this command.

The facing page shows the DELETE JOURNAL statement.

Access Privileges

To delete a journal table, the user name specified in the LOGON statement must have one of the following:

- The RESTORE privilege on the database or journal table being deleted
- Ownership of the database containing the journal table

Restrictions

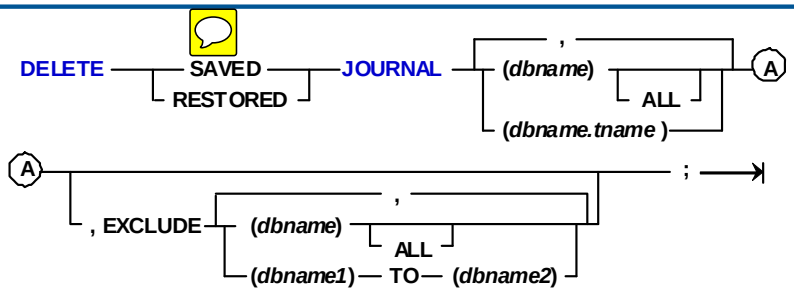
You cannot delete a saved subtable when all of the following conditions are true:

- A CHECKPOINT statement in the archive utilized an access lock, and
- The journal is not dual image, and
- One or more AMPs are off-line.

Transactions between an all-AMP archive and a single-AMP archive may not be consistent when a journal archive has all three of the above conditions. You cannot delete a saved subtable with an AMP off-line that does not have a dual journal.

The command does not delete the rows in the active journal.

DELETE JOURNAL Statement



To use the DELETE JOURNAL statement, you must have the RESTORE privilege or own the database that contains the journal. Note: you cannot delete rows from an active journal.

To create a saved checkpoint and delete the images in the permanent journal, you could execute the following script.

```
LOGON sysdba,dbapass;
CHECKPOINT (Payroll_Tab.Salaries_Jnl), WITH SAVE;
DELETE SAVED JOURNAL (Payroll_Tab.Salaries_Jnl);
LOGOFF;
```

Summary

The facing page summarizes some important concepts regarding this module.

Summary

- As like archive and restore operations, you use the ARC facility for recovery operations.
- Roll operations can use either current journals (active and saved subtables) or restored journals (restored subtable).
- The CHECKPOINT statement indicates a recovery point in a journal.
  - The CHECKPOINT WITH SAVE statement saves stored images before a row marker in an active subtable and appends them to the saved subtable.
- ROLLBACK commands help you recover from one or more transaction errors and reverses changes made to a database or table.
- ROLLFORWARD commands help you recover from hardware errors. These commands replace existing row data with after-change images.
- DELETE JOURNAL command erases the contents of either the restored subtable or the saved subtable in the permanent journal.

Module 60: Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.

Module 60: Review Questions

1. True or False. The DELETE JOURNAL command can be used to delete the active and the saved areas of the current journal.

2. In general, rollback operations help you recover from _____ failures and rollforward operations help you recover from _____ failures.

3. To use the ARCHIVE JOURNAL TABLE command to archive a permanent journal, the active journal images need to be moved to the saved area of the current journal. The command to do this is:

   _____

Notes

Module 61

Teradata Factory Recap

After completing this module, you will be able to:

- **Identify those Data Dictionary tables that need periodic maintenance.**
- **Identify those administrative functions that are NOT necessary with Teradata.**

Notes

Table of Contents

Teradata Factory Review – Week 1 61-4
Teradata Factory Review – Week 2 61-6
Dictionary Tables to Maintain 61-8
Plan and Follow-up 61-10
Things You Never Have to do with Teradata 61-12
Things You Never Have to do with Teradata (cont.) 61-14
Teradata Differentiators 61-16
Teradata Certification Tests 61-18

Teradata Factory Review – Week 1

The facing page lists some of the key topics that were covered in the first week of this course.

Teradata Factory Review – Week 1

Teradata Concepts – Big Picture view of Teradata

- Teradata Database concepts, architecture, and terminology
- Data protection mechanisms (RAID, Cliques, Clusters, Locks, and Journals)
- Teradata Systems and Configurations (e.g., Teradata 6650 and 6680)
- How Teradata uses memory and utilizes disk array storage

GOAL – Understand basic concepts of Teradata and how Teradata fits with MPP systems

Physical Database Design and Implementation – Detailed view of Teradata

- Data Distribution, Hashing, and Index Access
- Analyze Primary Index Criteria and choose Primary Indexes
- Analyze Secondary Index Criteria and choose Secondary Indexes
- Access Issues, Constraints, Sizing, and Statistics
- Understand Join Processing and interpret EXPLAIN plans of joins
- Additional Index Choices – Join Indexes, Hash Indexes, etc.

GOAL – Create tables with appropriate attributes and indexes.

Teradata Factory Review – Week 2

The facing page lists some of the key topics that were covered in the second week of this course.

Teradata Factory Review – Week 2

Teradata Application Utilities – Load and Export data

- Utilization of BTEQ, FastLoad, FastExport, MultiLoad, and TPump utilities

GOAL – Create load and export scripts to load/export data into/from Teradata tables

Teradata Database Administration

- Creating and using the Database Environment and Hierarchy
- The Data Dictionary/Directory 
- Space Allocation and Usage
- Management of Users and Databases
- Controlling access to system via Access Rights, Roles, and Profiles
- Monitoring system activity and logging user access and queries
- Workload Management (TASM)
- Utilization of tools such as Teradata Administrator and Viewpoint
- System utilities – administrative, maintenance, and recovery

GOAL – Administration of a Teradata Database using DBA commands and utilities

Dictionary Tables to Maintain

You need to maintain some dictionary tables. The following pages list these tables and describe the maintenance.

Dictionary Tables to Maintain

Reset accumulators and peak values using DBC.AMPUsage view and the ClearPeakDisk macro provided with the software.

- **DBC.Acctg**
  Resource usage by Account/user
- **DBC.DataBaseSpace**
  Dbase and Table space accounting

Teradata automatically maintains these tables, but good practices can reduce their size.

- **DBC.AccessRights**
  User Rights on objects
- **DBC.RoleGrants**
  Role rights on objects
- **DBC.Roles**
  Defined Roles
- **DBC.Accounts**
  Account Codes by user

Archive these logging tables (if desired) and purge information 60-90 days old. Retention depends on customer requirements.

- **DBC.SW_Event_Log**
  Database Console Log
- **DBC.ResUsage**
  Resource monitor tables
- **DBC.EventLog**
  Session logon/logoff history
- **DBC.AccLogTbl**
  Logged User-Object events
- **DBC.DBQL tables**
  Logged user/SQL activity

Purge these tables when the associated removable media is expired and over-written

- **DBC.RCEvent**
  Archive/Recovery events
- **DBC.RCConfiguration**
  Archive/Recovery config
- **DBC.RCMedia**
  VolSerial for Archive/Recovery

Plan and Follow-up

Establish a set of procedures that will help you administer the Teradata Database. Document these procedures and periodically refer to them.

Plan and Follow-up

Review the material you have learned and experienced in this course. From it, develop a checklist of tasks. For example:

1. **Set up a job that periodically checks the size of your dictionary tables.**
2. **Set up a job that periodically checks the size of your application databases. Evaluate them for even data distribution on AMPs. Ensure that user's permanent space is being used efficiently. Reallocate space if necessary.**
3. **Verify adequate Spool_Reserve.**
4. **Set up and document the definition of users, roles, profiles, privileges, and an accounting system (if you don't have one already).**
5. **Check the allocation of the Crashdumps database.**
6. **Install and run the ResUsage macros at regular intervals. Evaluate and review reports for even distribution of processing.**

Things You Never Have to do with Teradata

The facing page lists a number of database maintenance functions that you never have to do with the Teradata database.

Things You Never Have to do with Teradata

Implementation

1. Create and format data files to hold the data and the indexes.
2. Determine the physical location of each table and index partition or simple tablespace.
3. Write programs to determine how to divide data into partitions.
4. Code the space allocation for each partition or underlying file structures.
5. Embed partitioning assignments into CREATE TABLE statements.
6. Code the definition and allocation for temporary work space.
7. Create, size, and determine the content of tablespaces.
8. Associate tables and/or queries with degrees of parallelism.
9. Add hints or otherwise rewrite SQL.
10. Determining the level of parallelism to be assigned to tables or indexes.
11. Assign and manage special buffer pools for parallel processing.
12. Create rollback segments or log files.
13. Ensure that the data is spread evenly across disks and controllers.
14. Build summary tables before end users can access the data warehouse.
15. Carefully build indexes on tables and summary tables to support index only access for performance, based on known queries – these indexes may or may not aid in ad hoc access.
16. Build and partition materialized view logs.
17. Build and partition indexes on top of materialized views.
18. Determine how materialized views are updated, asynchronously or synchronously.

Things You Never Have to do with Teradata (cont.)

The facing page lists a number of database maintenance functions that you never have to do with the Teradata database.

Things You Never Have to do with Teradata (cont.)

Support

19. **Monitor partition size.**
20. **Monitor and tune temporary work and sort spaces.**
21. **Monitor and tune buffer pool assignments.**
22. **Monitor and tune parameters and control blocks that enable parallel execution.**
23. **Perform periodic table and index reorgs (unloads and reloads, dropping and rebuilding).**
24. **Convert data types of mainframe data sets prior to a data warehouse load.**
25. **Setting up multiple load jobs from a mainframe to the data warehouse in order to load a single table in parallel.**
26. **Manually restart the multi-step load process when failure occurs.**
27. **Sort and/or split the data before a load job.**

Growth and Leverage

28. **Alter the parallelism assignments as the number of users or data volume increases.**
29. **Expand partition boundaries or relocate partition data sets.**
30. **Add or delete table or index partitions as tables grow.**

Teradata Differentiators

The facing page identifies a number of key Teradata differentiators.

Teradata Differentiators

- **Product Maturity** – Teradata has focused on data warehousing needs since 1984.
- **Customer References** – Teradata's impressive list of customers includes leaders in their respective industries and also in the use of data warehousing technology.
- **Quickest Time to Solution** – By the nature of the inherent parallel architecture and self-managing features, Teradata provides the flexibility needed for rapid, initial implementation, and ongoing extensibility.
- **Lowest Total Cost of Ownership** – even the largest Teradata sites report having two or fewer full-time DBAs.
- **Complete Support Infrastructure** – Teradata is supported by the most skilled and experienced data warehousing professionals with 20+ years of experience.
- **Effortless Scalability** – unique, unconditional parallelism and automatic hashed data distribution are the key reasons behind Teradata's scalability.
- **High User Concurrency** – Teradata offers industry-leading performance to increasing numbers of satisfied users as the warehouse workload grows.
- **Complex and Ad Hoc Query Performance** – Teradata's parallel-aware, cost-based optimizer provides for advanced ad hoc and complex query environment.
- **Fast, Fail-safe Data Load Utilities** – Teradata ensures mission-critical availability of the information by allowing load/restore activities while users access the warehouse.
- **Seamless Mainframe Integration** – Teradata offers bi-directional, high-speed channel connectivity to leading mainframe environments.

Teradata Certification Tests

The facing page lists the various Teradata certification tests. Depending upon the tests that are completed, you can earn various Teradata Certified designations such as Teradata Certified Professional.

The Teradata 12 Certification tests require knowledge plus experience with Teradata. This manual will help you prepare for these Teradata 12 tests, but many of the test questions are scenario-based and Teradata experience is needed to answer these types of questions.

Disclaimer: The Teradata Certification tests include questions from a mix of sources and **require experience**, especially on tests other than the Basics test. However, some suggestions on Teradata Factory modules to concentrate on for the different tests include:

1 – Teradata 12 Basics (modules 1–8, 12, 16, 31, 32, 41)
2 – Teradata 12 SQL (Teradata WBT classes)
3 – Teradata 12 Physical Design and Implementation (modules 8, 10, 12–32, 40–43)
4 – Teradata 12 Database Administration (modules 8, 10, 12, 22–25, 30–32, 41–60)
5 – Teradata 12 Solutions Development (modules 12–40)
6 – Teradata 12 Enterprise Architecture (entire course)
7 – Teradata 12 Comprehensive Mastery (entire course)

Teradata Certification Tests

Teradata 12.0 Certification Tests

- ✓ 1 – Teradata 12 Basics
- 2 – Teradata 12 SQL
- ✓ 3 – Teradata 12 Physical Design and Implementation
- ✓ 4 – Teradata 12 Database Administration
- 5 – Teradata 12 Solutions Development
- 6 – Teradata 12 Enterprise Architecture

- 7 – Teradata 12 Comprehensive Mastery

By passing all seven Teradata 12 certification tests, you become a Teradata 12 Certified Master.

✓ This course (along with Teradata experience) will prepare you for these tests.

Options for Teradata V2R5 Certified Masters:

- **The Teradata 12 Qualifying Exam is available as an alternative to taking tests 1 – 6.**
- **To achieve the Teradata 12 Master certification …**
  1. **Pass the Teradata 12 Qualifying Exam OR pass each of the 6 tests**
  2. **Pass the Teradata 12 Comprehensive Mastery exam**

Notes

TERADATA
Raising Intelligence

Appendix A: The Lab Environment

This appendix describes the lab environment that will be used during the class.

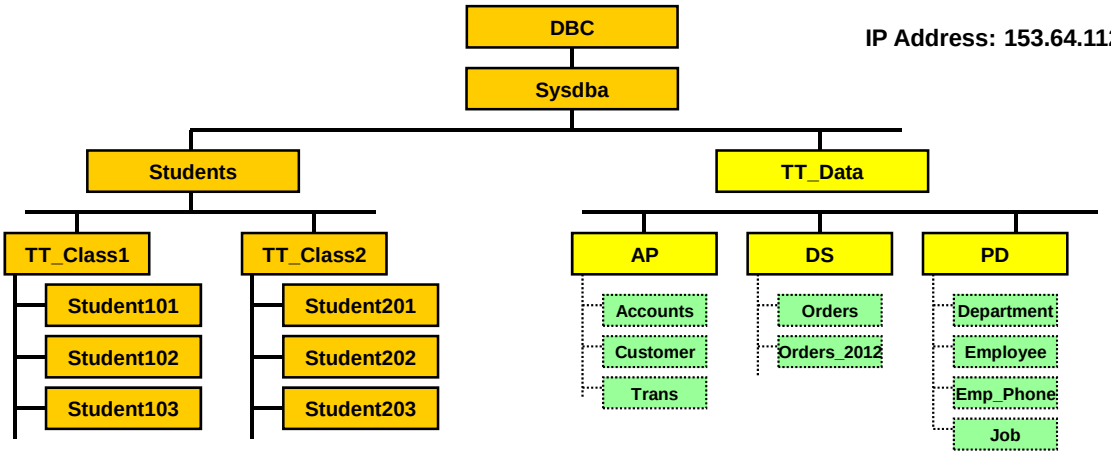
Lab Environment Notes

One way to connect from a Windows workstation to a Teradata server to a Linux system is to use a Secure Shell connection. This opens a terminal window which you can use to logon to Linux and enter Linux commands.

putty is an example of a utility that provides a secure shell connection

If you are executing BTEQ interactively, you cannot enter your password as part of the .LOGON statement. BTEQ will prompt you for your Teradata username password.

Lab Environment



IP Address: 153.64.112.7

Common Teradata SQL Commands:

To populate a newtable:

```
INSERT INTO newtable
  SELECT * FROM db.oldtab WITH [NO] DATA;
```

HELP USER username;	- lists objects owned by user
HELP 'SQL sqlcommand';	- help for an SQL command
SHOW TABLE tablename;	- shows table creation syntax
SELECT COUNT(*) FROM tname;	- provides count of rows
DELETE tablename ALL;	- deletes all rows in a table
DROP TABLE tablename;	- drops table from the database

Useful Linux Commands:

ls -l	- long listing of UNIX files)
more filename	- view contents of text file
vi filename	- create or edit a text file
cp file1 file2	- copies file1 to file2
mv oldname newname	- renames a file
rm filename	- delete a file
exit	- exits Linux

Notes

AP Tables

AP.Accounts (10,000 Rows)

Account Number	Street Number	Street	City	State	Zip Code	Balance Forward	Balance Current
PK							
UPI							
20024010	123	Harbor Blvd.	Torrance	CA	90323	1000.00	900.00
20031023	3456	186th St.	Glendale	CA	90451	1500.00	1700.00
20049873	100	Western Av.	Las Vegas	NV	97345	400.00	400.00
20031134	10	Heather Rd.	S. Monica	CA	92345	6020.00	5312.00
Integer	*Integer*	*Char(25)*	*Char(20)*	*Char(2)*	*Integer*	*Decimal(10,2)*	*Decimal(10,2)*

Note:
Column names that span 2 lines are connected with an underscore.

Ex: Account_Number

AP.Customer (7,000 Rows)

Customer Number	Last Name	First Name	Social Security
PK			
UPI			
13021	Smith	George	456788765
18765	Jones	Barbara	987453498
11023	Wilson	John	495028367
1123	Omaguchi	Sandra	234904587
Integer	*Char(30)*	*Char(20)*	*Integer*

AP.Trans (15,000 Rows)

Trans Number	Trans Date	Account Number	Trans ID	Trans Amount
PK				
		NUPI		
4653	2002-02-11	20024020	2009	-50.00
3241	2002-02-08	20034567	DEP	160.00
1298	2002-02-08	20024005	2987	-70.00
11026	2002-02-13	20024020	DEP	20.00
Integer	*Date*	*Integer*	*Char(4)*	*Decimal(10,2)*

Notes

DS Tables

Orders (31,200 Rows)

orderid	custid	orderstatus	totalprice	orderdate	orderpriority	clerk	location	shippriority	ordercomment
PK	FK,NN								
UPI									
100001	1001	C	1,005.00	2000-01-02	10	Jack ..	5	20	In Stock
103501	1451	C	1,005.00	2002-12-01	10	Dee ...	3	20	In Stock
101400	1080	C	1,150.00	2001-02-28	10	Fred ...	10	20	In Stock
Integer	*Integer*	*CHAR(1)*	*Decimal (10,2)*	*Date*	*SmallInt*	*Char(16)*	*SmallInt*	*Smallint*	*Varchar(79)*

Orders_2012 (12000 Rows) - Same Layout

Notes

PD Tables

Employee (1000 Rows)

Employee Number	Dept Number	Emp_Mgr Number	Job Code	Last Name	First Name	Salary Amount
PK	FK	FK	FK			
UPI						
100001	1001	?	3000	DeBosse	Ibee	200000.00
100797	1048	100791	3017	Myers	Ruth	41000.00
100002	1001	100001	3001	Smith	Steve	110000.00
Integer	*Integer*	*Integer*	*Integer*	*Char(20)*	*Varchar(20)*	*Decimal (10,2)*

Job (66 Rows)

Job Code	Job Description
PK	
UPI	
3000	President
3001	Senior Mgmt
3017	Analyst L2
Integer	*Char(20)*

Department (60 Rows)

Dept Number	Dept Name	Dept_Mgr Number	Budget Amount
PK		FK	
UPI			
1048	Design SW	100791	1000000.00
1050	Design Services	100811	1000000.00
1028	Engineering SW	100441	3000000.00
Integer	*Char(20)*	*Integer*	*Decimal (10,2)*

Emp_Phone (2000 Rows)

Employee Number	Area Code	Phone Number	Extension
PK			
FK			
NUPI			
100001	937	5100001	1001
100001	937	4100001	1001
100389	858	4852815	815
Integer	*SmallInt*	*Integer*	*Integer*

Note:
Column names that span 2 lines are connected with an underscore.
Ex: Employee_Number

vi or vim Notes

If you are not comfortable or familiar with vi or vim, you can create your script in a Windows Notepad, copy it, and paste it into a vi file.

1. Enter your commands (job/script) in a Notepad file.

2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function

Switch to your terminal window where Linux is running and …

3. **vi labx_1.btq** (or whatever filename you wish)

   enter an **i** by itself (do not press <Enter>)

   Use the mouse to choose the **Edit ➔ Paste** function

   Press the ESC key (multiple times does not hurt)

   Enter **:wq** (this saves the file and exits vi)

Another technique that can be used to create Linux scripts without using vi is to do the following:

1. Enter your commands (job/script) in a Notepad file.

2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function

Switch to your terminal window where Linux is running and …

3. **cat > labx_1.btq** (or whatever filename you wish)

   Use the mouse to choose the **Edit ➔ Paste** function

   To exit the cat command, press either the DELETE key or CNTL C.

vi or vim Commands

vim filename

Movement and other

h	move cursor to left
j	move cursor down a line
k	move cursor up a line
l	move cursor to right
space	move cursor to right
w	move a word to right
b	move a word to left
0	go to start of a line (zero)
$	go to end of a line
G	go to end of file
nG	go to line n
^f	scroll forward 1 screen
^b	scroll back 1 screen
^l	refresh the screen
/str	search forward for *str*
.	repeat last command

Modification

r	replace 1 character
x	delete char under cursor
X	delete char before cursor
5x	delete 5 characters
dw	delete a word
dd	delete the line
d$	delete the rest of the line
dG	delete rest of the file
cw	change a word
C	change rest of the line
J	join 2 lines together
yy	yank or copy a line
3yy	yank or copy 3 lines
p	put yanked line(s) after line
P	put yanked line(s) before line
u	undo last command

Input mode

i	insert before cursor
I	insert at beginning of line
a	insert after cursor
A	insert at end of line
o	open a new line below
O	open a new line above
R	Replace mode and insert after <cr>

ESC exits input & colon modes

Colon mode

:w	write or save the file
:w *fname*	write new file name
:wq	write and quit
:q!	quit and don't save
:e *fname*	edit another file
:r *fname*	read in a file
:!*cmd*	run UNIX command
:set smd	set showmode on

Notes

Module B

TERADATA
Raising Intelligence

Appendix B: Acronyms

This Appendix contains a listing of various Teradata acronyms.

Notes

Numbers

2PC	Two-phase Commit

-A-

ABF	Adaptive Bandwidth Feature
ABI	Application Binary Interface
ABIOS	Advanced BIOS
ABM	Asynchronous Balance Mode
ABRD	Automatic Baud Rate Detection
AC	Application Component, Alternating Current
ACE	Array Configuration Editor
ACF	Advanced Communications Function
ACF/NCP	Advanced Communication Function for the Network Control Program
ACF/SSP	ACF/Software Support Program
ACF/VTAM	Advanced Communication Function for the Virtual Telecommunications Access Method
ACID	Atomicity, Consistency, Isolation, Durability
ADAM	Application Data Access Manager
ADCOM	Advanced Distributed Communications System
ADCS	Advanced Data Communication System
ASCII	American Standard Code for Information Interchange
ADE	Application Development Environment
ADW	Active Data Warehouse
AFMS	Advanced Function Management System
AFMS/NDP	Advanced Function Management System/Normalized Data Protocol
AFMS/SEF	Advanced Function Management System/System Exchange Format
AI	Artificial Intelligence
AJI	Aggregate Join Index
ALC	Algorithmic Compression
AMP	Access Module Processor
ANSI	American National Standards Institute
ANSWER	Alpha-Numeric Single Wire Electronic Recorder
AOD	Application Output Definition
AOE	Application Operating Environment
AOS	Automated Order System
AP	Application Processor
API	Application Programming Interface
APPC	Advanced Program to Program Communication
AS	Administration Station
ASF	Archive Storage Facility
ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Module (Networking) Automated Teller Machine
AUI	Attachment Unit Interface
AWS	Administration WorkStation

-B-

BAM	BYNET Administrative Menus
BASIC	Beginner's All-purpose Symbolic Instruction Code
BCD	Binary-Coded Decimal
BDL	BYNET Data Link
BIC	BYNET Interface Card (Adapter)
BIM	Business Information Model; Business Impact Model
BIOS	Basic Input/Output System
BIST	Built-In Self Test
BIU	Battery Interface Unit
BLC	Block Level Compression
BLIP	BYNET Low Level Internet Protocol
BLLI	BYNET Low Latency Interface
BLM	BYNET Link Manager
BLOB	Binary Large Object
BMCA	BYNET Micro Channel Architecture adapter
BNC	Bayonet-Neill-Concelman (Ethernet connector)
BOM	Byte Order Mark (associated with Unicode)
BOOTP	BOOTstrap Protocol
BPCI	BYNET Peripheral Component Interconnect
bps	bits per second
BTAM	Basic Telecommunications Access Method
BTEQ	Basic Teradata **Q**uery facility
BYA4G	BYNET A Switch V2 – 4 ports – Gigahertz speed (60 MB/sec.)
BYA4M	BYNET A Switch V2 – 4 ports – Gigahertz speed (60 MB/sec.)
BYA4MS	BYNET A Switch V2 – 4 ports – Gigahertz speed (60 MB/sec.)
BYA4P	BYNET A Switch V1 – 4 ports – 10 MB/sec.
BYA8QX	BYNET A Switch V3 – 8 ports within BYA64GX or BYC64G
BYA16	BYNET A Switch V1 – 16 ports – 10 MB/sec.
BYA16G	BYNET A Switch V2 – 16 ports – Gigahertz speed (60 MB/sec.)
BYA64GX	BYNET A Switch V2 – 64 ports – Expandable - Gigahertz speed
BYAS	Component name for BYA16XS boards
BYB32	BYNET B Switch – 10 MB/sec.
BYB64G	BYNET B Switch V2
BYC64G	BYNET C Switch V3
BYCLK	BYNET Clock (BYNET V3)
BYNET	**B**an**y**an **Net**work - High speed interconnect
BYOX	BYNET Optical Externsion (BYNET V3)

-C-

CA	Cable Adapter
CAD/CAM	Computer-Aided Design/Computer-Aided Manufacturing
CAE	Computer Aided Engineering
CAI	Century Analysis, Inc., Computer-Aided Instruction Computer-Assisted Instruction
CASE	Computer Aided Software Engineering
CAT	Configuration and Test
CC	Cluster Controller, Common Cable, Common Carrier
CCB	Change Control Board, Character Control Block, Command Control Block
CCC	Common Carrier Communication
CCDLC	Common Carrier Data Link Control
CCITT	Consultative Committee on International Telephone and Telegraph
CCL	Customer Care Link (older software to alert remote support center)
CD	Compact Disk
CD-ROM	Compact Disk - Read Only Memory
CDS	Customer Data Storage
CE	Customer Engineer
CFM	Configuration File Management
CI	Cylinder Index
CICS	Customer Information Control System
CIM	Common Information Model (used as model for WBEM)
CJK	Chinese, Japanese, and Korean
CLAN	Cabinet Local Area Network
CLC	Cabinet Level Controller (5100 component)
CLCX	Cabinet Level Controller eXtended (5100 component)
CLI	Call Level Interface
CLOB	Character Large Object
CM	Communications Manager
CMB	Chassis Management Board
CMIC	Cabinet Management Interface Controller (for 5100)
CMIC	Chassis Management Interface Controller (for later systems)
CMOS	Complementary Metal Oxide Semiconductor
COBOL	Common Business-Oriented Language
COP	Communications Processor
COS	Corporate Office Server
CP	Column Partitioning (or Partitioned)
CPPI	Character PPI
CPU	Central Processing Unit
CR/LF	Carriage Return/Line Feed
CRC	Cyclic Redundancy Check
CRU	Customer Replaceable Unit
CSF	Customer Support Facility
CSI	Complex Service Interface
CTG	Channel Tailgate
CTS	Clear to Send

-D-

DA	Disk Array
DAC	Disk Array Controller
DAMC	Disk Array Module Cabinet
DARDAC	Dual Active Redundant Disk Array Controller
DASD	Direct-Access Storage Device
DAT	Digital Audio Tape, Dynamic Address Translation
DBA	Database Administrator
DBC	Database Computer (Teradata)
DBLT	Disk Bay Locator Table
DBMS	Database Management System
DBQL	Database Query Log
DBS	Database Subsystem
DC	Data Communications, Direct Current
DCE	Distributed Computing Environment
DCL	Data Control Language
DCRAM	Disk Cache Random Access Memory
DD	Derived Data
DDE	Direct Data Exchange
DDI	Device Driver Interface
DDL	Data Definition Language
DES	Data Encryption Standard
DEUI	Dual Ethernet UPS Interface (5400 UPS feature)
DIF	Document Interchange Facility, Data Interchange Format
DKI	Driver Kernel Interface
DIM	Diagnostic Interconnect Module (3600 component
DIMM	Dual In-line Memory Module
DIP	Database Initialization Procedure or Program
DISCO	Disconnect memory
DLPI	Data Link Provider Interface
DMA	Direct Memory Access
DMCA	Dual Micro Channel Architecture
DML	Data Manipulation Language
DMX	EMC^2 Direct Matrix Disk Array
DNS	Domain Name Server
DOD	Department Of Defense
DOS	Disk Operating System
DP	Diagnostic Processor
DR	Deficiency or Defect Report
DRAM	Dynamic Random Access Memory
DS	Desk Side
DSC	Data Stream Compatible
DSS	Decision Support System
DSU	Disk Storage Unit
DSW	Destination Selection Word
DTP	Distributed Transaction Processing
DTR	Data Terminal Ready

-E-

E2I	External-to-Internal
EA	EaseAdvantage
EAFW	EaseAdvantage Framework
EBCA	EISA Bus Channel Adapter
EBCDIC	Extended Binary Coded Decimal Interchange Code
ECC	Error Check and Correction
ECL	Emitter Coupler Logic
EDAC	Error Detection and Correction
EDI	Electronic Data Interchange
EEPROM	Electronically Erasable Programmable Read Only Memory
EGA	Enhanced Graphics Adapter (IBM)
EISA	Extended Industry Standard Architecture
EOE	Enhanced Operating Environment
EPROM	Erasable Programmable Read-Only Memory
ESCON	Enterprise System Connection
ESM	Environmental Services Monitor
EUC	Extended UNIX Code

-F-

4GL	Fourth Generation Language
FAX	Facsimile
FC	Fibre Channel
FDDI	Fiber Distributed Data Interface
FE	Field Engineer
FEU	Front End Unit
FICON	Fibre Connection (IBM)
FIFO	First In, First Out
FIPS	Federal Information Processing Standards
FK	Foreign Key
FMLI	Forms and Menu Language Interpreter
FORTRAN	FORmula TRANslation
FSB	Front Side Bus
FSG	File Segment Cache
FRU	Field Replaceable Unit
FTAM	File Transfer and Access Method (OSI)
FTP	File Transfer Protocol
FUD	Fear, Uncertainty, Doubt
FW	Firmware
FYI	For Your Information

-G-

GB	Gigabyte (one billion bytes)
GCA	General Customer Availability
GOSIP	Government Open Systems Interconnection Profile
GSC	Global Support Center
GUI	Graphical User Interface

-H-

HASP	Houston Automatic Spooling Program
HATP	High Availability Transaction Processing
HBN	Hash Bucket Number
HCA	Host Channel Adapter
HDA	Home Disk Address (used with Virtual Storage Services)
HDD	Hard Disk Drives
HDLC	High-level Data Link Control
HI	Hash Index
HOLAP	Hybrid On-Line Analytical Processing
HSN	Hot Standby Node
HW	Hardware
Hz	Hertz - cycles per second

-I-

iABI	Intel Application Binary Interface
I2E	Internal-to-External
I^2C or I2C	Inter-Integrated Circuit
IC	Integrated Circuit
ICI	Initial Certified Installation
ICMB	Intelligent Chassis Management Bus
ICSI	Integrated Complex Service Interface
IDE	Integrated Development Environment
IE	Information Engineering
IEEE	Institute of Electrical and Electronics Engineers
I/L	Memory Inter-Leave controller
IMS	Information Management System, Inventory Management System
IO	Input/Output
IP	Internet Protocol
IPMB	Intelligent Platform Management Bus
IPS	Integrated Peripheral Subsystem
IRQ	Interrupt Request
ISA	Industry-Standard Architecture
ISAM	Indexed Sequential Access Method
ISD	Interactive System Definition
ISDN	Integrated Services Digital Network
ISO	International Standards Organization
ISP	Internet Support Package
ISPF	Interactive System Productivity Facility
ISV	Independent Software Vendor
IT	Information Technology

-J-

JBOD	Just a Bunch of Disks
JCL	Job Control Language
JDBC™	Java Database Connectivity
JDO	Joint Development Operation (Teradata and NCR)
JES	Job Entry Subsystem
JFS	Journaling File System
JI	Join Index
JIS	Japanese Industrial Standards
JIT	Just-In-Time

-K-

Kb	Kilobit
KB	Kilobyte (1,024 bytes)
Kbaud	Kilo Baud
Kbps	Kilobits per second
KBPS	Kilobytes per second
KHz	Kilohertz
KSR	Keyboard Send/Receive

-L-

LAN	Local Area Network
LARC	Limited Address Range Cache
LCMP	Loosely Coupled Multiprocessing
LCD	Liquid Crystal Display
LDAP	Lightweight Directory Access Protocol
LDM	Limited Distance Modem, Local Domain Manager
LED	Light Emitting Diode
LFM	Log File Management
LM	Local Media Module
LOB	Large Object (either a BLOB or CLOB)
LPB	Local Peripheral Board
LPI	Language Processors Inc.
lpm	lines per minute (or line-per-minute)
LSU	Logical Storage Unit
LT/ST	Large Table/Small Table (join)
LUN	Logical Unit (disk array logical unit)

-M-

MA	Modular Array
MAC	Media Access Control, Medium Access Control, Message Authentication Code
MAP	Manufacturing Automation Protocol, Maintenance Analysis Procedures, Master Application Processor
Mb	Megabit
MB	Megabyte
Mbps	Megabits per second
MLAN	Management LAN
MCA	Micro Channel Architecture
MCCA	Micro Channel to Channel Adapter
MCIA	Micro Channel Interface Architecture
MHz	MegaHertz - million cycles per second
MI	Master Index
MIPS	Million Instructions per Second
MIS	Management Information System
MLPPI	Multi-Level Partitioned Primary Index
MO	Method of Operation
MOLAP	Multi-dimensional On-Line Analytical Processing
MPEG	Moving Picture Experts Group
MPL	Message Passing Layer
MPP	Massively Parallel Processing
MSU	Memory Storage Unit
MTBDL	Mean Time Between Data Loss
MTBF	Mean Time Between Failures
MTBR	Mean Time Between Repairs
MTDP	Micro Teradata Director Program
MVC	Multi-Value Compression
MVS	Multiple Virtual Storage (IBM mainframe OS)

-N-

NA	Network Agent
ND	No Duplicates
NetBIOS	Network Basic Input/Output System
NFS	Network File System
NI	Network Interface
NIST	National Institute of Standards and Technology
NLQ	Near Letter Quality
NN	No Nulls
NOBOM	No Byte Order Mark (associated with Unicode)
NoPI	No Primary Index Table (Teradata 13.0 feature)
NPPI	Non-Partitioned Primary Index
NRZ	Non-Return to Zero
Ns	Nanosecond
NSC	NCR Storage Cabinet
NTOS	NCR Teradata Operating System
NUPI	Non-Unique Primary Index
NUSI	Non-Unique Secondary Index
NVRAM	Non-Volatile Random Access Memory (read and write memory)

-O-

OCC	Open Cooperative Computing
OCCA	Open Cooperative Computing Architecture
OCR	Optical Character Reader, Optical Character Recognition
ODBC™	Open Database Connectivity
ODS	Operational Data Store
OE	Operating Environment
OEM	Original Equipment Manufacturer
OLAP	On-Line Analytical Processing
OLCP	On-Line Complex Processing
OLE	Object Linking and Embedding
OLTP	On-Line Transaction Processing
OMC	ORION Memory Controller chip
OMC-DC	ORION Memory Controller - DRAM Controller
OMC-DP	ORION Memory Controller - Data Path Controller
ONE	Open Network Environment
ONS	Open Networking System
OOP	Object Oriented Programming
OPB	ORION PCI Bridge chip
OPS	Oracle Parallel Server
OS	Operating System
OSA	Open Systems Architecture
OSF	Open Software Foundation
OSI	Open Systems Interconnect
OSI/DTP	Open Systems Interconnect/Distributed Transaction Processing
OSS	Open System Standards
OTOS	Open Teradata Operating System

-P-

PBX	Private Branch Exchange
PC	Personal Computer, Printed Circuit
PCEB	PCI EISA Bridge chip
PCI	Peripheral Component Interconnect
PCMCIA	Personal Computer Memory Card International Association
PC/RA	Probable Cause/Recommended Action
PCS	Personal Computer Support
PDB	Parallel Data Bus
PDCR	Performance Data and Collection Repository
PDE	Parallel Database Extensions
PDN	Public or Private Data Network
PE	Parsing Engine
PEP	Parsing Engine Processor
PIM	Plug-In Module (used with V3 BYNET 32 switch)
PK	Primary Key (or Parent Key)
PL/1	Programming Language I
PLAN	Private LAN
PM/PC	Performance Management/Production Control
PN	Primary Network
POPS	Parallel Object Processing System
POS	Point of Sale (noun), Point-of-Sale (terminal) (adj.)
POSIX	Portable Operating System Interface for Computing Environments
POST	Power-On System Test
PPI	Partitioned Primary Index
PQS	PCI Quad SCSI
PSA	Priority Scheduler Administrator
PSC	Power Shelf Controller
PSI	Power Supply Interface
PTB	Pass Through Board
PUT	Parallel Upgrade Tool

-Q-

QA	Quality Assurance
QCD	Query Capture Database
QFC	Quad Fibre Channel
QIC	Quarter Inch Cartridge

-R-

RAID	Redundant Array of Independent (formerly, Inexpensive) Disks
RAM	Random Access Memory
RASUI	Reliability, Availability, Serviceability, Usability, Installability
RC	Remote Client
RCMB	Rack (or Remote) Chassis Management Board
RDAC	Redundant Disk Array Controller (software)
RDBMS	Relational Database Management System
RFC	Request for Change
RFP	Request for Proposal
RFS	Remote File System
RGB	Red, Green, Blue
RISC	Reduced Instruction Set Computing
RJE	Remote Job Entry
RLM	Revision Level Manager
RM	Resource Manager (TOP END) or Rack Mount
ROLAP	Relational On-Line Analytical Processing
ROM	Read Only Memory
ROSE	Remote Operation Service Element
RPC	Remote Procedure Call
RPG	Report Program Generator
rpm	revolutions (or rotations) per minute
RSC	Remote System Call
RSS	Resource Sampling Subsystem (used for performance capture)
RTS	Ready to Send, Request to Send
RTS-CTS	Request to Send - Clear to Send

-S-

SAN	Storage Area Network
SAR	System Activity Report (sar – UNIX utility)
SCO	Santa Cruz Operation UNIX
SCSI	Standard (or Small) Computer System Interface
SDL	Screen Definition Language
SDW	Scalable Data Warehouse
SE	Systems Engineer
SEEPROM	Serial Electronically Erasable Programmable Read-Only Memory
SI	Self-Instruction
SIMM	Single In-line Memory Module
SKU	Stock-keeping Unit
SLAN	Service or System LAN (Local Area Network)
SLES	SUSE Linux Enterprise Server
SLC	Second Level Cache
SM3G	Server Management – 3rd Generation
SMB	Server Management Board
SMC	Server Management Chassis
SMP	Symmetrical Multi-Processing
SN	Secondary Network
SNA	Systems Network Architecture
SNAG	SNA Gateway
SNO	SuperNode
SNP	Super Node Processor
SOV	Single Operational View
SP	Stored Procedure
SPI	System Performance Investigator
SQL	Structured Query Language
SQLCA	Structured Query Language Communication Area
SQLDA	Structured Query Language Data Area
SRAM	Static Random Access Memory
SS	Support Sentinel
SSA	Symbolic Service Address, System-to-System Adapter
SSD	Solid State Disk or Drive
SSI	Single System Image, Standard Serial Interface
SSL	Secure Socket Layer (Networking)
SSP	Software Support Program, System Support Program (IBM)
STMP	Simple Mail Transfer Protocol
SUS	Startup Subsystem
SUSE	Software und System Entwicklung (German name which means Software and Systems Development)
SVID	System V Interface Definition
SVR4	System V Release 4
SW	Software
SWS	Service Workstation (used with 56xx and later systems)
SYSGEN	System Generation

-T-

3GL	Third Generation Language
TAM	Teradata Access Method (used with Replication Services)
TAP	Teradata Application Program
TARA	Tiered Archive Restore Architecture
TASM	Teradata Active System Management
TAXI	Transparent Asynchronous Xceiver Interface
TB	Terabyte (a trillion bytes)
TBA	To Be Announced
TBD	To Be Determined
TCAM	Telecommunications Access Method
TCP/IP	Transmission Control Protocol/Internet Protocol
TDAS	Teradata Dual Active Solution
TDQM	Teradata Dynamic Query Manager
TFTP	Trivial File Transfer Protocol
TLI	Transport Layer Interface
TLS	Transport Layer Security (Networking)
TM	Transaction Manager
TMSM	Teradata Multi-System Manager
TOD	Time of Day (noun) or Time-of-Day (adj.)
TOS	Teradata Operating System
TQD	Teradata Query Director
TP	Transaction Processing
TPA	Trusted Parallel Application or Architecture
TPD	Teradata Parallel Database
tps	transactions per second
TPT	Teradata Parallel Transporter
TSET	Teradata System Emulation Tool
TSO	Time-Sharing Option
TTL	Transistor-Transistor Logic
TTU	Teradata Tools and Utilities
TTY	Teletype (serial communication protocol)
TVI	Teradata Vital Infrastructure (software to alert remote support)
TVS	Teradata Virtual Storage
TVSS	Teradata Virtual Storage Services
TWA	Teradata Workload Analyzer

-U-

U	Unit of Measurement – 1.75” or 4.445 cm
UDF	User Defined Function
UDM	User Defined Method
UDP	Unreliable Data Protocol
UDT	User Defined Type
UFS	UNIX File System
UMB	Universal Management Board
UPI	Unique Primary Index
UPS	Uninterruptible Power Source
UPS-IS	Uninterruptible Power Source – Input Selector
USI	Unique Secondary Index

-V-

VAR	Value Added Reseller
VDC	Volts Direct Current
VDT	Video Display Terminal
VGA	Video Graphics Array
VLC	Value List Compression
VLF	Very Low Frequency
VLSI	Very Large Scale Integration
VM	Virtual Machine
VONUSI	Valued-ordered NUSI
VPIX	"DOS under UNIX"
VPROC	Virtual Processor
VSAM	Virtual System Access Method
VSS	Virtual Storage Services
VTAM	Virtual Telecommunications Access Method

-W-

WAL	Write Ahead Logic
WAN	Wide Area Network
WAWS	Windows AWS (Administration Workstation)
WBEM	Web-Based Enterprise Management
WCI	WAL Cylinder Index
WinDDI	Windows Data Dictionary Interface (application)
WIN-TCP	Wollongong Integrated Networking/Transmission Control Protocol
WMI	WAL Master Index
WORM	Write Once - Read Many Optical Disk
WTO	Write to Operator
WYSIWYG	What You See Is What You Get

-X-

XA	Extended Architecture
XMIT	Transmit
XML	Extensible Markup Language

-Y-

YMCA	Ynet Micro Channel Architecture adapter
YTD	Year-to-Date

Notes

Module C

Appendix C: Answers to Review Questions

This Appendix contains answers to the review questions for the course modules.

Notes

Module 1: Review Question Answers

1. Which feature allows the Teradata Database to process enormous volumes of data quickly? ____

   a. High availability software and hardware components
   b. High performance servers from Intel
   c. Proven Scalability
   d. Parallelism

2. The Teradata Database is primarily a ____ .

   a. Client
   b. Server

3. Which choice represents a quadrillion bytes or a Petabyte (PB) of data? ____

   a. 10^9
   b. 10^{12}
   c. 10^{15}
   d. 10^{18}

4. In a relational table, the set of columns that uniquely identifies a row is the ***Primary Key.***

Module 2: Review Question Answers

1. **What language is used to access a Teradata table?**

   SQL

2. **What are five Teradata database objects?**

   Tables, views, macros, triggers, and stored procedures

3. **What are four major components of the Teradata architecture?**

   PEs, AMPs, Vdisks, and Message Passing Layer

4. **What are views?**

   Filter (or subset) of rows and columns or one or more tables

5. **What are macros?**

   Predefined, stored set of SQL statements

Module 3: Review Question Answers

1. What are the two software elements that accompany an application on all client side environments?

   CLI/ODBC/JDBC/.NET and TDP/MTDP

2. What is the purpose of the PE?

   Parse, optimize, and dispatch queries

3. What is the purpose of the AMP?

   Manage and retrieve data from disk storage

4. How many sessions can a PE support?

   120

Match Quiz

i	1. CLI	a. Does Aggregating and Locking
f	2. MTDP	b. Validates SQL syntax
e	3. MOSI	c. Connects AMPs and PEs
b	4. Parser	d. Balances sessions across PEs
a	5. AMP	e. Provides Client side OS independence
c	6. Message Passing Layer	f. Library of Session Management Routines
d	7. TDP	g. PE S/W turns SQL into AMP steps
g	8. Optimizer	h. PE S/W sends plan steps to AMP
h	9. Dispatcher	i. Library of Teradata Service Routines
j	10. Parallelism	j. Foundation of Teradata architecture

Module 4: Review Question Answers

True or False

1. False — A database will always have tables.
2. True — A user will always have a password.
3. False — A user creating a subordinate user must give up some of his/her Perm Space.
4. False — Creating tables requires the definition of at least 1 column and a Primary Index.
5. True — The sum of all user and database Perm Space will equal the total space on the system.
6. False — The sum of all user and database Spool Space will equal the total space on the system.
7. True — Before a user can read a table, a database or table SELECT privilege must exist in the DD/D for that user.
8. False — Deleting a macro from a database reclaims Perm Space for the database.

9. Which statement is TRUE about PERM space? ____

   a. PERM space cannot be dynamically modified.
   b. The per/AMP limit of PERM space can be exceeded.
   c. Tables, index subtables, and stored procedures use PERM space.
   d. Maximum PERM space can be defined at the database or table level.

10. Which statement is TRUE about SPOOL space? ____

    a. SPOOL space cannot be dynamically modified.
    b. Maximum SPOOL space can be defined at the database or user level.
    c. The SPOOL limit is dependent on the database limit where the table is located.
    d. Maximum SPOOL space can be defined at a value greater than the immediate parent's value.

Module 5: Review Question Answers

Answer the following either as True or False as these apply to Primary Indexes:

True or False 1. UPI and NUPI equality value accesses are always a one-AMP operation.

True or False 2. UPI and NUPI indexes allow NULL in a primary index column.

True or False 3. UPI, NUPI, and NOPI tables allow duplicate rows in the table.

True or False 4. Only UPI can be used as a Primary Key implementation.

Fill in the Blanks

5. The output of the hashing algorithm is called the *row hash*.
6. To determine the target AMP, the Message Passing Layer must lookup an entry in the Hash Map based on the *hash bucket number*.
7. A Row ID consists of a row hash plus a *uniqueness* value.
8. A uniqueness value is required to produce a unique Row ID because of *hash synonyms* and *NUPI duplicates*.
9. Once the target AMP has been determined for a PI search, the *Master Index* for that AMP is accessed to determine the cylinder that may hold the row.
10. The Cylinder Index points us to the address and length of the data *block*.

Module 6: Review Question Answers

Fill each box with either Yes, No, or the appropriate number.

	USI Access	NUSI Access	FTS
# AMPs	*2*	*All*	*All*
# rows	*0 - 1*	*0 - n*	*0 - all*
Parallel Operation	*No*	*Yes*	*Yes*
Uses Hash Maps	*Yes*	*No*	*No*
Uses Separate Sub-table	*Yes*	*Yes*	*No*
Reads all data blocks of table	*No*	*No*	*Yes*

Module 7: Review Question Answers

Complete the following.

1. **Each AMP has its own memory and manages its own disk space and executes independently of other AMPs. This is referred to as a _shared nothing_ architecture.**

2. **The software component that allows the Teradata Database to execute in different operating system environments is the _PDE_.**

3. **A physical message passing interconnect is called the _BYNET_.**

4. **A clique provides protection from a _node_ failure.**

5. **If a node fails, all vprocs will migrate to the remaining nodes in the clique. This feature is referred to as _vproc migration_.**

6. **The _AWS_ or _SWS_ provides a single point of operational control for Teradata MPP systems.**

7. **A _TPA_ node is part of a system configuration, is connected to the BYNET, and executes the Teradata Database software.**

8. **A _NOTPA_ node is part of a system configuration, connects to the BYNET, and is used to execute application software other than Teradata Database software.**

9. **A _HSN_ node is part of a system configuration, connects to the BYNET, and is used as a spare node in the event of a node failure.**

Module 8: Review Question Answers

Match the item to a lettered description.

f	1. Database locks	a. Provides for TXN rollback in case of failure
c	2. Table locks	b. Teradata Backup and Recovery applications
h	3. Row Hash locks	c. Protects all rows of a table
i	4. FALLBACK	d. Logs changed rows for down AMP
k	5. Cluster	e. Provides for recovery to a point in time
d	6. Recovery journal	f. Applies to all tables and views within
a	7. Transient journal	g. Multi-platform archive utility
g	8. ARC	h. Lowest level of protection granularity
b	9. NetBackup/Tivoli	i. Protects tables from AMP failure
e	10. Permanent journal	j. Protects database from a physical drive failure
j	11. Disk Array	k. Group of AMPs used by Fallback

Module 9: Review Question Answers

1. What is a major difference between a 6650 system as compared to a 6690 system?

   6690 Systems have both spinning disks and solid state disks in the same cabinet.

2. What is a major difference between a 2650 node and a 2690 node?

   2690 nodes have hardware compression boards.

3. What does the acronym represent and briefly define the purpose of the following subsystems?

   BYNET *Banyan Network; high speed interconnect for data/message passing between nodes*

   SWS *Service Workstation; is dedicated to operations, system servicing, and maintenance*

4. Specify the names of the two TPA nodes in 6690 cabinet #2.

   SMP002-8 *SMP002-9*

Play the numbers games – match the number to a definition.

d	1.	3	a.	Typical # of AMPs per node in a 6650 3+1 clique
b	2.	8	b.	Maximum number of nodes that can be in a 2690 cabinet
f	3.	24	c.	Maximum number of drives in one NetApp 6844 disk array
a	4.	42	d.	Number of nodes in a 2650 clique
c	5.	128	e.	Large disk drive size (GB) for a 2690 disk array
e	6.	900	f.	Typical # of AMPs in a 2690 node

Module 10: Review Question Answers

1. Which two are placed into FSG cache?
   a. Hash maps
   b. Master Index
   c. Cylinder Indexes
   d. Permanent data blocks

2. What is the WAL Depot used for?
   a. UNDO Rows
   b. New data blocks
   c. Master Index updates
   d. Write-in-place data blocks

3. Which two are placed into the WAL Log?
   a. REDO Rows
   b. UNDO Rows
   c. New data blocks
   d. Master Index updates
   e. Write-in-place data blocks

4. Describe the fundamental relationship between Linux, logical units, and disk array controllers.

   Linux uses logical unit names to communicate with disk array controllers.

5. Describe the fundamental relationship between AMPs, Vdisks, Pdisks, Partitions, and LUNs.

   Each AMP is assigned to a Vdisk which is made up of 1 or more Pdisks. A Pdisk is assigned to a partition in a logical unit or LUN.

Module 11: Review Question Answers

1. List two capabilities of using Teradata Virtual Storage.

   Simplify adding storage space,
   Provides a multi-temperature data warehouse

2. List the two operational modes of Teradata Virtual Storage.

   Teradata Traditional *Intelligent Placement*

3. Which choice is associated with data temperature?

   a. Skewed data
   b. Frequency of access
   c. Solid State Disk Drives
   d. Inner tracks versus outer tracks on a spinning disk

4. Which data level is migrated from hot to cold storage?

   a. Row
   b. Block
   c. Cylinder
   d. Subtable

5. Which two types of data are typically considered to be HOT data?

   a. WAL
   b. DBC tables
   c. Spool data
   d. History data

Module 12: Review Question Answers

1. Which three are benefits to creating a data model in 3NF? ____ ____ ____

   a. Minimize redundancy
   b. To reduce update anomalies
   c. To improve distribution of data
   d. To improve flexibility of access
   e. To reduce number of I/Os to access data

2. Which data model would include the definition of a partitioned primary index? ____

   a. Logical data model
   b. Physical data model
   c. Business information model
   d. Extended logical data model

3. Which two factors should be considered when deciding to denormalize a table? ____ ____

   a. Volatility
   b. Performance
   c. Distribution of data
   d. Connectivity of users

4. Which is a benefit of implementing data types at the domain level? ____

   a. Reduce storage space
   b. Avoid data conversion
   c. Provides consistent display of data
   d. Reduce need for secondary indexes

Module 13: Review Question Answers

1. The Row Hash for a PI value of 824 is the same for the data types of INTEGER and DECIMAL(18,0). True or False. *True*
2. The first 16 or 20 bits of the Row Hash is referred to as the *hash* *bucket* *number*.
3. The Hash Map consists of entries or buckets which identify an *AMP* number for the Row Hash.
4. The Current Configuration *Primary* Hash Map is used to locate the AMP to locate/store a row based on PI value.
5. The *RECONFIG* utility is used to redistribute rows to a new system configuration with more AMPs.
6. When creating a new table, the Unique Value of a Table ID comes from the dictionary table named DBC. *Next*.
7. The Row ID consists of the *Row* *Hash* and the *Uniqueness* *Value*.
8. The *Master* *Index* contains a Cylinder Index Descriptor (CID) for each allocated Cylinder.
9. The *Cylinder* *Index* contains an entry for each data block in the cylinder.
10. The *Row* *Reference* *Array* consists of a set of 2 byte pointers to the data rows in data block.
11. The maximum block size is approximately *128 KB* and the maximum row size is approximately *64 KB*.
12. The Primary Index data value is used as a row qualifier to eliminate hash *synonyms*.

Module 14: Review Question Answers

1. When Teradata INSERTs a new row into a table, it first goes to the _____ to locate the proper cylinder for the new row.
   a. Cylinder Index
   b. Fallback AMP
   c. Free Cylinder List
   d. Master Index

2. When a new block is needed, the File System searches the Free Block List looking for the first Free Block whose size is equal to, or greater than the new block's requirement. It does not have to be an exact match.
   a. True
   b. False

3. Name the condition which occurs when there is no block on the Free Block List with enough sectors to accommodate the additional data during an INSERT or UPDATE.
   a. Mini Cylinder Pack
   b. Cylinder Migrate to a new cylinder
   c. Cylinder Migrate to an adjacent cylinder
   d. Cylinder Full

4. The _____ parameter can be set to control how completely cylinders are filled during loading and PackDisk.
   a. Free Space Percent
   b. DataBlockSize
   c. PermDBSize
   d. PermDBAllocUnit

Module 14: Review Question Answers

5. Number the following steps in sequence from 1 to 6 that the File System software will attempt to perform in order to insert a new row into an existing data block.

   5 Perform a Cylinder Migrate operation to an adjacent cylinder

   1 Simply insert the row into data block if enough contiguous free bytes in the block

   4 Perform a Block split

   6 Perform a Cylinder Migrate operation to a new cylinder

   2 Defragment the block and insert the row

   3 Expand or grow the block to hold the row

6. As part of a cylinder full condition, if the number of free sectors within a cylinder is greater than 25%, what operation will Teradata perform in the background? ***Cylinder Defragmentation***

7. If the number of free cylinders falls below a minimum threshold, what operation will Teradata perform in the background? ***Mini-Cylpack***

Module 15: Review Question Answers

1. Which two data interfaces are available with Teradata SQL Assistant?
   a. CLIv2
   b. JDBC
   c. ODBC
   d. Teradata .Net

2. Separate history database files are needed to maintain queries for different data sources.
   a. True
   b. False

3. Which piece of query information is not available in the History Window?
   a. User name
   b. Query band
   c. Elapsed time
   d. Data source name
   e. Number of rows returned

4. What are two techniques to execute multiple statements as a multi-statement request?

   Use the Execute Parallel icon 

   Use function key 9 – F9

Module 16: Exercise 2 – Choosing PI Candidates

ENTITY 1

100,000,000 Rows	A	B	C	D	E	F
PK/FK	PK,UA					
Distinct Values	100M	95M	300K	250K	40M	1M
Max Rows/Value	1	2	400	350	3	110
Max Rows/NULL	0	0	0	0	1.5M	0
Typical Rows/Value	1	1	325	300	2	90
PI/SI	UPI	NUPI	NUPI	NUPI		NUPI
Collect Statistics (Y/N)						

Module 16: Exercise 2 – Choosing PI Candidates

	ENTITY 2						
10,000,000 Rows	G	H	I	J	K	L	
PK/FK	PK,SA						
Distinct Values	10M	100K	9M	12	50	180K	
Max Rows/Value	1	200	2	1M	240K	60	
Max Rows/NULL	0	0	100K	0	0	0	
Typical Rows/Value	1	100	1	800K	190K	50	
PI/SI	UPI	NUPI				NUPI	
Collect Statistics (Y/N)							

Module 16: Exercise 2 – Choosing PI Candidates

	DEPENDENT						
5,000,000 Rows	A	M	N	O	P	Q	
PK/FK	PK				NN, ND		
	FK	SA					
Distinct Values	2M	50	90K	3M	5M	2M	
Max Rows/Value	4	200K	75	2	1	5	
Max Rows/NULL	0	0	0	390K	0	1M	
Typical Rows/Value	1	60K	50	1	1	1	
PI/SI	UPI				UPI		
	NUPI		NUPI				
Collect Statistics (Y/N)							

Module 16: Exercise 2 – Choosing PI Candidates

ASSOCIATIVE 1

300,000,000 Rows	A	G	R	S	
PK/FK	PK				
	FK	FK,SA			
Distinct Values	100M	10M	15K	800K	
Max Rows/Value	5	50	21K	400	
Max Rows/NULL	0	0	0	0	
Typical Rows/Value	3	30	19K	350	
	UPI				
PI/SI	NUPI	NUPI	NUPI?	NUPI	
Collect Statistics (Y/N)					

Module 16: Exercise 2 – Choosing PI Candidates

ASSOCIATIVE 2

100,000,000 Rows	A	M	G	T	U	
PK/FK	PK					
	FK		FK			
Distinct Values	50M		10M	560K	750	
Max Rows/Value	3		150	180	135K	
Max Rows/NULL	0		0	0	0	
Typical Rows/Value	1		8	170	100K	
	UPI					
PI/SI	NUPI		NUPI	NUPI		
Collect Statistics (Y/N)						

Module 16: Exercise 2 – Choosing PI Candidates

HISTORY

730,000,000 Rows	A	DATE	D	E	F	
PK/FK	PK					
	FK	SA				
Distinct Values	100M	730	N/A	N/A	N/A	
Max Rows/Value	18	1100K	N/A	N/A	N/A	
Max Rows/NULL	0	0	N/A	N/A	N/A	
Typical Rows/Value	3	900K	N/A	N/A	N/A	
PI/SI	UPI					
	NUPI					
Collect Statistics (Y/N)						

Module 16: Review Question Answers

1. Which trade-off must be balanced to make the best choice for a primary index? ____

   a. Access and volatility
   b. Access and block size
   c. Block size and volatility
   d. Access and distribution

2. When volatility is considered as one of the Primary Index choice criteria, what is analyzed? ____

   a. Degree of uniqueness
   b. How often the data values will change
   c. How often the fixed length rows will change
   d. How frequently the column is used for access

3. To optimize the use of disk space, the designer should choose a primary index that _____.

   a. is non-unique
   b. consists of one column
   c. is unique or nearly unique
   d. consists of multiple columns
   e. has fewer distinct values than AMPs

Module 16: Review Question Answers

4. For NoPI tables, what are 2 ways in which the Random Generator is executed?
   a. At the AMP level with FastLoad
   b. At the PE level for ad hoc SQL requests
   c. At the TPump client level for array insert operations
   d. At the AMP level for INSERT-SELECT into an empty NoPI table

5. Assume DBSControl flag #53 (Primary Index Default) is set to N (No Primary Index), which two indexes are created for TableX given the following DDL command?

```
CREATE TABLE TableX
        (col1     INTEGER NOT NULL UNIQUE
        ,col2     CHAR(10) NOT NULL PRIMARY KEY
        ,col3     CHAR(80));
```

   a. col1 will be a UPI
   b. col1 will be a USI
   c. col2 will be a UPI
   d. col2 will be a USI

6. Which two options are permitted for NoPI tables?
   a. Fallback
   b. MultiLoad
   c. Hash Index
   d. BLOBs and CLOBs

Module 17: Review Question Answers

1. In a PPI table, every row is uniquely identified by its *Partition # + Row Hash + Uniqueness Value.*
2. The Row Key consists of the *Partition # + Row Hash .*
3. In an NPPI table, the partition number defaults to *0 .*
4. True or (False.) For a PPI table, the partition number and the Row Hash are both used by the Message Passing Layer to determine which AMP(s) should receive the request.
5. Which two options apply to the RANGE_N expression in a partitioning expression? *b d*
   a. Ranges can be listed in descending order
   (b.) Allows use of NO RANGE OR UNKNOWN option
   c. Partitioning column must be part of the Primary Index
   (d.) Has a maximum of 65,535 partitions with Teradata Release 13.10
6. With a populated table, select 2 actions that are allowed with the ALTER TABLE command. *b d*
   a. Drop all of the ranges
   (b.) Add or drop ranges from the partition “ends”
   c. Change the columns that comprise the primary index
   (d.) Add or drop special partitions (NO RANGE, UNKNOWN)
7. Which 2 choices are advantages of partitioning a table? *a d*
   (a.) Fast delete of rows in partitions
   b. Fewer AMPs are involved when accessing data
   c. Faster access (than an NPPI table) if the table has a UPI
   (d.) Range queries can be executed without a secondary index

Module 17: Review Question Answers

Given this CREATE TABLE statement, answer the following questions.

```
CREATE TABLE     Orders
   (Order_id          INTEGER NOT NULL,
    Cust_id           INTEGER NOT NULL,
    Order_status      CHAR(1),
    Total_price       DECIMAL(9,2) NOT NULL,
    Order_date        DATE FORMAT 'YYYY-MM-DD' NOT NULL,
    Order_priority    SMALLINT,
    Clerk             CHAR(16),
    Ship_priority     SMALLINT,
    Order_Comment     VARCHAR(80) )
PRIMARY INDEX    (Order_id)
PARTITION BY RANGE_N (Order_date
      BETWEEN DATE '2003-01-01' AND DATE '2012-12-31'
      EACH INTERVAL '1' MONTH)
UNIQUE INDEX     (Order_id);
```

8. What is the name of partitioning column? *Order_date*
9. What is the time period for each partition? *1 Month*
10. Why is there a Unique Secondary Index specified instead of defining Order_id as a UPI? *c*
    a. This is a coding style choice.
    b. You cannot have a UPI when using a partitioned primary index.
    (c.) You cannot have a UPI if the partitioning column is not part of the primary index.
    d. This is a mistake. You cannot have a secondary and a primary index on the same column(s).

Module 18: Review Question Answers

1. Which two choices apply to Column Partitioning?
   a. SET table
   b. NoPI table
   c. Table with multi-level partitioning
   d. Table with existing row partitioning

2. What are two benefits of Column Partitioning?
   a. Reduced I/O
   b. Reduced CPU
   c. Reduced disk space usage
   d. Reduced tactical query response times

3. True or False? Deleting a row in a column partitioned table will reclaim table space.

4. True or False? In a multi-level columnar table, only one level may have column partitioning.

5. True or False? The preferred way to load a columnar table is using INSERT/SELECT.

Module 19: Review Question Answers

1. Because the row is hash-distributed on different columns, the subtable row will typically land on an AMP other than the one containing the data row. This index would be:
   a. UPI or NUPI
   b. USI
   c. NUSI
   d. None of the above

2. The Teradata DBS hashes the value and uses the Row Hash to find the desired rows. This is always a one-AMP operation. This index would be:
   a. UPI or NUPI
   b. USI
   c. NUSI
   d. None of the above

3. _____ is a process that determines common Row IDs between multiple NUSI values by a process of intersection.
   a. NUSI Bit Mapping
   b. Dual NUSI Access
   c. Full Table Scan
   d. NUSI Read

4. If aggregation is performed on a NUSI column, the Optimizer accesses the NUSI subtable and returns the result without accessing the base table. This is referred to as:
   a. NUSI bit mapping
   b. Full table scan
   c. Dual NUSI access
   d. Covering Index

Module 20: Exercise 3 – Choosing SI Candidates

ENTITY 1

100,000,000 Rows	A	B	C	D	E	F
PK/FK	PK,UA					
Distinct Values	100M	95M	300K	250K	40M	1M
Max Rows/Value	1	2	400	350	3	110
Max Rows/NULL	0	0	0	0	1.5M	0
Typical Rows/Value	1	1	325	300	2	90
	UPI	NUPI	NUPI	NUPI		NUPI
PI/SI	USI	NUSI	NUSI	NUSI	NUSI	NUSI
Collect Statistics (Y/N)						

Module 20: Exercise 3 – Choosing SI Candidates

	ENTITY 2						
10,000,000 Rows	G	H	I	J	K	L	
PK/FK	PK,SA						
Distinct Values	10M	100K	9M	12	50	180K	
Max Rows/Value	1	200	2	1M	240K	60	
Max Rows/NULL	0	0	100K	0	0	0	
Typical Rows/Value	1	100	1	800K	190K	50	
	UPI	NUPI				NUPI	
PI/SI	USI	NUSI	NUSI		NUSI	NUSI	
Collect Statistics (Y/N)							

Module 20: Exercise 3 – Choosing SI Candidates

	DEPENDENT						
5,000,000 Rows	A	M	N	O	P	Q	
PK/FK	PK				NN,ND		
	FK	SA					
Distinct Values	2M	50	90K	3M	5M	2M	
Max Rows/Value	4	200K	75	2	1	5	
Max Rows/NULL	0	0	0	390K	0	1M	
Typical Rows/Value	1	60K	50	1	1	1	
PI/SI	UPI				UPI		
	NUPI		NUPI				
	USI				USI		
	NUSI	NUSI	NUSI	NUSI		NUSI	
Collect Statistics (Y/N)							

Module 20: Exercise 3 – Choosing SI Candidates

ASSOCIATIVE 1

300,000,000 Rows	A	G	R	S	
PK/FK	PK				
	FK	FK,SA			
Distinct Values	100M	10M	15K	800K	
Max Rows/Value	5	50	21K	400	
Max Rows/NULL	0	0	0	0	
Typical Rows/Value	3	30	19K	350	
	UPI				
PI/SI	NUPI	NUPI	NUPI?	NUPI	
	USI				
	NUSI	NUSI	NUSI	NUSI	
Collect Statistics (Y/N)					

Module 20: Exercise 3 – Choosing SI Candidates

ASSOCIATIVE 2

100,000,000 Rows	A	M	G	T	U	
PK/FK	PK					
	FK		FK			
Distinct Values	50M		10M	560K	750	
Max Rows/Value	3		150	180	135K	
Max Rows/NULL	0		0	0	0	
Typical Rows/Value	1		8	170	100K	
	UPI					
PI/SI	NUPI		NUPI	NUPI		
	USI					
	NUSI		NUSI	NUSI	NUSI	
Collect Statistics (Y/N)						

Module 20: Exercise 3 – Choosing SI Candidates

HISTORY

730,000,000 Rows	A	DATE	D	E	F	
PK/FK	PK					
	FK	FK,SA				
Distinct Values	100M	730	N/A	N/A	N/A	
Max Rows/Value	18	1100K	N/A	N/A	N/A	
Max Rows/NULL	0	0	N/A	N/A	N/A	
Typical Rows/Value	3	900K	N/A	N/A	N/A	
PI/SI	UPI					
	NUPI					
	USI					
	NUSI	NUSI				
Collect Statistics (Y/N)						

Module 20: Exercise 4 – Eliminating SI Candidates

ENTITY 1

100,000,000 Rows	A	B	C	D	E	F
PK/FK	PK,UA					
Value Access	50K	0	0	0	0	0
Range Access	0	0	0	0	0	0
Distinct Values	100M	95M	300K	250K	40M	1M
Max Rows/Value	1	2	400	350	3	110
Max Rows/NULL	0	0	0	0	1.5M	0
Typical Rows/Value	1	1	325	300	2	90
Change Rating	0	3	1	1	1	1
PI/SI	UPI	~~NUPI~~	NUPI	NUPI		NUPI
	USI	~~NUSI~~	~~NUSI~~	~~NUSI~~	~~NUSI~~	~~NUSI~~
Collect Statistics (Y/N)						

Module 20: Exercise 4 – Eliminating SI Candidates

	ENTITY 2						
10,000,000 Rows	G	H	I	J	K	L	
PK/FK	PK,SA						
Value Access	5K	365	12	12	0	0	
Range Access	12	0	0	0	0	260	
Distinct Values	10M	100K	9M	12	50	180K	
Max Rows/Value	1	200	2	1M	240K	60	
Max Rows/NULL	0	0	100K	0	0	0	
Typical Rows/Value	1	100	1	800K	190K	50	
Change Rating	0	0	9	1	2	0	
	UPI	NUPI				NUPI	
PI/SI	USI	NUSI	~~NUSI~~		~~NUSI~~	VONUSI	
Collect Statistics (Y/N)							

Module 20: Exercise 4 – Eliminating SI Candidates

	DEPENDENT						
5,000,000 Rows	A	M	N	O	P	Q	
PK/FK	PK				NN,ND		
	FK	SA					
Value Access	0	0	0	0	0	0	
Range Access	0	0	0	0	0	0	
Distinct Values	2M	50	90K	3M	5M	2M	
Max Rows/Value	4	200K	75	2	1	5	
Max Rows/NULL	0	0	0	390K	0	1M	
Typical Rows/Value	1	60K	50	1	1	1	
Change Rating	0	0	3	1	0	1	
PI/SI	UPI				UPI		
	NUPI		~~NUPI~~				
	USI				USI		
	~~NUSI~~	~~NUSI~~	~~NUSI~~	~~NUSI~~		~~NUSI~~	
Collect Statistics (Y/N)							

Module 20: Exercise 4 – Eliminating SI Candidates

ASSOCIATIVE 1

300,000,000 Rows	A	G	R	S
PK/FK	PK			
	FK	FK,SA		
Value Access	260	0	0	0
Range Access	0	0	0	0
Distinct Values	100M	10M	15K	800K
Max Rows/Value	5	50	21K	400
Max Rows/NULL	0	0	0	0
Typical Rows/Value	3	30	19K	350
Change Rating	0	0	0	0
PI/SI	UPI			
	NUPI	NUPI	NUPI?	NUPI
	USI			
	NUSI	~~NUSI~~	~~NUSI~~	~~NUSI~~
Collect Statistics (Y/N)				

Module 20: Exercise 4 – Eliminating SI Candidates

ASSOCIATIVE 2

100,000,000 Rows	A	M	G	T	U	
PK/FK	PK					
	FK		FK			
Value Access	0		0	0	0	
Range Access	0		0	0	0	
Distinct Values	50M		10M	560K	750	
Max Rows/Value	3		150	180	135K	
Max Rows/NULL	0		0	0	0	
Typical Rows/Value	1		8	170	100K	
Change Rating	0		0	0	0	
PI/SI	UPI					
	NUPI		NUPI	NUPI		
	USI					
	~~NUSI~~		~~NUSI~~	~~NUSI~~	~~NUSI~~	
Collect Statistics (Y/N)						

Module 20: Exercise 4 – Eliminating SI Candidates

HISTORY

730,000,000 Rows	A	DATE	D	E	F	
PK/FK	PK					
	FK	SA				
Value Access	10M	5K	0	0	0	
Range Access	0	20K	0	0	0	
Distinct Values	100M	730	N/A	N/A	N/A	
Max Rows/Value	18	1100K	N/A	N/A	N/A	
Max Rows/NULL	0	0	N/A	N/A	N/A	
Typical Rows/Value	3	900K	N/A	N/A	N/A	
Change Rating	0	0	N/A	N/A	N/A	
PI/SI	UPI					
	NUPI					
	USI					
	NUSI	VONUSI				
Collect Statistics (Y/N)						

Module 20: Review Question Answers

1. With a NUPI, a technique to avoid a duplicate row check is to _____.

   a. use set tables
   b. use the NOT NULL constraint on the column
   c. create the table as MULTISET table
   d. compare data values byte-by-byte within a Row Hash in order to ensure uniqueness

2. Which type of usage normally applies to a USI? ____

   a. Range access
   b. NOT condition
   c. Equality value access
   d. Inequality value access

3. Which two types of usage normally apply to a composite NUSI that is hash-ordered? ____ ____

   a. Covering index
   b. Equality value access
   c. Inequality value access
   d. Non-covering range access

Module 21: Review Question Answers

1. Which one of the following situations requires the use of the Transient Journal?
   a. loading a table with FastLoad
   b. DELETE all the rows in a table
   c. UPDATE all the rows in a table
   d. INSERT/SELECT into an empty table

2. What is a negative impact of updating a UPI value?

   Very I/O intensive - requires that (internally) the data row be deleted and re-inserted into the table as well as updating the existing secondary indexes.

3. What are the 4 types of constraints?

   Primary Key *Unique* *References* *Check*

4. True or False? A primary key constraint is always implemented as a primary index.
5. True or False? A primary key constraint is always implemented as a unique index.
6. True or False? Multi-column constraints must be coded as table level constraints.
7. True or False? Only named check constraints may be modified.
8. True or False? Named primary key constraints may always be dropped if they are no longer needed.
9. True or False? Using the "START WITH 1" and "INCREMENT BY 1" options with an Identity column will provide sequential numbering with no gaps for the column.

Module 22: Review Question Answers

1. **Which of the following rules is not one of the relational modeling rules for a Primary Key?**
   a. **Must not contain NULL values**
   b. **Must be unique**
   c. **Must consist of a single column**
   d. **Must not change**

2. **Which choice cannot be referenced by a Foreign Key?**
   a. **Column defined as a USI with NOT NULL**
   b. **Column defined as UNIQUE with NOT NULL**
   c. **Column defined as a NUPI with NOT NULL**
   d. **Column defined as a Primary Key with NOT NULL**

3. ***True*** **or False. A reference index subtable is only created for standard (or full) referential integrity.**

4. **How is the reference index subtable hash distributed?**

   By the foreign key values.

5. **How can a reference index row be marked as "invalid"?**

   1) Adding a RI constraint to a populated table
   2) Revalidating a RI constraint after a table is restored

Module 23: Review Question Answers

1. Which of the following can be used with the COMPRESS option?
   a. Identity column
   b. Non-unique Primary Index
   c. USI as PK with Standard RI
   d. Non-unique Secondary Index

2. Which section of a row identifies the starting location of variable length column data and is present only if variable length columns without compression are declared?
   a. Presence Bits
   b. Column Offsets
   c. VARCHAR Columns
   d. Uncompressed Columns

3. How can you override the default that a column with a NULL value will require row space?
   a. Use the NOT NULL option on the column as part of the CREATE TABLE statement.
   b. Set the user's default so columns will default to COMPRESS when creating a table.
   c. Use the COMPRESS option on the column as part of the CREATE TABLE statement.
   d. Use the DEFAULT NULL option on the column as part of the CREATE TABLE statement.

4. What is the minimum space the table headers will take for a 6-column table on a 10 AMP system?
   a. 1024 bytes
   b. 4096 bytes
   c. 5120 bytes
   d. 10240 bytes

5. What DD view is used to get sizing information about tables? *DBC.TableSizeV*

Module 24: Review Question Answers

1. What must a REQUEST parcel contain? *At least 1 SQL statement*

2. Which two statements about the RESPOND parcel are true? ___
   - a. Identifies response buffer size.
   - b. Generates a SUCCESS/FAIL parcel.
   - c. Always followed by one or more DATA parcels.
   - d. May be sent by itself as a continuation request.

3. Match the six SQL Parser phases listed below with its correct description.

c Syntaxer	a. Determines whether the Requesting User ID has the necessary permissions
e Resolver	b. Create concrete steps
a Security	c. Checks the Request for valid syntax.
f Optimizer	d. Creates the steps for execution.
d Generator	e. Breaks down Views and Macros into their underlying table references
b Apply	f. Chooses the execution plan.

4. Which Parser phase benefits the most from the use of macros? ___
   - a. Generator
   - b. Resolver
   - c. Syntaxer
   - d. Apply

Module 24: Review Question Answers

5. What is the function of the Request-to-Steps (R-T-S) Cache? ____

   a. Stores the SQL text and AMP steps generated by the Parser.
   b. Resolves View and Macro references down to table references.
   c. Stores the most recently used DD information including SQL names, their related numeric IDs and Statistics.
   d. Analyzes the various ways an SQL Request can be executed and determines which of these is the most efficient.

6. Teradata's Late Binding Parser refers to Apply, which acts upon the _____ from the Generator and produces _____ by binding in the data values from the DATA parcel.

   a. Plastic Steps / Concrete Steps
   b. Interpretive Steps / Compiled Steps
   c. Processing Steps / Execution Steps
   d. AMP steps / Request-to-Steps Cache

Module 25: Review Question Answers

1. When alternatives are available, the Optimizer may require “hints” to ensure that it will make the best choices.
   - T. True
   - F. False

2. If you COLLECT STATISTICS for a NUPI in Teradata 14.0, the statistics are stored in _____.
   - a. DBC.StatsTbl
   - b. DBC.Indexes
   - c. DBC.TVFields
   - d. Data Dictionary (DD) cache

3. Dynamic AMP sample statistics are stored in the _____.
   - a. DBC.TVFields
   - b. DBC.Indexes
   - c. Data Dictionary (DD) cache
   - d. none of the above

4. You can use the _____ to display information (e.g., last collection date) about current column or index statistics.
   - a. EXPLAIN statement
   - b. HELP INDEX statement
   - c. SHOW TABLE statement
   - d. COLLECT STATISTICS statement
   - e. HELP STATISTICS statement

Module 26: Review Question Answers

Fill in the blanks.

1. ***Parallel*** **steps are multi-AMP processing steps that are numbered but execute at the same time.**

2. **The primary way to help the Optimizer make the best choices and ensure the most accurate EXPLAIN output is to *Collect Statistics* on appropriate indexes and columns.**

3. **An EXPLAIN plan will indicate "estimated with *High* confidence" when a value for an index is provided to retrieve the data and the index has collected statistics.**

4. **Name the two ways to EXPLAIN a Macro:**

   a. ***Hard-coded parameter values***
   b. ***Soft-coded parameter values***

(Continued on next page.)

Module 26: Review Question Answers

Match each EXPLAIN term to a definition.

i	1.	(Last Use)
c	2.	END TRANSACTION
h	3.	eliminating duplicate rows
l	4.	by way of the sort key in spool field
b	5.	does SMS (set manipulation step)
f	6.	does BMSMS (bit map ...)
e	7.	redistributed by hash code to all AMPs
d	8.	“Pseudo Table”
g	9.	all rows scan
k	10.	“a single partition of” or “n partitions of”
m	11.	“a rowkey-based merge join”
j	12.	group_amps operation
a	13.	“SORT to partition Spool m by rowkey”
e	14.	which is duplicated on all AMPs

a. The spool is to be ordered by partition and hash.

b. Combines answer sets using a UNION, EXCEPT (MINUS) or INTERSECT operator.

c. Indicates transaction locks are released and changes are committed.

d. Internal function to synchronize table-level locks across AMPs.

e. Indicates data is being relocated in preparation for a join.

f. Indicates that NUSI Bit Mapping is being used.

g. Indicates a full table scan.

h. Indicates that a DISTINCT operation is done to ensure that there are no duplicate rows.

i. Indicates that the Spool file will be released at the end of the step.

j. Subset of AMPs will be used instead of all AMPs.

k. Indicates partition elimination will occur.

l. Field1 is created to allow a tag sort.

m. Indicates an equality join based on partition and hash.

Module 27: Review Question Answers

1. To place an execution plan into the QCD database, preface a valid parsable Teradata SQL statement with ***INSERT EXPLAIN***.
2. When using Visual Explain to compare multiple plans, one plan must be selected as the ***base*** query.
3. List the 3 types of QCD users that access rights are associated with.

   Normal ***Power User*** ***Administrator***

Module 28: Review Question Answers

1. The best way to be sure what type of Join will occur is to use the EXPLAIN facility.

   a. True
   b. False

2. When two tables are to be Merge Joined, which is the best case of the following scenarios :

   a. The Join column is not a Primary Index of either table.
   b. The Join column is the Primary Index of one of the tables.
   c. The Join column(s) is the Primary Index of both tables.
   d. None of the above

3. Match the four join plans with its correct description.

_C_	Product Join	a.	Most efficient types of Join; the only type of join that doesn't always use all of the AMPs. The number of AMPs involved will vary.
_D_	Merge Join	b.	Based on set subtraction; used for finding rows that don't have a matching row in the other table. Queries with the NOT IN and EXCEPT operator lead to this type of join.
_A_	Nested Join	c.	Every qualifying row of one table is compared to every qualifying row in the other table. Rows that match on their WHERE conditions are then saved.
_B_	Exclusion Join	d.	Commonly done when the join condition is based on equality. Efficient because the every row is not compared with every row in other table.

Module 28: Review Question Answers

Fill in the blanks.

4. When joining two PPI tables that are not partitioned in the same manner, a technique available to the optimizer is referred to as the ___*sliding*___ window.

5. A direct merge join of two PPI tables when the tables have the same PI and identical partitioning expressions is referred to as a _*rowkey*_ - based merge join.

6. The term ___*partition elimination*___ refers to an automatic optimization in which the optimizer determines, based on query conditions, that some partitions can be skipped.

Module 29: Exercise 5 – Making Final Index Choices

ENTITY 1

100,000,000 Rows	A	B	C	D	E	F
PK/FK	PK,UA					
Value Access	50K	0	0	0	0	0
Range Access	0	0	0	0	0	0
Join Access	10M	0	0	0	0	0
Join Rows	10M	0	0	0	0	0
Distinct Values	100M	95M	300K	250K	40M	1M
Max Rows/Value	1	2	400	350	3	110
Max Rows/NULL	0	0	0	0	1.5M	0
Typical Rows/Value	1	1	325	300	2	90
Change Rating	0	3	1	1	1	1
PI/SI	UPI ~~USI~~		~~NUPI~~	~~NUPI~~		~~NUPI~~
Collect Statistics (Y/N)	Y*					

Y* – For large tables, a dynamic sample or COLLECT SAMPLE will suffice.

Module 29: Exercise 5 – Making Final Index Choices

ENTITY 2

10,000,000 Rows	G	H	I	J	K	L
PK/FK	PK,SA					
Value Access	5K	365	12	12	0	0
Range Access	12	0	0	0	0	260
Join Access	100M	0	0	0	0	0
Join Rows	100M	0	0	0	0	0
Distinct Values	10M	100K	9M	12	50	180K
Max Rows/Value	1	200	2	1M	240K	60
Max Rows/NULL	0	0	100K	0	0	0
Typical Rows/Value	1	100	1	800K	190K	50
Change Rating	0	0	9	1	2	0
PI/SI	UPI ~~USI~~	~~NUPI~~ NUSI				~~NUPI~~ VONUSI
Collect Statistics (Y/N)	Y*	Y	Y**	Y**		Y

Y* – For large tables, a dynamic sample or COLLECT SAMPLE will suffice.

Y** – Possibly only collect statistics (or SAMPLE) when needed. Column I may be too volatile for stats.

Module 29: Exercise 5 – Making Final Index Choices

	DEPENDENT						
5,000,000 Rows	A	M	N	O	P	Q	
PK/FK	PK				NN,ND		
	FK	SA					
Value Access	0	0	0	0	0	0	
Range Access	0	0	0	0	0	0	
Join Access	700K	0	0	0	0	0	
Join Rows	1M	0	0	0	0	0	
Distinct Values	2M	50	90K	3M	5M	2M	
Max Rows/Value	4	200K	75	2	1	5	
Max Rows/NULL	0	0	0	390K	0	1M	
Typical Rows/Value	1	60K	50	1	1	1	
Change Rating	0	0	3	1	0	1	
PI/SI	~~UPI~~				~~UPI~~		
	NUPI						
	USI				USI		
Collect Statistics (Y/N)	Y						

Module 29: Exercise 5 – Making Final Index Choices

ASSOCIATIVE 1

300,000,000 Rows	A	G	R	S
PK/FK	PK			
	FK	FK,SA		
Value Access	260	0	0	0
Range Access	0	0	0	0
Join Access	0	8M	0	0
Join Rows	0	300M	0	0
Distinct Values	100M	10M	15K	800K
Max Rows/Value	5	50	21K	400
Max Rows/NULL	0	0	0	0
Typical Rows/Value	3	30	19K	350
Change Rating	0	0	0	0
PI/SI	~~UPI~~ ~~NUPI~~	NUPI	~~NUPI?~~	~~NUPI~~
	USI			
	NUSI			
Collect Statistics (Y/N)	Y	Y		

Module 29: Exercise 5 – Making Final Index Choices

ASSOCIATIVE 2

100,000,000 Rows	A	M	G	T	U	
PK/FK	PK					
	FK		FK			
Value Access	0		0	0	0	
Range Access	0		0	0	0	
Join Access	7M		250K	0	0	
Join Rows	800M		20M	0	0	
Distinct Values	50M		10M	560K	750	
Max Rows/Value	3		150	180	135K	
Max Rows/NULL	0		0	0	0	
Typical Rows/Value	1		8	170	100K	
Change Rating	0		0	0	0	
PI/SI	~~UPI~~					
	NUPI		~~NUPI~~	~~NUPI~~		
	USI					
Collect Statistics (Y/N)	Y		Y			

What additional index choices would be available for the column G?

Note: Options that may improve the join performance on G: 1) Create a single table join index on G or 2) Create a covered NUSI on G (order by hash on G).

Module 29: Exercise 5 – Making Final Index Choices

HISTORY

730,000,000 Rows	A	DATE	D	E	F	
PK/FK	PK					
	FK	SA				
Value Access	10M	5K	0	0	0	
Range Access	0	20K	0	0	0	
Join Access	800M	0	0	0	0	
Join Rows	2.4B	0	0	0	0	
Distinct Values	100M	730	N/A	N/A	N/A	
Max Rows/Value	18	1100K	N/A	N/A	N/A	
Max Rows/NULL	0	0	N/A	N/A	N/A	
Typical Rows/Value	3	900K	N/A	N/A	N/A	
Change Rating	0	0	N/A	N/A	N/A	
PI/SI	~~UPI~~ NUPI					
	USI					
	NUSI	~~VOLATILE USI~~				
Collect Statistics (Y/N)	Y	Y				

What additional index choices would be available for the DATE column?

Possibly create a PPI on A partitioned by DATE. If so, maybe leave the NUSI on A for value access.

Module 29: Review Question Answers

1. When do Cartesian Product Joins generally occur?

   a. When the Join Column is the Primary Index of both tables.
   b. When a column appears in an Equality Join constraint.
   c. When an error is made in coding the SQL query.
   d. None of the above.

2. A Join Condition of (1=1) in an EXPLAIN output is usually indicative of _____.

   a. Cartesian product join
   b. Exclusion merge join
   c. Merge join
   d. Hash join

3. One of the ways in which Join Access demographics are expressed is _____ , which is a measure of how often all known transactions access rows from the table through a Join on this column.

   a. Join Access Rows
   b. Join Access Frequency
   c. High Access Rows
   d. Value and Join Access

Module 30: Review Question Answers

Check the box if the attribute applies to the index.

	Compressed Join Index Syntax	Non-Compressed Join Index Syntax	Aggregate Join Index	Sparse Join Index	Hash Index
May be created on a single table	✓	✓	✓	✓	✓
May be created on multiple tables	✓	✓	✓	✓	
Requires the use of SUM or COUNT functions			✓		
Requires a WHERE condition to limit rows stored in the index.				✓	
Automatically updated as base table rows are inserted or updated	✓	✓	✓	✓	✓
Automatically includes the base table PI value as part of the index					✓

Module 31: Review Question Answers

1. Which BTEQ setting controls Teradata vs. ANSI mode? *.SET SESSION TRANSACTION*
2. Which commands will not work in ANSI mode? *BT, ET*
3. *True* or False. The SQL Flagger is just a warning device and doesn't affect command execution.
4. True or *False*. Failure of an individual request in ANSI (or COMMIT) mode causes the entire transaction to be rolled back.
5. *True* or False. Logging off during an explicit transaction without either a COMMIT or ET will always result in a ROLLBACK.
6. True or *False*. HELP SESSION will show the session mode and the status of the SQL Flagger.
7. Where does a Volatile Temporary table get its space from? *Spool*
8. Where does a Global Temporary table get its space from? *Temporary*

Module 32: When Do Multiple Sessions Make Sense?

Multiple sessions improve performance ONLY for SQL requests that impact fewer than ALL AMPs.

TRANS_HISTORY

Trans_Number	Trans_Date	Account_Number	Trans_ID	Amount
PK		FK,NN		
USI		NUPI		
NUSI				

Which of the following batch requests would benefit from multiple sessions?

1. INSERT INTO Trans_History
   VALUES (:T_Nbr, DATE, :Acct_Nbr, :T_ID, :Amt);
2. SELECT * FROM Trans_History
   WHERE Trans_Number=:Trans_Number;
3. DELETE FROM Trans_History
   WHERE Trans_Date < DATE - 120;
4. DELETE FROM Trans_History
   WHERE Account_Number= :Account_Number;

Trans Type	Table or Row Lock	Multiple Sessions Useful or Not?
NUPI	Row Hash	Yes
NUSI	Full Table	No
FTS	Full Table	No
NUPI	Row Hash	Yes

Module 32: Review Question Answers

Answer True or False.

1. True or ***False***. With MultiLoad, you can import and export data.
2. ***True*** or False. In Teradata mode, a BTEQ DELETE ALL function does not use the Transient Journal to store before-images of deleted rows.
3. True or ***False***. An INSERT/SELECT of 1,000,000 rows into an empty table is only slightly faster than an INSERT/SELECT of 1,000,000 rows into a table with 1 row.

Match the Teradata Parallel Transporter operator with the corresponding Teradata utility.

1. _*a*_ UPDATE	a. MultiLoad
2. _*d*_ STREAM	b. FastLoad
3. _*b*_ LOAD	c. FastExport
4. _*c*_ EXPORT	d. TPump

Module 33: Review Question Answers

Answer True or False.

1. ***True*** **or False. With BTEQ you can import data from the host to Teradata AND export from Teradata to the host.**
2. **True or *False*. .EXPORT DATA sends results to a host file in field mode. *(Results are in record mode.)***
3. ***True*** **or False. INDICDATA is used to preserve nulls.**
4. ***True*** **or False. With BTEQ, you can use conditional logic to bypass statements based on a test of an error code.**
5. **True or *False*. It is useful to employ multiple sessions when ALL AMPS will be used for the transaction. *(It is useful when fewer than all AMPs are used.)***
6. ***True*** **or False. With .EXPORT, you can have output converted to a format that can be used with PC programs.**

Module 34: Review Question Answers

Match the item in the first column to a corresponding statement in the second column.

1. c Phase 1	a. Might be used if a zero date causes an error
2. g CHECKPOINT	b. Table status required for loading with FastLoad
3. h ERRORTABLE1	c. Records written in unsorted blocks
4. d ERRORTABLE2	d. Records rows with duplicate values for UPI
5. b Empty Table	e. Not permitted on table to be loaded with FastLoad
6. e Secondary Index	f. Points FastLoad to a record in an input file
7. j Conversion	g. Can be used to restart loading from a given point
8. a NULLIF	h. Records constraint violations
9. f RECORD	i. Builds the actual table blocks for the new table
10. i Phase 2	j. Transform one data type to another, once per column

Module 35: Review Question Answers

Match the item in the first column to its corresponding statement in the second column.

__*d*__ 1. .LOGTABLE	a. Connects sessions to Teradata
__*a*__ 2. .LOGON	b. Uses a single data record to set one or more utility variables
__*b*__ 3. .ACCEPT	c. System variable
__*e*__ 4. UPDATE	d. Identifies the log to create or acquire
__*c*__ 5. &SYSDATE	e. Teradata SQL statement permitted by Support Environment

Module 36: Review Question Answers

Answer True or False.

1. True or ***False***. **FastExport requires the use of a PI or USI in the SELECTs.**
2. True or ***False***. **The number of FastExport sessions (for a Linux server) defaults to the number of AMPs. *(The UNIX default is 4 sessions.)***
3. True or ***False***. **The maximum block size you can specify is 128 KB. *(It is 64 KB.)***
4. ***True*** **or False. You can export from multiple tables with FastExport.**
5. ***True*** **or False. You can use multiple SELECTs in one FastExport job.**
6. True or ***False***. **The default lock for a SELECT in a FastExport job is a table level ACCESS lock.**

Module 37: Review Question Answers

Answer True or False.

1. ***True*** or False. With MultiLoad, you can import data from the host into populated tables.
2. ***True*** or False. MultiLoad cannot process tables with USIs or Referential Integrity defined.
3. True or ***False***. MultiLoad allows changes to the value of a table's primary index.
4. ***True*** or False. MultiLoad allows you to change the value of a column based on its current value.
5. ***True*** or False. MultiLoad permits non-exclusive access to target tables from other users except during Application Phase.

Match the MultiLoad Phase in the first column to its corresponding task in the second column.

1. _*a*_ Preliminary	a. Acquires or creates Restart Log Table.
2. _*e*_ DML Transaction	b. Locks are released.
3. _*c*_ Acquisition	c. Applies (loads) data to the work tables.
4. _*d*_ Application	d. Execute mload for each target table as a single multi-statement request.
5. _*b*_ Cleanup	e. Stores DML steps in work tables

Module 38: Review Question Answers

1. **Complete the BEGIN statement to accomplish the following:**
   - **Specify an error limit count of 200,000 and an error percentage of 5%.**
   - **Specify a checkpoint at 500,000 records.**
   - **Request 16 sessions, but allow the job to run with only 8.**
   - **Set the number of hours to try to establish connection as 6.**

```
.LOGTABLE RestartLog_mld;
.LOGON _____;
.BEGIN  [IMPORT]  MLOAD TABLES  Trans_Hist
        ERRLIMIT  200000  5
        CHECKPOINT  500000
        SESSIONS  16  8
        TENACITY  6
;
.END MLOAD ;
```

Module 39: Review Question Answers

Match the item in the first column to its corresponding statement in the second column.

__*c*__ 1. TPump purpose	a. Query against TPump status table
__*e*__ 2. MultiLoad purpose	b. Concurrent updates on same table
__*b*__ 3. Row hash locking	c. Low-volume changes
__*d*__ 4. PACK	d. Use to specify how many statements to put in a multi-statement request
__*f*__ 5. MACRO	e. Large volume changes
__*a*__ 6. Statement rate change	f. Used instead of DML

Module 40: Various Ways of Performing an Update Solution

Answer: 2, 4, 3, 1 (Note: Timings are for an older and very small Teradata system).

- **UPDATE (Do the UPDATE statement as shown):**

  Timings: **Total time:** **29 minutes 9 seconds**

- **INSERT / SELECT the revised values into a new table; drop the old table and rename the new table.**

Timings:		
	Create new table	1 second
	Insert/Select 900000 rows	1 minute 26 seconds
	Drop Old Table	11 seconds
	Rename New Table	1 second
	Total Time:	**1 minute 39 seconds**

- **.EXPORT the new data values and the primary index columns to the Host and use MultiLoad UPDATE.**

Timings:		
	Export 900,000 rows	14 seconds
	MultiLoad/Update	3 minutes 27 seconds
	Total Time:	**3 minutes 41 seconds**

- **.EXPORT the whole rows to the Host selecting the updated values along with the rest of the record, and FastLoad the table.**

Timings:		
	Export 900,000 rows	16 seconds
	Delete old rows	2 seconds
	FastLoad the data	1 minute 59 seconds
	Total Time:	**2 minutes 17 seconds**

Module 40: Choosing a Utility Exercise Solution

1. **A sales table currently contains 24 months of transaction data. At the end of each month, 250 million rows are added for the current month and 250 million rows are removed for the oldest month There is enough PERM space to hold 30 months worth of data.**

   Which choice (from below) makes the most sense? ***d***

2. **The customer decides to partition the table by month and maintain each month's data in a separate partition. At this time, only the most recent 24 months need to be maintained. At the end of each month, data is loaded into a new monthly partition and the oldest month is removed. The partition expression does not include the NO RANGE partition.**

   Which choice (from below) makes the most sense? ***b***

Utility Choices:

a. **Use FastLoad to add new data to existing table, and ALTER TABLE to remove old data.**
b. **Use MultiLoad to add new data to existing table, and ALTER TABLE to remove old data.**
c. **Use FastLoad to add new data to existing table, and MultiLoad to remove old data.**
d. **Use MultiLoad to add new data to existing table, and MultiLoad to remove old data.**
e. **Use TPump to add new data, and TPump to remove old data.**
f. **Use TPump to add new data, and ALTER TABLE to remove old data.**

Module 41: Review Question Answers

1. True or False. — You should use system user DBC to create application users and databases.
   You should create and logon as an administrative user to perform these tasks.
2. True or False. — A database or user can have multiple Owners, but only one Creator.
3. True or False. — An Owner and a Parent are two different terms that mean the same thing.
4. True or False. — An Owner and a Creator are two different terms that mean the same thing.
   The term "Creator" is used to mean the one and only one user who creates a database object.
5. True or False. — An administrative user (e.g., Sysdba) will never have more permanent space than DBC.
   Sysdba can be allocated more permanent space than DBC.
6. True or False. — The GIVE statement transfers a database or user space to a recipient you specify. It does not automatically transfer all child databases.

   The GIVE statement automatically transfers all child databases, users, tables, view and macros the transferred object owns.

Module 42: Review Question Answers

1. True or *False*. The DBC.Databases view only contains information about databases; users are not included in this view.

2. *True* or False. The DBC.Users view only contains information about users; databases are not included in this view.

3. True or *False*. Queries that use restricted views usually take less time to execute than queries that use unrestricted views.

4. True or *False*. All of the data dictionary tables are Fallback protected.

5. If a child table exists and the parent table doesn't, the reference constraint is marked as _____.
   a. Inconsistent
   b. Unresolved
   c. Missing
   d. Invalid

6. After executing the ALTER TABLE … ADD FOREIGN KEY … statement, Foreign Key values that are missing in the parent table are marked in an error table and are known as _____ rows.
   a. Inconsistent
   b. Unresolved
   c. Missing
   d. Invalid

Module 43: Review Question Answers

1. True or False. Space limits are enforced at the table level.

2. True or False. When you use the GIVE statement to transfer a database/user, only the tables allocated to the original database/user are transferred to the new database/user.

3. True or False. You should reserve at least 25% of total available space for spool.

4. The DBC. *TablesizeV* view provides disk space usage at the table level and excludes table ALL.

5. The DBC. *DiskSpaceV* view only provides disk space usage at the database level.

Module 44: Review Question Answers

1. True or False. You can only give the authority to use the CREATE DATABASE and CREATE USER statements to certain types of users.

2. True or False. An individual user with a $L priority will always receive less CPU time than a user with a $M priority.

3. True or False. A user can use the MODIFY USER statement to change their password, default database, and date format.

4. When creating a new user, which option defaults to the immediate owner's value. ___

   a. SPOOL
   b. FALLBACK PROTECTION
   c. All of the Account_IDs
   d. DEFAULT DATABASE

5. When creating a new user, which options are required with the CREATE USER command. ___

   a. SPOOL
   b. PERMANENT
   c. User name
   d. PASSWORD

Module 45: Review Question Answers

Answer the following questions:

1. List 2 advantages of utilizing profiles.
   Change common user attributes for large groups of users.
   Specify password security controls for groups of users.

2. If a user's profile is set to NULL, which two are immediately affected in the current session.

   a. SPOOL value
   b. Default database
   c. Session priority
   d. TEMPORARY value

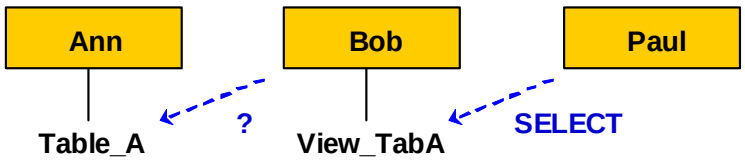
Match each term to the definition.

b	1. CREATE PROFILE	a. Lists all words that can not be included in passwords
c	2. DBC.ProfileInfo	b. System access right needed to create a profile
a	3. DBC.PasswordRestrictions	c. Lists the profiles currently in the system

Module 46: Review Question Answers

1. True or False — There are only two types of access rights or privileges: explicit and implicit.
2. True or False — The primary statements you use to manage access rights are GRANT, REVOKE, and GIVE.
3. The __ALL__ option on the GRANT command grants privileges to a database or user and all of its current and future descendants.
4. The __INSERT__, __REFERENCES__, __SELECT__, and __UPDATE__ access rights can be granted at the column level.
5. The __PUBLIC__ user is used to grant an access right to every user in the system.
6. Given the following: Ann owns Table_A, Bob creates View_TabA and grants SELECT on View_TabA to Paul.

   What access right does Ann give Bob on Table_A so Paul can use View_TabA to access Table_A?
   __SELECT WITH GRANT OPTION__



Module 47: Review Question Answers

Answer the following questions:

1. List 3 advantages of utilizing roles.

   Simplify access rights management
   The number of access rights in the DBC.AccessRights table is reduced.
   Improves performance and reduces dictionary contention

2. How many levels of role nesting are currently allowed? 1

3. True or False. A user can use the SET ROLE command to set their current role to any defined role in the system.

4. True or False. Roles may only be granted to users and other roles.

Match each term to the definition.

e	1. WITH ADMIN OPTION	a.	Established by the SET ROLE command
c	2. CREATE ROLE	b.	Lists the roles currently in the system
b	3. DBC.RoleInfo	c.	System access right needed to create a role
d	4. DBC.UserRoleRights	d.	Lists all of a user's role rights - including nested roles
f	5. DEFAULT ROLE	e.	Allows the user to assign other users to the role
a	6. Current role	f.	Option with the MODIFY USER statement

Module 48: Review Question Answers

1. True or False. Although usernames are the basis for identification in the system, username information usually is not protected information.

2. True or False. Once users have a username and password, they can access any information in the system.

3. True or False. To change the minimum number of characters in a valid password from 6 to 8, you would update the DBC.LogonRules table.

4. What does 1024 represent in the DBC.LogonRules view? *All of the hosts*

5. Which choices can be used to view host or mainframe sessions? *b and d*

   a. Sessions utility
   b. QrySessn utility
   c. Gtwglobal utility
   d. DBC.SessionInfo view

6. Which choice is used to determine why a user logon has failed? *b*

   a. DBC.Logons view
   b. DBC.LogOnOff view
   c. DBC.AccessLog view
   d. DBC.SessionInfo view
   e. DBC.LogonEvents view

Module 49: Review Question Answers

1. In order to use the BEGIN/END LOGGING commands, what is the name of the system macro you need execute permission on?
   DBC.AccLogRule

2. How is this macro initially created?
   When the DIP script (DIPACC) is executed.

3. What is a negative impact of the following statement?

   BEGIN LOGGING WITH TEXT ON EACH ……
   Potentially a lot of entries are placed in the dictionary and would require a lot of PERM space.

4. With DBQL, what is the size of the default text captured for queries? *200 characters*

5. *True* or False. With DBQL, the LIMIT SUMMARY option cannot be used with any other LIMIT.

6. True or *False*. With DBQL, the WITH SQL option only captures a maximum of 10,000 characters.

7. True or *False*. With DBQL, the option WITH ALL ON ALL is typically a good choice.

8. *True* or False. With DBQL, default rows are logged in the DBC.DBQLogTbl.

Module 50: Review Question Answers

1. **Given the following, what is minimum % of opportunities to use the CPU resource that the following Performance Groups (PG) can expect.**

   TM = *6% (60/100 x 10/100 = .06)* **DH =** *5% (20/100 x 30/120 = .05)*

Default		Tactical		DSS	
RP 0 - Weight 20		RP 1 - Weight 60		RP 2 - Weight 20	
PG Name	AG Wgt	PG Name	AG Wgt	PG Name	AG Wgt
L	5	TL	5	DL	5
M	10	TM	10	DM	10
H	20	TH	30	DH	30
R	40	TR	55	DR	75

2. **Without TASM workloads enabled, a user session is associated with a** *Performance Group* **which is effectively assigned to an** *Allocation Group* **.**

3. **With TASM workloads enabled, a user query is associated with a** *Workload* **which is effectively assigned to an** *Allocation Group* **.**

Module 51: Review Question Answers

1. What type of TASM filter or throttle rule is needed for the following restrictions?

Limit the number of concurrent sessions	*Object Throttle*
Reject queries based on max processing time	*Query Resource Filter*
Reject queries accessing a specific DB	*Object Access Filter*
Limit the number of FastLoad jobs	*Load Utility Throttle*
Delay more the 20 queries for a specific account	*Object Throttle*

2. What is the purpose of the default workload definition name "WD-Default"?
   a. Default workload for any queries with an enforcement policy of normal.
   b. Default workload for any queries that are not associated with a workload.
   c. Default workload for any queries assigned to Default resource partition.
   d. Default workload for any queries assigned to Standard resource partition.

3. Which query attribute is not used by the Parsing Engine software to classify a query into a Workload Definition (WD)?
   a. User name
   b. Account
   c. User Role
   d. User Profile
   e. Optimizer estimates

Module 51: Review Question Answers (cont.)

4. Which Teradata application can be used to initially define workload definitions?

   a. Teradata Manager
   b. Workload Analyzer
   c. Dynamic Workload Wizard
   d. Priority Scheduler Administrator

5. Place the following control options in the proper sequence from 1 to 4 as they are acted upon by Teradata software.

   _2_ a. Object throttles
   _4_ b. Exception criteria
   _3_ c. Workload throttles
   _1_ d. Object filters

Module 52: Review Question Answers

1. **List three portlets of Viewpoint.**

   Remote Database Console

   Query Monitor

   My Queries

Module 53: Review Question Answers

1. What are two methods of setting the ResUsage logging intervals?

   Supervisor SET commands *ctl utility*

2. Match the following tools to its description.

d	1. ResUsage tables	a. Set logging rates
c	2. Teradata Viewpoint	b. Emulates a target system
b	3. Teradata SET	c. Provides session level information
a	4. ctl	d. Holds historical resource data

Module 54: Review Question Answers

1. What is the operating system command to restart Teradata? ___*tpareset*___
2. What is the DB Window supervisor command to restart Teradata? ___*restart tpa*___
3. Which of the following choices will cause a Teradata restart? ___*c, e*___

   a. SWS hard drive failure
   b. Single drive failure in RAID 1 drive group
   c. Two drive failures in same RAID 1 drive group
   d. Single power supply failure in a TPA node
   e. TPA node CPU failure
   f. One of BYNETs fails
   g. LAN connection to SMP is lost

Module 55: Review Question Answers

1. What are two ways that you initiate a Teradata system utility (e.g., dbscontrol)?

   Command-line *Teradata DB Window*

2. Identify the purpose of the following DBS Control utility parameters.

CenturyBreak	*determines break point for 21st century; how 2-digit years are interpreted*
DateForm	*sets IntegerDate or ANSIDate default; how dates are displayed and exported*
MaxLoadTasks	*max # of concurrent FastLoad, MultiLoad, & FastExport jobs*
SessionMode	*sets default session mode – BTET or ANSI*

3. True or **False.** The Checktable utility has only two levels of internal table checking.

4. **True** or False. The Table Rebuild utility rebuilds tables differently depending on whether the table is a fallback, non-fallback or permanent journal table.

5. The *SCANDISK* utility does a consistency check within an AMP's file system.

6. The *CHECKTABLE* utility does a consistency check for a table across all AMPs.

7. The *VPROCMANAGER* utility can be used to set an offline AMP to online.

Module 56: Review Question Answers

1. ***True*** or False.	A permanent journal stores committed, uncommitted, and aborted changes to a row in a table.
2. True or ***False***.	A database or user can have many permanent journals.
3. True or ***False***.	Separate Permanent Journals are required for before and after images.
4. ***True*** or False.	The Saved and Active areas are both part of the Current Journal.
5. True or ***False***.	The CREATE JOURNAL statement may be used to create a permanent journal.
6. True or ***False***.	Tables that use a Permanent Journal must be in the same database as the Permanent Journal.

Module 58: Review Question Answers

1. ***True*** or False. The Archive and Recovery utility protects against more types of potential data loss than automatic data protection features.

2. True or ***False***. Recovery and FastLoad are about the same in ease and speed to recover data.

3. True or ***False***. An All-AMPs archive of a database archives all of the objects in the database.

4. True or ***False***. Archiving a partition of a PPI table places a partition-level lock on the partition being archived.

Module 59: Review Question Answers

1. True or False. You can use the RESTORE command to restore entities that are not defined in the data dictionary.

2. True or False. When you execute a RESTORE of a database, any tables or views created since the archive of the database are dropped when you restore the database.

3. True or False. You can use the COPY operation to copy tables, views, macros, and triggers from one system to another system.

4. The REVALIDATE REFERENCES FOR statement is used to validate Referential Integrity between tables that are identified as _____.
   a. Invalid
   b. Missing
   c. Unresolved
   d. Inconsistent

Module 60: Review Question Answers

1. True or *False*. The DELETE JOURNAL command can be used to delete the active and the saved areas of the current journal.

2. In general, rollback operations help you recover from *software* failures and rollforward operations help you recover from *hardware* failures.

3. To use the ARCHIVE JOURNAL TABLE command to archive a permanent journal, the active journal images need to be moved to the saved area of the current journal. The command to do this is:

   CHECKPOINT ,WITH SAVE

TERADATA
Raising Intelligence

Appendix D: Data Dictionary Views

This Appendix contains the Data Dictionary Views for Teradata 14.0.

Data Dictionary views are part of the Teradata Database Data Dictionary and reside in the space owned by the system user DBC. They provide information about users, their access rights, grants, and logons.

View definitions are stored in the table DBC.TVM. View column information is stored in DBC.TVFields.

The following are the view forms:

- Without the X (for example, DBC.AccountInfo and DBC.AccountInfoV), they display global information.
- With the X (for example, DBC.AccountInfoX and AccountInfoVX), they display information associated with the requesting user only.
- With the V (for example, AccessLogV), they display information associated with the Unicode version, where object name columns have a data type of VARCHAR(128).
- Without the V (for example, DBC.AccountInfo or DBC.AccountInfoX), they display information associated with the Compatibility version, where object name columns have a data type of CHAR(30).

Data Dictionary Views – Teradata 14.0

DBC.AccessLog[V]

User Type	Columns Selected
Security Administrator	LogDate LogTime LogonDate LogonTime LogicalHostId IFPNo SessionNo UserName AccountName OwnerName AccessType Frequency EventCount AccLogResult Result DatabaseName TVMName ColumnName StatementType StatementText QueryBand ProxyUser

Data Dictionary Views – Teradata 14.0

DBC.AccLogRules[V]

User Type	Columns Selected
Security Administrator	UserName DatabaseName TVMName AcrAlterFunction AcrCheckpoint AcrCreateDatabase AcrCreateFunction AcrCreateMacro AcrCreateTable AcrCreateUser AcrCreateView AcrCreateProcedure AcrCreExtProcedure AcrDelete AcrDropDatabase AcrDropFunction AcrDropMacro AcrDropProcedure AcrDropTable AcrDropUser AcrDropView AcrDump AcrExecute AcrExecuteFunction AcrExectuteProcedure AcrGrant AcrIndex

DBC.AccLogRules[V] (cont.)

User Type	Columns Selected
	AcrInsert AcrReference AcrRestore AcrSelect AcrUpdate AcrCreateTrigger AcrDropTrigger AcrCreateRole AcrDropRole AcrCreateProfile AcrDropProfile AcrAlterProcedure AcrRepControl AcrAlterExtProcedure AcrUDTUsage AcrUDTType AcrUDTMethod AcrCreAuthorization AcrDropAuthorization AcrStatistics AcrShow ArcCreOwnerProcedure ArcConnectThrough CreatorName CreateTimeStamp ArcCreateGLOP ArcDropGLOP ArcGLOPMember

Data Dictionary Views – Teradata 14.0

DBC.AccountInfo[V][X]

User Type	Columns Selected
Database Administrator End User Supervisor	UserName UserOrProfile AccountName

DBC.AllRights[V][X]

User Type	Columns Selected
Database Administrator	UserName DatabaseName TableName ColumnName AccessRight GrantAuthority GrantorName AllnessFlag CreatorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.AllRoleRights[V][X]

User Type	Columns Selected
Security Administrator Database Administrator Supervisor	UserName DatabaseName TableName ColumnName AccessRight GrantorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.AllSpace[V][X]

User Type	Columns Selected
Database Administrator End User Supervisor	Vproc DatabaseName AccountName TableName MaxPerm MaxSpool MaxTemp CurrentPerm CurrentSpool CurrentTemp PeakPerm PeakSpool PeakTemp MaxProfileSpool MaxProfileTemp

Data Dictionary Views – Teradata 14.0

DBC.AllTempTables[V][X]

User Type	Columns Selected
Database Administrator End User	HostNo SessionNo UserName B_DatabaseName B_TableName E_TableId

Data Dictionary Views – Teradata 14.0

DBC.All_RI_Children[V][X]

User Type	Columns Selected
All Users	IndexID IndexName ChildDB ChildTable ChildKeyColumn ParentDB ParentTable ParentKeyColumn InconsistencyFlag CreatorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.All_RI_Parents[V][X]

User Type	Columns Selected
All Users	IndexID IndexName ParentDB ParentTable ParentKeyColumn ChildDB ChildTable ChildKeyColumn InconsistencyFlag CreatorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.AMPUsage[V][X]

User Type	Columns Selected
Database Administrator	AccountName UserName CPUTime DiskIO CPUTimeNorm Vproc VprocType Model

Data Dictionary Views – Teradata 14.0

DBC.ArchiveLoggingObjsV[X]

User Type	Columns Selected
Database Administrator	DatabaseName TVMName LogLevel CreatorName CreateTimeStamp

DBC.Association[V][X]

User Type	Columns Selected
Operations Control	DatabaseName TableName EventNum Original_DataBaseName Original_TableName Original_TableKind Original_Version Original_ProtectionType Original_JournalFlag Original_CreatorName Original_CommentString

Data Dictionary Views – Teradata 14.0

DBC.Authorizations[V][X]

User Type	Columns Selected
Database Administrator	DatabaseName AuthorizationName AuthorizationID TableKind Version AuthorizationType AuthorizationSubType OSDomainName OSUserName

Data Dictionary Views – Teradata 14.0

DBC.BusinessCalendarExceptions (14.0)

User Type	Columns Selected
Database Administrator	CalendarName ExceptionIndicator ExceptionDate ExceptionReason CreatorName CreationTime

DBC.BusinessCalendarPatterns (14.0)

User Type	Columns Selected
Database Administrator	CalendarName DayName Pattern PatternComment CreatorName LastModified

Data Dictionary Views – Teradata 14.0

DBC.CharSets[V]

User Type	Columns Selected
End User	CharSetName

DBC.CharTranslations[V]

User Type	Columns Selected
End User	CharSetName CharSetId InstallFlag E2I E2IUp I2E I2EUp

DBC.Children[V][X]

User Type	Columns Selected
Supervisor	Child Parent

Data Dictionary Views – Teradata 14.0

DBC.Collations[V]

User Type	Columns Selected
End User	CollName CollInstall CollEqvClass CollOrderCS CollOrderUC

Data Dictionary Views – Teradata 14.0

DBC.Columns[V][X]

User Type	Columns Selected
DB Administrator End User	DatabaseName TableName ColumnName ColumnFormat ColumnTitle ColumnType ColumnUDTName ColumnLength DefaultValue Nullable CommentString DecimalTotalDigits DecimalFractionalDigits ColumnId UpperCaseFlag Compressible CompressValue ColumnConstraint ConstraintCount ConstraintId (14.0)

DBC.Columns[V][X] (cont.)

User Type	Columns Selected
DB Administrator End User	CreatorName CreateTimeStamp LastAlterName LastAlterTimeStamp CharType IdColType AccessCount LastAccessTimeStamp CompressValueList TimeDimension VTCheckType TTCheckType PartitioningColumn (14.0) ColumnPartitionNumber (14.0) ColumnPartitionFormat (14.0) ColumnPartitionAC (14.0) ArrayColNumberOfDimensions (14.0) ArrayColScope (14.0) ArrayColElementType (14.0) ArrayColElementUdtName (14.0)

Data Dictionary Views – Teradata 14.0

DBC.ColumnStats[V]

User Type	Columns Selected
All Users	DatabaseName TableName ColumnName StatsName (14.0) StatsSource (14.0) DBSVersion (14.0) IndexNumber (14.0) SampleSignature (14.0) SampleSizePct (14.0) ThresholdSignature (14.0) RowCount (14.0) UniqueValueCount (14.0) NullCount (14.0) AllNullCount (14.0) HighModeFreq (14.0) TimeStamp (14.0) LastAlterTimeStamp (14.0)

DBC.ConnectRules[V][X]

User Type	Columns Selected
Security Administrator	TrustUser ProxyUser ProxyUserType GrantStatus WithoutRole ProxyRole1 ProxyRole2 ProxyRole3 ProxyRole4 ProxyRole5 ProxyRole6 ProxyRole7 ProxyRole8 ProxyRole9 ProxyRole10 ProxyRole11 ProxyRole12 ProxyRole13 ProxyRole14 ProxyRole15 CreatorName CreateTimeStamp TrustOnly (14.0)

Data Dictionary Views – Teradata 14.0

DBC.ConstraintFunctionsV (14.0)

User Type	Columns Selected
Security Administrator	ConstraintName Action DatabaseName FunctionName

DBC.ConstraintValuesV (14.0)

User Type	Columns Selected
Security Administrator	ConstraintName ValueName ValueConstant ValueisBitPos

Data Dictionary Views – Teradata 14.0

DBC.CostProfiles_V

User Type	Columns Selected
All Users	ProfileTypeName ProfileName ProfileCat ProfileDesc

DBC.CostProfileTypes_V

User Type	Columns Selected
All Users	ProfileTypeName ProfileTypeDesc

Data Dictionary Views – Teradata 14.0

DBC.CostProfileValues_V

User Type	Columns Selected
All Users	ProfileName ProfileId ConstName ConstId ConstCat ConstVal ConstDesc

DBC.CSPSessionInfo[V]

User Type	Columns Selected
Operations Control	SessionNo HostNo StartMBox LogonSource

Data Dictionary Views – Teradata 14.0

DBC.Databases[V][X]

User Type	Columns Selected
Database Administrator End User	DatabaseName CreatorName OwnerName AccountName ProtectionType JournalFlag PermSpace SpoolSpace TempSpace CommentString CreateTimeStamp LastAlterName LastAlterTimeStamp DBKind AccessCount LastAccessTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.Databases2[V][X]

User Type	Columns Selected
Database Administrator	DatabaseName DatabaseId UnResolvedRICount

DBC.Database_Default_Journals[V][X]

User Type	Columns Selected
Database Administrator End User	DatabaseName Journal_DB JournalName

DBC.DBCInfo[V]

User Type	Columns Selected
All Users	InfoKey InfoData

Data Dictionary Views – Teradata 14.0

DBC.DBQLRules[V]

User Type	Columns Selected
All Users	UserName AccountString ApplName TypeofRule ExplainFlag ObjFlag SqlFlag StepFlag XMLPlanFlag SummaryFlag ThresholdFlag TextSizeLimit SummaryVal1 SummaryVal2 SummaryVal3 TypeofCriterion

Data Dictionary Views – Teradata 14.0

DBC.DeleteAccessLog[V][X]

User Type	Columns Selected
Security Administrator	LogDate LogTime

DBC.DeleteOldInDoubt[V]

User Type	Columns Selected
Database Administrator	LogicalHostId CoordTaskId LogonUserName CommitOrRollback CompletionDate UserLogonTime SessionNumber RunUnitId ResolvingUserLogonName UserLogonDate CompletionTime Options

Data Dictionary Views – Teradata 14.0

DBC.DiskSpace[V][X]

User Type	Columns Selected
Database Administrator End User Supervisor	Vproc DatabaseName AccountName MaxPerm MaxSpool MaxTemp CurrentPerm CurrentSpool CurrentTemp PeakPerm PeakSpool PeakTemp MaxProfileSpool MaxProfileTemp

Data Dictionary Views – Teradata 14.0

DBC.ErrorTblsV[X]

User Type	Columns Selected
Operations Control	LogicalHostId ErrTblDbName ErrTblName BaseTblDbName BaseTblName CreatorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.Events[V][X]

User Type	Columns Selected
Operations Control	CreateDate CreateTime EventNum EventType UserName DatabaseName ObjectType AllAMPsFlag RestartSeqNum OperationInProcess TableName CheckpointName LinkingEventNum DataSetName LockMode JournalUsed JournalSaved IndexPresent DupeDumpSet

Data Dictionary Views – Teradata 14.0

DBC.Events_Configuration[V][X]

User Type	Columns Selected
Operations Control	CreateDate CreateTime EventNum EventType UserName LogProcessor PhyProcessor Vproc ProcessorState RestartSeqNum

Data Dictionary Views – Teradata 14.0

DBC.Events_Media[V][X]

User Type	Columns Selected
Operations Control	CreateDate CreateTime EventNum EventType UserName DataSetName VolSerialId VolSequenceNum DupeDumpSet

DBC.ExportWidthV (14.0)

User Type	Columns Selected
All Users	ExportDefinitionName ExportWidthRuleSet

Data Dictionary Views – Teradata 14.0

DBC.ExternalSPs[V][X]

User Type	Columns Selected
Database Administrator	DatabaseName ExternalProcedureName ExternalProcedureID NumParameters ExternalName SrcFileLanguage NoSQLDataAccess ParameterStyle ExecProtectionMode ExtFileReference CharacterType Platform RoutineKind ParameterUDTIDs AuthIDUsed AppCategory GLOPSetDatabaseName GLOPSetMemberName

Data Dictionary Views – Teradata 14.0

DBC.Functions[V][X]

User Type	Columns Selected
Database Administrator	DatabaseName FunctionName SpecificName FunctionId NumParameters ParameterDataTypes FunctionType ExternalName SrcFileLanguage NoSQLDataAccess ParameterStyle DeterministicOpt NullCall PrepareCount ExecProtectionMode ExtFileReference CharacterType PlatformInterimFldSize RoutineKind ParameterUDTIds MaxOutParameters GLOPSetDatabaseName GLOPSetMemberName

Data Dictionary Views – Teradata 14.0

DBC.HostsInfo[V]

User Type	Columns Selected
Database Administrator End User	LogicalHostId HostName DefaultCharSet

DBC. IndexConstraints[V]

User Type	Columns Selected
Database Administrator End User	DatabaseName TableName IndexName IndexNumber ConstraintType ConstraintText ConstraintCollation CollationName CreatorName CreateTimestamp CharSetID SessionMode ResolvedCurrent_Date ResolvedCurrent_TimeStamp

Note:
The DBC.PartitioningConstraints views are the preferred views for information about partitioning constraints.

Data Dictionary Views – Teradata 14.0

DBC.IndexStats[V]

User Type	Columns Selected
Database Administrator End User	DatabaseName TableName ColumnName StatsName (14.0) StatsSource (14.0) DBSVersion (14.0) IndexNumber (14.0) SampleSignature (14.0) SampleSizePct (14.0) ThresholdSignature (14.0) RowCount (14.0) UniqueValueCount (14.0) NullCount (14.0) AllNullCount (14.0) HighModeFreq (14.0) CollectTimeStamp LastAlterTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.Indices[V][X]

User Type	Columns Selected
Database Administrator End User	DatabaseName TableName IndexNumber IndexType UniqueFlag IndexName ColumnName ColumnPosition CreatorName CreateTimeStamp LastAlterName LastAlterTimeStamp IndexMode IndexMode AccessCount LastAccessTimeStamp UniqueOrPK VTConstraintType TTConstraintType SystemDefinedJI

Data Dictionary Views – Teradata 14.0

DBC.InDoubtLog[V]

User Type	Columns Selected
Database Administrator	LogicalHostId CoordTaskId LogonUserName UserLogonDate CompletionDate CommitOrRollBack SessionNumber RunUnitId ResolvingUserLogonName UserLogonTime CompletionTime Options

DBC.JoinIndexesV

User Type	Columns Selected
Database Administrator	DatabaseName TableName JoinIdxDatabaseName JoinIdxName IndexType

Data Dictionary Views – Teradata 14.0

DBC.Journals[V][X]

User Type	Columns Selected
Database Administrator End User	Tables_DB TableName Journals_DB JournalName

Data Dictionary Views – Teradata 14.0

DBC.LogOnOff[V][X]

User Type	Columns Selected
Database Administrator Security Administrator Supervisor	LogDate LogTime UserName AccountName Event LogicalHostId IFPNo SessionNo LogonDate LogonTime LogonSource ClientODBCDriverVersion (14.0) ClientNetDataProviderVersion (14.0) ClientODBCDriverManagerVersion (14.0) ClientNetFrameworkVersion (14.0) ClientAttributesEx (14.0) ClientJDBCDriverVersion (14.0) ClientJavaVersion (14.0) RecoverableNetworkProtocol (14.0) LogonRedrive (14.0)

Data Dictionary Views – Teradata 14.0

DBC.LogonRules[V]

User Type	Columns Selected
Security Administrator	UserName LogicalHostId LogonStatus NullPassword CreatorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.MultiColumnStatsV

User Type	Columns Selected
Database Administrator End User	DatabaseName TableName ColumnName StatsName (14.0) StatsSource (14.0) DBSVersion (14.0) IndexNumber (14.0) SampleSignature (14.0) SampleSizePct (14.0) ThresholdSignature (14.0) RowCount (14.0) UniqueValueCount (14.0) NullCount (14.0) AllNullCount (14.0) HighModeFreq (14.0) CollectTimeStamp (14.0) LastAlterTimeStamp (14.0)

Data Dictionary Views – Teradata 14.0

DBC.PartitioningConstraintsV[X] (14.0)

User Type	Columns Selected
Database Administrator End User	DatabaseName TableName IndexName IndexNumber ConstraintType ConstraintText ConstraintCollation CollationName CreatorName CreateTimeStamp CharSetID SessionMode ResolvedCurrent_Date ResolvedCurrent_TimeStamp DefinedCombinedPartitions MaxCombinedPartitions PartitioningLevels ColumnPartitioningLevel

Data Dictionary Views – Teradata 14.0

DBC.ProfileAsgdSecConstraintsV[X] (14.0)

User Type	Columns Selected
All Users	ProfileName ConstraintName ValueName IsDefault Assignor

Data Dictionary Views – Teradata 14.0

DBC.ProfileInfo[V][X]

User Type	Columns Selected
Database Administrator End User Security Administrator Supervisor	ProfileName DefaultAccount DefaultDB SpoolSpace TempSpace ExpirePassword PasswordMinChar PassordMaxChar PasswordDigits PasswordSpecChar PasswordRestrictWords MaxLogonAttempts LockedUserExpire PasswordReuse CommentString CreatorName CreateTimeStamp LastAlterName LastAlterTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.QryLog[V]

User Type	Columns Selected
Database Administrator	ProcID CollectTimeStamp QueryID UserID UserName DefaultDatabase AcctString ExpandAccString SessionID LogicalHostID RequestNum InternalRequestNum LogonDateTime AccStringTime AccStringHour AccStringDate AppID ClientID ClientAddress QueryBand ProfileID StartTime FirstStepTime FirstRespTime ElapsedTime NumSteps NumStepswPar

DBC.QryLog[V] (cont.)

User Type	Columns Selected
Database Administrator	MaxStepsInPar MaxStepsInPar NumResultRows TotalIOCount AMPCPUTime ParserCPUTime UtilityRowCount ErrorCode ErrorText WarningOnly AbortFlag CacheFlag StatementType StatementGroup QueryText NumOfActiveAMPs MaxAMPCPUTime MaxCPUAmpNumber MinAmpCPUTime MaxAmpIO MaxIOAmpNumber MinAmpIO SpoolUsage LSN EstResultRows EstProcTime EstMaxRowCount

Data Dictionary Views – Teradata 14.0

DBC.QryLog[V] (cont.)

User Type	Columns Selected
Database Administrator	AMPCPUTimeNorm ParserCPUTimeNorm MaxAMPCPUTimeNorm MaxCPUAmpNumberNorm MinAmpCPUTimeNorm ParExpreqTime ProxyUser ProxyRole SessionTemporalQualifer CalendarName (14.0) CPUDecayLevel (14.0) IODecayLevel (14.0) TacticalCPUException (14.0) TacticalIOException (14.0) SeqRespTime (14.0) ReqIOKB (14.0) ReqPhysIO (14.0) ReqPhysIOKB (14.0) DataCollectAlg (14.0) CallNestingLevel (14.0) NumRequestCtx (14.0) KeepFlag (14.0) QueryRedriven (14.0) ReDriveKind (14.0)

Data Dictionary Views – Teradata 14.0

DBC.QryLogEventsHis[V]

User Type	Columns Selected
Database Administrator	ProcID CollectTimeStamp EntryTS EntryKind EntryID EntryName EventValue LastValue Activity ActivityId ActivityName ConfigId SeqNo Spare1 Spare2 Spare3

Data Dictionary Views – Teradata 14.0

DBC.QryLogEvents[V]

User Type	Columns Selected
Database Administrator	ProcID CollectTimeStamp SessionID LogicalHostID WDID OpEnvID SysConID EventTime EventCode EventSubCode EventInfo

Data Dictionary Views – Teradata 14.0

DBC.QryLogExceptions[V]

User Type	Columns Selected
Database Administrator	ProcID CollectTimeStamp Query ID UserName SessionID RequestNum LogicalHostID AcctString WDID OpEnvID SysConID ClassificationTime ExceptionTime ExceptionValue ExceptionAction NewWDID ExceptionCode ExceptionSubCode ErrorText ExtraInfo RuleID WarningOnly RejectionCat

Data Dictionary Views – Teradata 14.0

DBC.QryLogExplain[V]

User Type	Columns Selected
Database Administrator	ProcID CollectTimeStamp Query ID ExpRowNo ExplainText

Data Dictionary Views – Teradata 14.0

DBC.QryLogObjects[V]

User Type	Columns Selected
Database Administrator	ProcID CollectTimeStamp QueryID ObjectDatabaseName ObjectTableName ObjectColumnName ObjectID ObjectNum ObjectType FreqofUse TypeofUse

DBC.QryLogSQL[V]

User Type	Columns Selected
Database Administrator	ProcID CollectTimeStamp Query ID SqlRowNo SqlTextInfo

Data Dictionary Views – Teradata 14.0

DBC.QryLogSteps[V]

User Type	Columns Selected
DB Administrator	ProcID CollectTimestamp QueryID StepLev1Num StepLev2Num StepName StepStartTime StepStopTime ElapsedTime EstProcTime EstCPUCost CPUtime IOcount EstRowCount RowCount RowCount2 NumOfActiveAMPs MaxAmpCPUTime MaxCPUAmpNumber MinAmpCPUTime MaxAmpIO MaxIOAmpNumber

DBC.QryLogSteps[V] (cont.)

User Type	Columns Selected
DB Administrator	MinAmpIO LastRespTime SpoolUsage MaxAMPSpool MaxSpoolAmpNumber MinAMPSpool StepWD LSN UtilityTableId RowsWComprColumns EstIOCost EstNetCost EstHRCost CPUTimeNorm MaxAmpCPUTimeNorm MaxCPUAmpNumberNorm MinAmpCPUTimeNorm NumCombinedPartitions (14.0) NumContexts (14.0) NumCPReferences (14.0)

Data Dictionary Views – Teradata 14.0

DBC.QryLogSummary[V]

User Type	Columns Selected
Database Administrator	ProcID CollectTimeStamp UserID AcctString LogicalHostID AppID ClientID ClientAddr ProfileID SessionID QueryCount ValueType QuerySeconds AverageTime TotalIOCount AverageIO AMPCPUTime AverageAmpCPU

DBC.QryLogSummary[V] (cont.)

User Type	Columns Selected
DB Administrator	ParserCPUTime AverageParserCPU AMPCPUTimeNorm AverageAmpCPUNorm ParserCPUTimeNorm AverageParserCPUNorm LowHist HighHist

Data Dictionary Views – Teradata 14.0

DBC.QryLogTDWM[V]

User Type	Columns Selected
DB Administrator	ProcID CollectTimeStamp QueryID UserID UserName DefaultDatabase AcctString LastStateChange DelayTime WDDelayTime WDID OpEnvID SysConID LSN NoClassification WDOverride SLGMet ExceptionValue FinalWDID

DBC.QryLogTDWM[V] (cont.)

User Type	Columns Selected
DB Administrator	TDWMEstMaxRows TDWMEstLastRows TDWMEstTotalTime TDWMAllAmpFlag TDWMConfLevelUsed StatementGroup SessionWDID

Data Dictionary Views – Teradata 14.0

DBC.QryLogTDWMSum[V]

User Type	Columns Selected
Security Administrator	ProcID CollectTimeStamp WDID OpEnvID SysConID StartColTime Arrivals ActiveCount Completions MinRespTime MaxRespTime AvgRespTime MinCPUTime MaxCPUTime AvgCPUTime DelayedCount AvgDelayTime ExceptionAbCount ExceptionMvCount ExceptionCoCount ExceptionCount AbortCount ErrorCount

DBC.QryLogTDWMSum[V] (cont.)

User Type	Columns Selected
Security Administrator	MetSLGCount MetSLGCount RejectedCount MovedInCount IntervalDelayCnt DelayedQueries

Data Dictionary Views – Teradata 14.0

DBC.QryLogXMLV

User Type	Columns Selected
Database Administrator	ProdID CollectTimeStamp QueryID XMLRowNo XMLTextInfo

DBC.RCC_Configuration[V][X]

User Type	Columns Selected
Operations Control	EventNum LogProcessor PhyProcessor ProcessorState RestartSeqNum Vproc

DBC.RCC_Media[V][X]

User Type	Columns Selected
Operations Control	EventNum VolSerialId VolSequenceNum DupeDumpSet

Data Dictionary Views – Teradata 14.0

DBC.ReconfigDeleteOrderV

User Type	Columns Selected
All Users	EventNum OrderNumber DatabaseName TableName CheckTableOption ProcessOffline

DBC.ReconfigInfoV

User Type	Columns Selected
All Users	ReconfigId Description ReconfigType BeginTimeStamp EndTimeStamp BeginRedistTimeStamp EndRedistTimeStamp BeginDelTimeStamp EndDelTimeStamp Status CurrByteCount CurrTabRedistCount CurrTabDeleteCount EstRemainRedistSecs EstRemainDeleteSecs AddAmpCount DelAmpCount MovAmpCount ModAmpCount NodeCount TotTaskCount TotTableCount TotByteCount TotCatchUpByteCount TotJournalByteCount ActualRedistSecs ActualDeleteSecs

DBC.ReconfigInfoV (cont.)

User Type	Columns Selected
All Users	MetSLGCount EstRedistSecs EstDeleteSecs BeginCalcHBTimeStamp EndCalcHBTimeStamp BeginWrSpaceTimeStamp EndWrSpaceTimeStamp BeginPHBNewTimeStamp EndPHBNewTimeStamp BeginFBHBNewTimeStamp EndFBHBNewTimeStamp BeginWrPHBTimeStamp EndWrPHBTimeStamp BeginWrFBHBTimeStamp EndWrFBHBTimeStamp BeginWrCfgTimeStamp EndWrCfgTimeStamp BeginWrCfgNewTimeStamp EndWrCfgNewTimeStamp BeginWrBkupIdTimeStamp EndWrBkupIdTimeStamp BeginWrBMTimeStamp EndWrBMTimeStamp BeginDelHBNewTimeStamp EndDelHBNewTimeStamp BeginVProcCfgTimeStamp EndVProcCfgTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.ReconfigRedistOrderV

User Type	Columns Selected
All Users	OrderNumber DatabaseName TableName ProcessOffline

Data Dictionary Views – Teradata 14.0

DBC.ReconfigTableStatsV

User Type	Columns Selected
All Users	DatabaseName TableName ReconfigId Phase Status BeginTimeStamp EndTimeStamp TotRowCount TotByteCount TotCPUSecs TotIOCount LowRowCount LowRowCountAmp HighRowCount HighRowCountAmp LowByteCount LowByteCountAmp HighByteCount HighByteCountAmp LowCPUSecsCount LowCPUSecsCountAmp HighCPUSecsCount

DBC.ReconfigTableStatsV (cont.)

User Type	Columns Selected
All Users	HighCPUSecsCountAmp LowIOCount LowIOCountAmp HighIOCount HighIOCountAmp MetSLGCount ActualRedistSecs ActualDeleteSecs EstRedistSecs EstDeleteSecs FSGIOCount FSysReadCount FSysWriteCount FSysMiscCount MsgRcvCount MsgSendCount MsgMiscCount MsgWaitRcvTime MsgWaitSendTime MsgWaitMiscTime NoMemFlushCount CkptFlushCount

Data Dictionary Views – Teradata 14.0

DBC.RepCaptureRulesV

User Type	Columns Selected
All users	GroupName RuleSetName GroupName ObjectKind DefaultOpt LikePattern LikeEscape NotLikePattern NotLikeEscape

DBC.RepTables[V][X]

User Type	Columns Selected
All users	GroupName TableName

Data Dictionary Views – Teradata 14.0

DBC.ResolvedDTSV[X] (14.0)

User Type	Columns Selected
All users	ResolvedCurrent_Date ResolvedCurrent_TimeStamp TableName DatabaseName

DBC.RestrictedWords[V]

User Type	Columns Selected
All users	RestrictedWord

Data Dictionary Views – Teradata 14.0

DBC.RI_Child_Tables[V][X]

User Type	Columns Selected
All users	IndexID IndexName ChildDbID ChildTID ChildKeyFID ParentDbID ParentTID ParentKeyFID InconsistencyFlag CreatorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.RI_Distinct_Children[V][X]

User Type	Columns Selected
All users	IndexID IndexName ChildDB ChildTable ParentDB ParentTable InconsistencyFlag CreatorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.RI_Distinct_Parents[V][X]

User Type	Columns Selected
All users	IndexID IndexName ParentDB ParentTable ChildDB ChildTable InconsistencyFlag CreatorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.RI_Parent_Tables[V][X]

User Type	Columns Selected
All users	IndexID IndexName ParentDbID ParentTID ParentKeyFID ChildDbID ChildTID ChildKeyFID InconsistencyFlag CreatorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.RoleInfo[V][X]

User Type	Columns Selected
Database Administrator End User	RoleName CreatorName CommentString CreateTimeStamp ExtRole

DBC.RoleMembers[V][X]

User Type	Columns Selected
Database Administrator End User	RoleName Grantee GranteeKind Grantor WhenGranted DefaultRole WithAdmin

Data Dictionary Views – Teradata 14.0

DBC.SecConstraintsV[X] (14.0)

User Type	Columns Selected
All Users	ConstraintName DataType Nullable SizeInBytes AssigneeCount Creator CreateTimeStamp

DBC.SecurityDefaults[V]

User Type	Columns Selected
Security Administrator	ExpirePassword PasswordMinChar PasswordMaxChar PasswordDigits PasswordSpecChar PasswordRestrictWords MaxLogonAttempts LockedUserExpire PasswordReuse

DBC.SecurityLog[V][X] Note: Older view – use AccessLogV

User Type	Columns Selected
Security Administrator	LogDate LogTime LogType UserName AccountName DatabaseName TableName Text

Data Dictionary Views – Teradata 14.0

DBC.SessionInfo[V][X]

User Type	Columns Selected
Database Administrator End User Security Administrator Supervisor	UserName AccountName SessionNo DefaultDataBase IFPNo Partition LogicalHostId HostNo CurrentCollation LogonDate LogonTime LogonSequenceNo LogonSource ExpiredPassword TwoPCMode Transaction_Mode CurrentRole ProfileName LogonAcct LDAP AuditTrailID CurIsolationLevel QueryBand ProxyUser ProxyCurRole

DBC.SessionInfo[V][X] (cont.)

User Type	Columns Selected (14.0)
Database Administrator End User Security Administrator Supervisor	TemporalQualifier CalendarName ExtendedLogonSource ClientIpAddress ClientProgramName ClientSystemUserId ClientConnectionType ClientCoordName ClientEnvName ClientJobId ClientJobName ClientOsName ClientProcThreadId ClientSecProdGrp ClientSecProdUserId ClientTcpPortNumber ClientTdHostName ClientTerminalId ClientTransactionId ClientUserOperId ClientVmName ClientVmUserId MechanismName ClientTDPReleaseId ClientCLIv2ReleaseId

Data Dictionary Views – Teradata 14.0

DBC.SessionInfo[V][X] (cont.)

User Type	Columns Selected (14.0)
Database Administrator End User Security Administrator Supervisor	ClientSessionDesc ClientWorkload ClientJobData ClientODBCDriverVersion ClientNetDataProviderVersion ClientODBCDriverManagerVersion ClientNetFrameworkVersion ClientAttributesEx ClientJDBCDriverVersion ClientJavaVersion ExportDefinitionName ExportWidthRuleSet RecoverableNetworkProtocol LogonRedrive

Data Dictionary Views – Teradata 14.0

DBC.ShowColChecks[V][X]

User Type	Columns Selected
Database Administrator End User	DatabaseName TableName ColumnName ColCheck CreatorName CreateTimeStamp

DBC.ShowTblChecks[V][X]

User Type	Columns Selected
Database Administrator End User	DatabaseName TableName CheckName TblCheck CreatorName CreateTimeStamp

DBC.Software_Event_Log[V]

User Type	Columns Selected
Operations Control	TheDate TheTime Event_Tag Category Severity PMA Vproc Partition Task TheFunction Function SW_Version Line Text

Data Dictionary Views – Teradata 14.0

DBC.Table_LevelConstraints[V][X]

User Type	Columns Selected
Database Administrator End User	DatabaseName TableName ConstraintName ConstraintText CreatorName CreateTimeStamp VTCheckType TTCheckType

Data Dictionary Views – Teradata 14.0

DBC.Tables[V][X]

User Type	Columns Selected
DB Administrator End User	DatabaseName TableName Version TableKind ProtectionType JournalFlag CreatorName RequestText CommentString ParentCount ChildCount NamedTblCheckCount UnnamedTblCheckExist PrimaryKeyIndexId RepStatus CreateTimeStamp LastAlterName LastAlterTimeStamp RequestTxtOverFlow AccessCount LastAccessTimeStamp

DBC.Tables[V][X]

User Type	Columns Selected
DB Administrator End User	UtilVersion QueueFlag CommitOpt TransLog CheckOpt TemporalProperty ResolvedCurrent_Date ResolvedCurrent_Timestamp SystemDefinedJI VTQualifier TTQualifier PartitioningLevels (14.0) PIColumnCount (14.0)

Data Dictionary Views – Teradata 14.0

DBC.Tables2[V][X]

User Type	Columns Selected
Database Administrator Supervisor	TVMName TVMId DatabaseId ParentCount ChildCount

DBC.Tables3VX

User Type	Columns Selected
Database Administrator Supervisor	DatabaseName TableName FieldName TableId FieldId

Data Dictionary Views – Teradata 14.0

DBC.TableSize[V][X]

User Type	Columns Selected
Database Administrator End User	Vproc DatabaseName AccountName TableName CurrentPerm PeakPerm

DBC.TableText[V][X]

User Type	Columns Selected
Database Administrator End User	DatabaseName TableName TableKind RequestText LineNo

Data Dictionary Views – Teradata 14.0

DBC.TempTableStatsV (14.0)

User Type	Columns Selected
Database Administrator End User	DatabaseName TableName ColumnName StatsName StatsSource ValidQuery DBSVersion IndexNumber SampleSignature SampleSizePct ThresholdSignature MaxIntervals MaxValueLength RowCount UniqueValueCount PNullUniqueValueCount NullCount AllNullCount HighModeFreq PNullHighModeFreq CreateTimeStamp LastTimeStamp LastAlterTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.Triggers[V][X]

User Type	Columns Selected
Database Administrator	DatabaseName SubjectTableDataBaseName TableName TriggerName EnabledFlag ActionTime Event Kind OrderNumber TriggerComment RequestText CreatorName CreateTimeStamp LastAlterName LastAlterTimeStamp AccessCount LastAccessTimeStamp CreateTxtOverflow VTEventType TTEventType

Data Dictionary Views – Teradata 14.0

DBC.User_Default_Journals[V][X]

User Type	Columns Selected
End User	UserName Journal_DB JournalName

DBC.UsrAsgdSecConstraintsV[X] (14.0)

User Type	Columns Selected
All Users	UserName ConstraintName ValueName IsDefault Assignor

Data Dictionary Views – Teradata 14.0

DBC.UserGrantedRights[V]

User Type	Columns Selected
End User	DatabaseName TableName ColumnName Grantee GrantAuthority AccessRight AllnessFlag CreatorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.UserRights[V]

User Type	Columns Selected
End User	DatabaseName TableName ColumnName AccessRight GrantAuthority GrantorName CreatorName CreateTimeStamp

DBC.UserRoleRights[V]

User Type	Columns Selected
End User	RoleName DatabaseName TableName ColumnName AccessRight GrantorName CreateTimeStamp

Data Dictionary Views – Teradata 14.0

DBC.Users[V]

User Type	Columns Selected
DB Administrator End User Supervisor	UserName CreatorName PasswordLastModDate PasswordLasModTIme OwnerName PermSpace SpoolSpace TempSpace ProtectionType JournalFlag StartupString DefaultAccount DefaultDataBase CommentString DefaultCollation PasswordChgDate LockedDate LockedTime LockedCount

DBC.Users[V] (cont.)

User Type	Columns Selected
DB Administrator End User Supervisor	TimeZoneHour TimeZoneMinute DefaultDateForm CreateTimeStamp LastAlterTime LastAlterTimeStamp DefaultCharType RoleName ProfileName AccessCount LastAccessTimeStamp

TERADATA
Raising Intelligence

Appendix E: Solutions to Lab Exercises

This Appendix contains possible solutions to the lab exercises.

Notes

Lab Solutions for Lab 15-1

Lab Exercise 15-1 Solutions

7. **Using the Query window, execute the following query.**

```
INSERT INTO Old_Orders SELECT * FROM DS.Orders
WHERE o_orderdate BETWEEN '2008-07-01' AND '2008-09-30';
```

   Use the "Format Query" option to format the query.

   How many rows are in the Old_Orders table? ***1200***

8. **Using the History window, recall the query from step #7 and modify it to add orders from '2006-10-01' through '2006-12-31'.**

   How many rows are in the Old_Orders table? ***2400***

9. **Execute the following query by using the drag and drop object feature of SQL Assistant.**

```
SELECT custid, SUM (totalprice)
FROM    Old_Orders
GROUP BY 1
ORDER  BY 1;
```

   Use the "Add Totals" feature to automatically generate a total sum for all of the orders.

   What is the sum of the orders using this feature? ***3,065,969.88***

Lab Solutions for Lab 15-1 (cont.)

11. **Using the Query Builder feature, create a view named "Old_Orders_v" for the Old_Orders table that includes the following columns and only includes orders for December, 2008.**

    orderid, custid, totalprice, orderdate

```
CREATE VIEW Old_Orders_v AS
    SELECT orderid, custid, totalprice, orderdate FROM Old_Orders
    WHERE orderdate BETWEEN '2008-12-01' AND '2008-12-31';
```

    SELECT all of the rows from the view named "Old_Orders_v".

    How many rows are displayed from this view? ***400***

12. **Using the Query Builder feature, create a simple macro named "Old_Orders_m" which selects all of the orders from the view named "Old_Orders_v".**

```
CREATE MACRO Old_Orders_m
AS (SELECT * From Old_Orders_v; );
```

    Execute the macro "Old_Orders_m".

```
EXEC Old_Orders_m;
```

    What is the sum of the orders for December using this "Add Totals" feature? ***517,341.92***

Lab Solutions for Lab 16-1

Lab Exercise 16-1 Solutions

Purpose

In this lab, you will use Teradata SQL Assistant to evaluate various columns of table as primary index candidates.

What you need

Populated PD.Employee table; your empty Employee table

Tasks

1. INSERT/SELECT all rows from the populated PD.Employee table to your "Employee" table. Verify the number of rows in your table.

   INSERT INTO Employee SELECT * FROM PD.Employee;

   SELECT COUNT(*) FROM Employee; Count = _____

   INSERT INTO Employee SELECT * FROM PD.Employee;
   SELECT COUNT(*) FROM Employee; Count = 1000

Lab Solutions for Lab 16-1 (cont.)

2. Collect column demographics for each of these columns in Employee and determine if the column would be a primary index candidate or not.

   By using the SHOW TABLE Employee command, you should be able to complete the Employee_number information without executing any SQL.

	Distinct Values	Max Rows for a Value	Max Rows NULL	Avg Rows per Value	Candidate for PI (Y/N)
Employee_Number	1000	1	0	1	Yes
Dept_Number	61	40	0	16	No
Job_Code	52	97	0	19	No
Last_name	464	42	0	2	Yes?

```
SELECT COUNT(DISTINCT(Dept_number))  FROM Employee;
SELECT Dept_number, COUNT(*)         FROM Employee GROUP BY 1 ORDER BY 2 DESC;
SELECT COUNT(*)                      FROM Employee WHERE Dept_number IS NULL;
SELECT COUNT(*) / COUNT(DISTINCT(Dept_number)) FROM Employee;
```

Lab Solutions for Lab 16-2

Lab Exercise 16-2 Solutions

Tasks

1. Use SHOW TABLE command to determine which column is the Primary Index. PI = _____

```
SHOW TABLE Employee;
    CREATE SET TABLE Student130.Employee , FALLBACK,
        NO BEFORE JOURNAL,
        NO AFTER JOURNAL,
        CHECKSUM = DEFAULT,
        DEFAULT MERGEBLOCKRATIO
            ( employee_number      INTEGER NOT NULL,
              dept_number          INTEGER,
              emp_mgr_number       INTEGER,
              job_code             INTEGER,
              last_name            CHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
              first_name           VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
              salary_amount        DECIMAL(10,2))
    UNIQUE PRIMARY INDEX (employee_number);


    Primary Index = Employee_number
```

Lab Solutions for Lab 16-2 (cont.)

1. (cont.) Determine the AMP space usage of your Employee table using DBC.TableSizeV.

```
SELECT    Vproc, TableName (CHAR(15)), CurrentPerm
FROM      DBC.TableSizeV
WHERE     DatabaseName = DATABASE AND TableName = 'Employee'
ORDER BY  1 ;
```

Vproc	TableName	CurrentPerm	
0	Employee	6,656	
1	Employee	7,168	
2	Employee	6,656	
3	Employee	7,168	
4	Employee	8,192	(max)
5	Employee	7,168	
6	Employee	6,656	
7	Employee	6,144	(min)
8	Employee	6,144	
9	Employee	6,144	
10	Employee	6,656	
11	Employee	6,656	
12	Employee	7,168	
13	Employee	6,656	
:	:	:	
25	Employee	7,168	

Note: This result is based on an 26-AMP system.

Lab Solutions for Lab 16-2 (cont.)

2. Create a new table named Employee_2 with the same columns as Employee except specify Last_Name as the Primary Index.

```
CREATE TABLE Employee_2, FALLBACK
    ( employee_number   INTEGER NOT NULL
     ,dept_number       INTEGER
     ,emp_mgr_number    INTEGER
     ,job_code          INTEGER
     ,last_name         CHAR(20)
     ,first_name        VARCHAR(20)
     ,salary_amount     DECIMAL(10,2) )
PRIMARY INDEX (last_name);
```

Use INSERT/SELECT to populate Employee_2 from Employee.

```
INSERT INTO Employee_2 SELECT * FROM Employee;
```

Lab Solutions for Lab 16-2 (cont.)

2. (cont.) Determine the AMP space usage of your Employee_2 table using DBC.TableSizeV.

```
SELECT    Vproc, TableName (CHAR(15)), CurrentPerm
FROM      DBC.TableSizeV
WHERE     DatabaseName = DATABASE AND TableName = 'Employee_2'
ORDER BY  1 ;
```

Vproc	TableName	CurrentPerm	
0	Employee_2	15,872	(max)
1	Employee_2	7,168	
2	Employee_2	4,608	
3	Employee_2	6,144	
4	Employee_2	3,584	(min)
5	Employee_2	7,168	
6	Employee_2	4,608	
7	Employee_2	7,680	
8	Employee_2	5,632	
9	Employee_2	9,216	
10	Employee_2	5,632	
11	Employee_2	4,608	
12	Employee_2	5,632	
13	Employee_2	15,872	
:	:	:	
25	Employee_2	5,632	

Note: This result is based on an 26-AMP system.

Lab Solutions for Lab 17-1

Lab Exercise 17-1 Solutions

1. **INSERT/SELECT all rows from the populated DS.Orders table to your "Orders" table. Verify the number of rows in your table.**

```
INSERT INTO Orders          SELECT * FROM  DS.Orders;
INSERT INTO Orders_2012     SELECT * FROM  DS.Orders_2012;
SELECT COUNT(*)             FROM Orders;                  Count =  31,200
SELECT COUNT(*)             FROM Orders_2012;             Count =  12,000
```

2. **Use the SHOW TABLE for Orders to help create a new, similar table (same column names and definitions, etc.) named "Orders_PPI" that has a PPI.**

```
CREATE TABLE Orders_PPI
  (orderid                  INTEGER NOT NULL,
   custid                   INTEGER NOT NULL,
   orderstatus              CHAR(1),
   totalprice               DECIMAL(9,2) NOT NULL,
   orderdate                DATE FORMAT 'YYYY-MM-DD' NOT NULL,
   orderpriority            SMALLINT,
   clerk                    CHAR(16),
   location                 SMALLINT,
   shippriority             SMALLINT,
   ordercomment             VARCHAR(79))
PRIMARY INDEX ( orderid )
   PARTITION BY RANGE_N (orderdate
   BETWEEN DATE '2003-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' MONTH, NO RANGE );
```

How many partitions are logically defined for the Orders_PPI table? 121

Lab Solutions for Lab 17-1 (cont.)

3. INSERT/SELECT all of the rows from your Orders table into the Orders_PPI table. Verify the number of rows in your table. Count = _____

```
INSERT INTO Orders_PPI  SELECT * FROM Orders;
SELECT COUNT(*) FROM Orders;
```

Count = 31,200

How many partitions would you estimate have data at this time? 108

4. Use the PARTITION key word to list the partitions and the number of rows in various partitions.

```
SELECT      COUNT(DISTINCT(PARTITION)) FROM  Orders_PPI;

SELECT      PARTITION, COUNT(*)
FROM        Orders_PPI
GROUP BY    1
ORDER BY    1;
```

How many partitions actually have data? 108
How many rows are in each partition for the year 2003? 100
How many rows are in each partition for the year 2011? 600

PARTITION	Count(*)
1	100
:	:
72	400
73	400
:	:
108	600

Lab Solutions for Lab 17-1 (cont.)

5. Use INSERT/SELECT to add the rows from the DS.Orders_2012 table to your Orders_PPI table. Verify the number of rows in your table. Count = _____

```
INSERT INTO Orders_PPI   SELECT * FROM Orders_2012;

SELECT COUNT(*) FROM Orders_PPI;          Count =  43,200
```

Use the PARTITION key word to determine the number of partitions used and number of rows in various partitions.

```
SELECT      COUNT(DISTINCT(PARTITION))
FROM        Orders_PPI;
```

How many partitions actually have data? 120

```
SELECT      PARTITION, COUNT(*)
FROM        Orders_PPI
WHERE       orderdate BETWEEN '2012-01-01' AND '2012-12-31'
GROUP BY    1
ORDER BY    1;
```

How many rows are in each partition for 2012? 1000

Lab Solutions for Lab 17-1 (cont.)

6. INSERT the following row (using these values) into the Orders_PPI table.

```
INSERT INTO Orders_PPI
      VALUES (100000, 1000, 'C', 1000, '2000-12-31', 10, 'EF Codd', 5, 20, 'old order');
```

How many partitions are in Orders_PPI? 121

```
SELECT  COUNT(DISTINCT(PARTITION))  FROM  Orders_PPI;
```

What is the partition number (highest partition #) of the NO RANGE partition? 121

```
SELECT  MAX(PARTITION) FROM  Orders_PPI;
      or
SELECT  PARTITION  FROM Orders_PPI  WHERE orderdate ='2000-12-31';
```

Lab Solutions for Lab 17-1 (cont.)

7. (Optional) Create a new table named "Orders_PPI_ML" with multi-level partitioning.

```
CREATE TABLE Orders_PPI_ML
      (orderid INTEGER NOT NULL,
       custid INTEGER NOT NULL,
       orderstatus CHAR(1),
       totalprice DECIMAL(9,2) NOT NULL,
       orderdate DATE FORMAT 'YYYY-MM-DD' NOT NULL,
       orderpriority SMALLINT,
       clerk CHAR(16),
       location SMALLINT,
       shippriority SMALLINT,
       ordercomment VARCHAR(79))
PRIMARY INDEX ( orderid )
       PARTITION BY (
       RANGE_N (orderdate BETWEEN DATE '2003-01-01' AND DATE '2012-12-31'
                                          EACH INTERVAL '1' MONTH , NO RANGE),
       RANGE_N (location BETWEEN 1 AND 10 EACH 1, NO RANGE OR UNKNOWN) )
UNIQUE INDEX (orderid) ;
```

8. (Optional) Populate the Orders_PPI_ML table from the Orders and Orders_2012 tables using INSERT/SELECT. Verify the number of rows in Orders_PPI_ML.

```
INSERT INTO  Orders_PPI_ML  SELECT * FROM Orders;
INSERT INTO  Orders_PPI_ML  SELECT * FROM Orders_2012;
SELECT COUNT(*) FROM Orders_PPI_ML ;
```

Count = 43,200

Lab Solutions for Lab 17-1 (cont.)

9. (Optional) For the Orders_PPI_ML table, use the PARTITION key word to answer the following questions.

   How many partitions actually have data? 1200

```
SELECT COUNT(DISTINCT(PARTITION)) FROM Orders_PPI_ML;
```

   What is the highest partition number? 1319

```
SELECT MAX(PARTITION) FROM Orders_PPI_ML;
```

   What is the partition number for orders placed from location #1 in January, 2012? 1189

```
SELECT DISTINCT(PARTITION) FROM Orders_PPI_ML
WHERE location = 1
AND orderdate BETWEEN '2012-01-01' AND '2012-01-31';
```

   What is the partition number for orders placed from location #1 in February, 2012? 1200

```
SELECT DISTINCT(PARTITION) FROM Orders_PPI_ML
WHERE location = 1
AND orderdate BETWEEN '2012-02-01' AND '2012-02-29';
```

   Is there a difference of 11 partitions between these 2 months? Yes

   Why or why not? The second level of partitioning was based on 10 locations plus the NO RANGE partition for a difference of 11.

Lab Solutions for Lab 17-1 (cont.)

10. (Optional) Before altering the table, verify the number of rows in Orders_PPI. Count = 43,201

```
SELECT COUNT(*) FROM  Orders_PPI;
```
Count = 43,201

Use the ALTER TABLE command on Orders_PPI.

```
ALTER TABLE Orders_PPI  MODIFY PRIMARY INDEX
      DROP RANGE      BETWEEN DATE '2003-01-01' AND DATE '2003-12-31'
                      EACH INTERVAL '1' MONTH
      ADD RANGE       BETWEEN DATE '2013-01-01' AND DATE '2013-12-31'
                      EACH INTERVAL '1' MONTH
      WITH DELETE;
```

How many partitions currently have data rows? 109

```
SELECT COUNT(DISTINCT(PARTITION))  FROM  Orders_PPI;
```
Count = 109

How many rows now exist in the table? 43,201 Has the row count changed? No

```
SELECT COUNT(*) FROM  Orders_PPI;
```
Count = 43,201

If the row count did not change, why not?

Because this table had NO RANGE defined, the rows from the dropped partitions are moved into the NO RANGE partition.

Lab Solutions for Lab 18-1

Lab Exercise 18-1

Purpose

In this lab, you will use Teradata SQL Assistant to create tables with column partitioning in various ways. These tables will be used in later labs.

What you need

Populated DS tables and Orders and Orders_2012 tables in your database

Tasks

1. Use the SHOW TABLE for Orders to help create a new, similar table (same column names and definitions, etc.) that does NOT have a primary index and name this table "Orders_NoPI".

```
CREATE MULTISET TABLE Orders_NoPI, FALLBACK
      (orderid  INTEGER NOT NULL,
      custid   INTEGER NOT NULL,
      orderstatus          CHAR(1),
      totalprice           DECIMAL(9,2) NOT NULL,
      orderdate            DATE FORMAT 'YYYY-MM-DD' NOT NULL,
      orderpriority        SMALLINT,
      clerk    CHAR(16),
      location  SMALLINT,
      shippriority         SMALLINT,
      ordercomment         VARCHAR(79))
NO PRIMARY INDEX;
```

Lab Solutions for Lab 18-1 (cont.)

2. Populate the Orders_NoPI table (via INSERT/SELECT) with all of the rows from the DS.Orders and DS.Orders_2012 tables.

```
INSERT INTO Orders_NoPI   SELECT * FROM  DS.Orders;
INSERT INTO Orders_NoPI   SELECT * FROM  DS.Orders_2012;
```

Verify the number of rows in your table. Count = 43,200 (count should be 43,200)

3. Using SHOW TABLE for Orders_NoPI, create a new column partitioned table named "Orders_CP".

```
CREATE MULTISET TABLE Orders_CP, FALLBACK
     (orderid                INTEGER NOT NULL,
      custid                 INTEGER NOT NULL,
      orderstatus            CHAR(1),
      totalprice             DECIMAL(9,2) NOT NULL,
      orderdate              DATE FORMAT 'YYYY-MM-DD' NOT NULL,
      orderpriority          SMALLINT,
      clerk                  CHAR(16),
      location               SMALLINT,
      shippriority           SMALLINT,
      ordercomment           VARCHAR(79))
NO PRIMARY INDEX
PARTITION BY COLUMN;
```

Populate the Orders_CP table (via INSERT/SELECT) from the Orders_NoPI table.

```
INSERT INTO Orders_CP   SELECT * FROM  Orders_NoPI;
```

Lab Solutions for Lab 18-1 (cont.)

4. Verify the number of rows in your table and fine how many partitions are defined. Count = 43,200

```
SELECT COUNT(*) FROM  Orders_CP;
```

How many partitions actually have data? 1

```
SELECT COUNT(DISTINCT(PARTITION)) FROM  Orders_CP;
```

Note: Even though each column stored in separate partitions internally, the table only has 1 logical partition.

Lab Solutions for Lab 18-1 (cont.)

5. Use the SHOW TABLE for Order_NoPI to create a new column partitioned table named "Orders_CP_NoAC" without automatic compression on each column.

```
CREATE MULTISET TABLE Orders_CP_noAC, FALLBACK
      (orderid                INTEGER NOT NULL,
       custid                 INTEGER NOT NULL,
       orderstatus            CHAR(1),
       totalprice             DECIMAL(9,2) NOT NULL,
       orderdate              DATE FORMAT 'YYYY-MM-DD' NOT NULL,
       orderpriority          SMALLINT,
       clerk                  CHAR(16),
       location               SMALLINT,
       shippriority           SMALLINT,
       ordercomment           VARCHAR(79))
NO PRIMARY INDEX  PARTITION BY COLUMN
      (orderid                NO AUTO COMPRESS,
       custid                 NO AUTO COMPRESS,
       orderstatus            NO AUTO COMPRESS,
       totalprice             NO AUTO COMPRESS,
       orderdate              NO AUTO COMPRESS,
       orderpriority          NO AUTO COMPRESS,
       clerk                  NO AUTO COMPRESS,
       location               NO AUTO COMPRESS,
       shippriority           NO AUTO COMPRESS,
       ordercomment           NO AUTO COMPRESS);
```

Populate the Orders_CP_NoAC table (via INSERT/SELECT) from the Orders_NoPI table.

```
INSERT INTO Orders_CP_noAC   SELECT * FROM  Orders_NoPI;
```

Lab Solutions for Lab 18-1 (cont.)

6. (Optional) Use the SHOW TABLE for Order_CP to create a new column partitioned table named "Orders_CP_TP based on the following:

```
CREATE MULTISET TABLE Orders_CP_TP, FALLBACK
      (orderid              INTEGER NOT NULL,
       custid               INTEGER NOT NULL,
       orderstatus          CHAR(1),
       totalprice           DECIMAL(9,2) NOT NULL,
       orderdate            DATE FORMAT 'YYYY-MM-DD' NOT NULL,
       orderpriority        SMALLINT,
       clerk                CHAR(16),
       location             SMALLINT,
       shippriority         SMALLINT,
       ordercomment         VARCHAR(79))
NO PRIMARY INDEX
PARTITION BY
(COLUMN,
 RANGE_N (orderdate
      BETWEEN DATE '2003-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' MONTH ));
```

Populate the Orders_CP_TP table (via INSERT/SELECT) from the Orders_NoPI table.

```
INSERT INTO Orders_CP_TP   SELECT * FROM  Orders_NoPI;
```

Lab Solutions for Lab 18-1 (cont.)

7. (Optional) Use the PARTITION key word to determine the number of partitions defined in the Orders_CP_TP.

   How many partitions actually have data? 120

```
SELECT COUNT(DISTINCT(PARTITION)) FROM  Orders_CP_TP;
```

8. (Optional) Determine the AMP space usage of the Orders_CP, Orders_CP_noAC, and Orders_CP_TP tables using DBC.TableSizeV.

```
SELECT    TableName (CHAR(15)), SUM(CurrentPerm)
FROM      DBC.TableSizeV
WHERE     DatabaseName = DATABASE
AND       TableName In ('Orders_CP',  'Orders_CP_noAC', 'Orders_CP_TP')
GROUP BY 1
ORDER BY 1;
```

TableName	Sum(CurrentPerm)
Orders_CP	3,100,672
Orders_CP_noAC	4,532,224
Orders_CP_TP	5,909,504

Lab Solutions for Lab 23-1

Lab Exercise 23-1 Solutions

1. Populate your Accounts table from the AP.Accounts table using the INSERT/SELECT statement:

   ```
   INSERT INTO Accounts SELECT * FROM AP.Accounts;
   ```

   Using the DBC.TableSizeV view, what is the amount of Perm space used. Accounts = 1,844,224

2. Create a new table, named "Accounts_MVC", based on the Accounts table except compress the following city names: Culver City, Hermosa Beach, Los Angeles, and Santa Monica

   ```
   CREATE SET TABLE Accounts_MVC, FALLBACK
          (Account_Number          INTEGER NOT NULL,
           Street_Number           INTEGER,
           Street                  CHAR(25),
           City                    CHAR(20)
                   COMPRESS ('Hermosa Beach', 'Culver City', 'Los Angeles','Santa Monica'),
           State                   CHAR(2),
           Zip_Code                INTEGER,
           Balance_Forward         DECIMAL(10,2),
           Balance_Current         DECIMAL(10,2))
      PRIMARY INDEX ( Account_Number );
   ```

   Populate your Accounts_MVC table from the AP.Accounts table using INSERT/SELECT.

   Using the DBC.TableSizeV view, what is the amount of Perm space used. Accounts_MVC = 1,444,864

Lab Solutions for Lab 23-2

Lab Exercise 23-2 Solutions

1. **Determine the size of your empty Trans table using the DBC.TablesizeV view (SELECT with and without the SUM aggregate function).**

```
SELECT SUM(CurrentPerm)  FROM  DBC.TablesizeV
WHERE  DatabaseName = DATABASE   AND  TableName = 'Trans';
```

```
Sum(CurrentPerm)
          26,624
```

Size of empty Trans = 26,624 (Captured on a 26 AMP system)

What size are the table headers on each AMP? 1024

2. **Since the typical row length is 38 bytes (see facing page), estimate the size of this table assuming it will have 15,000 rows.**

38 x 15,000 = 570,000 x 2 (Fallback) = 1,140,000 bytes

Estimated size of Trans = 1,140,000

3. **Populate your Trans table from the AP.Trans table using the following INSERT/SELECT statement:**

```
INSERT INTO Trans SELECT * FROM AP.Trans;
```

Use the SELECT COUNT(*) function to verify the number of rows. 15,000

```
SELECT COUNT(*) FROM Trans;      Count(*)
                                    15000
```

Lab Solutions for Lab 23-2 (cont.)

4. Using the DBC.TablesizeV view, determine the actual size of the Trans table using the SUM function.

   Size of populated Trans = 1,185,792

```
SELECT    SUM(CurrentPerm) FROM  DBC.TablesizeV
WHERE     DatabaseName = DATABASE   AND  TableName = 'Trans' ;

Sum(CurrentPerm)
      1,185,792                          (Estimated size was 1,140,000)
```

5. Create a USI on the Trans_Number column.

```
CREATE UNIQUE INDEX (Trans_Number) on Trans;
```

   Estimate the size of the USI = 1,020,000

   (4 + 29 + 1) x 15,000 = 510,000 x 2 (Fallback) = 1,020,000 bytes

   Actual size of the table with a USI = 2,222,080

```
SELECT    SUM(CurrentPerm)  FROM  DBC.TablesizeV
WHERE     DatabaseName = DATABASE  AND  TableName = 'Trans' ;

Sum(CurrentPerm)
      2,222,080
```

   Actual size of the USI = 1,036,288 (using the empirical sizing technique)

   2,222,080 - 1,185,792 = 1,036,288

Lab Solutions for Lab 23-2 (cont.)

6. Create a NUSI on the Trans_ID column.

```
CREATE INDEX (Trans_ID) on Trans;
```

Estimate the size of the NUSI = _____

```
SELECT COUNT(DISTINCT(Trans_ID)) FROM Trans;
```

```
Count(Distinct(TRANS_ID))
                        975
```

15,000 / 975 = 15 (approximately)

(15,000 x 8) + (975 x (4 + 21 + 1) x 15) = 500,250 bytes approx.

Estimate of NUSI = 500,250 x 2 (Fallback) = 1,000,500 bytes

Actual size of the table including indexes = 2,856,960

```
SELECT      SUM(CurrentPerm) FROM  DBC.TablesizeV
WHERE       DatabaseName = DATABASE
AND         TableName = 'Trans' ;
```

```
Sum(CurrentPerm)
           2,856,960
```

The NUSI calculation is not as close because the NUSI has one value that has a large number of duplicate values.

Actual size of the NUSI = 634,880 (using the empirical sizing technique)

2,856,960 - 2,222,080 = 634,880

Lab Solutions for Lab 23-3

Lab Exercise 23-3 Solutions

1. **Populate the Employee and Emp_Phone tables**

```
INSERT INTO Employee     SELECT * FROM PD.Employee;
INSERT INTO Emp_Phone  SELECT * FROM PD.Emp_Phone;
```

2. **Using the DBC.TablesizeV view, determine the actual size of the Emp_Phone table by using the SUM aggregate function.**

   Size of populated Emp_Phone = 154,624

```
SELECT      SUM (CurrentPerm)  FROM  DBC.TablesizeV
WHERE       DatabaseName = DATABASE
AND         TableName = 'Emp_Phone' ;
```

Sum(CurrentPerm)
154,624

3. **The Foreign key is Employee_Number in the Emp_Phone table and the Primary Key is the Employee_Number in the Employee table.**

   Create a References constraint on Employee_Number using the following SQL statements.

```
ALTER TABLE Emp_Phone ADD CONSTRAINT fk1 FOREIGN KEY (Employee_Number)
     REFERENCES Employee (Employee_Number);
```

   (use the HELP CONSTRAINT Emp_Phone.fk1; to view constraint information.

```
HELP CONSTRAINT Emp_Phone.fk1;
```

Name	Type	State	Index ID	Foreign Key Columns	...
FK1	REFERENCE	VALID	0	EMPLOYEE_NUMBER	...

Lab Solutions for Lab 23-3 (cont.)

4. Using the DBC.TablesizeV view, determine the actual size of the Emp_Phone table by using the SUM aggregate function.

```
SELECT COUNT(DISTINCT(Employee_Number)) AS "Count"
FROM    Emp_Phone;
```

```
Count
1000
```

(4 + 25 + 1) x 1,000 = 30,000 x 2 (Fallback) = 60,000 bytes approx.

Estimated size of the Reference Index = 60,000

Size of Emp_Phone table with references index = 233,472

```
SELECT      SUM (CurrentPerm)  FROM   DBC.TablesizeV
WHERE       DatabaseName = DATABASE
AND         TableName = 'Emp_Phone' ;
```

```
Sum(CurrentPerm)
          233,472
```

Size of references index = 78,848

233,472 - 154,624 = 78,848

Lab Solutions for Lab 27-1

Lab Exercise 27-1 Solutions

1. **Delete all of the rows in the Trans table.**

```
DELETE Trans ALL;
```

The Trans table should have a USI on Trans_Number and a NUSI on Trans_ID from a previous lab. Verify with HELP INDEX Trans;

```
HELP INDEX Trans;
```

Unique?	Primary or Secondary?	Column Names	Index Id
N	P	ACCOUNT_NUMBER	1
Y	S	TRANS_NUMBER	4
N	S	TRANS_ID	8

2. **Collect statistics on the Trans table primary and secondary indexes. Use the Help Statistics command after collecting statistics on all of the indexes.**

```
COLLECT STATISTICS ON Trans COLUMN Account_Number;
COLLECT STATISTICS ON Trans COLUMN Trans_Number;
COLLECT STATISTICS ON Trans COLUMN Trans_ID;
```

```
HELP STATISTICS Trans;
```

Date	Time	Unique Values	Column Names
12/02/25	11:13:36	0	*
12/02/25	11:17:12	0	TRANS_NUMBER
12/02/25	11:17:13	0	ACCOUNT_NUMBER
12/02/25	11:17:13	0	TRANS_ID

Lab Solutions for Lab 27-1 (cont.)

3. Populate your Trans table from AP.Trans using the INSERT/SELECT function. Verify (SELECT COUNT) that your Trans table has 15,000 rows.

```
INSERT INTO Trans SELECT * FROM AP.TRANS;
SELECT COUNT(*) FROM Trans;
Count(*)
   15000
```

Use the Help Statistics command after populating the Trans table. Do the statistics reflect the status of table? NO How many unique values are there for each column? 0

```
HELP STATISTICS Trans;
```

Date	Time	Unique Values	Column Names
12/02/25	11:13:36	0	*
12/02/25	11:17:12	0	TRANS_NUMBER
12/02/25	11:17:13	0	ACCOUNT_NUMBER
12/02/25	11:17:13	0	TRANS_ID

4. Recollect statistics on the Trans table. Use the Help Statistics command after populating the Trans table. Do the statistics reflect the status of table? YES How many unique values are there for Trans_ID? 975

```
COLLECT STATISTICS ON Trans;
HELP STATISTICS Trans;
```

Date	Time	Unique Values	Column Names
12/02/25	11:13:36	15000	*
12/02/25	11:19:57	15000	TRANS_NUMBER
12/02/25	11:19:57	10000	ACCOUNT_NUMBER
12/02/25	11:19:58	975	TRANS_ID

Lab Solutions for Lab 27-2

Lab Exercise 27-2 Solutions

1. **Collect statistics on orderid and orderdate for Orders and Orders_PPI.**

```
COLLECT STATISTICS ON Orders COLUMN orderid;
COLLECT STATISTICS ON Orders COLUMN orderdate;
COLLECT STATISTICS ON Orders_PPI COLUMN orderid;
COLLECT STATISTICS ON Orders_PPI COLUMN orderdate;
```

Execute the following two Explains.

```
EXPLAIN SELECT * FROM Orders
    WHERE orderdate BETWEEN '2012-01-01' AND '2012-01-31';
```

How many AMPs are accessed in this operation? All
Is this a full table scan? Yes Why or why not? Because orderdate is not indexed or partitioned

```
EXPLAIN SELECT * FROM Orders_PPI
    WHERE orderdate BETWEEN '2012-01-01' AND '2012-01-31';
```

How many AMPs are accessed in this operation? All
Is this a full table scan? No Why or why not? Values are specified for a partitioning column

Is partitioning beneficial for this type of query? Yes

Lab Solutions for Lab 27-2 (cont.)

2. Execute the following two Explains.

```
EXPLAIN SELECT * FROM Orders
        WHERE orderid = 418200;
```

How many AMPs are accessed in this operation? One
How is this row retrieved by Teradata? Via the Primary index

```
EXPLAIN SELECT * FROM Orders_PPI
        WHERE orderid = 418200;
```

How many AMPs are accessed in this operation? One
How many partitions are scanned to locate this row? All
How is this row retrieved by Teradata? By scanning each partition for the primary index value.

Lab Solutions for Lab 27-2 (cont.)

3. Execute the following two Explains.

   EXPLAIN DELETE FROM Orders WHERE orderdate BETWEEN '2009-01-01' AND '2009-12-31';

   How many AMPs are accessed in this operation? All
   Is this a full table scan? Yes Why or why not? No index qualification nor any partitioning

   EXPLAIN DELETE FROM Orders_PPI
   WHERE orderdate BETWEEN '2009-01-01' AND '2009-12-31';

   How many AMPs are accessed in this operation? All
   Is this a full table scan? No Why or why not? Values are specified for a partitioning column
   How many partitions are scanned to delete these rows? 12

4. Collect statistics on orderid for the Orders_CP table and execute the following two Explains.

   EXPLAIN SELECT Orderid FROM Orderid_CP;

   How many column partitions are accessed? 2
   What is the relative cost for this EXPLAIN? .03

   EXPLAIN SELECT Orderid, Custid FROM Orders_CP;

   How many column partitions are accessed? 3
   What is the relative cost for this EXPLAIN? .04

Lab Solutions for Lab 30-1

Lab Exercise 30-1 Solutions

1. **Use INSERT/SELECT to place all rows from the populated PD tables into your empty tables. Verify the number of rows in your tables.**

```
INSERT INTO Employee SELECT * FROM PD.Employee;
SELECT COUNT(*) FROM Employee;    Count = 1000

INSERT INTO Department SELECT * FROM PD.Department;
SELECT COUNT(*) FROM Department;    Count = 60

INSERT INTO Job SELECT * FROM PD.Job;
SELECT COUNT(*) FROM Job;    Count = 66
```

2. **EXPLAIN the following SQL statement.**

```
SELECT      Last_Name, First_Name, Dept_Name, Job_Desc
FROM        Employee E
INNER JOIN  Department D    ON  E.Dept_Number = D.Dept_Number
INNER JOIN  Job J           ON  E.Job_Code = J.Job_Code
ORDER BY    3, 1, 2;
```

What is estimated time cost for this EXPLAIN? 0.10 seconds

Lab Solutions for Lab 30-1 (cont.)

3. Create a "non-compressed" join index which includes the following columns of Employee, Department, and Job.

```
CREATE JOIN INDEX EDJ_JI AS
SELECT      Last_Name, First_Name, D.Dept_Number, Dept_Name, J.Job_Code, Job_Desc
FROM        Employee E
INNER JOIN  Department D        ON  E.Dept_Number = D.Dept_Number
INNER JOIN  Job J               ON  E.job_code = J.job_code;
```

Execute the HELP USER command. What is the object type of the Join Index? I

4. EXPLAIN the following SQL statement (same SQL as step #2)

What is estimated time cost for this EXPLAIN? 0.03 seconds

Is the join index used? Yes

How much space does the join index require (use the DBC.TableSizeV view)? 125,592

```
SELECT      TableName , SUM(CurrentPerm)
FROM        DBC.TableSizeV
WHERE       DatabaseName = USER  AND  TableName = 'EDJ_JI'
GROUP BY    1
ORDER BY    1;
```

TableName	Sum(CurrentPerm)
EDJ_JI	125,592

Lab Solutions for Lab 30-1 (cont.)

5. EXPLAIN the following SQL statement – Salary_Amount has been added as a projected column.

```
SELECT      Last_Name, First_Name, Dept_Name, Job_Desc, Salary_Amount
FROM        Employee E
INNER JOIN  Department D   ON  E.Dept_Number = D.Dept_Number
INNER JOIN  Job J          ON  E.Job_Code = J.Job_Code
ORDER BY    3, 1, 2;
```

   What is estimated time cost for this EXPLAIN? 0.10 seconds

   Is the join index used? No If not, why not? Salary_Amount is not in the Join Index. The join index rows and the associated Employee row are not on the same AMP. Therefore, the optimizer chooses to join the two tables together.

6. Drop the Join Index.

```
DROP JOIN INDEX EDJ_JI;
```

Lab Solutions for Lab 30-1 (cont.)

7. (Optional) Create a new "non-compressed" join index similar to step #3 except include the Primary Index of Employee_Number for the Join Index.

```
CREATE JOIN INDEX EDJ_JI2 AS
SELECT      Employee_Number, Last_Name, First_Name,
            D.Dept_Number, Dept_Name, J.Job_Code, Job_Desc
FROM        Employee E
INNER JOIN  Department D        ON  E.Dept_Number = D.Dept_Number
INNER JOIN  Job J               ON  E.job_code = J.job_code
PRIMARY INDEX (Employee_Number);
```

8. (Optional) EXPLAIN the SQL statement from step #5.

   Is the join index used? Yes If so, why? Because a join index row and the associated employee row are on same AMP – the join index and the Employee table can be joined together with matching PI values. The cost is .04 seconds versus .10 seconds without the join index.

9. (Optional) Drop the Join Index.

```
DROP JOIN INDEX EDJ_JI2;
```

Lab Solution for Lab 33-1

Lab Exercise 33-1 Solution

more lab33_12.btq

```
.LOGON tdt5b/student130,xxx
.EXPORT DATA FILE = data33_1, LIMIT=4000, CLOSE
SELECT * FROM AP.Accounts;
.EXPORT RESET
.QUIT
```

bteq < lab33_12.btq

Note:

This general technique to create data files will be used for many class exercises, but will not always be shown.

Lab Solution for Lab 33-1

Lab Exercise 33-1 Solution

more lab33_18.btq

```
.SET SESSIONS 8
.LOGON tdt5b/student130,xxx
.IMPORT DATA FILE = data33_1
.QUIET ON
.REPEAT * PACK 10
USING     in_account_number       (INTEGER),
          in_street_number        (INTEGER),
          in_street               (CHAR(25)),
          in_city                 (CHAR(20)),
          in_state                (CHAR(2)),
          in_zip_code             (INTEGER),
          in_balance_forward      (DECIMAL(10,2)),
          in_balance_current      (DECIMAL(10,2))
INSERT INTO Accounts VALUES
        (:in_account_number, :in_street_number, :in_street, :in_city, :in_state,
         :in_zip_code, :in_balance_forward, :in_balance_current);
.QUIT
```

bteq < lab33_18.btq

Lab Solution for Lab 33-2

Lab Exercise 33-2 Solution

more lab33_22.btq

```
.LOGON tdt5b/student130,xxx
.IMPORT DATA FILE = data33_2
.EXPORT REPORT FILE = report33_2
.WIDTH 80
.REPEAT *
  USING         in_customer_number (INTEGER)
  SELECT        customer_number       (TITLE '')
                , last_name (CHAR(10))  (TITLE '')
                , first_name (CHAR(10))  (TITLE '')
                , social_security           (TITLE '')
  FROM          AP.Customer
  WHERE         customer_number = :in_customer_number;
.EXPORT RESET
.QUIT
```

more report33_2

```
9000      Underwood          Anne          213633756
  :           :                :               :
8501      Atchison           Jose          213631261
```

Lab Solutions for Lab 34-1

Lab Exercise 34-1 Solution

more lab34_12.fld

```
LOGON tdt5b/student130,xxx;
BEGIN LOADING Customer
   ERRORFILES cust_e1, cust_e2;
DEFINE     in_cust          (INTEGER),
           in_lname         (CHAR(30)),
           in_fname         (CHAR(20)),
           in_social        (INTEGER)
   FILE = data34_1;
INSERT INTO Customer     VALUES
          ( :in_cust,
            :in_lname,
            :in_fname,
            :in_social);
LOGOFF;
```

Or

```
LOGON tdt5b/student130,xxx;
BEGIN LOADING Customer
   ERRORFILES cust_e1, cust_e2;
DEFINE FILE = data34_1;
INSERT INTO Customer.* ;
LOGOFF;
```

fastload < lab34_12.fld

more lab34_13.fld

```
LOGON tdt5b/student130,xxx;
BEGIN LOADING Customer
   ERRORFILES cust_e1, cust_e2;
DEFINE     in_cust          (INTEGER),
           in_lname         (CHAR(30)),
           in_fname         (CHAR(20)),
           in_social        (INTEGER)
   FILE = data34_2;
INSERT INTO Customer     VALUES
          ( :in_cust,
            :in_lname,
            :in_fname,
            :in_social);
END LOADING;
LOGOFF;
```

Or

```
LOGON tdt5b/student130,xxx;
BEGIN LOADING Customer
   ERRORFILES cust_e1, cust_e2;
DEFINE FILE = data34_2;
INSERT INTO Customer.* ;
END LOADING;
LOGOFF;
```

fastload < lab34_13.fld

Lab Solution for Lab 34-2

Lab Exercise 34-2 Solution

more lab34_22.fld

```
LOGON tdt5b/student130,xxx;
BEGIN LOADING Trans ERRORFILES trans_e1, trans_e2;
DEFINE    in_transno        (INTEGER),
          in_transdate      (CHAR(10), NULLIF='0000-00-00'),
          in_accno          (INTEGER),
          in_trans_id       (CHAR(4)),
          in_trans_amt      (DECIMAL(10,2))
FILE = data34_3;
INSERT INTO Trans
VALUES  ( :in_transno,
          :in_transdate     (FORMAT 'YYYY-MM-DD'),
          :in_accno,
          :in_trans_id,
          :in_trans_amt );
END LOADING;
LOGOFF;
```

fastload < lab34_22.fld

From bteq:

```
SELECT COUNT(*) FROM Trans WHERE Trans_Date IS NULL;

COUNT(*)
      150
```

Lab Solution for 35-1

Lab Exercise 35-1 Solution

more lab35_12.fxp

```
.LOGTABLE Restartlog3512_fxp;
.LOGON tdt5b/student130,xxx;
.ACCEPT cnum, lname, fname, socsec FROM FILE data35_1;
.SET filename TO 'Customer';
INSERT INTO &filename
      VALUES (&cnum, '&lname', '&fname', &socsec) ;
.LOGOFF;
```

fexp < lab35_12.fxp

Lab Solutions for Lab 36-1

Lab Exercise 36-1 Solutions

more lab36_12.fxp

```
.LOGTABLE Restartlog3612_fxp ;
.LOGON tdt5b/student130,xxx;
.BEGIN EXPORT;
.EXPORT OUTFILE data36_1;
   SELECT        T.trans_number
                 ,A.account_number
                 ,A.number
                 ,A.street
                 ,A.city
                 ,A.state
                 ,A.zip_code
   FROM          AP.Accounts A
   INNER JOIN    AP.Trans T
   ON A.Account_number = T.Account_number;
.END EXPORT;
.LOGOFF;
```

fexp < lab36_12.fxp

(alternative join syntax)

more lab36_12a.fxp

```
.LOGTABLE Restartlog3612a_fxp;
.LOGON tdt5b/student130,xxx;
.BEGIN EXPORT;
.EXPORT OUTFILE data36_1;
   SELECT        T.trans_number
                 ,A.account_number
                 ,A.number
                 ,A.street
                 ,A.city
                 ,A.state
                 ,A.zip_code
   FROM           AP.Accounts A
                 , AP.Trans T
   WHERE A.Account_number = T.Account_number;
.END EXPORT;
.LOGOFF;
```

fexp < lab36_12a.fxp

Lab Solutions for Lab 36-2 (ACCEPT and CASE)

Lab 36-2 Solutions

more lab36_22a.fxp

```
.LOGTABLE Restartlog3622a_fxp;
.LOGON tdt5b/student130,xxx;
.SET LoVal TO  500;
.SET HiVal  TO  9499;
.ACCEPT city FROM FILE data36_2;

.BEGIN EXPORT;
.EXPORT OUTFILE report36_a MODE RECORD FORMAT TEXT;

 SELECT     CAST (Account_Number                                          AS CHAR(12)),
            CAST (City                                                    AS CHAR(12)),
            CAST (CAST (Balance_Current  AS FORMAT '$,$$$,$$9.99')        AS CHAR(12)),
            CAST ( (CASE   WHEN Balance_Current < &LoVal     THEN ' Below MIN'
                           WHEN Balance_Current > &HiVal     THEN ' Above MAX'
                    END)   AS CHAR(10) )
 FROM       AP.Accounts
 WHERE      City = '&city'
 AND        (Balance_Current < &LoVal OR Balance_Current > &HiVal)
 ORDER BY   Account_Number;
.END EXPORT;
.LOGOFF;
```

fexp < lab36_22a.fxp

Lab Solutions for Lab 36-2 (ACCEPT and UNION)

```
more lab36_22b.fxp
   .LOGTABLE Restartlog3622b_fxp;
   .LOGON tdt5b/student130,xxx;
   .SET LoVal TO 500;
   .SET HiVal TO 9499;
   .ACCEPT city FROM FILE data36_2;

   .BEGIN EXPORT;
   .EXPORT OUTFILE report36_b MODE RECORD FORMAT TEXT;

   SELECT  Account_Number (CHAR(12)), City (CHAR(12)), Balance_Current (CHAR(12)),
           ' Above MAX' (CHAR(10))
   FROM    AP.Accounts
   WHERE   City = '&city'   AND   Balance_Current > &HiVal

   UNION

   SELECT  Account_Number (CHAR(12)), City (CHAR(12)), Balance_Current (CHAR(12)),
           ' Below MIN' (CHAR(10))
   FROM    AP.Accounts
   WHERE   City = '&city'   AND   Balance_Current < &LoVal
   ORDER BY 1;
   .END EXPORT;
   .LOGOFF;

fexp < lab36_22b.fxp
```

Lab Solutions for Lab 36-2 (.IMPORT)

```
more lab36_22c.fxp
   .LOGTABLE Restartlog3622c_fxp;
   .LOGON tdt5b/student130,xxx;
   .SET LoVal TO  500;
   .SET HiVal  TO  9499;
   .BEGIN EXPORT;
   .LAYOUT        Record_Layout;
   .FIELD         city      *    CHAR(15);

   .IMPORT        INFILE data36_2c  FORMAT TEXT LAYOUT Record_Layout ;

   .EXPORT OUTFILE report36_c MODE RECORD FORMAT TEXT;
   SELECT         CAST (Account_Number                                          AS CHAR(12)),
                  CAST (City                                                    AS CHAR(12)),
                  CAST (CAST (Balance_Current  AS FORMAT '$,$$$,$$9.99')        AS CHAR(12)),
                  CAST ( (CASE   WHEN Balance_Current < &LoVal          THEN ' Below MIN'
                                 WHEN Balance_Current > &HiVal          THEN ' Above MAX'
                          END)   AS CHAR(10) )
    FROM          AP.Accounts
    WHERE         City = :city
    AND           (Balance_Current < &LoVal OR Balance_Current > &HiVal)
    ORDER BY      Account_Number ;
   .END EXPORT;
   .LOGOFF;
fexp < lab36_22c.fxp
```

Lab Solution for Lab 37-1

Lab 37-1 Solution

more lab37_13.mld

```
.LOGTABLE Restartlog3713_mld;
.LOGON tdt5b/student130,xxx;
.BEGIN MLOAD TABLES Accounts, Customer, Trans ;
.Layout  Record_Layout;
     .FIELD     Table_Code      *     CHAR(1) ;
     .FIELD     PI_Value        *     INTEGER ;
.DML LABEL Del_Acct ;
     DELETE FROM Accounts    WHERE Account_Number = :PI_Value ;
.DML LABEL Del_Cust ;
     DELETE FROM Customer    WHERE Customer_Number = :PI_Value ;
.DML LABEL Del_Trans ;
     DELETE FROM Trans       WHERE Account_Number = :PI_Value ;
.IMPORT INFILE data37_1
     LAYOUT Record_Layout
     APPLY Del_Acct          WHERE Table_Code = 'A'
     APPLY Del_Cust          WHERE Table_Code = 'C'
     APPLY Del_Trans         WHERE Table_Code = 'T' ;
.END MLOAD ;
.LOGOFF ;
```

mload < lab37_13.mld | tee lab37_13.out

Lab Solution for Lab 38-1

Lab Exercise 38-1 Solution

more lab38_13.mld

```
.LOGTABLE Log3813_mld;
.LOGON tdt5b/student130,xxx;
.BEGIN MLOAD TABLES
          Accounts, Customer, Trans;
.LAYOUT   Record_Layout;
.FIELD    in_code          1   CHAR(1);
.FIELD    in_account1      2   INTEGER;
.FIELD    in_streetnum     *   INTEGER;
.FIELD    in_streetname    *   CHAR(25);
.FIELD    in_city          *   CHAR(20);
.FIELD    in_state         *   CHAR(2);
.FIELD    in_zip_code      *   INTEGER;
.FIELD    in_balancefor    *   DECIMAL (10,2);
.FIELD    in_balancecur    *   DECIMAL (10,2);
.FIELD    in_custno        2   INTEGER;
.FIELD    in_lastname      *   CHAR(30);
.FIELD    in_firstname     *   CHAR(20);
.FIELD    in_socsec        *   INTEGER;
.FIELD    in_transno       2   INTEGER;
.FIELD    in_transdate     *   CHAR(10);
.FIELD    in_account2      *   INTEGER;
.FIELD    in_transid       *   CHAR(4);
.FIELD    in_transamount   *   DECIMAL(10,2);
```

(continued)

```
.DML LABEL Insert_Acct;
 INSERT INTO Accounts VALUES
 (:in_account1, :in_streetnum, :in_streetname,
  :in_city, :in_state, :in_zip_code, :in_balancefor,
  :in_balancecur) ;

.DML LABEL Insert_Cust;
 INSERT INTO Customer  VALUES
 ( :in_custno, :in_lastname,
   :in_firstname, :in_socsec );

.DML LABEL Insert_Trans;
 INSERT INTO Trans VALUES
 ( :in_transno, :in_transdate, :in_account2,
   :in_transid, :in_transamount);

.IMPORT INFILE data38_1
    LAYOUT  Record_Layout
    APPLY     Insert_Acct   WHERE in_code = 'A'
    APPLY     Insert_Cust   WHERE in_code = 'C'
    APPLY     Insert_Trans  WHERE in_code = 'T';

.END MLOAD;
.LOGOFF;
```

mload < lab38_13.mld | tee lab38_13.out

Lab Solution for Lab 38-2

Lab Exercise 38-2 Solution

```
more lab38_23.mld
   .LOGTABLE Log3823_mld;
   .LOGON tdt5b/student130,xxx;
   .BEGIN IMPORT MLOAD TABLES Accounts;
   .LAYOUT Record_Layout;
      .FIELD    in_account        1    INTEGER;
      .FIELD    in_streetnum      *    INTEGER;
      .FIELD    in_streetname     *    CHAR(25);
      .FIELD    in_city           *    CHAR(20);
      .FIELD    in_state          *    CHAR(2);
      .FIELD    in_zip_code       *    INTEGER;
      .FIELD    in_balancefor     *    DECIMAL (10,2);
      .FIELD    in_balancecur     *    DECIMAL (10,2);
   .DML LABEL Update_Account  DO INSERT FOR MISSING UPDATE ROWS;
    UPDATE  Accounts            SET      Balance_Current = :in_balancecur
                                WHERE    Account_Number = :in_account;

    INSERT INTO Accounts VALUES  (:in_account, :in_streetnum, :in_streeetname, :in_city,
                                  :in_state, :in_zip_code, :in_balancefor, :in_balancecur);
   .IMPORT INFILE  data38_2  LAYOUT  Record_Layout  APPLY Update_Account;
   .END MLOAD;
   .LOGOFF;

mload < lab38_23.mld | tee lab38_23.out
```

Lab Solution for Lab 39-1

Lab Exercise 39-1 Solution

```
more lab39_13a.tpp
   .LOGTABLE Log813a_tpp;
   .LOGON tdt5b/student130,xxx;
   .BEGIN LOAD SESSIONS 4  PACK 20  RATE 4800 SERIALIZE ON;
   .LAYOUT Record_Layout;
      .FIELD    in_accountno      1    INTEGER         KEY;
      .FIELD    in_stnum          *    INTEGER;
      .FIELD    in_stname         *    CHAR(25);
      .FIELD    in_city           *    CHAR(20);
      .FIELD    in_state          *    CHAR(2);
      .FIELD    in_zip_code       *    INTEGER;
      .FIELD    in_balancefor     *    DECIMAL (10,2);
      .FIELD    in_balancecur     *    DECIMAL (10,2);
   .DML LABEL Update_Account  DO INSERT FOR MISSING UPDATE ROWS
      USE (in_accountno,in_stnum,in_stname,in_city,in_state,in_zip_code,in_balancefor,in_balancecur);
    UPDATE  Accounts              SET       Balance_Current = :in_balancecur
                                  WHERE     Account_Number = :in_accountno;
    INSERT INTO Accounts VALUES  (:in_accountno, :in_stnum, :in_stname, :in_city,
                                   :in_state, :in_zip_code, :in_balancefor, :in_balancecur);
   .IMPORT INFILE data39_1  LAYOUT Record_Layout  APPLY Update_Account;
   .END LOAD;
   .LOGOFF;

tpump < lab39_13a.tpp | tee lab39_13a.out
```

Lab Solution for Lab 39-1 (Alternate Solution using SQL UPSERT)

Lab Exercise 39-1 Alternate Solution

```
more lab39_13b.tpp
   .LOGTABLE Log813b_tpp;
   .LOGON tdt5b/student130,xxx;
   .BEGIN LOAD SESSIONS 4  PACK 20  RATE 4800 SERIALIZE ON;
   .LAYOUT Record_Layout;
      .FIELD    in_accountno      1    INTEGER        KEY;
      .FIELD    in_stnum          *    INTEGER;
      .FIELD    in_stname         *    CHAR(25);
      .FIELD    in_city           *    CHAR(20);
      .FIELD    in_state          *    CHAR(2);
      .FIELD    in_zip_code       *    INTEGER;
      .FIELD    in_balancefor     *    DECIMAL (10,2);
      .FIELD    in_balancecur     *    DECIMAL (10,2);
   .DML LABEL Update_Account
      USE (in_accountno,in_stnum,in_stname,in_city,in_state,in_zip_code,in_balancefor,in_balancecur);
    UPDATE  Accounts              SET       Balance_Current = :in_balancecur
                                  WHERE     Account_Number = :in_accountno
    ELSE INSERT INTO Accounts VALUES  (:in_accountno, :in_stnum, :in_stname, :in_city,
                                       :in_state, :in_zip_code, :in_balancefor, :in_balancecur);
   .IMPORT INFILE data39_1  LAYOUT Record_Layout  APPLY Update_Account;
   .END LOAD;
   .LOGOFF;

tpump < lab39_13b.tpp | tee lab39_13b.out
```

Lab Solutions for Lab 42-1

Lab Exercise 42-1 Solutions

1. **Using the DBC.DBCInfoV view, find the release and version of the system on which you are logged on:**

```
SELECT    *
FROM      DBC.DBCInfoV;
```

InfoKey	InfoData	
RELEASE	14.00.00.01	(PDE Release #)
VERSION	14.00.00.01	(Teradata Version #)

2. **Using the DBC.ChildrenV view, list your parents' userids:**

```
SELECT    Parent
FROM      DBC.ChildrenV
WHERE     Child = USER;
```

Parent
DBC
Sysdba
Students
TT_Class1

Lab Solutions for Lab 42-1 (cont.)

3. Using the DBC.DatabasesV view, find your:

```
SELECT    OwnerName, CreatorName, AccountName, PermSpace, SpoolSpace, TempSpace
FROM      DBC.DatabasesV
WHERE     Databasename = USER;
```

Immediate parent's name	TT_Class1	
Creator's name	TT_Class1	
Default account code	$M0+EDUC&S&D&H	
Perm space limit	1,999,999,976	(2 GB)
Spool space limit	4,999,999,992	(5 GB)
Temp space limit	999,999,988	(1 GB)

4. Using the DBC.UsersV view, find your:

```
SELECT    DefaultDatabase, DefaultCollation,
          DefaultDateForm, CreateTimeStamp, PasswordLastModDate
FROM      DBC.UsersV
WHERE     UserName = USER;
```

Default database name	?
Default collation sequence	H
Default date format	?
Create Time Stamp	2012-02-10 11:50:45
Last password modification date	2012-02-10

Lab Solutions for Lab 42-1 (cont.)

4. **OPTIONAL: SHOW this view. Note the WHERE conditions. (Remember, this is a restricted view, even though it does not have an [X] suffix.)**

```
REPLACE VIEW DBC.UsersV
    AS SELECT
        dbase.DatabaseName(NAMED UserName),
        dbase.CreatorName,
    ...
```

5. **Using the DBC.TablesV view, find the number of tables in the DD/D (User DBC) that are:**

Fallback protected **Count is 152**

```
SELECT      Count(*)
FROM        DBC.TablesV
WHERE       DatabaseName = 'DBC'
AND         TableKind IN ('T', 'O')  AND  ProtectionType = 'F';
```

Not Fallback protected **Count is 15**

```
SELECT      Count(*)
FROM        DBC.TablesV
WHERE       DatabaseName = 'DBC'
AND         TableKind IN ('T', 'O')  AND  ProtectionType = 'N';
```

Lab Solutions for Lab 42-1 (cont.)

5. Modify the query to find the number of tables OTHER THAN DD/D (not DBC tables) that are:

Fallback protected — Count will vary - 1685

```
SELECT      Count(*)
FROM        DBC.TablesV
WHERE       DatabaseName  NE 'DBC'
AND         TableKind IN ('T', 'O')
AND         ProtectionType = 'F';
```

Not Fallback protected — Count will vary - 317

```
SELECT      Count(*)
FROM        DBC.TablesV
WHERE       DatabaseName  NE  'DBC'
AND         TableKind IN ('T', 'O')
AND         ProtectionType = 'N';
```

Lab Solutions for Lab 42-1 (cont.)

6. Using the DBC.PartitioningConstraintsV view, answer the following questions; hint, use the ColumnPartitioningLevel column to determine if a table is column partitioned or not.

   Note: Your answers may vary

   For your user, how many tables have a PPI? 2
   For your user, how many tables have column partitioning? 3

   For the system, how many tables have a PPI? 108
   For the system, how many tables have column partitioning? 94

   What is the constraint type for PPI tables? O

```
SELECT    TableName,
          ColumnPartitioningLevel     AS "CP",
          PartitioningLevels          AS "# Levels",
          ConstraintText              AS "Text"
FROM      DBC.PartitioningConstraintsV
WHERE     DatabaseName = USER;

SELECT    COUNT(*)
FROM      DBC.PartitioningConstraintsV
WHERE     ColumnPartitioningLevel = 0;          108

SELECT    COUNT(*)
FROM      DBC.PartitioningConstraintsV
WHERE     ColumnPartitioningLevel <> 0;         94
```

Lab Solutions for Lab 42-1 (cont.)

7. Using the DBC.ColumnsV view, find the number of columns in the entire system defined with *default values:*

   Number of columns Count will vary - 1936

```
SELECT   'Column count:'
         ,COUNT (*)
FROM     DBC.ColumnsV
WHERE    DefaultValue IS NOT NULL;
```

'Column count:'	Count(*)
Column count:	1936

   OPTIONAL: Modify the query to find the number of objects that have columns defined with *default values:*

   Number of tables Count will vary - 166

```
SELECT   'Object Count:'
         ,COUNT (DISTINCT (DatabaseName || '.' || TableName))
FROM     DBC.ColumnsV
WHERE    DefaultValue IS NOT NULL;
```

'Object Count:'	Count(Distinct(TableName))
Object Count:	166

Lab Solutions for Lab 42-1 (cont.)

8. Using the DBC.IndicesV view, find the number of tables OTHER THAN Dictionary tables that have non-unique primary indexes (NUPI):

   Number of tables Count will vary - 1187

```
SELECT    COUNT (DISTINCT(DatabaseName || '.' || TableName))
FROM      DBC.IndicesV
WHERE     IndexType IN ('P', 'Q')
AND       UniqueFlag = 'N'
AND       DatabaseName NE 'DBC';
```

```
Count(Distinct(TableName))
                      1187
```

Note: The IndexType of 'P' is used for the primary index of non-partitioned tables.
The IndexType of 'Q' is used for the primary index of partitioned tables.

Lab Solutions for Lab 42-2 (optional)

Lab Exercise 42-2 (optional) Solutions

1. **Use INSERT/SELECT to place all rows from the populated PD tables into your empty tables. Verify the number of rows in your tables.**

```
INSERT INTO Employee SELECT * FROM PD.Employee;
SELECT COUNT(*) FROM Employee;    Count = 1000

INSERT INTO Department SELECT * FROM PD.Department;
SELECT COUNT(*) FROM Department;    Count = 60

INSERT INTO Job SELECT * FROM PD.Job;
SELECT COUNT(*) FROM Job;    Count = 66

INSERT INTO Emp_Phone SELECT * FROM PD.Emp_Phone
SELECT COUNT(*) FROM Emp_Phone;    Count = 2000
```

2. **Use the GRANT statement to GRANT yourself the REFERENCES access rights on the tables.**

```
GRANT REFERENCES ON Employee TO student130;
GRANT REFERENCES ON Department TO student130;
GRANT REFERENCES ON Job TO student130;
```

Lab Solutions for Lab 42-2 (optional – cont.)

3. Create a References constraint between the Employee.Dept_Number column and the Department.Dept_Number column.

```
ALTER TABLE Employee ADD CONSTRAINT emp_dept_ref
      FOREIGN KEY (dept_number) REFERENCES
      Department (dept_number) ;
```

What is the name of the Employee RI error table? EMPLOYEE_0
How many rows are in this table? 1
Which department is not represented in the department table? 1000

4. Use the DBC.All_RI_ChildrenV view (qualify the ChildDB to your database) and verify this References constraint.

What is the IndexID of this constraint? 0

```
SELECT      Indexid    (FORMAT 'z9')  AS ID
           ,IndexName
           ,ChildTable
           ,ChildKeyColumn
           ,ParentTable
           ,ParentKeyColumn
FROM        DBC.ALL_RI_ChildrenV
WHERE       ChildDB = USER
ORDER BY    1;
```

ID	IndexName	ChildTable	ChildKeyColumn	ParentTable	ParentKeyColumn
0	emp_dept_ref	Employee	dept_number	Department	dept_number

Lab Solutions for Lab 44-1

Lab Exercise 44-1

1. **Using the DBC.DiskSpaceV view, find the total disk storage capacity of the system on which you are logged on:**

   Total capacity **1,739,098,519,142**

```
SELECT SUM(MaxPerm) (FORMAT 'zzz,zzz,zzz,999')  FROM DBC.DiskSpaceV;

     Sum(MaxPerm)
1,739,098,519,142
```

2. **Using the same view, find how much of the space is currently in use:**

   Current space utilization **3,709,982,208**

```
SELECT SUM(CurrentPerm) (FORMAT 'zzz,zzz,zzz,999') FROM DBC.DiskSpaceV;

  Sum(CurrentPerm)
     3,709,982,208
```

   Write a query to show what percentage of system capacity is currently in use.

```
SELECT ( (SUM(CurrentPerm) ) / NULLIFZERO (SUM(MaxPerm)) * 100) (FORMAT 'z9.99')
            AS "% Used Perm Space"
FROM        DBC.DiskSpaceV;

% Used Perm Space
                0.21
```

Lab Solutions for Lab 44-1 (cont.)

2. (cont.)

OPTIONAL: Write a query to show which databases/users are currently using (current perm) the largest percentage of their max perm space limit (group by database/user).

```
SELECT      Databasename
           ,( (SUM(CurrentPerm) ) / NULLIFZERO (SUM(MaxPerm)) * 100) (FORMAT 'z9.99')
                AS "% Used Perm Space"
FROM        DBC.DiskSpaceV
GROUP BY  1
ORDER BY  2 DESC;
```

DatabaseName	% Used Perm Space
TD_SYSFNLIB	59.21
MYQCD	29.75
student206	3.08
TDStats	2.99
SYSSPATIAL	2.55
student223	2.48

Lab Solutions for Lab 44-1 (cont.)

3. Using the DBC.DatabasesV view, find the total number of databases and users defined in the system.

```
SELECT    COUNT(*) AS "Total Count"
FROM      DBC.DataBasesV;
```

Total Count
200

Total row count (databases and users) 200 (Answers will vary)

How many users are there? 174 (Answers will vary)

```
SELECT    COUNT(*) AS "User Count"
FROM      DBC.DataBasesV
WHERE     DBKind = 'U';
```

User Count
174

How many databases are there ? 26 (Answers will vary)

```
SELECT    COUNT(*) AS "DB Count"
FROM      DBC.DataBasesV
WHERE     DBKind = 'D';
```

DB Count
26

Lab Solutions for Lab 44-1 (cont.)

3. (cont.)

Who is the creator of AP? SYSDBA
Who is the owner of AP? TT_Data

```
SELECT      CreatorName, OwnerName
FROM        DBC.DataBasesV
WHERE       DatabaseName = 'AP';
```

CreatorName	OwnerName
SYSDBA	TT_Data

Lab Solutions for Lab 44-1 (cont.)

4. Using the DBC.TableSizeV view, find the name and size of each table in the DBC user. List the Data Dictionary tables in DESCending order by size.

```
SELECT    TableName, SUM(CurrentPerm)  (FORMAT 'zzz,zzz,zzz')
FROM      DBC.TableSizeV
WHERE     DatabaseName = 'DBC'
GROUP BY  1
ORDER BY  2 DESC;
```

TableName	Sum(CurrentPerm)
EventLog	2,118,117,376
TransientJournal	103,088,128
TVM	34,121,728
TVFields	19,522,048
TextTbl	18,756,608
DataBaseSpace	16,474,112

5. Using the DBC.AMPUsage view, find the number of AMP vprocs defined on your system.

   (HINT: Use a WHERE condition to reduce the number of DD/D table rows considered.)

```
SELECT COUNT(DISTINCT (Vproc))     AS "# of AMPS"
FROM   DBC.AMPUsageV               WHERE  UserName = USER;
```

# of AMPS
26

Lab Solutions for Lab 44-1 (cont.)

6. Using the DBC.AccountInfoV view, list all of your valid account codes.

```
SELECT    *
FROM      DBC.AccountInfoV
WHERE     UserName = USER;
```

UserName	AccountName
student130	$M0+EDUC&S&D&H
student130	$L0+EDUC&S&D&H

7. Using the DBC.AMPUsageV view, write a query to show the number of AMP CPU seconds and logical disk I/Os that have been charged to your:

```
SELECT    UserName            (CHAR(12))
          ,SUM (CPUTime)      (FORMAT 'zzzz.99')
          ,SUM (DiskIO)       (FORMAT 'zzz,zzz,zzz')
FROM      DBC.AMPUsageV
WHERE     UserName = USER
GROUP BY  UserName ;
```

UserName	Sum(CpuTime)	Sum(DiskIO)
student130	111.53	1,021,628

Lab Solutions for Lab 45-1

Lab Exercise 45-1 Solutions

1. Create a user profile with a profile name that is the same as your user name (studentxxx_P).

```
CREATE PROFILE student130_P AS
      ACCOUNT              =  '$M',
      DEFAULT DATABASE     =  student130
      SPOOL                =  50E6,
      TEMPORARY            =  80E6,
      PASSWORD             = (EXPIRE = 91,  MINCHAR = 6, MAXLOGONATTEMPTS = 3,
                              LOCKEDUSEREXPIRE = 5, REUSE = 365, DIGITS='R',
                              RESTRICTWORDS='Y', SPECCHAR='P');
```

2. Use the DBC.ProfileInfoV view to display information about profiles in the system.

```
SELECT      ProfileName
           ,DefaultAccount AS "Def Acct"
           ,DefaultDB
           ,SpoolSpace
           ,TempSpace
FROM        DBC.ProfileInfoV
ORDER BY    1;
```

Lab Solutions for Lab 45-1 (cont.)

3. Create two new users in the system as follows:

```
CREATE USER student130_A  AS PERM = 0, PASSWORD = ***, PROFILE = student130_P;
CREATE USER student130_B  AS PERM = 0, PASSWORD = ***, PROFILE = student130_P;
```

4. Logon to Teradata as studentxxx_A.

   Were you prompted to enter a new password? YES

   Why were you prompted to enter a new password? Because EXPIRE was not equal to 0.

Lab Solutions for Lab 47-1

Lab Exercise 47-1

1. **Using the DBC.AllRightsV view, find the total number of rows in the DBC.AccessRights table assigned to users.**

```
SELECT COUNT(*)      FROM  DBC.AllRightsV;

Count(*)
  64,816      Total row count (AllRights view) (Answers will vary)
```

Using the DBC.AllRoleRightsV view, find the total number of rows in the DBC.AccessRights table assigned to roles.

```
SELECT COUNT(*)      FROM  DBC.AllRoleRightsV;

Count(*)
    155       Total row count (AllRoleRights view)      (Answers will vary)
```

2. **Using the DBC.UserRightsV view, take a look at the databases and tables on which you currently hold rights.**

```
SELECT COUNT(*)      FROM   DBC.UserRightsV;

Count(*)
    355       Total number of rows returned             (Answers will vary)
```

How do you think most of these privileges were granted?

These are automatic rights that you received when your user and tables were created.

Lab Solutions for Lab 47-1 (cont.)

2. (cont.)

Execute the following SQL command and then recheck the number of Access Rights you have.

```
CREATE TABLE Emp_Phone2 AS PD.Emp_Phone WITH NO DATA;
```

Total number of user rights returned **367** **(Answers will vary)**

```
SELECT COUNT(*)    FROM  DBC.UserRightsV;

Count(*)
      367
```

How many new access rights were created? **12**

Lab Solutions for Lab 47-1 (cont.)

3. For your Emp_Phone2 table, use the GRANT command to give the SELECT access right to the database AP.

```
GRANT SELECT ON Emp_Phone2 TO AP;
```

Use the GRANT command to give SELECT WITH GRANT access right to the database PD.

```
GRANT SELECT ON Emp_Phone2 TO PD WITH GRANT OPTION;
```

Check the total number of user rights returned 367

```
SELECT COUNT(*)    FROM  DBC.UserRightsV;

Count(*)
      367
```

Did this count change? No

If not, why not? These new rights are associated with users different than yourself.

Use the DBC.UserGrantedRightsV view to show any user rights that you may have explicitly granted.

How many rows are returned with this view? 92 (answers will vary)

```
SELECT    Count(*) FROM  DBC.UserGrantedRightsV;
```

Lab Solutions for Lab 47-2

Lab Exercise 47-2 Solutions

1. Using the DBC.RoleInfoV view, find the total number of roles defined in the system?

   SELECT COUNT(*) FROM DBC.RoleInfoV; Count = 423

   Using the DBC.RoleInfoVX view, what is the number of roles that you have created?

   SELECT COUNT(*) FROM DBC.RoleInfoVX; Count = 0

   Using the DBC.RoleMembersVXview, what is your (studentxxx) default role?

   SELECT * FROM DBC.RoleMembersVX; TT_Access_R

   Using the DBC.RoleMembersVXview, which roles do you have the "With Admin" option?

   SELECT * FROM DBC.RoleMembersVX; Role1_130, Role2_130, Role3_130

2. Grant the following access rights to the specified roles as follows:

```
GRANT SELECT                    ON Orders       TO Role1_130;
GRANT SELECT                    ON Orders_2012  TO Role1_130;
GRANT SELECT                    ON Orders_PPI   TO Role2_130;
GRANT INSERT, UPDATE, DELETE    ON Orders_PPI   TO Role3_130;
```

Lab Solutions for Lab 47-2 (cont.)

3. Grant Role1_xxx to studentxxx_A; GRANT Role1_130 to student130_A;
   Grant Role2_xxx to studentxxx_B; GRANT Role2_130 to student130_B;
   Grant Role2_xxx to Role3_xxx; GRANT Role2_130 TO Role3_130;
   Grant Role3_xxx to studentxxx_B; GRANT Role3_130 TO student130_B;

4. Modify the two users in the system as follows:

   MODIFY USER student130_A AS DEFAULT ROLE = Role1_130;
   MODIFY USER student130_B AS DEFAULT ROLE = Role2_130;

5. As "studentxxx_A", execute the following SQL statements and indicate if SELECT is allowed or not.

   SELECT COUNT(*) FROM Orders; Permitted or not? YES
   SELECT COUNT(*) FROM Orders_PPI; Permitted or not? NO

Lab Solutions for Lab 47-2 (cont.)

6. As "studentxxx_A", use the DBC.RoleMembersV and DBC.UserRoleRightsV views to view the current role of the user, any nested roles, and access rights for the roles.

```
SELECT  *   FROM DBC.RoleMembersVX;

SELECT      RoleName, DatabaseName, TableName, ColumnName, AccessRight
FROM        DBC.UserRoleRIghtsV
ORDER BY  1;
```

How many roles are available to studentxxx_A?	1
What is the default role for studentxxx_A?	Role1_130
Does studentxxx_A have the "With Admin" option on any roles?	No
How many user role rights are available to studentxxx_A?	2

OPTIONAL

7. Logon as "studentxxx_B" and set the password if requested.

8. As "studentxxx_B", execute the following SQL statements and indicate if SELECT is allowed or not.

SELECT COUNT(*) FROM Orders;	Permitted or not? No
SELECT COUNT(*) FROM Orders_PPI;	Permitted or not? Yes
DELETE Orders_PPI;	Permitted or not? No

Lab Solutions for Lab 47-2 (cont.)

9. As "studentxxx_B", use the DBC.RoleMembersVX and DBC.UserRoleRightsV views to view the current role of the user, any nested roles, and access rights for the roles.

```
SELECT *        FROM  DBC.RoleMembersVX;

SELECT          RoleName, DatabaseName, TableName, ColumnName, AccessRight
FROM            DBC.UserRoleRIghtsV
ORDER BY        1;
```

How many roles are available to studentxxx_B? 2

What is the default role for studentxxx_B? Role2_xxx

Does studentxxx_B have the "With Admin" option on any roles? No

How many user role rights are available to studentxxx_B? 1

10. As "studentxxx_B", use the SET ROLE command to set the current role to "Role3_xxx".

```
SET ROLE Role3_130;
```

SELECT COUNT(*) FROM Orders; Permitted or not? No

SELECT COUNT(*) FROM Orders_PPI; Permitted or not? Yes

DELETE Orders_PPI; Permitted or not? Yes

11. Log off as "studentxxx_A" and "studentxxx_B". Using your initial user logon name, DROP the two users and the profile you created.

```
DROP USER student130_A;   DROP USER student130_B;   DROP PROFILE student130_P;
```

Lab Solutions for Lab 49-1

Lab Exercise 49-1 Solutions

1. What system security defaults are in effect for your system? SELECT * FROM DBC.SysSecDefaults;

Number of days to expire password:	0	
Minimum number of characters required:	3	
Maximum number of characters required:	15	
Are digits allowed?	Yes X	No____
Are special characters allowed?	Yes X	No____
Maximum failed logons permitted (0=never lock):	3	
Minutes to elapse before unlocking:	5	
Days to expire before password reuse:	0	

2. Are these the security defaults that are in effect for your username? Yes or *No*.

   SELECT * FROM DBC.ProfileInfoVX; (minimum password length and unlock times are different)

3. Is a Profile in effect for your username? If so, what is the name of your Profile? *Student_P*

   SELECT Profile;

4. If a Profile is being used, which attributes in the Profile override the system security defaults?

   Minimum Password Length, Password Special Characters, Password Restricted Words

Lab Solutions for Lab 49-1 (cont.)

5. Using the DBC.LogOnOffV view, list the "BAD" logon attempts on your system that have occurred during the last ten days. Qualify the SELECT using LIKE 'BAD%'.

   Number of Bad Logons (System) 148 (Answers will vary)

```
SELECT    COUNT(*)
FROM      DBC.LogonoffV
WHERE     EVENT LIKE 'BAD%'
AND       LOGDATE > CURRENT_DATE - 10;
```

   Number of Bad Logons (Your UserName) 2 (Answers will vary)

```
SELECT    COUNT(*)
FROM      DBC.LogonoffV
WHERE     EVENT LIKE 'BAD%'
AND       LOGDATE > CURRENT_DATE - 10
AND       UserName = USER;
```

6. Using the DBC.SessionInfoV, list the sessions currently logged on your system.

   Total number of Sessions (System) 28 (Answers will vary)
   Total number of sessions (Your UserName) 2 (Answers will vary)

```
SELECT COUNT(*)  FROM DBC.SessionInfoV;
SELECT COUNT(*)  FROM DBC.SessionInfoV WHERE UserName = USER;
```

Lab Solutions for Lab 49-2

Lab Exercise 49-2 Solutions

Tasks

1. **Using the DBC.AccLogRules view, list the access log rules that are in effect for your username.**

```
SELECT * from DBC.AccLogRules WHERE Username=USER;
```

Which codes are being logged and what type of logging is being captured?

	Code	Type of Logging	SQL Function Being Logged
Ex.	*CDB*	*B +*	*Create Database*
	CTB	*B +*	*Create Table*
	CUS	*B +*	*Create User*
	DDB	*B +*	*Drop Database*
	DTB	*B +*	*Drop Table*
	DUS	*B +*	*Drop User*
	EXE	*E*	*Execute*
	CPO	*B +*	*Create Profile*
	DPO	*B +*	*Drop Profile*
	CTR	*B +*	*Create Role*
	DTR	*B +*	*Drop Role*

2. **How many different access logging rules are there for all users?** Count = 91 (answers will vary)

```
SELECT COUNT(*) from DBC.AccLogRulesV;
```

Lab Solutions for Lab 49-2 (cont.)

3. Execute the following statement.

   CREATE DATABASE Test130 FROM DBC AS PERM=0;

   (this command should fail – 3524: access right violation)

4. Using the DBC.AccessLog view, list the access log entries for the last two weeks for your username.

   How many entries are in this log have been granted? 257 (answers will vary)

   ```
   SELECT COUNT(*) FROM DBC.AccessLog WHERE "AccLogResult" = 'G' AND Username=USER
   AND Logdate > Current_Date - 14;
   ```

   How many entries are in this log have been denied? 1 (at least one; answers will vary)

   ```
   SELECT COUNT(*) FROM DBC.AccessLog WHERE "AccLogResult" = 'D' AND Username=USER
   AND Logdate > Current_Date - 14;
   ```

   What is the difference between the Create Table and the Execute command log entries?

   SQL text is not captured for Execute commands.

Lab Solutions for Lab 49-2 (cont.)

5. Using the DBC.DBQLRulesV view, list the attributes of query log rule that is in effect for your username.

```
SELECT * from DBC.DBQLRulesV WHERE Username=USER;
```

Explain Text Logged	*F*	
Objects Logged	*F*	
Full SQL Logged	*F*	
Execution Steps Logged	*F*	
Summary	*F*	If Summary, times ____ ____ ____
Threshold	*T*	If threshold, time = *100 millisec.*
SQL Text Size	*200*	
Type of Criterion	*CPU time*	

6. Using the DBC.QryLogV view, list the logged queries for your username and how many are there?

```
SELECT COUNT (*) FROM DBC.QryLogV WHERE Username = USER;
```

Count = *11 (answers will vary)*

Using this view, how many queries are logged for all of the users with usernames like 'student%'?

```
SELECT COUNT (*) FROM DBC.QryLogV WHERE Username LIKE 'student%';
```

Count = *445 (answers will vary)*

Lab Solutions for Lab 49-2 (cont.)

7. **(Optional) Using the DBC.QryLogSummaryV view, what is the count of queries that have been executed for your username? (Hint: Join the DBC.QryLogSummaryV to the DBC.QryLogV)**

```
SELECT      SUM(S.QueryCount)
FROM        DBC.QryLogSummaryV S
INNER JOIN  DBC.QryLogV Q
ON          S.UserID = Q.UserID
WHERE       Q.UserName LIKE 'student140%';
```

Count = *1967 (answers will vary)*

(Optional) Using this view, what is the count of queries that have executed for all of the users with usernames like 'student%'?

```
SELECT      SUM(S.QueryCount)
FROM        DBC.QryLogSummaryV S
INNER JOIN  DBC.QryLogV Q
ON          S.UserID = Q.UserID
WHERE       Q.UserName LIKE 'student%';
```

Count = *213,646 (answers will vary)*

Notes

TERADATA
Raising Intelligence

Appendix F: Miscellaneous System Utility and Platform Details

This Appendix contains details on ...

- System utilities and MultiTool
- 15xx and 25xx Appliances
- 48xx/52xx Systems
- 49xx/53xx Systems
- 5400/5450 Systems
- 5500/5550/5555 Systems
- 5600/5650 Systems
- BYNET
- Disk Arrays
- Server Management 2nd and 3rd Generation

Table of Contents

Locations from which Utilities can be Executed F-4
Teradata Console Task F-6
Teradata MultiTool (Windows and Linux) F-8
DB Window via MultiTool F-10
PUT and Disk Arrays F-12
Making Sense of the Different Platforms F-14
MP-RAS Coexistence Combinations F-16
Linux Coexistence Combinations F-18
LSI Logic – 6843 Disk Array F-20
Disk Array Providers F-22
6844 Disk Array F-24
SMP Servers F-26
Teradata 2500 Appliance F-28
Teradata 255x Appliance F-30
Teradata 2500/255x Cabinets F-32
Teradata 1550 Appliance F-34
Teradata 1550 Cabinets F-36
Teradata 255x and 2580 Appliances F-38
Teradata 2580 Appliance F-38
Teradata 2500 Appliance F-38
Teradata 1600 Appliance F-40
Teradata 1550 (not shown) F-40
Teradata 255x and 1600 Cabinets F-42
Appliance Configuration Examples F-44
Appliance Configuration Examples (cont.) F-46
4800/4850 and 5200/5250 Systems F-48
4851/4855 and 5251/5255 Systems F-50
4900 and 5300 Systems F-52
4980 and 5380 Systems F-54
4980/5380 Processing Node F-56
4980/5380 System and Expansion Racks F-58
Example 1: 4980 System F-60
Example 2: 5380 System – 8 Nodes F-62
Example 3: 5380 System – 16 Nodes F-64
Example 4: 5380 System – 32 Nodes F-66
5400 Processing Node F-68
5400/5450 Systems F-70
5400/5450 Cabinets F-72
Teradata 555x Systems F-74
Example of 5500C Coexistence with a 5380 ..... F-76
5500 Teradata Configuration Examples F-78
5555H Example F-80
5555H System – 8 (6+2) Nodes F-82
5555H System – 16 (12+4) Nodes F-84
5555H System – 32 (24+8) Nodes F-86
Teradata 5600 Systems F-88
Examples of Teradata 555x and 56xx Cabinets F-90

5600H and 6844 Disk Arrays....F-92
5600H System – 9 (6+3) Nodes....F-94
Teradata 5650 Systems....F-96
Example of Teradata 5650 Cabinets....F-98
5650H and 6844 Disk Arrays....F-100
5650H System – 9 (6+3) Nodes....F-102
5650H System – 18 (12+6) Nodes....F-104
5650H System – 36 (24+12) Nodes....F-106
Teradata Configuration Examples....F-108
What is BYNET Version 2.1?....F-110
BYNET Interface Cards (BIC) or Adapters....F-112
BYNET Switches....F-114
BYNET Switches (BYA64GX and BYB64G)....F-116
BYNET™ Software....F-120
Enterprise Storage Solutions....F-122
SMP Connectivity – Fibre Channel....F-124
6841-2456 Fibre Channel Disk Array....F-124
LSI Logic – Models 1000, 2000, & 3000....F-126
5380 and 6841-2456 Disk Arrays....F-128
Server Management (2nd Generation)....F-130
Server Management with AWS....F-132
Server Management (3rd Generation) – SM3G....F-134
SM3G Architecture Description....F-136
SM3G Fault Tolerance....F-140

Locations from which Utilities can be Executed

The Teradata software includes two different types of utilities: host-based utilities and AMP-based utilities.

Host-based Utilities

Host-based utilities are programs/applications that you install on a host system. The term "host" may refer to a channel-attached host or a network-attached host. Host-based utilities run under the host operating system.

The Archive and Recovery Utility (ARC) and DUMP Unload/Load Utility (DUL, DULtape) are discussed later in this course.

AMP-based Utilities

You initialize AMP-based utilities using the Teradata Database Window. A console interface called Host Utility Console (HUTCNS) is a host-based utility that runs on a channel-attached mainframe and provides access to a number of AMP-based utilities.

The diagram on the facing page shows AMP-based utilities available through HUTCNS on a channel-attached mainframe and those available through the database window.

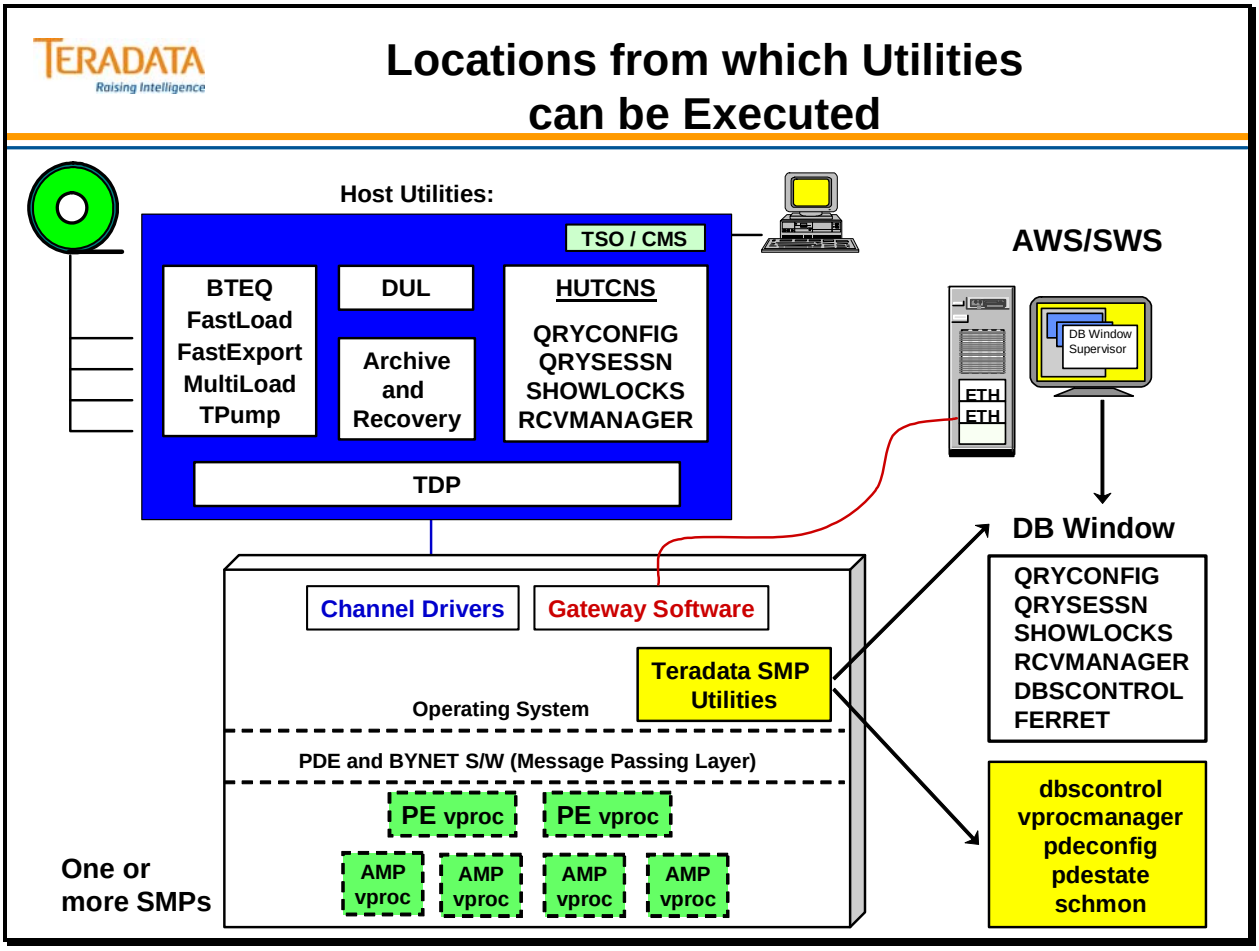
Example of Utilities available through HUTCNS:

Session Status	(QRYSESSN)	enter SES
Configuration Display	(QRYCONFIG)	enter CON
Locks Display	(SHOWLOCKS)	enter LOC
Recovery Manager	(RCVMANAGER)	enter RCV

Examples of Utilities available through the Teradata DB Window:

Ferret → Showspace	Displays, by AMP vproc or by Vdisk, the number of physical cylinders in use for permanent table storage and spool, as well as the average percent cylinder utilization.
Ferret → Packdisk	Packs system and user data stored on permanent data cylinders to a specified density. You can select one or more vprocs to pack.
Abort Host	Aborts all outstanding transactions running on behalf of a host that has crashed.

Locations from which Utilities can be Executed



Teradata Console Task

The Teradata Console (CNS) task is responsible for managing the Teradata DB Window.

There are 3 ways in which the CNS task can be invoked.

- Xdbw (Teradata Database window)
- /usr/ntos/bin/cnsterm (command-line interface)
- /usr/ntos/bin/cnstool (command-line interface)

cnstool and cnsterm commands

cnstool and **cnsterm** are command-line interfaces to the Teradata DB Console functions. **cnstool** is available with both UNIX and Windows 2000 systems. **cnsterm** is only available with UNIX MP-RAS systems.

Starting cnstool

After executing **cnstool**, commands directed to the Supervisor or any application area have to be preceded with the appropriate number. Window numbers 1 through 4 are the console utility windows, 5 is the Database I/O window, and 6 is the Supervisor Window.

Only the root user is allowed to use this command: **# cnstool**

For example, to enter a Supervisor command such as get version: **6:get version**

To start a utility such as qrysessn, enter the following:

6: start qryconfig (assume qryconfig is started in area 1)
1:offline; (offline is a qryconfig option)

To exit from cnstool, enter either **Del** or **Control C.**

Starting cnsterm

When you start cnsterm, the only command line option is the window number.

Only the root user is allowed to use this command: **# cnsterm *n***

where ***n*** is the window number.

For example, to display the supervisor: **# cnsterm 6**

To exit from any screen, press the keyboard's "break" or interrupt key. (You can check your interrupt key setting with the UNIX command, stty.) The interrupt key is often set to **Del** or **Control C**

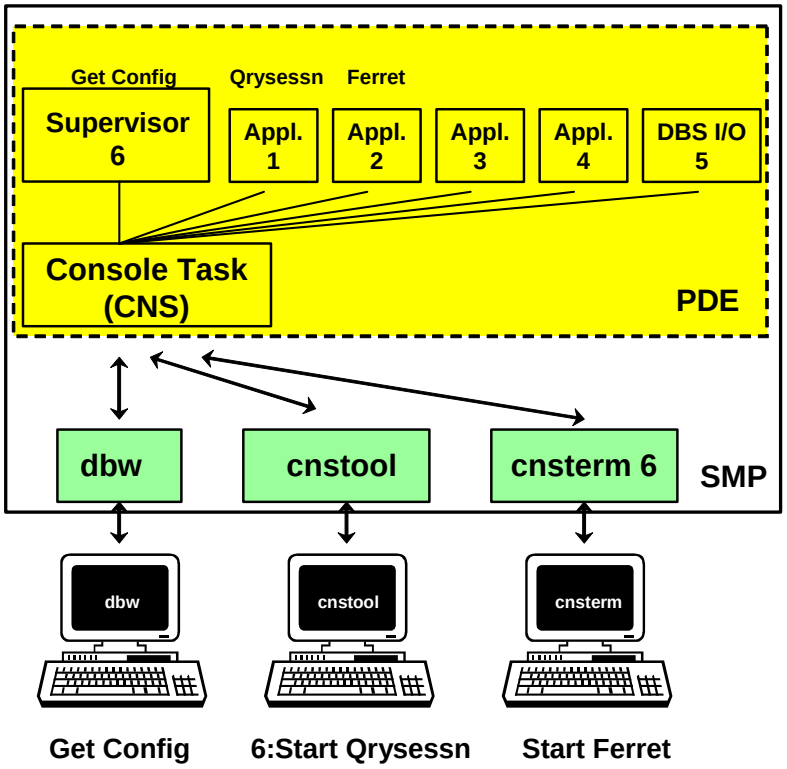
Teradata Console Task

The Supervisor is accessed via ...

- **dbw (Database Window)**
- **cnstool (command line)**
- **cnsterm (command line, e.g. Linux)**

Note: With Linux, use tdatcmd to set the PATH for Teradata command-line utilities.

```
# tdatcmd
# vprocmanager
```



Teradata MultiTool (Windows and Linux)

Teradata MultiTool provides a Graphical User Interface (GUI) on Windows and Linux that provides Teradata administrators and support personnel with a Windows interface to command-line-based Teradata and PDE tasks. The interface is additional to the existing user interfaces.

Teradata MultiTool is a Graphical User Interface (GUI) that you can use to start specific utilities. You can also start many utilities from the Supervisor Window within Teradata MultiTool.

To start Teradata MultiTool, do the following:

Start → Programs → Teradata RBDMS → Teradata MultiTool

PDE State Arrow

State information is received when you execute a pdestate -a command. In general, the following applies:

- An upward-pointing green arrow indicates a component is UP.
- A downward-pointing red arrow indicates a component is DOWN.
- An animated green or red arrow indicates that the PDE is in transition.

DBS State Arrow

In the DBS area, the following applies:

- An upward-pointing green arrow indicates a component is UP.
- A downward-pointing red arrow indicates a component is DOWN.
- An animated green or red arrow indicates that the Teradata RDBMS is in transition.

In addition, an upward-pointing arrow indicates the following:

- Green indicates that logons are enabled.
- Red indicates that logons are disabled.
- Three faces indicate that users are logged on. If no users are logged on, the faces are not present, and the RDBMS is quiescent.

Applications (Tools) that can be started via MultiTool

- Control GDO Editor (CTL) display and modify the fields of the PDE GDO (Globally Distributed Object).

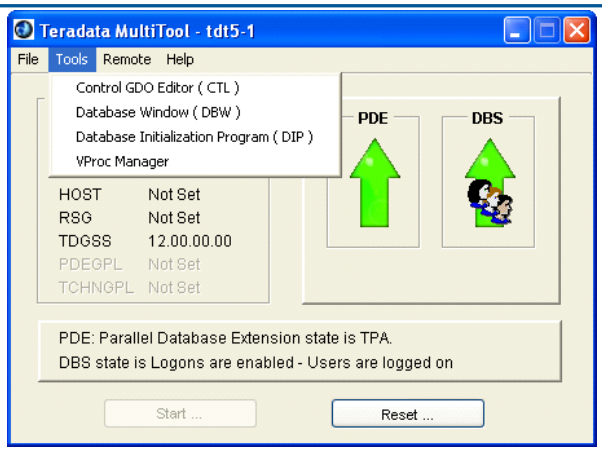
- Database Window (DBW) - activate the Teradata DB Window

- Database Initialization Program (DIP) executes one or more of the standard Database Initialization Program (SQL) scripts packaged with Teradata.

- Vproc Manager – numerous support functions such as change vproc states, initialize and boot a specific vproc, etc.

Teradata MultiTool (Windows and Linux)

Example of Main Window of MultiTool



The Teradata MultiTool utility (Windows and Linux) can be used to …

- **Start and reset Teradata**
- **Check status and versions of Teradata software**
- **Initiate various support applications**
  - **Control GDO Editor (CTL) - same function as xctl**
  - **Database Window (DBW) - activate the Teradata DB Window**
  - **Database Initialization Program (DIP) scripts**
  - **Vproc Manager - change state of vprocs, initialize a Vdisk, restart Teradata**
- **Connect to a remote Teradata node**

DB Window via MultiTool

The DBW may also be started via the MultiTool program.

In the Teradata MultiTool main window, select Tools -> Database Window (DBW) and the DBW should appear.

You can create files that log all output that appears in the Supervisor window or any of the application windows. These logs might be useful when you want to review or print information.

When you log all windows, standard log files are opened. If a log file already exists, the system overwrites the old log with the new log.

To start a log, Select the “File” menu and then select “Enable Logging”. The Select Logging File dialog appears and complete the dialog box.

The following list identifies the default file names, which are located in the *drive*:\Program Files\NCR\LPDE directory, where *drive* is the drive where you installed the default files.

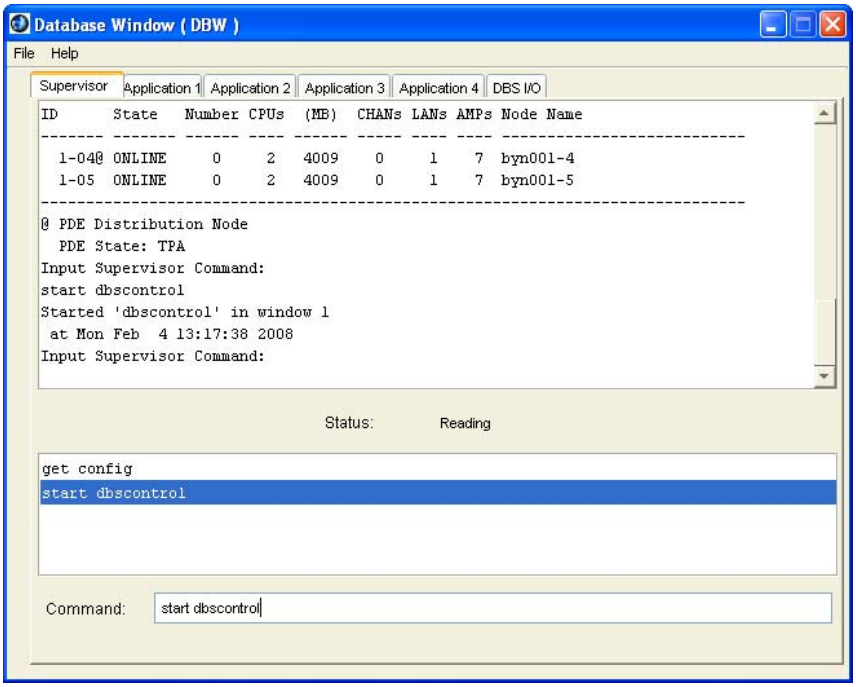
- Supervisor – SupvLog
- Application window 1 – App1Log
- Application window 2 – App2Log
- Application window 3 – App3Log
- Application window 4 – App4Log

DB Window via MultiTool

This example shows the Teradata DB Window supervisor function started via MultiTool.

The various functions appear as tabs within the window.

This example shows execution of a simple command (get config) and starting a utility.



From Supervisor, enter: START DBSCONTROL

PUT and Disk Arrays

The **PUT** (Parallel Upgrade Tool) utility is used to configure a Teradata Database environment. This utility performs many functions in the configuration of a Teradata Database system. One of the key functions is to scan a system for disk arrays and configure those disk arrays for use with the Teradata Database.

The chart on the facing page shows the **PUT** default configurations for different disk arrays.

If the default configuration is not desired, then either Symplicity or ACE utilities can be used to manually configure an array.

Notes:

RAID 1 – Classic mirroring (no striping); commonly used with the Teradata Database
RAID 1 + 0 – Striped Mirroring
RAID 5 – commonly used with the Teradata Database with 4, 9, 18, 36, or 73 GB disk drives

PUT and Disk Arrays

The PUT (Parallel Upgrade Tool) utility is used to configure a Teradata Database environment.

- One of its key functions is to scan a system for disk arrays and configure those disk arrays for use with Teradata. Depending on the type of array, RAID Level, size and number of disks, this utility will configure arrays differently.
- With Teradata Virtual Storage (discussed in another module), Storage Initialization & Profiling is called from PUT before adding new devices.
- This chart lists some of the configuration defaults (not an inclusive chart).

# of disks in array	RAID Level	Max # of Disks	Disks/ Group	# of Groups	Total LUNs	Disk Size (GB) **73**	**146***	**300***
						LUN Sizes (GB)		
40	1	40	2	20	20	73	146	300
40	5	40	4	10	10	219	438	900
52	1	52	2	26	26	73	146	300
52	5	52	4	13	13	219	438	900
60	1	64	2	30	30	73	146	300
60	5	64	4	15	15	219	438	900
64	1	64	2	32	32	73	146	300
64	5	64	4	16	16	219	438	900
108	1	128	2	54	54		146	300

- Note that RAID 5 is rarely used with 146 GB and larger disk drives.

Making Sense of the Different Platforms

The facing page attempts to provide some perspective of the different platforms.

The 4400, 4800, 4850, 5200, and 5250 nodes are based on the Intel Eclipse chassis and Aspen baseboard technology. These nodes are often referred to as Eclipse nodes.

The 4455, 4851, 4855, 5251, and 5255 nodes are based on the Intel Koa baseboard technology. These nodes may be referred to as Koa nodes.

The 4470, 4900 and 5300 nodes are based on the INTEL Dodson baseboard technology and may be referred to as Dodson nodes.

The 4475, 4950 and 5350 nodes are based on the INTEL Hodges baseboard technology and may be referred to as Hodges nodes.

The 4480, 4980, and 5380 nodes are based on the INTEL Harlingen baseboard technology and may be referred to as Harlingen nodes.

The 5400 and 5450 nodes are based on the INTEL Jarrell baseboard technology and may be referred to as Jarrell nodes.

The 155x, 25xx, and 55xx nodes are based on the INTEL Alcolu baseboard technology and may be referred to as Alcolu nodes.

The following dates indicate when these systems were generally available to customers (GCA – General Customer Availability).

– 5100M	January, 1996 (not described in this course)
– 4700/5150	January, 1998 (not described in this course)
– 4800/5200	April, 1999
– 4850/5250	June, 2000
– 4851/4855/5251/5255	July, 2001
– 4900/5300	March, 2002
– 4950/5350	December, 2002
– 4980/5380	August, 2003
– 5400E/5400H	March, 2005
– 5450E/5450H	April, 2006
– 5500E/5500C/5500H	March, 2007
– 2500/5550H	January, 2008
– 2550	October, 2008
– 1550	December, 2008
– 2555/5555C/H	March, 2009
– 1600/2580/5600C/H	March, 2010
– 5650C/H	July, 2010
– 6650C/H and 6680	April, 2011

Making Sense of the Different Platforms

	SMP	MPP	
2003 2004	4475/4480 Intel Xeon 2.8/3.06 GHz	4950/4980 (1 – 4 nodes) Intel Xeon 2.8/3.06 GHz BYNET V2.1	5350/5380 (2 – 512 nodes) Intel Xeon 2.8/3.06 GHz BYNET V2.1
2005 2006	540S Intel Xeon 3.4 GHz	5400/5450E (1 – 4 nodes) Intel Xeon 3.6/3.8 GHz BYNET V2.1	5400/5450H (1–1024 nodes) Intel Xeon 3.6/3.8 GHz BYNET V3.0
2007	550S/550P 1 or 2 Intel Dual-core Xeon CPUs 2.33 GHz	5500E (1 – 2 nodes) 1 or 2 Intel Dual-core Xeon CPUs 2.66 GHz Gbit Ethernet Switch	5500C/H (1–1024 nodes) One/Two Intel Dual-core Xeon CPUs 2.66 GHz BYNET V3.1
2008 2009	551 1 Intel Quad-core Xeon CPU 2.33 GHz		5550/5555H (1–1024 nodes) Two Intel Quad-core Xeon CPUs 2.33 GHz BYNET V3.1/V3.2
2010	560 2 Intel Six-core Xeon CPU 2.93 GHz		5600/5650H (1–4096 nodes) Two Intel quad or six-core CPUs 2.66/2.93 GHz BYNET V4.0

MP-RAS Coexistence Combinations

Teradata provides investment protection by allowing coexistence of different platforms in a single system. Different options are available to the customer - upgrades, expansion, migration, or coexistence.

Coexistence systems contain a mixture of nodes and storage that operate as a single MPP system running the same software. Each system is managed with the same AWS.

All coexistence opportunities must be assessed and approved by Teradata Development Division and the Global Sales Support (GSS) to ensure a proper configuration.

5350 (and later) Coexistence with 5400 and 5450

Coexistence of a 5400/5450 with 5350 and later generations usually makes more sense and is recommended. The power difference between 5350 and 5400 (est. 60%) should result in a manageable number of Vprocs on the 5400 nodes – memory contention should not be an issue.

In addition, a 5400/5450 coexistence system requires a Windows 2003 AWS 4.x that supports SM3G. Additional information on Server Management with coexistence will be covered in the Server Management lesson.

5250/5300 Coexistence with the 5400 and 5450

Although 5400/5450 coexistence is supported back to 5250 (where the 5250 is the oldest node technology in the configuration), it may not be in the customers' best interest to have 5250 through 5300 nodes coexist with 5400/5450 nodes because of sub-optimal parallel efficiency.

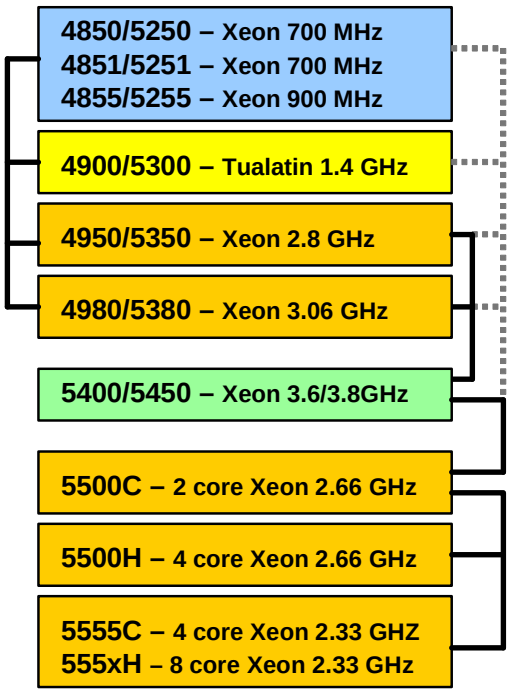
Coexistence & Parallel Efficiency

When planning coexistence, several factors must be evaluated:

- Increasing power differential between oldest and newest nodes in multi-generation coexistence systems usually results in a high number of AMPs (workload) on latest nodes.
- The resulting AMP contention for increasingly limited memory can result in an imbalance – the newest nodes become the pacing nodes and customers do not achieve the expected parallel efficiency (obtain expected value for investment).
- Problem increases with the number of generations being supported.

MP-RAS Coexistence Combinations

Teradata provides investment protection to customers by allowing coexistence of different generations in the same system.



Older generations are supported, but coexistence is more complicated to achieve.

5500C is needed for coexistence with nodes prior to 5500.

32-bit Coexistence with MP-RAS systems

Goal is to have Parallel Efficiency:

Utilization of one set of cliques at 100% and the other sets of cliques as close to 100% as possible.

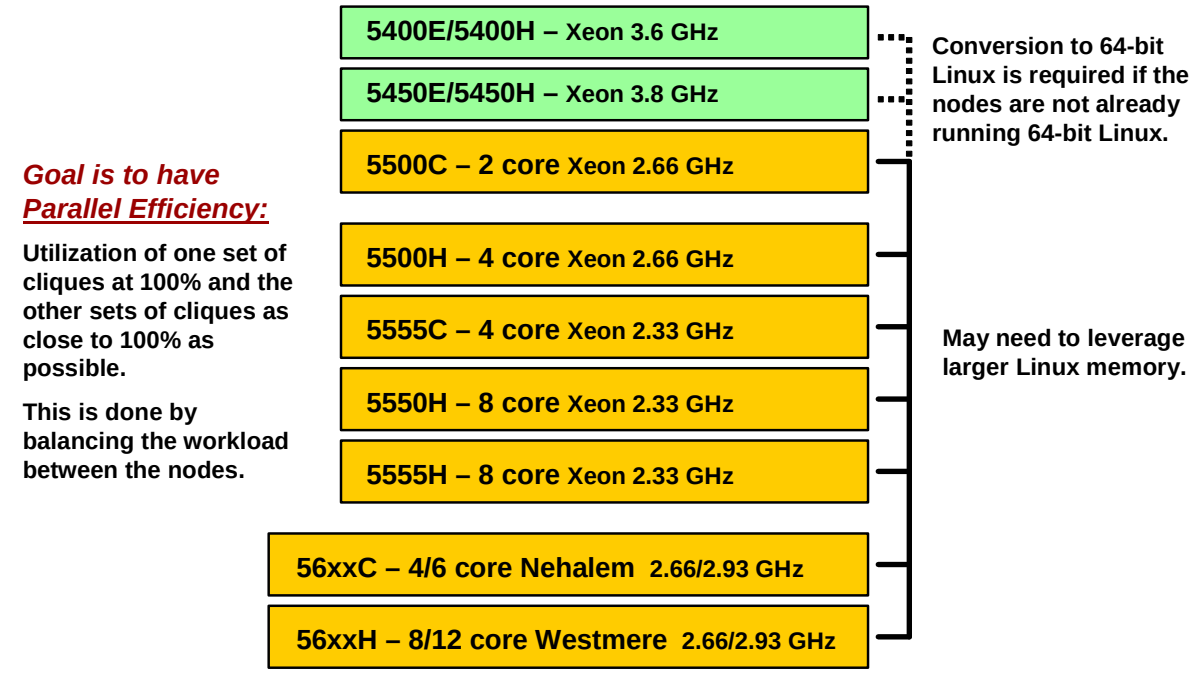
This is done by balancing the workload between the nodes.

Linux Coexistence Combinations

The facing page illustrates possible Linux coexistence combinations.

Linux Coexistence Combinations

In a 64-bit Linux environment, coexistence across multiple generations is supported with 55xx and 56xx systems.



Goal is to have Parallel Efficiency:

Utilization of one set of cliques at 100% and the other sets of cliques as close to 100% as possible.

This is done by balancing the workload between the nodes.

LSI Logic – 6843 Disk Array

The Storage Cabinet (model 6700-4000) supports 1 or 2 6843 disk arrays.

Characteristics of this array include:

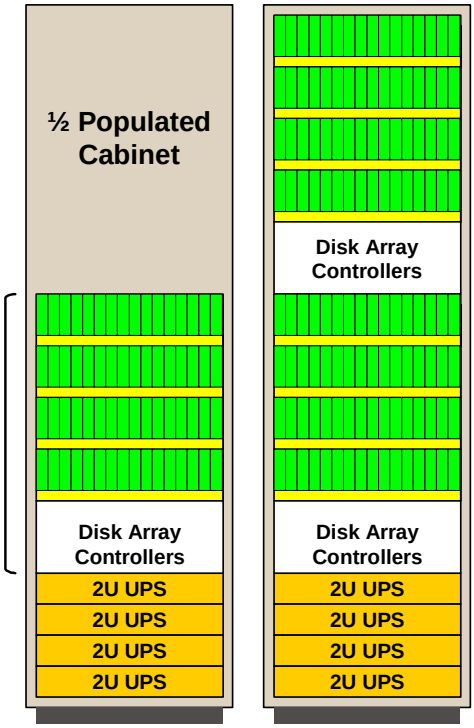
- Each array has one controller module (dual array controllers that support 4 Gbit Fibre Channel speeds.
- Cabinet has two arrays, therefore fewer drives per array.
- 2 arrays per cabinet provide better performance for Teradata.
- Drive trays support up to 16 disks – 73 (older), 146, or 300 GB disk drives.
- Each array supports 4 drive trays – maximum of 64 disks

LSI Logic – 6843 Disk Array

The Storage Cabinet (model 6700-4000) supports one or two 6843 disk arrays.

Common characteristics:

- Each array has one controller module (dual array controllers) that support 4 Gbit Fibre Channel speeds.
- Cabinet has two arrays, therefore fewer drives per array.
- Drive trays support up to 16 disks – 73 (older), 146, and 300 GB disk drives – 15K rpm.
- Each array supports 4 drive trays – maximum of 64 disks



6700-4000

Disk Array Providers

Disk array subsystems are primarily available from two vendors – NetApp (formerly LSI Logic™ Engenio®) and EMC^2 ™.

This chart identifies some of the older releases of LSI Logic (Engenio) Disk Arrays.

Key for this chart: NSC – NCR Storage Cabinet; WES – WorldMark Enterprise Storage

Release	Key Feature	Disk Array Class/Model	O.S. Support	GCA
Enterprise Storage (5.5)	Fibre Channel (2 Gbit)	6841-2456	MP-RAS, Win 2000	08/2003
Enterprise Storage (6.0)	Fibre Channel (2 Gbit)	6841-6456	MP-RAS, Win 2000	08/2003
Enterprise Storage (6.1)	Fibre Channel (2 Gbit; DAP-1)	6841-7456 6842-1000	MP-RAS, Win 2003, Linux	07/2004 05/2005
Enterprise Storage (7.0)	Fibre Channel (4 Gbit; DAP-3)	6843-1000, 2000, 3000 and 6287	MP-RAS, Win 2003, Linux	07/2006
Enterprise (7.1)	Fibre Channel (4 Gbit; DAP-3)	6843-4000	MP-RAS, Win 2003, Linux	01/2008

Examples of EMC^2 disk arrays for a 54xx system include:

- 6291-1002 (a.k.a., EMC2 Symmetrix DMX-2 1000 M2) – supports up to 144 disks (73 GB); typically used with 1 or 2 nodes and up to 100 drives in Teradata Database configurations. This is a standard single-wide cabinet.
- 6291-2000 (a.k.a., EMC2 Symmetrix DMX-2 2000 M2) – supports up to 288 disks (73 GB); typically used with 3+ nodes and up to 196 drives in Teradata Database configurations. This is a double-wide cabinet.

Although the DMX models support up to 144/288 drives and Teradata systems may have as many 100/196 drives, typically only 100 (96 + 4) or 196 (192 + 4) of the drives are used in Teradata Database configurations. 80 or 160 drives are used for AMP storage. The 4 additional drives are used as following: 2 for hot or dynamic spares and 2 drives (volumes) are needed by EMC^2 software and work space.

Disk Array Providers

Two primary storage vendors:

- **NetApp – (LSI Logic, Inc.™ sold its Engenio® external storage systems in 2011)**
- **EMC²™**

Examples of NetApp or Engenio® disk arrays are:

- **6844 (uses 6701-2000 storage cabinet)**
  - **supports up to 128 HDD per disk array; 128 drives per rack; 16 drives/tray**
- **3650 (Solid state disk array in 6680 cabinet)**
  - **supports up to 12 SSD per array;**

Example of an EMC²™ disk array is:

- **EMC² Symmetrix DMX-4 disk array (Model 4500) with 192 disk drives (+ 4 hot spare)**
  - **Design center configuration – each 5600 node accesses 192 drives configured with 480 pdisks used by 40 AMPs**

Note: This presentation will provide examples of NetApp disk arrays.

6844 Disk Array

The Storage Cabinet (model 6701-2000) supports one 6844 disk array.

Characteristics of this array include:

- Cabinet has one array, 30+% faster than 6843.
- Drive trays support up to 16 disk drives – 146, 300, 450, and 600 GB disk drives – 15K rpm.
- Available in three configurations:
  - 4-tray (up to 64 drives)
  - 6-tray (up to 96 drives)
  - 8 tray (up to 128 drives)
- Not supported with UNIX MP-RAS systems.
- Hardware features and characteristics
  - Integrated Fibre Channel harness
  - Integrated AC power harness
  - Integrated Ethernet harness
  - Two AC boxes
  - 30A, 32A, & 50A models

The following chart compares the 6843 (previous model) and 6844 arrays.

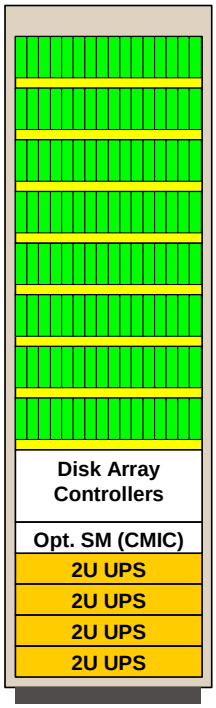
Feature	**6843**	**6844**
Data Protection	DAP1 & 3 (Controller & Drives)	- *same* -
Drive Ports	4 per controller	8 per controller
Max Host Ports	4 per controller	8 per controller
Host-Ports Expandable	No	Yes – via Host-Interface Cards (HICs)
Daisy-Chained Drive Trays	Yes	No

LSI Logic – 6844 Disk Array

The Storage Cabinet (model 6701-2000) supports one 6844 disk array.

Characteristics:

- Each array has one controller module (dual array controllers).
- Cabinet has one array, 30+% faster than 6843.
- Drive trays support up to 16 disk drives – 146, 300, 450, and 600 GB disk drives – 15K rpm.
- Available in three configurations:
  - 4-tray (up to 64 drives)
  - 6-tray (up to 96 drives)
  - 8 tray (up to 128 drives)
- Drive and host ports per controller is 8 versus 4 for 6843
- Drive trays are not chained together as 6843
- AC Power
  - Two 50 AMP Domestic (USA)
  - Four 30 AMP Domestic (USA)
  - Four 32 AMP, three phase international
- Not supported with UNIX MP-RAS systems.



6701-2000

SMP Servers

The following servers are entry-level data warehousing system servers. These systems can execute the UNIX MP-RAS, Windows Server 2003, or SUSE Linux operating systems.

These servers, when combined with the Teradata Database, external storage, connectivity features and applications, provide a complete entry-level data warehousing environment.

4480 Server

The base components of the 4480 server (not shown) are the same as the 4980/5380 processing node. Characteristics include:

- Two Intel XEON CPUs – 3.06 GHz with 512KB cache, 533 MHz front side bus
- Memory – 1 GB to 6 GB (Teradata maximum is 4 GB)
- I/O Slots - 6 PCI slots:
  - Two 64-bit/100 MHz slots (PCI Bus 1, slots 1, 2)
  - Three 32-bit/33 MHz slots (PCI Bus 0, slots 3, 4, and 5)
  - One 64-bit/133 MHz slot (PCI Bus 2, slot 6)
- Drive Bays
  - 1 flex drive (3.5")
  - 2 removable media bays (5.25”) populated with a CD-ROM drive and a tape drive
  - 5 hot-pluggable disk bays housing two 73 GB hard drives standard and an optional three disk drives

540S Server

The 540S server is also an entry-level data warehousing system server. The 540S platform is based on the Dell 2800 server. Characteristics include:

- Two Intel XEON processors – 3.4 GHz with 1 MB cache, 800 MHz front side bus
- Memory: 1 GB to 4 GB (6 GB for Linux)
- Two 73 GB Seagate hard drives (10K RPM)
- Tape drive
- Available adapters:
  - Fibre channel disk storage: LSI Logic 2 Gb PCI-X quad port (LSI7402XP)
  - Fibre channel tape storage: LSI Logic 2 Gb PCI-X quad port (LSI7402XP)
  - Teradata IBM host channel: PCI-X ESCON (PXSA4), FICON (PXFA)
  - Network communication: Ethernet 10/100/1000T copper dual port, 1 Gb Ethernet fiber single port
  - PCI single port serial card: SIIG CyberSerial PCI (JJ-PO1012)

550 and 551 S/P Servers

The facing page also illustrates the 550S, 550P, and 551P servers and the special rack used for these servers.

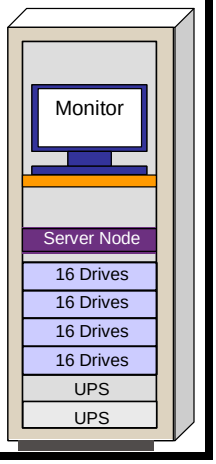
SMP Servers

SMP Servers

- Designed as an entry-level Teradata data warehousing servers

540S Server – pedestal or deskside server

- 540S is based on Dell server
  - Utilizes two Intel® Xeon® 3.4 GHz CPUs; 800 MHz FSB
- Utilizes external deskside disk arrays – e.g., 6287 (July, 2006)



550S/550P/551P Servers

- Single node, disk storage (drive trays), monitor, and 2 UPS in a rack (no AWS)
  - 550S – one Intel® Dual-core Xeon® 2.33 GHz CPU
  - 550P – one or two Intel® Dual-core Xeon® 2.66 GHz CPUs
  - 551P – one Intel® Quad-core Xeon® 2.33 GHz CPUs

Teradata 2500 Appliance

There is only one Teradata configuration for the 2500. Each disk array will be fully populated with 32 disks and each node is configured with 32 AMPs.

Miscellaneous 2500 Notes

- Smallest System: 2 Nodes - 1 Base cabinet
- Largest System: 48 nodes - 1 Base cabinet and 23 Expansion cabinets.
  - Expansions must be added in 2 node increments.
- Each 2500 system will have a Teradata AWS.
- The 2500 is released as a Linux only platform.

Down Node Protection

In a 2500 cabinet, a node is cabled to only one array (no cliques), therefore a node failure on the 2500 platform will cause the AMPs assigned to the failed node to become offline. The 2500 platform can use fallback protection to enable the system to run with offline AMPs.

Utilities considerations for the 2500 are:

- **Included in Bundled 2500 Price**

  TPT Load and Export Operators, ODBC, JDBC, CLI, OLE DB Provider, .NET Data Provider, Plug-in for Eclipse, SQL Assistant, SQL Assistant/Web Edition, BTEQ, Teradata Administrator, Meta Data Services, Data Connector, and MultiTool

- **Optional**

  TPT Update Operator, TPT Operator Stream, Teradata Manager, Teradata Warehouse Miner, Teradata Analyst Pak, Query Director, Teradata Replication Services

- **Available only to existing users**

  FastLoad, MultiLoad, Fast Export, and TPump

- **Limited or restricted use software**

  Priority Scheduler and “schmon”

- **Not supported with the 2500**

  TASM, Workload Analyzer, Dynamic Workload Manager, Mainframe Channel Connection

Teradata 2500 Appliance

- **Teradata Data Warehouse Appliance 2500**
  - Teradata 6.2.1, 64-bit Novel SUSE Linux 9, and bundled utilities
  - Typical configuration – 32 AMPs and 2 PEs per node
  - Configured as single-node cliques
- **Fully-integrated cabinet design**
  - **Two Nodes**
    - Each node has 2 Intel Dual-core CPUs at 2.66 GHz; total of 8 cores in the cabinet
    - 8 GB of memory per node; total of 16 GB memory in cabinet
    - 32 AMPs per node with shared LUNs
  - **Two Disk Arrays** – total of (64) 300 GB enterprise-class Fibre-Channel drives
    - 32 Drives per Node; RAID1 disk mirroring, Fallback is optional
  - 6.12 TB customer data per cabinet (assumes 30% data compression)
  - Scales up to 24 cabinets, 48 nodes, and 146 TB customer data
  - Nodes are interconnected via BYNET software over redundant Ethernet switches
- **Features not available for the Teradata 2500**
  - Multi-node cliques are not supported for automatic failover and recovery
  - RAID 5, hot standby nodes, hot spare disk drives, and dual active is not supported
  - Cannot co-exist with other Teradata systems

Teradata 255x Appliance

There is only one Teradata configuration for the 255x. Nodes are purchased in 2 node increments. Each of two nodes will be assigned to a disk array that is fully populated with 72 disks. Each node is assigned to 36 disks.

The 2550 node uses the Intel Xeon Clovertown CPU (8 MB of level-two cache) whereas the 2555 node uses the Intel Xeon Harpertown CPU. The Harpertown CPU has 12 MB of level-two cache and improved internal pipelines for a performance gain of approximately 5%.

Miscellaneous 255x Notes

- Smallest System: 2 Nodes - 1 Base cabinet
- Largest System: 44 nodes - 1 Base cabinet and 10 Expansion cabinets.
  - Expansions must be added in 2 node increments.
- Each 255x system will have a Teradata AWS.
- The 255x is released as a Linux only platform.

Down Node Protection

In a 255x cabinet, two nodes are configured in a clique. In the event of a node failure, PEs and AMPs can migrate to the remaining node within the clique.

Utilities considerations for the 255x are:

- **Included in Bundled 255x Price**

  TPT Load and Export Operators, ODBC, JDBC, CLI, OLE DB Provider, .NET Data Provider, Plug-in for Eclipse, SQL Assistant, SQL Assistant/Web Edition, BTEQ, Teradata Administrator, Teradata Manager, Meta Data Services, Data Connector, and MultiTool

- **Optional**

  TPT Update Operator, TPT Operator Stream, Teradata Manager, Teradata Warehouse Miner, Teradata Analyst Pak, Query Director, Teradata Replication Services

- **Available only to existing users**

  FastLoad, MultiLoad, Fast Export, and TPump

- **Limited or restricted use software**

  Priority Scheduler and “schmon”

Teradata 2550/2555 Appliance

- **Teradata Data Warehouse Appliance 2550/2555**
  - Teradata 12.0, 64-bit Novel SUSE Linux 10, and bundled utilities
  - Typical configuration – 36 AMPs and 2 PEs per node
- **Fully-integrated cabinet design**
  - Up to 4 Nodes in a cabinet – purchased in 2 node increments
    - Each node has 2 Intel Quad-core Xeon CPUs at 2.33 GHz; nodes are 40% to 45% faster
    - 32 GB of memory per node; maximum total of 128 GB memory in cabinet
  - Up to 2 Disk Arrays in cabinet – total of (144) 300 GB enterprise-class SAS drives
    - 36 Drives per Node; RAID1 disk mirroring, Fallback is optional
  - 12.6 TB customer data per cabinet (assumes 4 nodes and 30% data compression)
  - Scales up to 11 cabinets, 44 nodes, and 140 TB customer data
  - Nodes are interconnected via BYNET software over redundant Ethernet switches
  - Configured with 2 node cliques – provides continuity in event of node failure
  - Optional – Teradata Managed Server within cabinet (e.g., SAS or Viewpoint)
  - Optional – Mainframe channel connectivity (ESCON or FICON) in separate cabinet
- **Features not available for the Teradata 2550/2555**
  - RAID 5, hot standby nodes, hot spare disk drives, and dual active is not supported
  - Cannot co-exist with other Teradata systems

Teradata 2500/255x Cabinets

There are no optional configurations for the 2500 cabinet; every cabinet must be configured as shown on the facing page. The 255x cabinet may have 2 nodes with 1 array or 4 nodes with 2 arrays.

Teradata 2500 Node Characteristics:

- Same basic node as 5500 – includes 2 dual-core 2.66 GHz CPUs with 8 GB of memory per node. Each node is pre-configured with 32 AMPs utilizing shared LUNs in the associated disk array.

- It has all the Server Management and AWS capabilities used in the MPP systems.

Teradata 255x Node Characteristics:

- Same basic node as 555x – includes 2 quad-core 2.33 GHz CPUs with 32 GB of memory per node. Each node is pre-configured with 36 AMPs utilizing shared LUNs in the associated disk array.

- The 2550 node uses the Intel Xeon Clovertown CPU (8 MB of level-two cache) whereas the 2555 node uses the Intel Xeon Harpertown CPU. The Harpertown CPU has 12 MB of level-two cache and improved internal pipelines for a performance gain of approximately 5%.

- It has all the Server Management and AWS capabilities used in the MPP systems.

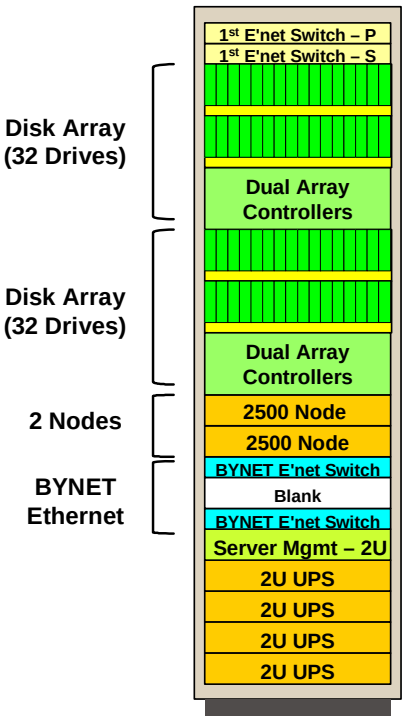
BYNET features of the Teradata 25xx include:

- BYNET switching is provided by redundant, dual-active Gigabit Ethernet Switches.

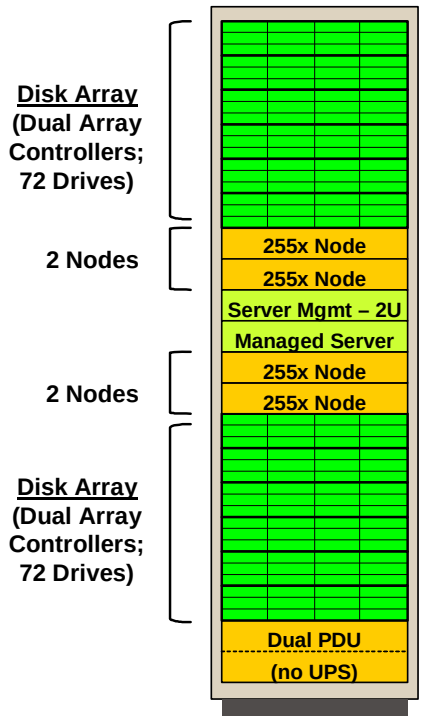
- 25xx processing node on-board Ethernet interfaces are connected to the BYNET Gigabit Ethernet Switches.
  - eth0 (the connector on the right) is connected to BYNET 0.
  - eth1 (the connector on the left) is connected to BYNET 1.

- All connections are copper. Cables are available in 5, 10, 20, and 30 meter lengths.

Teradata 2500/2550/2555 Cabinets



2500 Cabinet



2550/2555 Cabinet

Teradata 1550 Appliance

The Teradata Extreme Data Appliance 1550 allows you to gain deep strategic intelligence from extremely large amounts of detailed data. It supports very high-volume, non-enterprise data/analysis requirements for a small number of power users in specific workgroups or projects that are outside of your enterprise data warehouse (EDW).

This appliance is based on the field proven Teradata Purpose-Built 5550 processing nodes and provides the same scalability and data warehouse capabilities expected from Teradata.

Key features include:

- Extremely large user data capacities with 34 TB per node of uncompressed user data. With a typical 40% compression level, the usable capacity per node expands to 50TB.
- This means that the appliance can support a 1PB data store/analysis need with only 20 Teradata nodes.
- Featuring massively parallel processing (MPP) architecture, the Teradata BYNET® system interconnect with high-speed, fault tolerant, optimized messaging between nodes is a key scalability ingredient.
- The Teradata Extreme Data Appliance has scale out capability with up to 1,024 nodes, thereby enabling the potential of a huge data warehouse of more than 50PB.
- Utilizes a density optimized version of Teradata Storage that is based on industry-standard, commercial high capacity drives and high-performance, industry leading disk array technology.
- Storage consists of a fixed configuration of 124 commercial data disk drives each with 1TB of capacity. The storage array also includes four hot standby drives to ensure minimal impact and data loss exposure from drive failures – an important capability due to the large amount of data in each array.
- Pre-configured – to meet the demands of very large data set analysis with simple-to-order and easy-to-expand increments of a single Teradata node with storage.
- All the indexing capabilities of Teradata Database are included on this platform. These features, such as Partitioned Primary Index, help you efficiently organize and query the very large amounts of data.

Teradata 1550 Appliance

- **Teradata Extreme Data Appliance 1550**
  - Teradata 12.0 and 64-bit Novel SUSE Linux 10
  - Each node is configured with 62 AMPs and each AMP has 1 TB of Vdisk space
  - Intent is to support small number of analytical queries accessing a large amount of data.
- **Processor Node cabinet characteristics**
  - Up to 9 Nodes in a cabinet
    - Each node has 2 Intel Quad-core Xeon CPUs at 2.33 GHz
    - 32 GB of memory per node
  - System can scale up to 1024 nodes and up to 50 PB of
  - Nodes are interconnected via MPP BYNET V3
  - Optional – HSN (Hot Standby Node) in 4 node (3+1) clique environments
  - Optional – Teradata Managed Server within cabinet (e.g., SAS or Viewpoint)
  - Optional – Mainframe connectivity (ESCON or FICON) – requires channel node
- **Enterprise Storage cabinet characteristics**
  - Up to 124 disk drives + 4 HSD (Hot Spare Drives) in a cabinet
  - Drive pairs configured with RAID 1
  - Each disk drive is 1 TB capacity, SATA interface, 7.2K RPM
  - Storage connectivity is 4 Gb Quad Fibre Channel

Teradata 1550 Cabinets

The facing page illustrates the processor and storage cabinets for the Teradata Extreme Data Appliance.

The 1550 cabinet characteristics include:

- One to nine Teradata processing nodes (typical max is 8)
- BYNET switches
- Server management server and network
- UPS, Dual AC distribution, cooling fans
- Patented enhanced airflow

Key characteristics of the 1550 Processing Node include:

- 1550 Processors
  - Up to two Quad Core Intel® Xeon® 5300 Series 2.33GHz processors with 8MB Advanced Transfer L2 Cache Memory
  - 32GB using DDR2 667MHz fully buffered DIMM with ECC for Teradata
- Database running on Novell® SUSE®Linux 64-bit
- Internal Node Data Storage Devices
  - Two hot-swappable 73GB or 146GB SAS® hard drives (four max)
  - One CD/DVD-ROM drive
  - One 4mm 36/72GB tape drive per cabinet (standard)

Key characteristics of the 1550 Enterprise Storage Cabinet

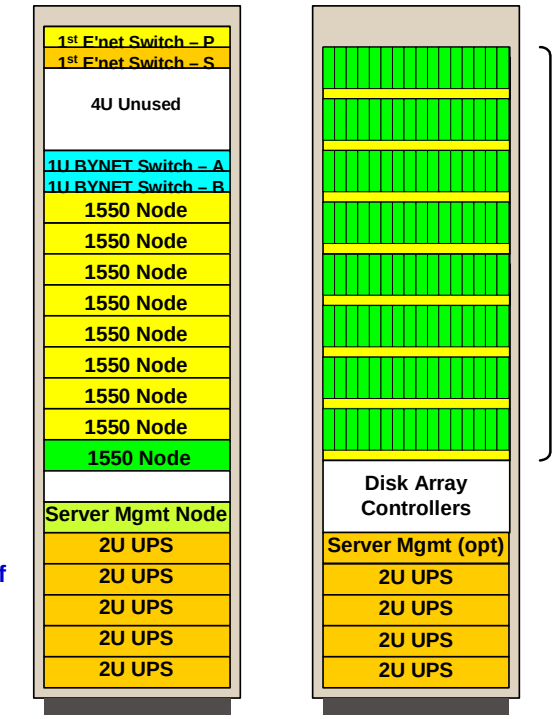
- Disk Drive Supported - SATA interface, 7.2K RPM, 1TB capacity
- Up to 124 drives = 124 TB spinning disk capacity plus 4 host spare disks
- Connects directly to Teradata nodes and certified for operation with Teradata Database by using Quad Fibre Channel Adapters which support 4Gb/sec connectivity between nodes and the multi-ported disk arrays.
- RAID 1 Mirroring

Teradata 1550 Cabinets

Processing Node
- Dual Quad Core CPUs
- 32 GB of memory
- Typically configured with 62 AMPs

One node is effectively connected to a storage cabinet.

One node will have 62 AMPs accessing 62 TB of Vdisk space.



1550 Processor Cabinet 1550 Storage Cabinet

Disk Array
- Dual Array Controllers
- 8 Drive Trays, each with 16 disks
- 124 Drives + 4 HSD
- 1 TB SATA Disk Drives
- 62 RAID 1 drive pairs

Teradata 255x and 2580 Appliances

These systems utilize these Linux nodes that are purchased in 2 node increments. With 255x systems, each of two nodes will be assigned to a disk array that is fully populated with 72 disks. Each node is assigned to 36 disks.

The 2550 node uses the Intel Xeon Clovertown CPU (8 MB of level-two cache) whereas the 2555 node uses the Intel Xeon Harpertown CPU. The Harpertown CPU has 12 MB of level-two cache and improved internal pipelines for a performance gain of approximately 5%.

The smallest system is 2 Nodes - 1 Base cabinet and the largest system is 44 nodes - 1 Base cabinet and 10 Expansion cabinets.

In a 255x cabinet, two nodes are configured in a clique. In the event of a node failure, PEs and AMPs can migrate to the remaining node within the clique.

Teradata 2580 Appliance

Characteristics of a 2580 appliance node include:

- Dell R710 TMS using quad core Intel® Nehalem-EP Processor (Bluefish)
- 96 GB memory – 8 GB DIMMS
- 1 less adapter slot and 2 less onboard network ports
- SATA Tape support in SWS rather than node
- 300 & 450 GB 15K6 SAS drives & 1 TB 7.2K SAS drives
- (1) CMIC based on 55xx node – SMWeb
- Channel Node based on 55xx
- Co-resident with 2550 & 2555

Teradata 2500 Appliance

This is the original Teradata appliance and is not described on the facing page. Characteristics include:

- Fully-integrated cabinet design
  - Two Nodes
    - **Each node has 2 Intel Dual-core CPUs at 2.66 GHz; total of 8 cores in the cabinet; 8 GB of memory per node;**
  - Two Disk Arrays – total of (64) 300 GB enterprise-class Fibre-Channel drives
    - **32 Drives per Node; RAID1 disk mirroring, Fallback is optional**
  - 6.12 TB customer data per cabinet (assumes 30% data compression)
  - Scales up to 24 cabinets, 48 nodes, and 146 TB customer data
  - Nodes are interconnected via BYNET software over redundant Ethernet switches
  - Typical configuration – 32 AMPs and 2 PEs per node
  - Configured as single-node cliques

Teradata 255x and 2580 Appliances

- **Teradata Data Warehouse Appliance (2550, 2555, and 2580)**
  - Fully integrated cabinet design with nodes and disk arrays in the same cabinet
    - Up to 4 Nodes in a cabinet – purchased in 2 node increments
    - Up to 2 Disk Arrays in cabinet with 144 drives – RAID1; Fallback is optional
  - Typical configuration – 36 AMPs and 2 PEs per node
  - Nodes are interconnected via BYNET software over redundant Ethernet switches
- **Teradata 255x Systems**
  - Nodes use 2 Intel Quad-core Xeon CPUs at 2.33 GHz; 2555 nodes are 5% faster
    - 32 GB of memory per node
  - 36 SAS Drives per node - 300 GB @ 15K drives
  - Scales up to 11 cabinets, 44 nodes, and 140 TB customer data
- **Teradata Data Warehouse Appliance 2580**
  - Dell R710 node with 2 Intel quad-core Nehalem CPUs with 96 GB memory
  - Available SAS disk drives – 300, 450 GB @ 15K drives, or 1 TB @ 7.2K drives
  - Optional – up to 2 Dell R710 TMS and 2 channel nodes (based on 55xx nodes)
  - Co-residence with 2550 and 2555 is supported

Teradata 1600 Appliance

The Teradata Extreme Data Appliance 1600 allows you to gain deep strategic intelligence from extremely large amounts of detailed data. It supports very high-volume, non-enterprise data/analysis requirements for a small number of power users in specific workgroups or projects that are outside of your enterprise data warehouse (EDW).

Characteristics of the 1600 are listed on the facing page.

Teradata 1550 (not shown)

The Teradata 1550 consists of separate processor and storage cabinets. For 1 fully populated processor cabinet of 8 1550 nodes, there will an associated 8 storage cabinets.

The 1550 cabinet characteristics include:

- One to nine Teradata processing nodes (typical max is 8)
- BYNET switches
- Server management server and network
- UPS, Dual AC distribution, cooling fans
- Patented enhanced airflow

Key characteristics of the 1550 Processing Node include:

- 1550 Processors
  - Up to two Quad Core Intel® Xeon® 5300 Series 2.33GHz processors with 8MB Advanced Transfer L2 Cache Memory
  - 32GB using DDR2 667MHz fully buffered DIMM with ECC for Teradata
- Database running on Novell® SUSE® Linux 64-bit

Key characteristics of the 1550 Enterprise Storage Cabinet

- Disk Drive Supported - SATA interface, 7.2K RPM, 1TB capacity
- Up to 124 drives = 124 TB spinning disk capacity plus 4 host spare disks
- Connects directly to Teradata nodes and certified for operation with Teradata Database by using Quad Fibre Channel Adapters which support 4 Gb/sec connectivity between nodes and the multi-ported disk arrays.
- RAID 1 Mirroring

Teradata 1600 Appliance

- **Teradata Extreme Data Appliance 1600**
  - Node is based on Dell R710 chassis – 1+1 node in a clique (HSN is optional)
  - BYNET over 1 Gb Ethernet for systems up to 23 TPA nodes (46 including HSN)
    - BYNET v4 for systems > 23 TPA nodes
  - Up to 58 TB of user data in one cabinet
  - Co-residence with 1550 and 1555 systems is supported (with BYNET v4)
  - Uses SWS and SMWeb for server management
- **Processor, Node, and Cabinet characteristics**
  - Up to 4 Nodes in a cabinet (1+1)
    - Each node has 2 Intel Quad-core Nehalem CPUs @ 2.66 GHz with Hyper-Threading
    - 48 GB of memory per node
  - System can scale to 50 PB of user data with BYNET v4
  - Optional – HSN (Hot Standby Node) in 2 node (1+1) clique environments
  - Optional – up to 5 TMS (Teradata Managed Server) nodes (R710 TMS)
  - Optional – up to 5 Channel Server (ESCON or FICON) nodes
- **Storage within 1600 cabinet**
  - Up to 144 disk drives – 72 drives per node – RAID 1 drive pairs
  - Nodes are configured with 36 AMPs – 2 disks per AMP
  - SAS disk drives are 1 TB capacity @ 7.2K RPM

Teradata 255x and 1600 Cabinets

There are no optional configurations for the 2500 cabinet; every cabinet must be configured as shown on the facing page. The 255x cabinet may have 2 nodes with 1 array or 4 nodes with 2 arrays.

Teradata 2500 Node Characteristics:

- Same basic node as 5500 – includes 2 dual-core 2.66 GHz CPUs with 8 GB of memory per node. Each node is pre-configured with 32 AMPs utilizing shared LUNs in the associated disk array.

- It has all the Server Management and AWS capabilities used in the MPP systems.

Teradata 255x Node Characteristics:

- Same basic node as 555x – includes 2 quad-core 2.33 GHz CPUs with 32 GB of memory per node. Each node is pre-configured with 36 AMPs utilizing shared LUNs in the associated disk array.

- The 2550 node uses the Intel Xeon Clovertown CPU (8 MB of level-two cache) whereas the 2555 node uses the Intel Xeon Harpertown CPU. The Harpertown CPU has 12 MB of level-two cache and improved internal pipelines for a performance gain of approximately 5%.

- It has all the Server Management and AWS capabilities used in the MPP systems.

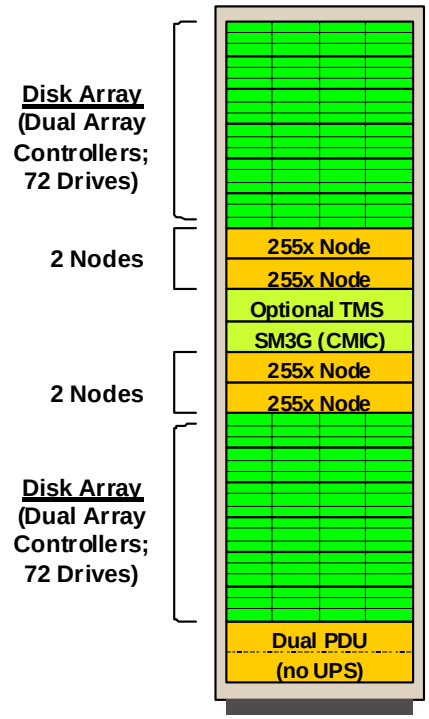
BYNET features of the Teradata 25xx include:

- BYNET switching is provided by redundant, dual-active Gigabit Ethernet Switches.

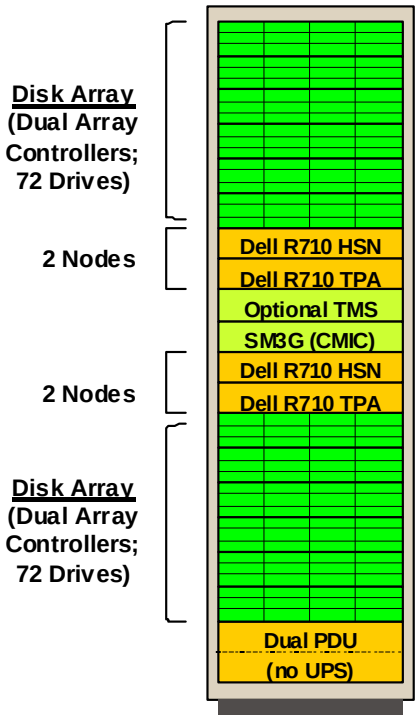
- 25xx processing node on-board Ethernet interfaces are connected to the BYNET Gigabit Ethernet Switches.
  - eth0 (the connector on the right) is connected to BYNET 0.
  - eth1 (the connector on the left) is connected to BYNET 1.

- All connections are copper. Cables are available in 5, 10, 20, and 30 meter lengths.

Teradata 255x and 1600 Cabinets



255x Cabinet



1600 Cabinet

Appliance Configuration Examples

The examples on the facing page show a typical AMP and Disk configurations for 255x, 1550, and 1600 systems.

Notes:

- 2500 systems utilize Fibre Channel disks
- 255x systems utilize SAS disks (Serial Attached SCSI)
- 1550 systems utilize SATA disks (Serial Advanced Technology Attachment)
- 1600 systems utilize SAS disks (300 GB, 450 GB, or 1 TB)

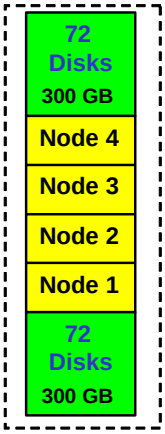
The typical configuration for a 2500 (not shown) is as follows:

- 32 Disks / Node (RAID 1)
- 8 GB Memory / Node
- 32 AMPs / Node
- 1 Node Cliques

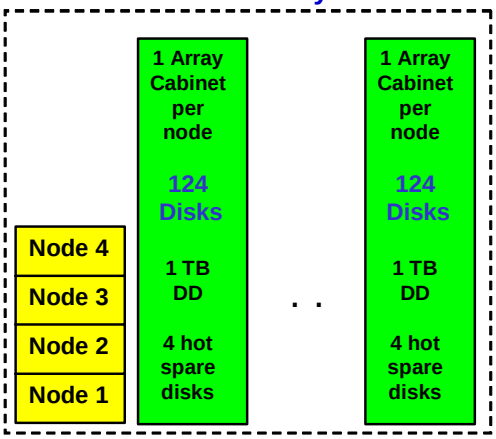
2500 Cabinet effectively has 2 Nodes managing 6.12 TB with 30% data compression.

Appliance Configuration Examples

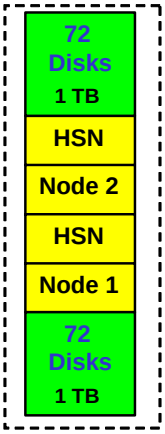
255x, 2580



155x and Disk Array Cabinets



1600



255x, 2580 – Linux

36 AMPs / Node
2 Node Cliques

36 Disks / Node (RAID 1)
32 GB Memory / Node

Cabinet

4 Nodes; 12.6 TB with 30% data compression

155x – Linux

62 AMPs / Node
Cliques optional – require HSN

124 Disks per Node (RAID 1)
32 GB Memory per Node

Each Vdisk – 2 Disks (RAID 1)
Each Vdisk – 1 TB*

Node – 62 AMPs x 1 TB = 62 TB*

1600 – Linux

36 AMPs / Node
2 (1+1) Cliques

72 Disks / Node (RAID 1)
48 GB Memory / Node

Cabinet

2 Nodes; 58 TB with 30% data compression

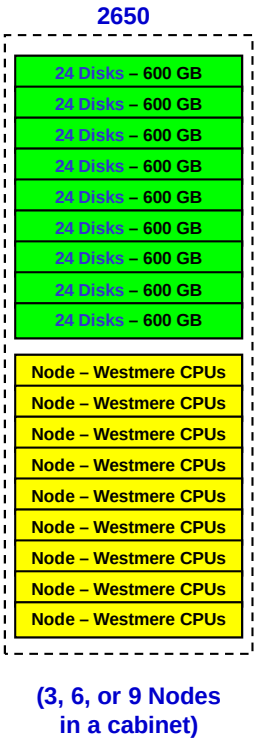
Appliance Configuration Examples (cont.)

The examples on the facing page show a typical AMP and Disk configurations for 2650 and 4600 systems.

Notes:

- 2650 systems utilize SAS disks (Serial Attached SCSI) – 300 GB and 600 GB disk drives
- 2650 systems can utilize 2 TB SATA disks (Serial Advanced Technology Attachment)
- 4600 systems utilize 300 GB SSD (Solid State Disks)

Appliance Configuration Examples (Cont.)



2650 Disk Options

- 300 or 600 GB SAS Disks 2.5" – 216 in cabinet
- 2 TB Disks 3.5" – 108 in cabinet

2650 Clique (e.g., 600 GB)

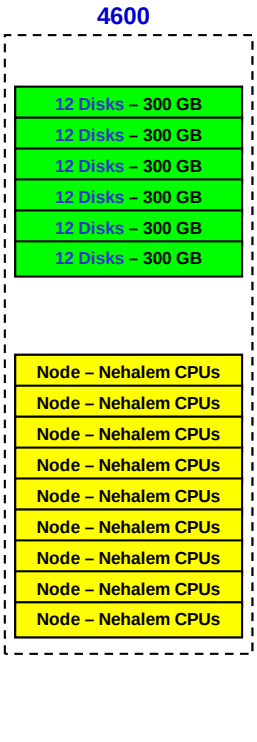
- 3 Node Cliques (3+0) share 3 drive trays
- 96 GB Memory / Node
- 24 AMPs / Node
- 72 AMPs /Clique

- 24 Disks / Node (RAID 1)
- 72 Disks / Clique

- Spinning Disks – 43.2 TB
- Max Perm – 19.4 TB
- Customer Data – 10.7 TB (after DBC and spool space)

2650 Cabinet with 9 nodes

- 216 AMPs
- 864 GB memory in cabinet



4600 Disks

- 300 GB Solid State Drives (SSD)
- 3.5" – 72 SSD in cabinet

4600 Clique (e.g., 600 GB)

- 3 Node Cliques (3+0) share 2 drive trays
- 96 GB Memory / Node
- 32 AMPs / Node
- 96 AMPs /Clique

- 8 Disks / Node
- 24 Disks / Clique

- SSD Space – 7.2 TB
- Max Perm – 6.5 TB
- Customer Data – 1.8 TB (after DBC, Spool and Fallback)

4600 Cabinet with 9 nodes

- 288 AMPs
- 864 GB memory in cabinet

4800/4850 and 5200/5250 Systems

The 4800/4850 and 5200/5250 systems utilize a rack-based cabinet. Like other Teradata systems, the 4800/4850 and 5200/5250 systems are composed of different type of subsystems.

The processing node is housed in an 11U "chassis" which is mounted in rack-based cabinet. A 4800/4850 or 5200/5250 cabinet is capable of housing two processing nodes.

4800 and 5200 Systems

The key component is the SMP node – it is based on the Intel 100 MHz internal bus architecture, uses the Intel® Pentium® II Xeon™ 450 MHz or Intel® Pentium® III Xeon™ 500 or 550 MHz CPUs, and has 3 PCI buses. Simply stated, the 4800/5200 SMP is a faster computing engine than the previous 4700/5150.

4800 and 5200 systems can be upgraded to 4850 and 5250 systems.

4850 and 5250 Systems

The 4850 and 5250 systems are very similar to the 4800 and 5200 systems. The key difference is that 4850 and 5250 SMPs utilize the Intel® Pentium® III Xeon™ 700 MHz CPU.

The 4850/5250 SMP chassis is 11U in height, same height as the 4800/5200 SMP chassis.

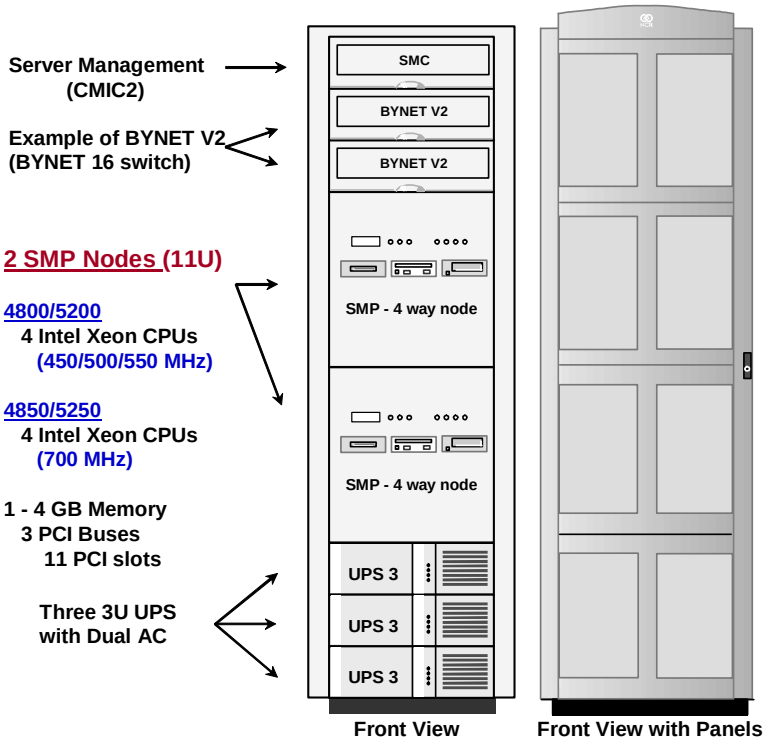
The following table lists the height of each chassis:

Chassis	Height
Server Management	3U (13.3 cm, 5.25 in)
BYA16G	3U (13.3 cm, 5.25 in)
SMP – 4800/4850/5200/5250	11U (48.9 cm, 19.25 in.)
UPS – 4800/4850/5200/5250	3U (13.3 cm, 5.25 in.)

4800/4850 and 5200/5250 Systems

Notes:

- **4800/4850 – up to 4 SMPs and uses BYNET 4 PCI board switches.**
- **5200/5250 – up to 512 SMPs and uses BYNET chassis switches.**
- **SMPs and components are housed in chassis modules which are mounted in a rack-based cabinet.**
- **SMPs use Intel® Pentium® III Xeon™ CPUs, a 100 MHz system bus, and leverage 4400 SMP technology.**



4851/4855 and 5251/5255 Systems

The 4851/4855 and 5251/5255 systems have a different SMP architecture than the 4800/4850/5200/5250 SMPs. The module chassis for these SMPs is 7U in height as compared to 11U for the 4800/4850/5200/5250 SMPs.

4851/5251 Systems

The 4851 and 5251 SMPs utilize the Intel® Pentium® III Xeon™ 700 MHz CPU. The 4851 and 5251 systems will primarily be used in a co-existence environment with existing 4850 and 5250 systems.

4855/5255 Systems

One key performance difference is that 4855 and 5255 SMPs utilize the Intel® Pentium® III Xeon™ 900 MHz CPU. This faster CPU provides approximately a 15% performance gain.

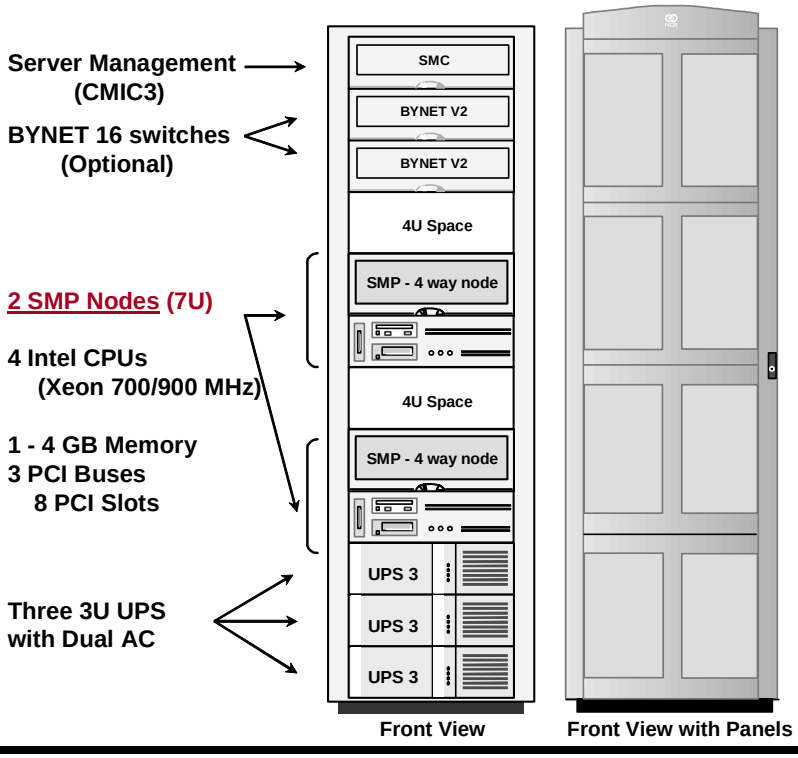
The following table lists the height of each chassis:

Chassis	Height
Service Management	3U (13.3 cm, 5.25 in)
BYA16G	3U (13.3 cm, 5.25 in)
SMP – 4851/4855/5251/5255	7U (31.1 cm, 12.25 in.)
UPS – 4851/4855/5251/5255	3U (13.3 cm, 5.25 in.)

4851/4855 and 5251/5255 Systems

Notes:

- 4851/4855 – up to 4 SMPs; use BYNET 4 V2 switches.
- 5251/5255 – up to 512 SMPs; use BYNET V2 switches.
- SMPs are based on the 4455 SMP architecture and are housed in a chassis that is 7U in height.
- 4855/5255 SMPs use Intel® Pentium® III 900 Xeon™ MHz CPUs, a 100 MHz system bus, and leverage 4455 SMP technology (4851/5251 SMPs use 700 MHz CPUs).



4900 and 5300 Systems

The 4900 and 5300 systems have a new SMP architecture. The module chassis for these new SMPs is 5U in height as compared to 7U for the 4851/4855/5251/5255 SMPs.

The 4900 and 5300 SMPs utilize the Intel® Pentium® III Tualatin 1.4 GHz CPU.

One key difference with this generation is that a cabinet or rack can now hold 4 SMPs rather than two.

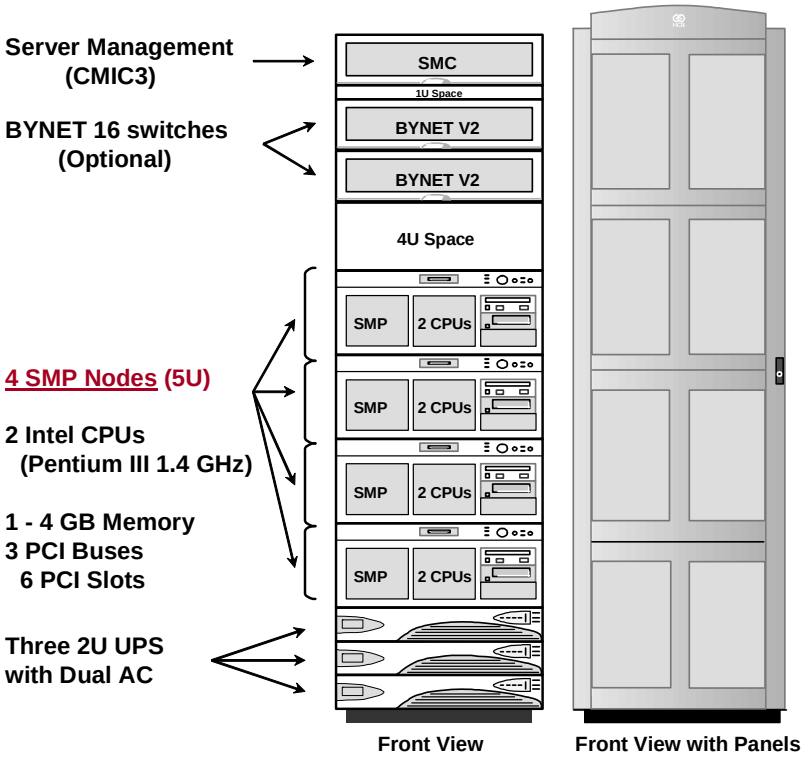
The following table lists the height of each chassis:

Chassis	Height
Service Management	3U (13.3 cm, 5.25 in)
BYA16G	3U (13.3 cm, 5.25 in)
SMP – 4900/5300	5U (22.2 cm, 8.75 in.)
UPS – 4900/5300	2U (8.9 cm, 3.5 in.)

4900 and 5300 Systems

Notes:

- 4900 – up to 4 SMPs; uses BYNET 4 V2.1 switch.
- 5300 – up to 512 SMPs; use BYNET V2 switches.
- SMPs are housed in a chassis that is 5U in height and this allows for 4 SMPs in the rack.
- Each 4900/5300 SMP has two Intel® Pentium® III 1.4 GHz CPUs and a 133 MHz system bus.



4980 and 5380 Systems

MPP systems represent Teradata's large system configurations to support the Teradata Database. The 4980 and 5380 systems were released on July 24, 2003. The 4950/5350 systems were released on December 20, 2002.

Each 4950/5350 SMP utilizes two Intel Pentium IV Xeon 2.8 GHz CPUs and a 400 MHz FSB (Front Side Bus).

Each 4980/5380 SMP utilizes two Intel Pentium IV Xeon 3.06 GHz CPUs that utilize Hyper-Threading and a 533 MHz FSB (Front Side Bus).

Characteristics or both systems include:

- Shared memory – standard 2 GB; can be configured with up to 4 GB for Teradata Database nodes (TPA nodes).
- I/O architecture – PCI, 3 PCI buses
- 2 internal 36 GB disks and removable media devices

System features:

- Improved system MTBF - four nodes per cabinet usually means fewer cabinets.
- Expandable In-cabinet upgrade; add nodes to partially populated rack
- TPA nodes running MP-RAS can be integrated with non-TPA nodes running Windows 2000 in same system.
- Uses existing BYNET 2.0 fabric switches; uses BYNET BIC Release 2.1 interface cards - increases the PCI I/O throughput of the BICs.
- With the release of the 5380, a new BYNET 16 (BYA16G) switch 1U switch chassis is also available.
- Server Management features utilizes the CMIC3; 4980/5380 utilizes a new 11 slot SMC (vs. previous 10 slot SMC) and uses a 2U UPS chassis with integrated IS (Input Selector).
- AWS capabilities – the Windows 2000 AWS can be used with MP-RAS systems
- AWS can monitor mixed operating systems in a single MPP system

4980 and 5380 Systems

4980 and 5380 systems leverage a common hardware building block – the same processing node or SMP.

- **Nodes are 5U in height; up to 4 nodes can be placed in one rack or cabinet**
- **4980/5380 SMPs utilize two Intel Pentium IV Xeon 3.06 GHz CPUs (with Hyper-Threading) and a 533 MHz FSB**

Multiple independent nodes (SMPs) work together as a system through these technologies.

- **Teradata Database - parallel database software**
- **BYNET - system interconnect**
- **AWS - single point of operational control**

4980 systems scale to 4 nodes; 5380 systems scale to 512 nodes.

TPA nodes running UNIX MP-RAS can be integrated with non-TPA nodes running Windows 2000 in same system.

4980/5380 Processing Node

The Processing node is comprised completely of Intel designed components. The older 4950/5350 components consist of an Intel Hudson III chassis kit and an Intel Hodges boxed board set kit. The 4980/5380 systems utilize the Intel Harlingen boxed board set kit.

A description of key components for a 4980/5380 is:

- Intel Hudson III Node Chassis – chassis used for node

- Intel Harlingen boxed board set (SE7501HG2) with two 3.06 GHz CPUs with Hyper-Threading enabled and has a 533 MHz FSB.

  – 2-Way Pentium 4 Xeon processor 3.06 GHz
  – Integrated ATI PCI video
  – Integrated Adaptec dual-channel Ultra 160 SCSI
  – Integrated dual 10/100Mb Ethernet
  – Six 72-bit sockets for 184-pin, 200 MHz, 2.5 V, DDR200 or DDR266 compliant, registered, ECC, SDRAM single-sided or double-sided memory modules (DIMM)
  – Entry server SSI/ATX form factor (12” x 13”)
  – 6 full length PCI slots

- Note: Some of the 4950/5350 processing nodes utilized the Hodges baseboard and some of the newer 4950/5350 processing nodes utilized the Harlingen baseboard. Regardless of the baseboard used in a 4950/5350, the FSB bus speed is set to 400 MHz.

4980/5380 Memory

The baseboard provides six DIMM sockets supporting three pairs of DIMMs: PC1600 (DDR200) for the 4950/5350, upward compatible with PC2100 (DRR266) for the 4980/5380. Memory is partitioned in three banks and can be implemented with either single-sided (one row) or double-sided (two rows) DIMMs, allowing for a maximum memory capacity of 12 GB using 2-GB DIMMs.

PCI Slots

The 4980/5380 has 6 PCI slots.

- Two 64-bit/100 MHz slots
- Three 32-bit/33MHz slots
- One 64-bit/133 MHz slot

4980/5380 Processing Nodes

4980/5380 Processing Node characteristics:

- **Two Intel Pentium IV Xeon 3.06 GHz CPUs (Hyper-Threading) and a 533 MHz FSB**
- **Typical memory is 4 GB for TPA nodes (required for Teradata V2R5.1 and above)**
- **Two 36 GB internal disk drives and local media devices**
  - **Used to hold Operating System and Teradata software**
  - **Mirrored for redundancy (via software mirroring)**
- **6 PCI slots via 3 PCI buses – fast (133 MHz) and wide (64-bit). Adapter examples:**
  - **LSI Quad FC (Fibre Channel) host adapter**
  - **High Performance PQS (PCI Quad SCSI) host adapter**
  - **PCI Bus ESCON Adapter (PBSA) – connects an ESCON channel.**
  - **BIC2MS for 4980; connects to BYNET 4 switches**
  - **BIC4MS for 5380; connects to BYNET 16 or 64 switches**
  - **Networking adapters – e.g., GigaBit Ethernet**
- **Integrated dual 10/100 Ethernet adapters**

Can be added and/or upgraded in the field

4980/5380 System and Expansion Racks

4980 and 5380 systems use industry standard rack mount architecture and individual chassis that conform to industry standards. Types of chassis that can be placed in a 4980/5380 cabinet include:

- Processing Node
- BYNET V2 16 Node Switch (BYA16G)
- Server Management
- Uninterruptible Power Supply

The rack is referred to as a 40U rack. A U represents a "unit of vertical measurement" in an industry standard rack. 1U = 4.445 cm or 1.75" high. Therefore, this cabinet has 40U or 40 x 1.75" (70") of usable space. The following table lists the height of each chassis:

Chassis	Height
Service Management	3U (13.3 cm, 5.25 in)
BYA16G	3U (13.3 cm, 5.25 in) 1U (4.445 cm, 1.75 in.)
4950/5350 or 4980/5380 SMP Node	5U (22.2 cm, 8.75 in.)
UPS – 4950/5350 or 4980/5380	2U (8.9 cm, 3.5 in.)

A **Base** (or **System**) rack contains either BYNET 4 (or BYNET 16 switches) and SMP nodes. An **Expansion** rack contains SMP nodes, but no BYNET switches.

Depending on the number of nodes required in the configuration:

- 1 Node: Use the 4950 or 4980 Single Node cabinet

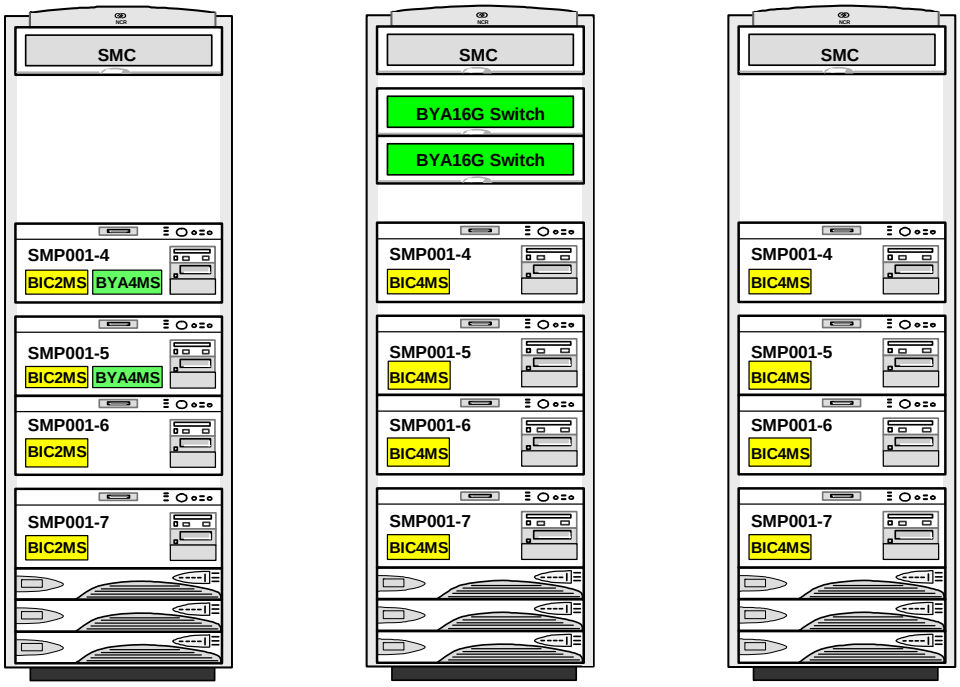
- 2 – 4 Nodes: Use an appropriate combination of 4950/4980 series Single, Two, Three and Four Nodes cabinet configurations; one base cabinet is required to provide the BYNET switches. A BYNET switch is always required in multi-node systems. The Internal BYNET switch (BYA4M) is used in two different nodes.

- 5 – 16 Nodes: Use an appropriate combination of 5350/5380 series Base and Expansion cabinets. One 5350 Base cabinet is required to provide the BYNET switches.

- 17 – 64 Nodes: Use an appropriate combination of 5350/5380 series Expansion cabinets along with two BYNET 64 switch cabinets. Two BYNET 64 cabinets are required to provide redundant BYNET fabrics.

- 65 – 512 Nodes: Use an appropriate combination of 5350/5380 series Expansion cabinets along with an appropriate number of BYNET 512 switch cabinets. Two BYNET 512 cabinets are required to provide redundant BYNET fabrics for every 64 nodes.

4980/5380 System and Expansion Racks



4980 Base or System Rack

5380 Base or System Rack

5380 Expansion Rack

Example 1: 4980 System

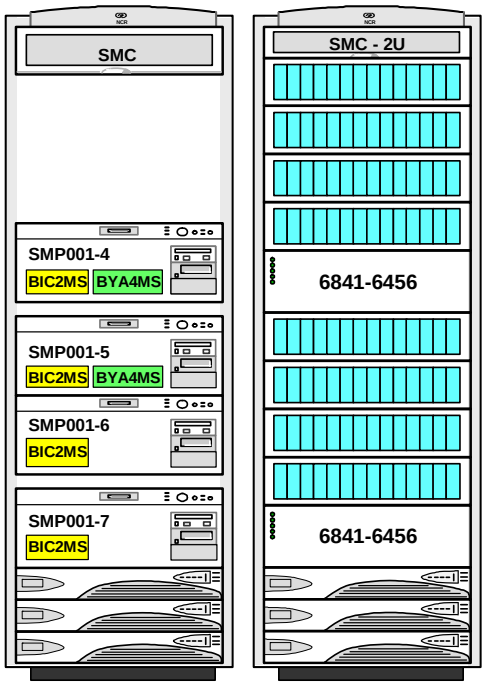
The facing page contains an example of an 4980 system. This example illustrates a configuration where 4 SMPs are Fibre-channel connected to 2 disk arrays.

This configuration from a Teradata Database perspective will be described in more detail later in this module.

Example 1: 4980 System

Notes:

- 4980 Cabinet with BYNET 4 switches.
- Each SMP has a BIC adapter to connect to the BYNET 4 switches.
- 4980 systems can be upgraded to 5380 systems.
  - Requires different BYNET switches
- Existing 4900 SMPs (in 4900 racks) can be upgraded to 4980 SMPs.
  - SMP chassis replacement
- Typical configuration is 1 4980 processor rack with 1 storage rack using 112 drives (73 GB).



Enterprise Storage
6841-6456

Example 2: 5380 System – 8 Nodes

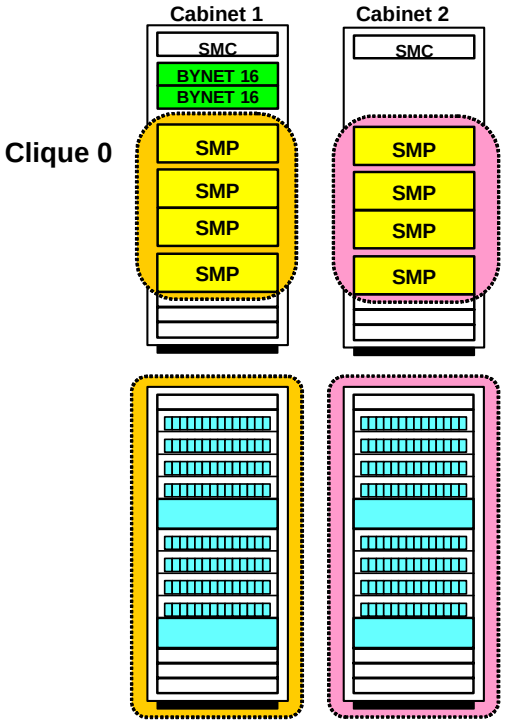
The facing page contains an example configuration of an 8-node 5380 system utilizing the 6841-6456 disk arrays. With the 6841-6456 disk array, a typical configuration is to configure cliques of 4 nodes sharing 4 disk arrays.

What additional racks (cabinets) are needed to upgrade this system to 16 nodes?

Answer:

2 additional 5380 “Expansion” racks, each with 4 SMPs
2 additional 6841-6456 Storage racks

Example 2: 5380 System – 8 Nodes



6841-6456 Disk Arrays

Clique 0
- SMP001-4
- SMP001-5
- SMP001-6
- SMP001-7

Clique 1
- SMP002-4
- SMP002-5
- SMP002-6
- SMP002-7

What racks are needed to grow this system to 16 nodes?

Example 3: 5380 System – 16 Nodes

The facing page contains an example configuration of a 16-node 5380 system utilizing the 6841-6456 disk arrays. With the 6841-6456 disk array, a typical configuration is to utilize cliques of 4 nodes sharing 2 disk arrays.

What additional racks (cabinets) are needed to upgrade this system to 32 nodes?

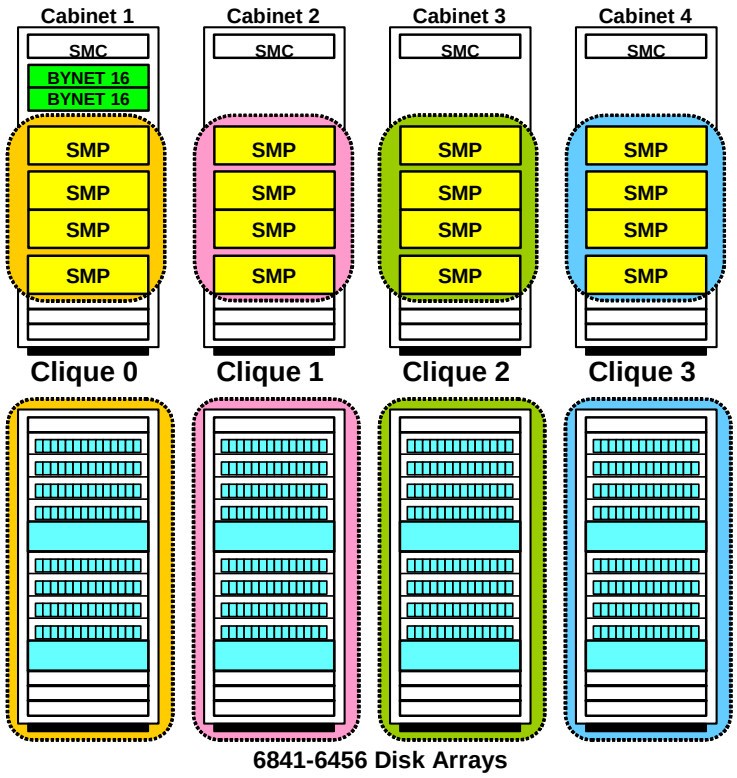
Answer:

2 BYNET V2 64 Node Switch racks
4 additional 5380 “Expansion” racks, each with 4 SMPs

4 additional 6841-6456 Storage racks

The 5380 system can scale up to 16 processing nodes using the BYNET V2 16 switch (BYA16G). This is a BYNET Release 2 implementation (i.e., 60 MB /sec).

Example 3: 5380 System – 16 Nodes



Clique 0
- SMP001-4
- SMP001-5
- SMP001-6
- SMP001-7

Clique 1
- SMP002-4
- SMP002-5
- SMP002-6
- SMP002-7

Clique 2
- SMP003-4
- SMP003-5
- SMP003-6
- SMP003-7

Clique 3
- SMP004-4
- SMP004-5
- SMP004-6
- SMP004-7

What racks are needed to grow this system to 32 nodes?

Example 4: 5380 System – 32 Nodes

The facing page contains an example configuration of a 32-node 5380 system utilizing the 6841-6456 disk arrays. As mentioned earlier, with the 6841-6456 disk array, a typical configuration is to configure cliques of 4 nodes sharing 2 disk arrays.

The 5380 system can scale to 64 processing nodes using the BYNET V2 64 Node switch (BYA64GX) which is housed in a BYNET V2 Switch cabinet. Note that there are two BYNET V2 Switch cabinets in the illustration on the facing page.

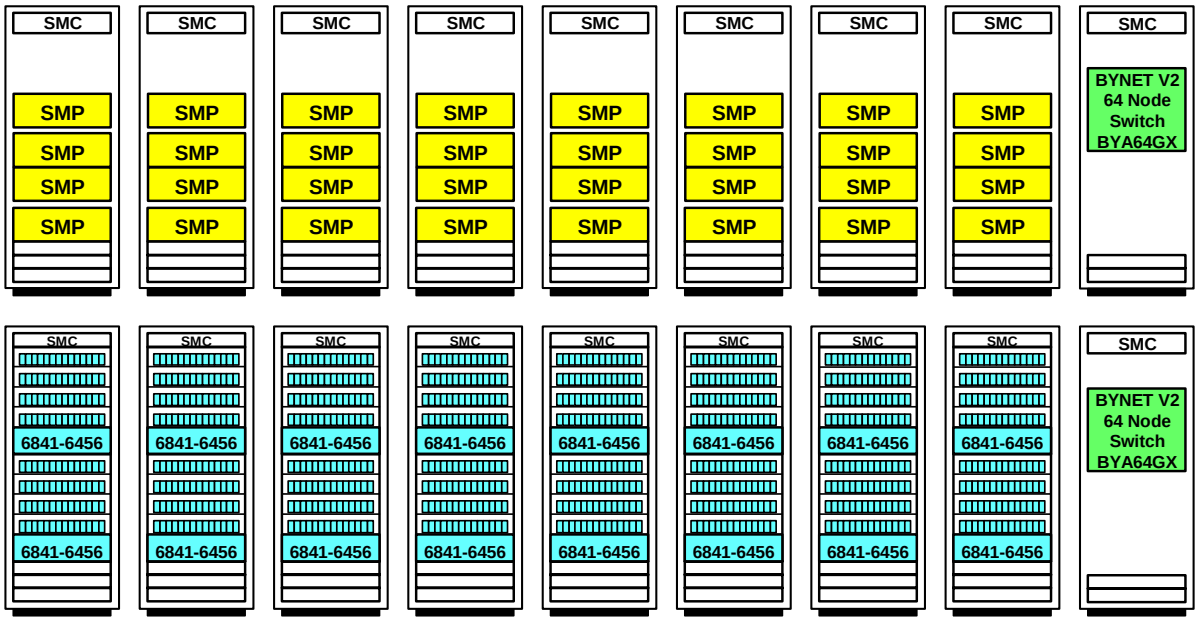
This is also a BYNET Release 2 implementation (i.e., 60 MB /sec).

What types of cabinets are needed to upgrade this system to 64 nodes?

Answer:

8 additional 5380 “Expansion” racks, each with 4 SMPs
8 additional 6841-6456 Storage racks

Example 4: 5380 System – 32 Nodes



6841-6456 Disk Arrays

What additional racks are needed to upgrade this system to 64 nodes?

5400 Processing Node

Each 5400E and 5400H Node Chassis is configured with the two Intel processors running at 3.6 GHz. These latest generation Intel® Xeon™ processors integrate Demand Based Switching (DBS) with Enhanced Intel SpeedStep® Technology to adjust power and lower the processor's power demand.

The new Intel® Extended Memory 64 Technology (Intel® EM64T), enables 64-bit memory addressability. The new processor contains enhancements to Intel Hyper-Threading Technology and expanded Streaming SIMD Extensions 3 (SSE3) Instructions to improve thread synchronization for better system responsiveness.

5400E/5400H Memory

Each 5400E and 5400H node includes 4 GB of memory bundled as an every unit item. The node contains six DIMM memory slots supporting DDR-2 400 MHz memory or DDR 266/333 MHz memory.

The 5400E and 5400H node chassis are shipped fully populated with memory and therefore do not require memory upgrade. Non-Teradata Windows nodes may be ordered with either 4 GB or 6 GB of memory.

Drives and Local Media Devices

The node chassis configuration will support:

- Up to a maximum of 4 hot-swappable SCSI hard drives ; comes with standard two 36 GB disk drives
- A slim-line CDROM drive (every unit item)
- Flex drive (every unit item)
- Tape Drive (optional)

PCI Slots

The 4980/5380 has 6 PCI slots. The slots support fast (133 MHz) and wide (64-bit) PCI cards. **SCSI, Token Ring and FDDI adapters are not available for the 5400 platform.**

Two Fibre Channel Storage adapters are available with the 5400, a quad port adapter for connection to Disk Arrays and Tape subsystems, and a dual port adapter for connection to Tape subsystems only.

Quad Fibre Channel Adapter: The quad Fibre Channel Storage Adapter provides connectivity to Enterprise Storage Quad Modular Fibre Channel Arrays, EMC DMX External Disk Storage Subsystems, and Tape subsystems. The quad Fibre Channel adapter offers four 2 Gb channels for connectivity.

Dual Fibre Channel Adapter: The dual Fibre Channel Storage Adapter provides connectivity to Tape Subsystems for MP-RAS and Windows Server 2003. The dual Fibre Channel adapter offers two 2 Gb channels for connectivity.

5400 Processing Nodes

5400 Processing Node characteristics:

- **Two Intel Pentium IV Xeon 3.6 GHz CPUs (Hyper-Threading) and a 800 MHz FSB**
  - **Intel "Jarrell" Baseboard**
- **4 GB Memory for TPA nodes**
- **Two 36 GB internal disk drives and local media devices**
  - **Used to hold Operating System and Teradata software**
  - **Mirrored for redundancy (via software mirroring)**
- **Integrated dual 10/100 Ethernet adapters for Server Management**
- **6 PCI slots via 3 PCI buses – fast (133 MHz) and wide (64-bit). Adapter examples:**
  - **LSI Quad FC (Fibre Channel) host adapter**
  - **PCI Bus ESCON Adapter (PBSA) – connects an ESCON channel.**
  - **BIC2QC for 5400E; connects to BYNET 4 switches**
  - **BIC2Q for 5400H; connects to BYNET 16 or 64 switches**
  - **Networking adapters – e.g., GigaBit Ethernet**
- **Can be added and/or upgraded in the field.**

5400/5450 Systems

The 5400/5450 Platform supports the Teradata Warehouse solution. There are two models of the 5400/5450 Platform, the 5400/5450**E** and the 5400/5450**H**.

The major difference between a 5400 and a 5450 node is the speed of the processing node:

- 5400 – two Intel Xeon 3.6 GHz CPUs each with 1 MB of cache memory
- 5450 – two Intel Xeon 3.8 GHz CPUs each with 2 MB of cache memory

54xxE Systems

The defining characteristic of the 5400/5450E is its node configuration which ranges from 1 to 4 nodes. These four nodes can be Teradata processing (i.e., TPA or Trusted Parallel Application) nodes and / or non-Teradata (i.e., non-TPA) nodes.

The 5400/5450E cabinet can not be reconfigured to expand to more than four nodes. This is because 5400/5450E cabinets are pre-wired with structured cabling to support only 4 nodes.

The 5400/5450E cabinet also contains components which support Server Management, BYNET Interconnect and Power Management. The 5400/5450E resides in a single industry standard rack, which has cabling to support up to four nodes.

54xxH Systems

The 5400/5450H models are targeted to the full-scale large data warehouse. The 5400/5450H models offer expansion capabilities from 1 to 1024 nodes of TPA and non-TPA nodes. The power of the Teradata database combined with the throughput, power and performance of both the Intel® Xeon™ processor and BYNET V3 technologies offers unsurpassed performance and capacity within the scalable data warehouse.

The 5400/5450H Platform is housed in one or more industry standard racks which can support up to 10 nodes (TPA and / or non-TPA nodes). When more than 16 TPA nodes are present, BYNET Switch Cabinets are required to support dual BYNET node interconnects (called BYNET fabrics).

Optionally, the 5400/5450H can support Large Cliques and/or Hot Standby Nodes. Large Cliques allow for up to eight nodes to be configured in a clique (standard cliques are four nodes) thus reducing the amount of degradation to less than 15% in the event of a node failure. Large Clique configurations require Fibre Channel Switches to reside in the 54xxH rack cabinet. These racks are frequently referred to as 54xxH FC cabinets or racks.

A Hot Standby Node (HSN) prevents system degradation by providing a “standby” node in the event of a node failure. With standard sized cliques, one HSN is needed for every three TPA nodes (referred to as a “3+1” configuration). When HSN is combined with Large Cliques, only one HSN is needed for every seven TPA nodes (referred to as a “7+1” configuration).

5400/5450 Systems

There are basically two 5400/5450 systems.

- 5400 and 5450E systems scale up to 4 nodes – uses a BYNET 4 V2.1 switch.
- 5400 and 5450H systems scale up to 1024 nodes.

Key features of the 54xx systems include:

- **Processing nodes that are 2U in size**
  - 5400 nodes – two Intel® Xeon® 3.6 GHz CPUs (each with 1 MB cache)
  - 5450 nodes – two Intel® Xeon® 3.8 GHz CPUs (each with 2 MB cache)
  - These nodes support both 32-bit and 64-bit operating systems.
- **New Cabinet design (more modern design) – allows up to 10 nodes in rack**
  - Improved cooling and integrated cabling (cables bundled in a harness)
- **BYNET Version 3**
  - Faster interconnect – 90 MB/sec (versus 60 MB/sec) per BYNET
  - Support for up to 1024 nodes; optical connection beyond 512 nodes
- **Improved 3rd Generation Server Management (SM3G)**
  - Ethernet-based server management
  - SM Chassis (CMIC) is a standard processing node – fewer required in system

5400/5450 Cabinets

The facing page illustrates the 54xx system family.

The 54xx system platforms also use industry standard rack mount architecture. The newly designed 54xx rack provides for better air flow and cooling. Similar to previous rack-based systems, this rack contains individual subsystem chassis that are housed in standard rack frames. Subsystems are self-contained, and their configurations — either internal or within a system — are redundant. The design ensures overall system reliability, enhances its serviceability, and enables time and cost efficient upgrades.

The Server Management (SM) chassis is redesigned with the 54xx and is located above the UPS chassis modules, instead of at the top of the cabinet.

The key chassis in the rack/cabinet is the SMP chassis. The SMP node chassis is 2U in height. Each 5400 SMP has two Intel® Pentium® IV Xeon 3.6 GHz CPUs that utilize Hyper-Threading and have 1 MB cache. Each 5450 SMP has two Intel® Pentium® IV Xeon 3.8 GHz CPUs that utilize Hyper-Threading and have 2 MB cache. Each node is also configured with 4 GB of memory.

The 54xxE rack-based cabinet houses up to 4 SMPs, BYNET V2.1, and has 3 UPS chassis.

The 54xxH rack-based cabinet houses up to 10 SMPs, optionally BYNET V3.0 switches, and has 5 UPS chassis. To accommodate the increased node density of the 54xxH, the number of UPS chassis modules per rack is five.

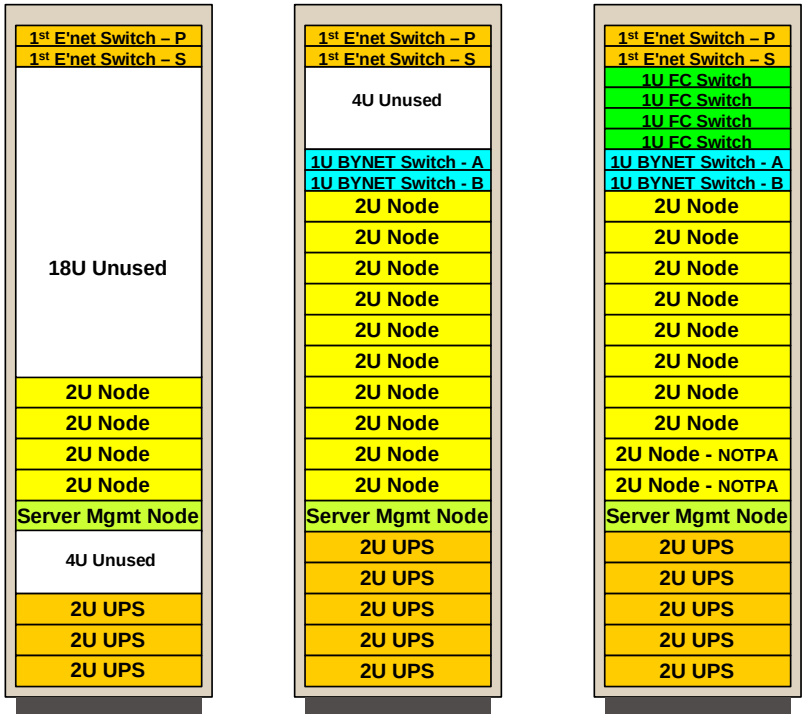
The 54xxH FC (Fibre Channel) or LC (Large Clique) rack-based cabinet houses up to 10 SMPs, optionally BYNET V3.0 switches, and has 5 UPS chassis. Usually, only 8 of the 10 nodes can be TPA nodes.

The following table lists the height of each chassis:

Chassis	Height
Service Management Chassis (3rd generation)	2U (8.9 cm, 3.5 in)
BYA32G-A and BYA32G-B	1U (4.5 cm, 1.75 in)
SMP – 5400/5450	2U (8.9 cm, 3.5 in)
UPS	2U (8.9 cm, 3.5 in)

SM3G – Server Management 3rd Generation

5400/5450 Cabinets



Terminology:

System Cabinet – includes BYNET switches

Expansion Cabinet – doesn't include BYNET switches

Teradata 555x Systems

The 555x processing nodes utilize the quad-core Intel CPUs to access Teradata database.

Teradata 5500 Systems

The predecessor to the 555x was the 5500 system. The three 5500 models are:

- 5500E – one or two nodes with one or two Intel Xeon dual-core 2.66 GHz CPUs
- 5500C – one Intel Xeon dual-core 2.66 GHz CPU with 2 MB of cache memory
- 5500H – two Intel Xeon dual-core 2.66 GHz CPUs with 4 MB of cache memory

The 5500E System is a special system and cabinet configuration used for entry-level data warehousing environments. The 5500E cabinet can not be reconfigured to expand to more than two nodes. This is because 5500E cabinets are pre-wired with structured cabling to support only 2 nodes. Therefore, the rack for the 5500E is not upgradeable and does not have cable harnesses for additional nodes. This cabinet is a full-height rack and only has 32Amps of power as compared to 50 Amps of power for 5500C and 5500H cabinets. The 5500E cabinet also does not have traditional BYNET switches. This special implementation of the "BYNET" interconnect uses a dedicated Ethernet switch for continuous connectivity between the nodes and no HSN (Hot Spare Node) is available.

5550H and 5555 C/H Systems

These models are targeted to the full-scale large data warehouse. These models offer expansion capabilities from 1 to 1024 nodes of TPA and non-TPA nodes. The power of the Teradata database combined with the throughput, power and performance of both the Intel® Xeon™ quad-core processors and BYNET V3 technologies offers unsurpassed performance and capacity within the scalable data warehouse.

The 5550H and 5555 C/H platforms are housed in one or more industry standard racks which can support up to 9 nodes (or a combination (up to 9) of TPA, channel, and/or managed server nodes). When more than 16 TPA or channel nodes are present, BYNET Switch Cabinets are required to support dual BYNET node interconnects (called BYNET fabrics).

Typically the 5550H and 5555 C/H platforms are configured with Hot Standby Nodes. Key characteristics of the 5550 are listed on the facing page. These nodes may have 16 GB of memory, but commonly are configured with 32 GB of memory. 32 GB is recommended for customers who meet any of the following criteria:

- Current customer has a I/O limited configuration and needs more bandwidth in their 5550 configuration
- Greater than a 500 AMP system or use large capacity drives - i.e., 300 GB drives
- Greater than 108 disks per node
- Customer who has less than 78 MB FSG cache and is moving to the 5550
- For a customer who may have had to lower their FSG cache below 65% and is moving to the 5550

Teradata 555x Systems

Features of the 555x nodes include:

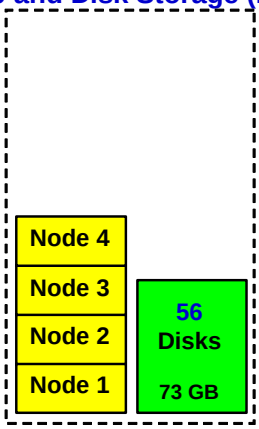
- **Processing node utilizes quad-core CPUs.**
  - **Utilizes two Intel® Xeon® 2.33 GHz quad-core CPUs**
    - **5550 nodes have 8 MB L2 cache per CPU; 5555 nodes have 12 MB L2 cache per CPU**
  - **Linux nodes start with 16 GB of memory; can be upgraded to 32 GB.**
  - **Linux (64 bit) is only offered for new systems; requires Teradata V26.2 or higher.**
    - **MP-RAS is only available for co-existence and existing system expansions.**
    - **Windows 2003 is not supported on 555x nodes.**
  - **5500 server nodes are upgradeable to 555x server nodes**
- **Available systems**
  - **5550H or 5555H – utilizes BYNET v3 or v4; supports up to 1024/4096 nodes.**
  - **5555C – only has 1 quad-core CPU; may be used for coexistence**
    - **Note: There is no 5550C model**
- **Largest recommended clique size is 3 nodes + 1 Hot Standby Node (HSN)**
  - **Typical LSI Disk Array is model 6843-4000 with 146 GB or 300 GB drives**
  - **Typical design center configuration is 3+1 nodes with 5 arrays (2½ racks)**
  - **Large cliques are not available with 555x systems.**

Example of 5500C Coexistence with a 5380

The example on the facing page shows the typical AMP and Disk configurations for a 5500C coexistence with a 5380.

Example of 5500C Coexistence with 5380

5380 and Disk Storage (Raid 1)



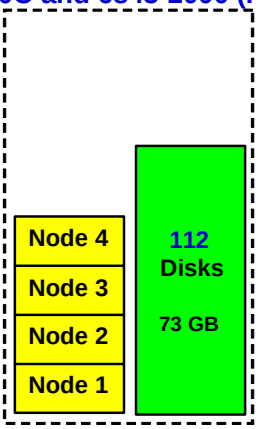
5380 (4 nodes/clique)

7 AMPs / Node
28 AMPs / Clique

4 Nodes – 56 Disks
14 Disks per Node
Each Vdisk – 2 Disks (RAID 1)
Each Vdisk – 73 GB*

Clique – 28 AMPs x 73 GB = 2044 GB*

5500C and 6843-2000 (Raid 1)



5500C (4 nodes/clique)

14 AMPs / Node
56 AMPs / Clique

4 Nodes – 112 Disks
28 Disks per Node
Each Vdisk – 2 Disks (RAID 1)
Each Vdisk – 73 GB*

Clique – 56 AMPs x 73 GB = 4088 GB*

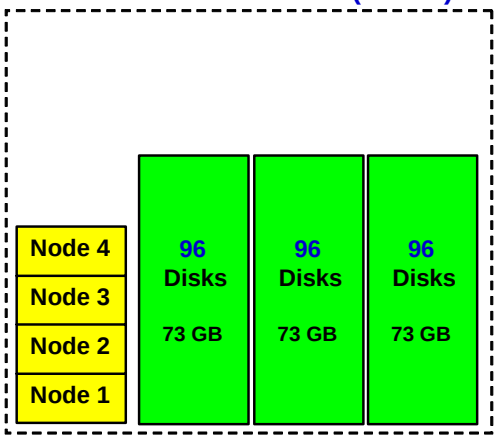
* Actual MaxPerm space will be less.

5500 Teradata Configuration Examples

The examples on the facing page show a typical AMP and Disk configurations for 5500 cliques.

5500 Teradata Configuration Examples

5500H and 6843-2000 (Raid 1)



5500H (4 nodes/clique) – Linux

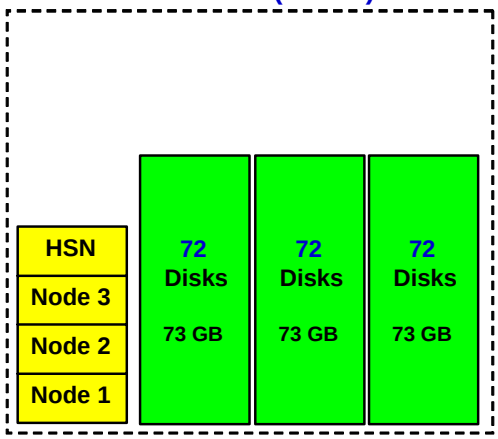
72 Disks per Node
4 Nodes – 288 Disks

18 AMPs / Node
72 AMPs / Clique

Each Vdisk – 4 Disks (RAID 1)
Each Vdisk – 146 GB*

Clique – 72 AMPs x 146 GB = 10,512 GB*

5500H and 6843-2000 (Raid 1) with HSN



5500H (4 nodes/clique) – Linux

72 Disks per Node
3 Nodes – 216 Disks

18 AMPs / Node
54 AMPs / Clique

Each Vdisk – 4 Disks (RAID 1)
Each Vdisk – 146 GB*

Clique – 54 AMPs x 146 GB = 7,884 GB*

* Actual MaxPerm space will be less.

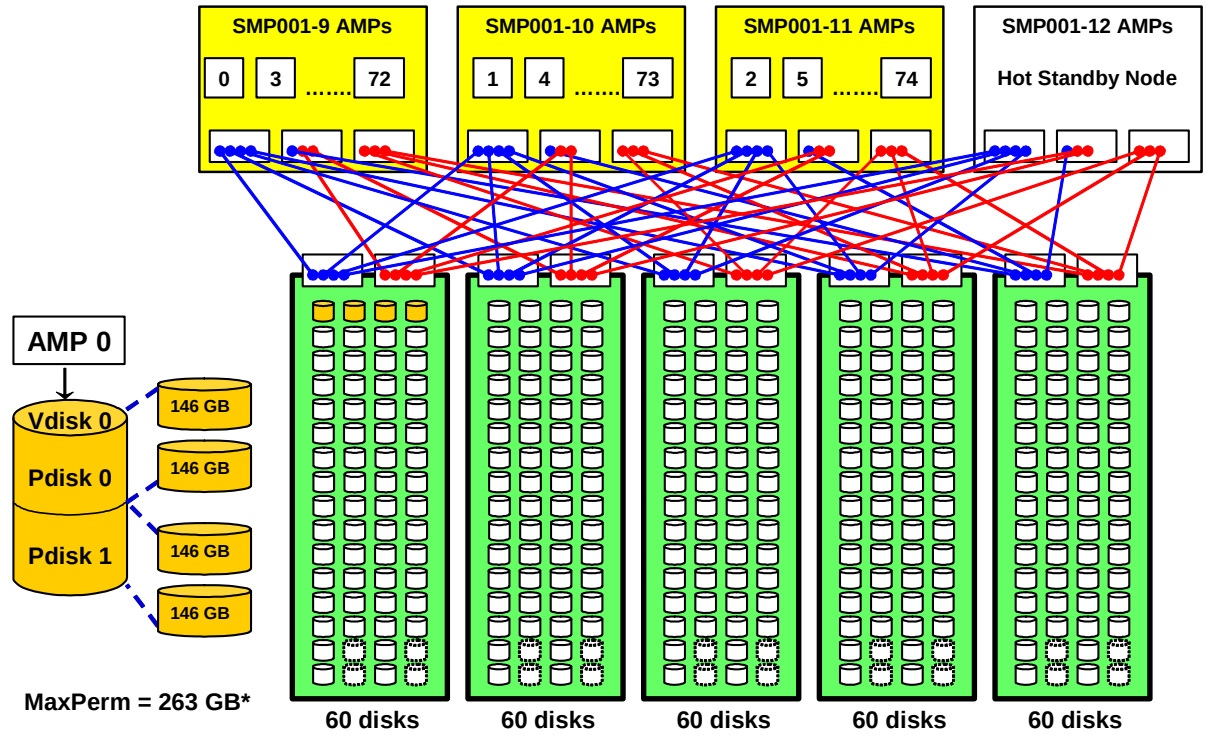
5555H Example

The facing page contains an example of a 4-node clique sharing **five** 6843-4000 disk arrays. Three nodes will be TPA nodes and 1 node will be a hot standby node.

Each 5555 node has 3 Quad Fibre Channel (4 Gbit/sec.) Host Bus Adapters (HBA). These Fibre Channel cables are point-to-point connections.

Note the distribution of AMPs among the 5555 nodes. The typical design center configuration is to configure 25 AMPs per node effectively utilizing 100 disks per node.

5555H Example



5555H System – 8 (6+2) Nodes

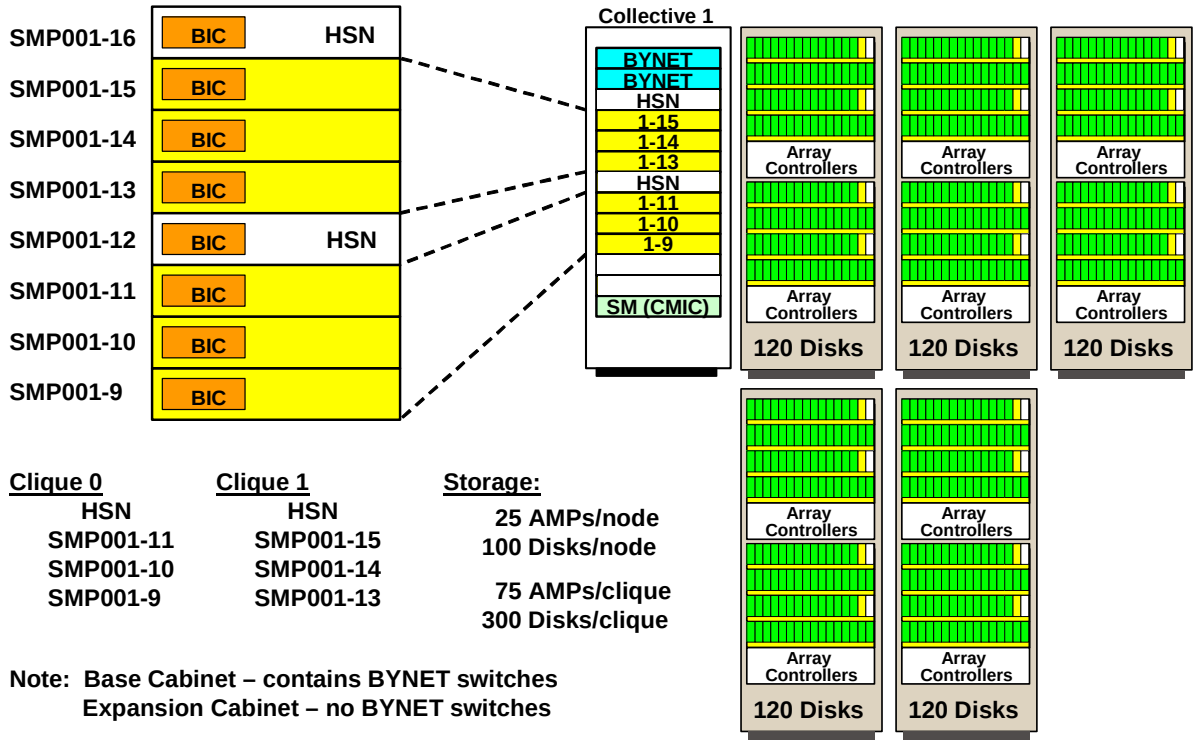
The facing page contains an example configuration of an 8-node 5555H system utilizing the 6843-4000 disk arrays. This example illustrates two 4-node cliques. Each clique has three TPA nodes and 1 hot standby node.

What additional racks (cabinets) are needed to upgrade this system to 16 (12+4) nodes?

Answer:

1 additional 5555H “Expansion” rack with 8 (6+2) nodes
5 additional 6843-4000 Storage racks

5555H System – 8 (6+2) Nodes



Clique 0
HSN
SMP001-11
SMP001-10
SMP001-9

Clique 1
HSN
SMP001-15
SMP001-14
SMP001-13

Storage:
25 AMPs/node
100 Disks/node

75 AMPs/clique
300 Disks/clique

Note: Base Cabinet – contains BYNET switches
Expansion Cabinet – no BYNET switches

5555H System – 16 (12+4) Nodes

The facing page contains an example configuration of a 16-node 5555 system utilizing the 6843-4000 disk arrays. Each clique has three TPA nodes and 1 hot standby node.

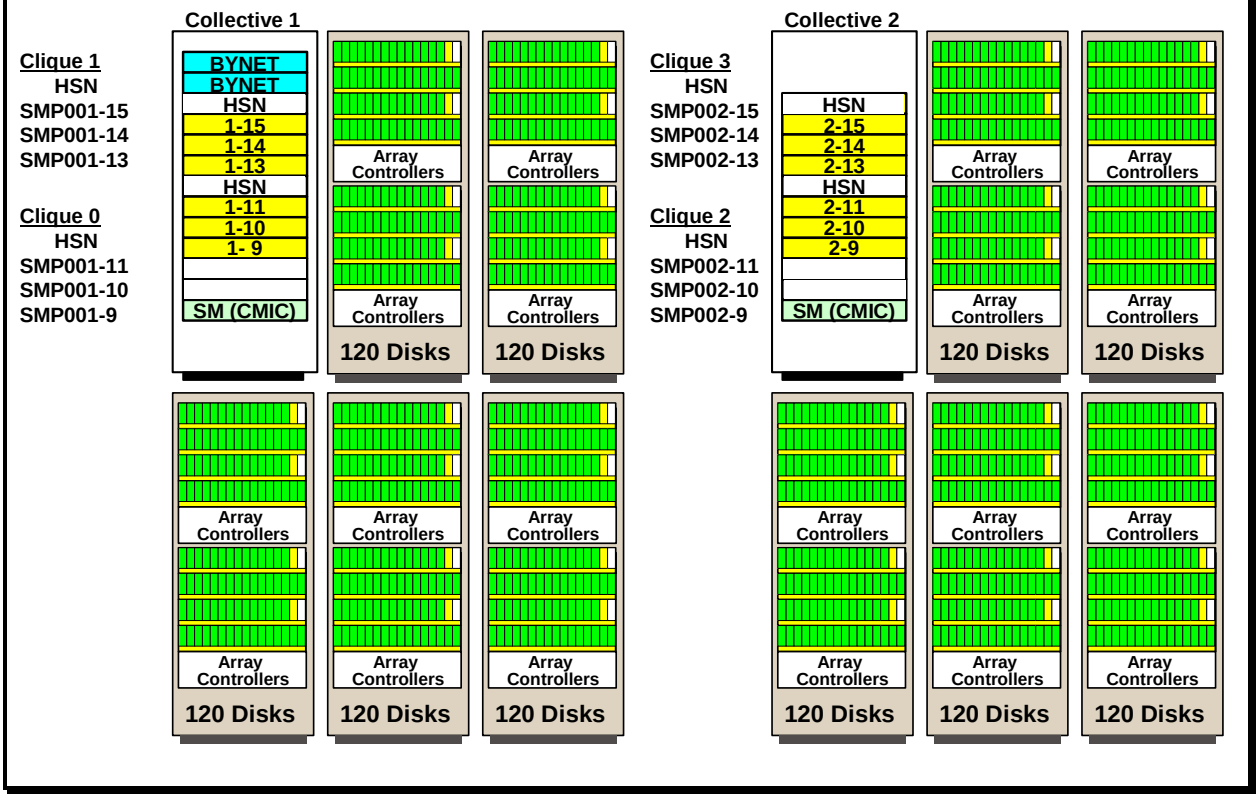
This will be a BYNET Release 3 implementation (i.e., 375 MB /sec per node).

What types of cabinets are needed to upgrade this system to 32 (24 + 8) nodes?

Answer:

2 BYNET switch racks
2 additional 5555H "Expansion" racks, each with 8 (6+2) nodes
10 additional 6843-4000 Storage racks

5555H System – 16 (12+4) Nodes



5555H System – 32 (24+8) Nodes

The facing page contains an example configuration of a 32-node 5555 system utilizing the 6843-4000 disk arrays. Each clique has three TPA nodes and 1 hot standby node. Each clique will share 5 disk arrays.

The Teradata 5555H system can scale to 64 processing nodes using the BYNET V3 64 Node switch (BYA64GX) which is housed in a BYNET V3 Switch cabinet. Note that there are two BYNET V3 Switch cabinets in the illustration on the facing page.

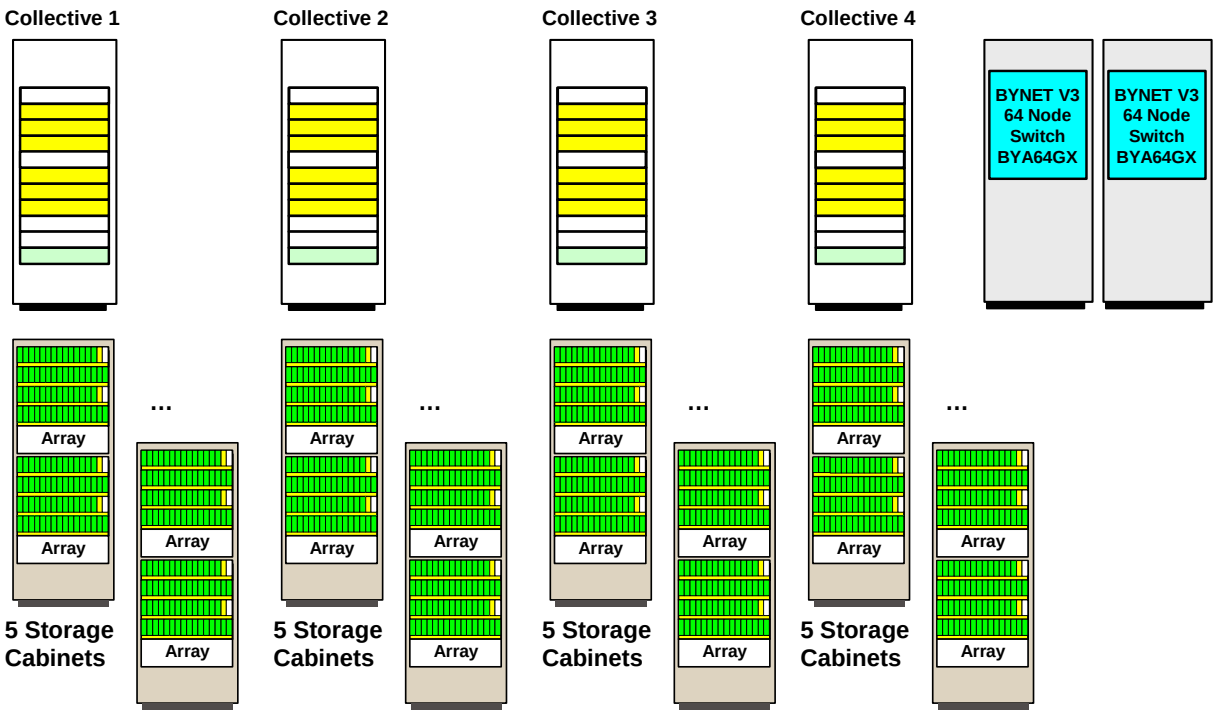
This will be a BYNET Release 3 implementation (i.e., 375 MB /sec per node).

What types of cabinets are needed to upgrade this system to 64 nodes?

Answer:

4 additional 5555H "Expansion" racks, each with 8 nodes
20 additional 6843-4000 Storage racks

5555H System – 32 (24+8) Nodes



Teradata 5600 Systems

The 5600 processing nodes are the newest release of Teradata Servers which supports the Teradata Warehouse solution. These nodes utilize the Intel Nehalem™ quad-core CPUs with hyper-threading enabled.

5600C and 5600H Systems

These models are targeted to the full-scale large data warehouse. These models offer expansion capabilities from 1 to 4096 nodes of the types TPA, HSN, non-TPA nodes, Channel, and TMS. The power of the Teradata database combined with the throughput, power and performance of both the Intel® Nehalem™ quad-core processors with hyper-threading and BYNET v4 technologies offers unsurpassed performance and capacity within the scalable data warehouse.

The 5600C and 5600H platforms are housed in one or more industry standard racks which can support up to 9 nodes (or a combination (up to 9) of TPA, channel, and/or managed server nodes). When more than 16 TPA are present, BYNET Switch Cabinets are required to support dual BYNET node interconnects (called BYNET fabrics).

- Typically, the 5600C and 5600H platforms are configured with Hot Standby Nodes. Key characteristics of the 5600 are listed on the facing page. 5600H nodes start with 48 GB of memory and may have 96 GB of memory.

5600 Node Details

- 5600H Node (dual CPUs) – Tylersburg Two Sockets @ 2.66 GHZ -Urbana Baseboard / Nehalem – Quad Core CPU w/ 12MB L2 Cache and HT enabled.
  - Memory - up 96 GB w/ 12GB D/R DIMMS
- 5600C Node (single CPU) – Tylersburg Two Sockets @ 2.66 GHZ -Urbana Baseboard / Nehalem – Quad Core CPU w/ 12MB L2 Cache and HT enabled
  - Memory - up to 24 GB Max – 4GB D/R DIMMS and 48 GB w/ 8GB D/R DIMMS
- 5600 Channel Server (single CPU) –Tylersburg Two Sockets @ 2.66 GHZ -Urbana Baseboard / Nehalem – Quad Core CPU w/ 12MB L2 Cache and HT enabled
  - Memory - up to 24 GB Max – 4GB D/R DIMMS
- TMS Nodes - Dell Node R710 – new 2U (dual CPU) Dell node based on Nehalem processors – (2.4 GHZ CPU)
  - Memory - up to 72 GB w/ 4GB DIMMS and up to 144GB w/ 8GB DIMMS

Teradata 5600 Systems

Features of the 5600 node include:

- **Processing node utilizes Intel quad-core Nehalem CPUs with hyper-threading enabled.**
  - **Utilizes one or two Intel® 2.66 GHz quad-core CPUs**
    - **5600C nodes utilize one quad-core CPU; may be used for coexistence**
    - **5600H nodes utilize two quad-core CPUs**
  - **5600H nodes start with 48 GB of memory; can be upgraded to 96 GB.**
    - **5600C nodes start with 24 GB of memory**
  - **Linux operating system: SLES 10; requires Teradata V12.0 or higher.**
  - **5600C server nodes are upgradeable to 5600H server nodes**
- **Available 5600 systems**
  - **5600C or 5600H – utilize BYNET v4 with support up to 1024 nodes.**
- **Largest recommended clique size is 2 TPA + 1 HSN**
  - **Typical LSI Disk Array is model 6844 with 300 or 450 GB drives**
  - **Typical design center configuration (LSI) is 2+1 nodes with 3 arrays (3 racks)**
  - **Large cliques are not available with 5600 systems.**

Examples of Teradata 555x and 56xx Cabinets

The facing page illustrates various Teradata 555x and 56xx cabinets.

55xx systems also use the 54xx industry standard rack mount cabinet which provide for excellent air flow and cooling. Similar to previous rack-based systems, this rack contains individual subsystem chassis that are housed in standard rack frames. Subsystems are self-contained, and their configurations — either internal or within a system — are redundant. The design ensures overall system reliability, enhances its serviceability, and enables time and cost efficient upgrades.

The 56xx cabinet is similar to the 55xx cabinet, but is approximately 4" deeper.

The key chassis in the rack/cabinet is the node chassis. The SMP node chassis is 2U in height. The 5500 SMP chassis is the same height as the 54xx chassis, but is approximately 2 inches deeper. The baseboards are different between the 54xx and 5500 systems.

Important Note: A Hot Standby Node is strongly recommended with 555x and 56xx cliques. For 555x systems, maximum of three TPA nodes with one HSN node. For 5600 systems, maximum of two TPA nodes with one HSN node.

Large Cliques (older configurations)

Teradata allows clique sizes up to 8 nodes via a set of Fiber Channel switches between the nodes and the disk arrays. Large clique configurations were common with the 54xx systems and available (and not as common) with 55xx systems.

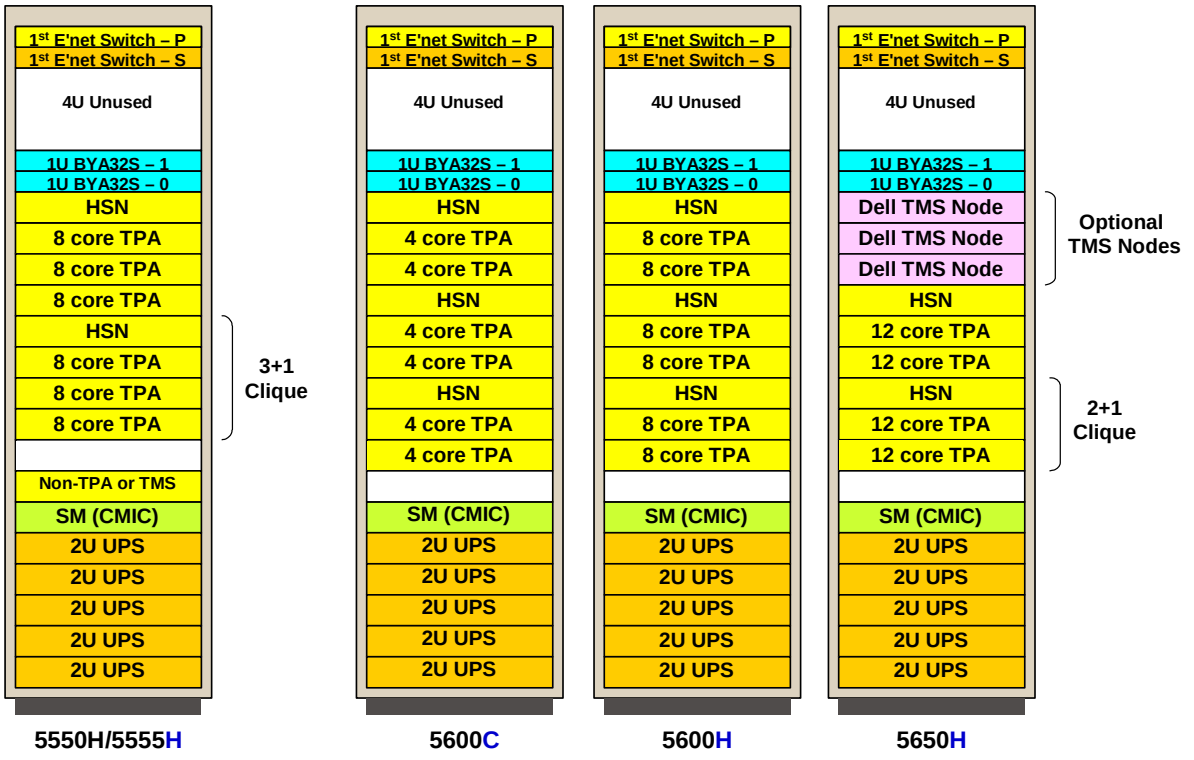
The 5500H LC (Large Clique) rack-based cabinet houses up to 9 SMPs, optionally BYNET V3 switches, and has 5 UPS chassis. Only 8 of the SMP nodes can be TPA nodes. If a HSN (Hot Standby Node) is configured, then the maximum number of TPA nodes is 7.

Optionally, the 5500 can support Large Cliques and/or Hot Standby Nodes. Large Cliques allow for up to eight nodes to be configured in a clique (standard cliques are four nodes) thus reducing the amount of degradation to less than 15% in the event of a node failure. Large Clique configurations require Fibre Channel Switches to reside in the 54xxH rack cabinet. These racks are frequently referred to as 5500C or 5500H LC (Large Clique) cabinets or racks. Large Cliques are available with the 5500, but are not recommended.

A Hot Standby Node (HSN) prevents system degradation by providing a "standby" node in the event of a node failure. With standard sized cliques, one HSN is needed for every three TPA nodes (referred to as a "3+1" configuration).

Important Note: The Large Clique configuration is NOT available with the 5550H, 5555C, 5555H, 56xxC, and 56xxH systems.

Examples of Teradata 555x and 56xx Cabinets



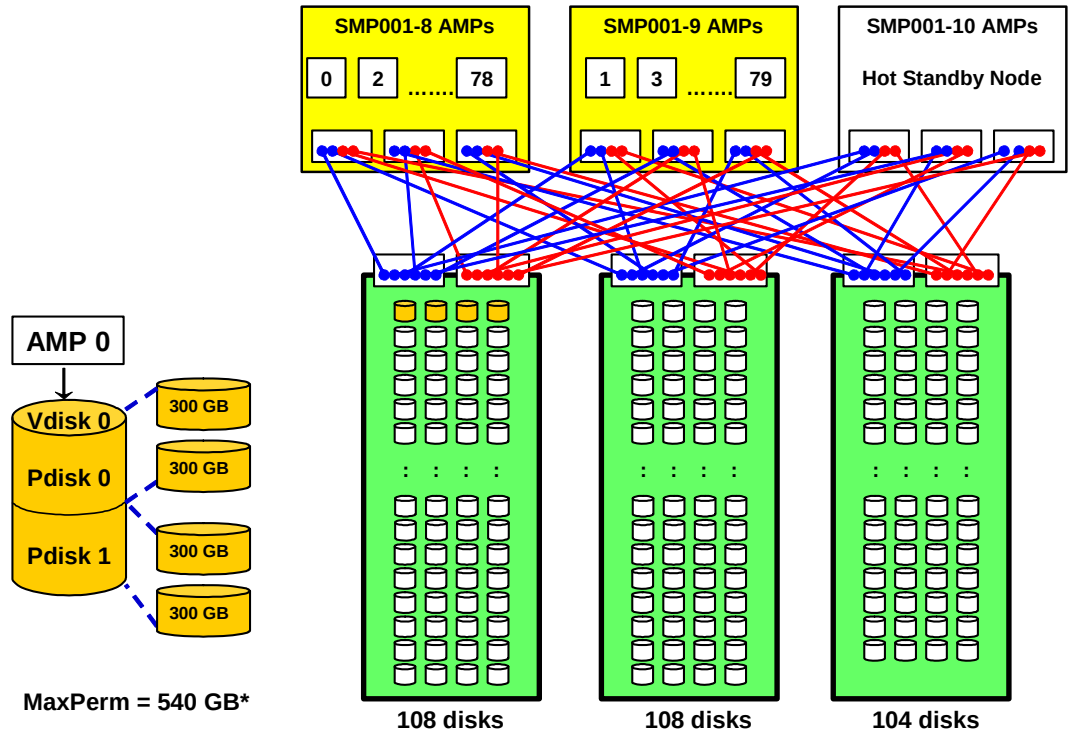
5600H and 6844 Disk Arrays

The facing page contains an example of a 3-node clique sharing **three** 6844 disk arrays. Two nodes will be TPA nodes and 1 node will be a hot standby node.

Each 5600 node has 3 Quad Fibre Channel (4 Gbit/sec.) Host Bus Adapters (HBA). These Fibre Channel cables are point-to-point connections.

Note the distribution of AMPs among the 5600 nodes. The typical design center configuration is to configure 40 AMPs per node effectively utilizing 120 disks per node.

5600H and 6844 Disk Arrays



5600H System – 9 (6+3) Nodes

The facing page contains an example configuration of a 9-node 5600H system utilizing the 6844 disk arrays. This example illustrates three 3-node (2+1) cliques. Each clique has two TPA nodes and 1 hot standby node.

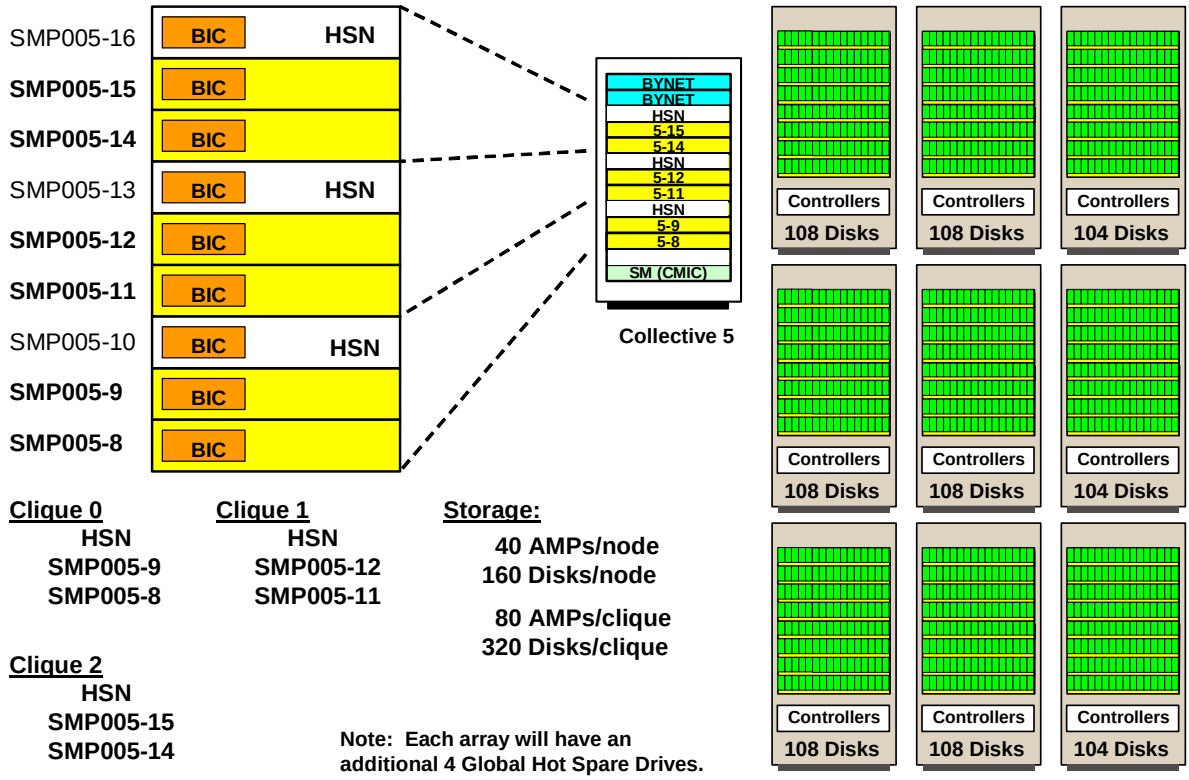
This will be a BYNET Release 4 implementation (i.e., 960 MB /sec per node).

What additional racks (cabinets) are needed to upgrade this system to 18 (12+6) nodes?

Answer:

1 additional 5600H "Expansion" rack with 9 (6+3) nodes
9 additional 6844 Storage racks

5600H System – 9 (6+3) Nodes



Clique 0
HSN
SMP005-9
SMP005-8

Clique 1
HSN
SMP005-12
SMP005-11

Clique 2
HSN
SMP005-15
SMP005-14

Storage:
40 AMPs/node
160 Disks/node

80 AMPs/clique
320 Disks/clique

Note: Each array will have an additional 4 Global Hot Spare Drives.

Teradata 5650 Systems

The 5650 processing nodes are the newest release of Teradata Servers which supports the Teradata Warehouse solution. These nodes utilize the Intel Westmere™ six-core CPUs with hyper-threading enabled.

5650C and 5650H Systems

These models are targeted to the full-scale large data warehouse. These models offer expansion capabilities from 1 to 4096 nodes of the types TPA, HSN, non-TPA nodes, Channel, and TMS. The power of the Teradata database combined with the throughput, power and performance of both the Intel® Westmere™ six-core processors with hyper-threading and BYNET v4 technologies offers unsurpassed performance and capacity within the scalable data warehouse.

The 5650C and 5650H platforms are housed in one or more industry standard racks which can support up to 9 nodes (or a combination (up to 9) of TPA, channel, and/or managed server nodes). When more than 16 TPA are present, BYNET Switch Cabinets are required to support dual BYNET node interconnects (called BYNET fabrics).

- Typically, the 5650C and 5650H platforms are configured with Hot Standby Nodes. Key characteristics of the 5650 are listed on the facing page. 5650C nodes typically have 48 GB of memory and 5600H nodes have 96 GB of memory.

5650 Node Details

- 5650H Node : Intel Urbana Node - Tylersburg chipset, Two Sockets , w/ Westmere CPU (Six Core) @ 2.93GHZ - HT Enabled
  - Memory Size: TPA Node: 96 GB max w/ 1333MHZ DDR3 8GB DIMMS
  - Two 450 GB mirrored disk drives plus one 300 GB disk drive for dumps
- 5650C Node : Intel Urbana Node - Tylersburg chipset, One Socket , w/ Westmere CPU (Six Core) @ 2.93GHZ - HT Enabled
  - Memory Size: TPA Node: 48 GB max w/ 1333MHZ DDR3 8GB DIMMS
- 5600 Channel Server (single CPU) –Tylersburg Two Sockets @ 2.66 GHZ -Urbana Baseboard / Nehalem – Quad Core CPU w/ 12MB L2 Cache and HT enabled
  - Memory - up to 24 GB Max – 4GB D/R DIMMS
- TMS Nodes - Dell Node R710 – new 2U (dual CPU) Dell node based on Nehalem processors – (2.4 GHZ CPU)
  - Memory - up to 72 GB w/ 4GB DIMMS and up to 144GB w/ 8GB DIMMS

Teradata 5650 Systems

Features of the 5650 node include:

- **Processing node utilizes Intel six-core Westmere CPUs with hyper-threading enabled.**
  - **Utilizes one or two Intel® 2.93 GHz six-core CPUs**
    - **5650C nodes utilize 1 socket with one six-core CPU; may be used for coexistence**
    - **5650H nodes utilize 2 sockets with two six-core CPUs**
  - **5650H nodes start with 96 GB of memory**
    - **5650C nodes start with 48 GB of memory**
  - **Linux operating system: SLES 10; requires Teradata 12.00.03.003 or Teradata 13.00.00.19 or higher. PDE support for > 16 logical processors is needed.**
  - **5650C server nodes are upgradeable to 5650H server nodes**
- **Available 5650 systems**
  - **5650C or 5650H – utilize BYNET v4 with support up to 4096 nodes.**
- **Largest recommended clique size is 2 TPA + 1 HSN**
  - **LSI Disk Array is model 6844 with 300, 450, or 600 GB drives**
  - **Typical design center configuration (LSI) is 2+1 nodes with 3 arrays (3 racks)**
  - **Typical node configuration is 47 AMPs with 4 disks/AMP**

Example of Teradata 5650 Cabinets

The facing page illustrates various Teradata 5650 cabinet configurations.

56xx and later systems utilize an industry standard rack mount cabinet which provide for excellent air flow and cooling. Similar to previous rack-based systems, this rack contains individual subsystem chassis that are housed in standard rack frames. Subsystems are self-contained, and their configurations — either internal or within a system — are redundant. The design ensures overall system reliability, enhances its serviceability, and enables time and cost efficient upgrades. The 56xx cabinet is similar to the 55xx cabinet, but is approximately 4" deeper.

The key chassis in the rack/cabinet is the node chassis. The SMP node chassis is 2U in height.

A Hot Standby Node is required with each 5650 clique.

- For 56xx systems, a clique has a maximum of two TPA nodes with one HSN node.

Large Cliques (older configurations)

Teradata allows clique sizes up to 8 nodes via a set of Fiber Channel switches between the nodes and the disk arrays. Large clique configurations were common with the 54xx systems and available (and not as common) with 55xx systems.

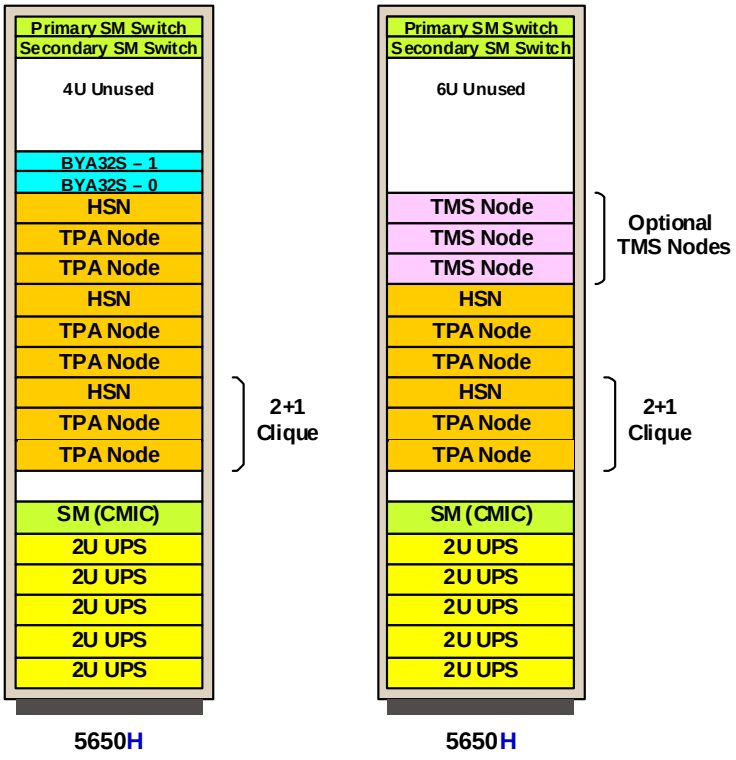
The 5500H LC (Large Clique) rack-based cabinet houses up to 9 SMPs, optionally BYNET V3 switches, and has 5 UPS chassis. Only 8 of the SMP nodes can be TPA nodes. If a HSN (Hot Standby Node) is configured, then the maximum number of TPA nodes is 7.

Optionally, the 5500 can support Large Cliques and/or Hot Standby Nodes. Large Cliques allow for up to eight nodes to be configured in a clique (standard cliques are four nodes) thus reducing the amount of degradation to less than 15% in the event of a node failure. Large Clique configurations require Fibre Channel Switches to reside in the 54xxH rack cabinet. These racks are frequently referred to as 5500C or 5500H LC (Large Clique) cabinets or racks. Large Cliques are available with the 5500, but are not recommended.

A Hot Standby Node (HSN) prevents system degradation by providing a "standby" node in the event of a node failure. With standard sized cliques, one HSN is needed for every three TPA nodes (referred to as a "3+1" configuration).

Note: The Large Clique configuration is NOT available with the 5550H, and later systems.

Example of Teradata 5650 Cabinets



5650H and 6844 Disk Arrays

The facing page contains an example of a 3-node clique sharing **three** 6844 disk arrays. Two nodes will be TPA nodes and 1 node will be a hot standby node.

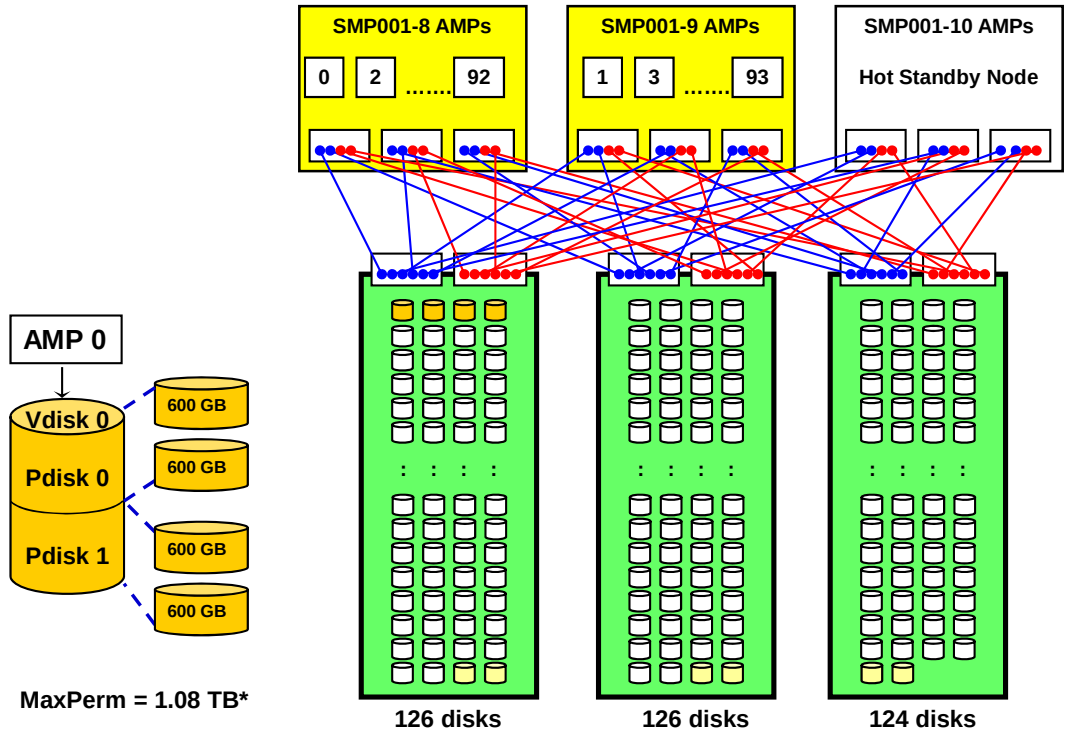
Each 5650 node has 3 Quad Fibre Channel (8 Gbit/sec.) Host Bus Adapters (HBA). These Fibre Channel cables are point-to-point connections.

Note the distribution of AMPs among the 5650 nodes. The typical design center configuration is to configure 47 AMPs per node effectively utilizing 188 disks per node.

In this example, a clique will consist of 2 TPA nodes sharing 376 disks implemented in 3 6844 disk arrays which require 3 storage cabinets.

In the illustration on the facing page, disks shown in white are data disks and disks shown in light yellow color are hot spare drives.

5650H and 6844 Disk Arrays



MaxPerm = 1.08 TB*

* Actual space is app. 90%.

2+1 node clique sharing 376 drives; 47 AMPs/node; Linux System

5650H System – 9 (6+3) Nodes

The facing page contains an example configuration of a 9-node 5650H system utilizing the 6844 disk arrays. This example illustrates three 3-node (2+1) cliques. Each clique has two TPA nodes and 1 hot standby node.

In this example, a clique will consist of 2 TPA nodes sharing 376 disks implemented in 3 6844 disk arrays which require 3 storage cabinets. Therefore, 3 cliques will require 9 disk arrays or 9 storage cabinets.

This will be a BYNET Release 4 implementation (i.e., 960 MB /sec per node).

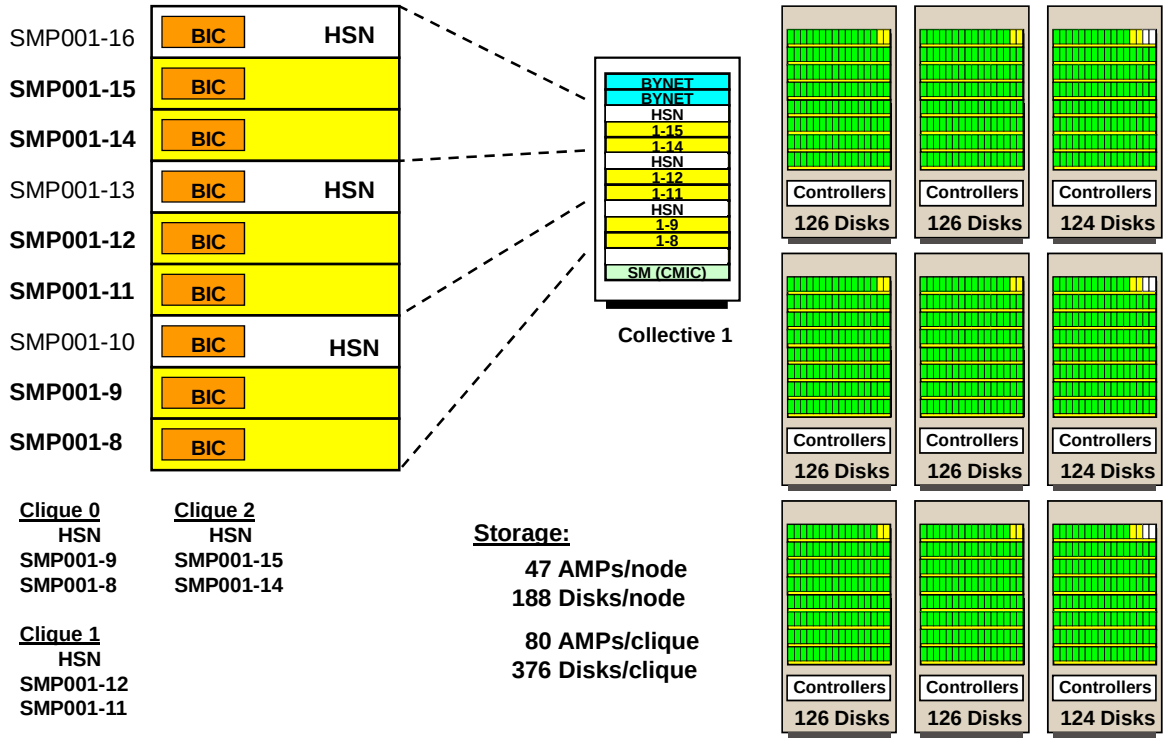
In the illustration on the facing page, disk areas shown in green color are used as data disks, disk areas shown in yellow color are hot spare drives, and disk areas shown in white are empty.

What additional racks (cabinets) are needed to upgrade this system to 18 (12+6) nodes?

Answer:

1 additional 5650H “Expansion” rack with 9 (6+3) nodes
9 additional 6844 Storage racks

5650H System – 9 (6+3) Nodes



Clique 0
HSN
SMP001-9
SMP001-8

Clique 2
HSN
SMP001-15
SMP001-14

Clique 1
HSN
SMP001-12
SMP001-11

Storage:

47 AMPs/node
188 Disks/node

80 AMPs/clique
376 Disks/clique

What cabinets are needed to grow this to a 18 (12+6) node system?

5650H System – 18 (12+6) Nodes

The facing page contains an example configuration of an 18-node 5650 system utilizing the 6844 disk arrays. Each clique has two TPA nodes and 1 hot standby node.

In this example, a clique will consist of 2 TPA nodes sharing 376 disks implemented in 3 6844 disk arrays which require 3 storage cabinets. Therefore, 6 cliques will require 18 disk arrays or 18 storage cabinets.

This will be a BYNET Release 4 implementation (i.e., 960 MB /sec per node).

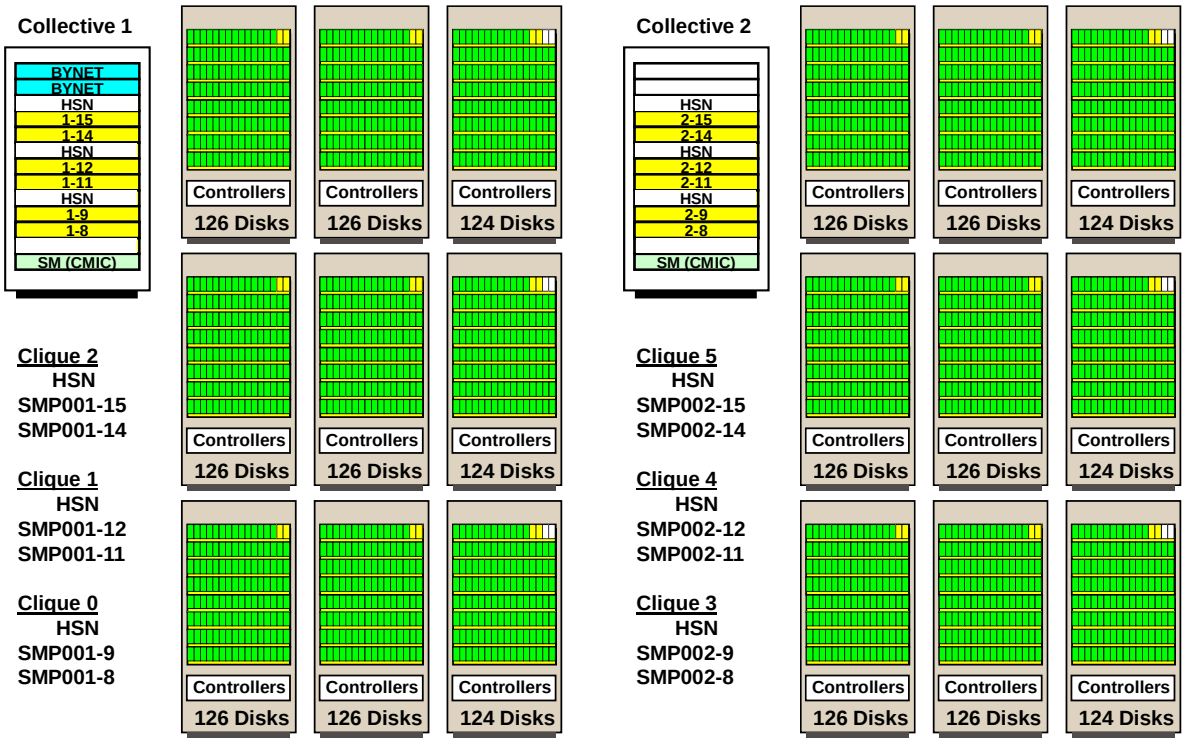
In the illustration on the facing page, disk areas shown in green color are used as data disks, disk areas shown in yellow color are hot spare drives, and disk areas shown in white are empty.

What types of cabinets are needed to upgrade this system to 36 (24 + 12) nodes?

Answer:

2 BYNET switch racks with BYNET 64 switches (e.g., BYA64S)
2 additional 5650H "Expansion" racks, each with 9 (6+3) nodes
18 additional 6844 Storage racks

5650H System – 18 (12+6) Nodes



What cabinets are needed to grow this to a 36 (24+12) node system?

5650H System – 36 (24+12) Nodes

The facing page contains an example configuration of a 36-node 5650 system utilizing the 6844 disk arrays. Each clique has two TPA nodes and 1 hot standby node.

In this example, a clique will consist of 2 TPA nodes sharing 376 disks implemented in 3 6844 disk arrays which require 3 storage cabinets. Therefore, 12 cliques will require 36 disk arrays or 36 storage cabinets.

This will be a BYNET Release 4 implementation (i.e., 960 MB /sec per node).

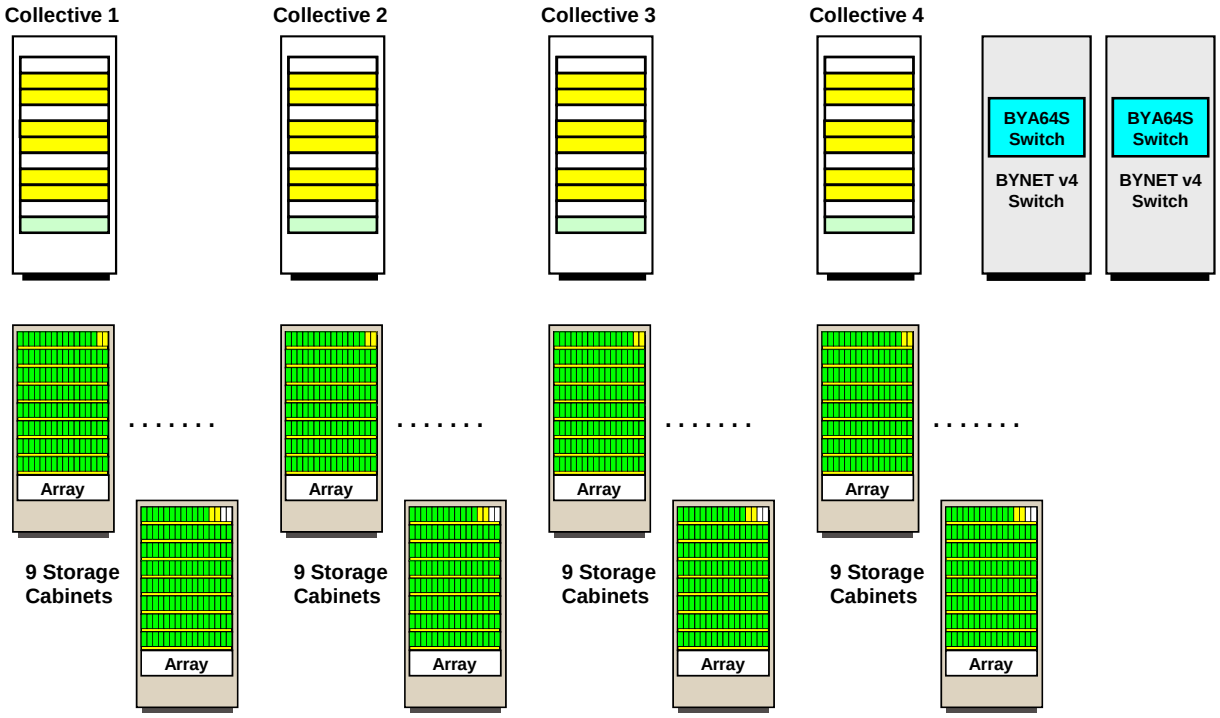
In the illustration on the facing page, disk areas shown in green color are used as data disks, disk areas shown in yellow color are hot spare drives, and disk areas shown in white are empty.

What types of cabinets are needed to upgrade this system to 63 (42 + 21) nodes?

Answer:

3 additional 5650H "Expansion" racks, each with 9 nodes
27 additional 6844 Storage racks

5650H System – 36 (24+12) Nodes



What cabinets are needed to grow this to a 63 (42+21) node system?

Teradata Configuration Examples

The examples on the facing page show a typical AMP and Disk configuration for 5555 and 5600 cliques.

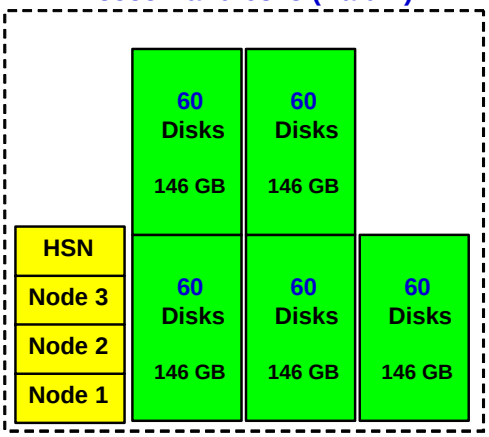
The 5555 example on the left uses 146 GB disk drives and the 5600 example on the right uses 300 GB disk drives.

An example of a 5600 would be 40 AMPs per node with 4 disks per AMP. Typical clique characteristics are:

- 2+1 nodes
- Total of 80 AMPs and 320 disks
- 3 LSI disk arrays

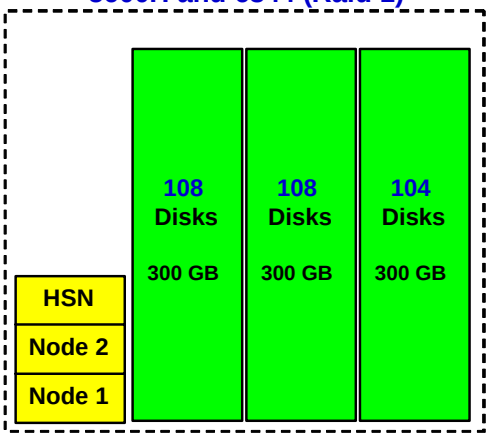
Teradata Configuration Examples

5555H and 6843 (Raid 1)



Note:
Each disk array will typically have additional global hot spare drives.

5600H and 6844 (Raid 1)



5555H (3+1 nodes/clique) – Linux

25 AMPs / Node
75 AMPs / Clique

100 Disks per Node
300 Disks per Clique

Each Vdisk – 4 Disks (RAID 1)
Each Vdisk – 263 GB*

Clique – 75 AMPs x 263 GB = 19.7 TB*

5600H (2+1 nodes/clique) – Linux

40 AMPs / Node
80 AMPs / Clique

160 Disks per Node
320 Disks per Clique

Each Vdisk – 4 Disks (RAID 1)
Each Vdisk – 540 GB*

Clique – 80 AMPs x 540 GB = 43.2 TB*

* **Actual MaxPerm space is app. 90%.**

What is BYNET Version 2.1?

The facing page lists the performance improvements of BYNET Version 2.1 as compared to BYNET Version 2.0. BYNET v2.1 is an incremental release to BYNET v2.0 that increases the PCI throughput of the BYNET interface adapters.

The SMP PCI interface (BIC) is changed to 64-bit (Wide) and 66 MHz (Fast). It is also downward compatible (64 bit or 32 bit @ 66 or 33 MHz). The new BICs are the BIC2M and the BIC4M.

The major benefit is that the PCI I/O throughput of the BIC is quadrupled to eliminate a potential bottleneck. Although it might seem that since a BYNET channel is full duplex, it could send/receive at 120 MB/sec. In reality, the bandwidth is about 95 MB/sec.

Since there are two BYNETs, then 2 x 95 MB/sec = 190 MB/sec.

However, BYNET v2.0 BICs are PCI 32-bit, 33 MHz adapters and have a PCI throughput limit of 100 MB/sec. Even though the BYNET can send/receive 195 MB/sec., the BIC limits the throughput to 100 MB/sec.

BYNET v2.1 BICs are PCI 64-bit, 66 MHz adapters and have a PCI throughput of 400 MB/sec.

There have been no changes to BYNET v2.0 switch infrastructure with BYNET v2.1. The same BYNET connectors, cables, and switch chassis modules are used.

Another enhancement is the BYA4M switch that supports PCI Narrow/Fast so as to not degrade PCI fast bus.

BYNET v2.1 is interoperable with Bynet v2.0.

What is BYNET Version 2.1?

- **An incremental release to BYNET v2.0 that increases the PCI throughput of the BYNET interface adapters.**
- **The SMP PCI interface is changed to 64-bit (Wide) and 66 MHz (Fast). It is also downward compatible (64 bit or 32 bit @ 66 or 33 MHz).**
  - **BIC2M for 4900**
  - **BIC4M for 5300**
- **PCI I/O throughput of the BIC is quadrupled to eliminate a potential bottleneck.**
- **One BYNET channel is full duplex - the actual throughput is about 95 MB/sec.**
  - **Therefore, two BYNET channels is 2 x 95 MB/sec. = 190 MB/sec.**
  - **BYNET v2.0 BIC (PCI 32-bit, 33 MHz) PCI throughput is limited to 100 MB/sec.**
  - **BYNET v2.1 BIC (PCI 64-bit, 66 MHz) PCI throughput is 400 MB/sec.**
- **No changes to BYNET v2.0 switch infrastructure.**
- **BYA4M switch supports PCI Narrow/Fast so as to not degrade PCI fast bus.**
- **BYNET v2.1 is interoperable with Bynet v2.0.**

BYNET Interface Cards (BIC) or Adapters

BIC2G BYNET Adapter

The WorldMark 4800 BYNET uses the BIC2G and BYA4 switches to connect up to 4 SMP nodes. The role of the BIC2G Adapter board is to provide a processing node with access to two independent BYNET networks via a single PCI slot. One BIC2G board is required in each SMP node.

The BIC2G is a PCI 2.1 compliant 32 bit / 33 MHz adapter. Following the standard WorldMark 4800/5200 SMP configuration, the BIC2G adapter is installed in PCI slot 1 and is assigned IRQ 5 from the SSU (identified as a multifunction adapter).

The BIC2G adapter (used with 4800 systems) actually has 4 channels or ports.

- Two 125 Megabit/sec. channels, compatible with BYNET V1.0 and V1.1 switches.
- Two 1 Gigabit/sec. channels, compatible with BYNET V2.0 switches.

BIC4G BYNET Adapter

The BIC4G adapter board interfaces the WorldMark 4850, 5200, and 5250 SMP nodes to the BYNET network. Each BIC4G provides the circuitry to connect to up to four BYNET Version 2 networks. All four of the ports on the BIC4G are based on a 1 Gbit/sec Fibre Channel interface.

Important: Only 2 BYNET networks are supported on the current release of the MPP systems.

One BIC adapter is housed in each SMP node chassis this adapter provides the interface to both networks.

The BIC4G is a PCI 2.1 compliant 32 bit / 33 MHz adapter. Following the standard WorldMark 4850, 5200, 5250 SMP configuration, the BIC4G adapter is installed in PCI slot 1 and is assigned IRQ 5 from the SSU (identified as a multifunction adapter).

BYNET Adapters

BIC2G BYNET Adapter
- Used with 4800 SMPs
- 4 ports - two V1 channels and two V2 channels

BIC4G BYNET Adapter
- Used with 52xx SMPs
- 4 ports - four V2 channels

BIC2C BYNET Adapter
- Used with 485x SMPs
- 2 ports - two V2 channels

BIC2M BYNET Adapter
- Used with 4900 SMPs
- 2 ports - two V2.1 channels

BIC4M BYNET Adapter
- Used with 5300 SMPs
- 4 ports - two V2.1 channels

BYNET Switches

BYNET 4 Switch (BYA4P) - a PCI card designed to interconnect up to 4 SMPs. This switch is effectively a BYNET V1.1 switch (10 MB/sec.) and is used in the 4800. The BYA4P is a PCI card that is placed into a PCI slot of an SMP.

BYNET V2 4 Switch (BYA4G) - PCI card designed to interconnect up to 4 SMPs. This switch is a BYNET V2 switch (60 MB/sec.) designed for 485x systems. The BYA4G is a PCI card that is placed into a PCI slot of an SMP.

BYNET V2.1 4 Switch (BYA4M) - PCI card designed to interconnect up to 4 SMPs. This switch is a new BYNET V2.1 switch (60 MB/sec.) designed for 4900 systems. The BYA4M is a PCI card that is placed into a PCI slot of an SMP.

BYNET V2 16 Node Switch (BYA16G) – this V2 switch (60 MB/sec.) allows up to 16 5200 SMPs to interconnect. This 3U chassis switch resides in the 5200/5300 System Cabinet.

BYNET V2 64 Node Switch (BYA64GX chassis) – this V2 switch is actually composed of 8 BYA8X switch boards in the BYA64GX chassis. Each BYA8X switch board allows up to 8 SMPs to interconnect (i.e., 8 switches x 8 SMPs each = 64 SMPs). The BYA64GX is actually a backpanel that allows the 8 BYA8X switch boards to interconnect. The BYA64GX also includes a Diagnostic Processor (DP) board. This 12U chassis resides in either the BYNET V2 64 Node Switch cabinet or the BYNET V2 64/512 Node Expansion Cabinet.

> Note: BYA8X switch board (in BYA64GX chassis): This is Stage A base switch board. Each board supports 8 links to nodes. The BYA64GX chassis can contain a maximum of 8 BYA8X switches, allowing for 64 links to nodes. In systems greater than 64 nodes, the BYA8GX switch boards also connect the BYA64GX chassis to BYB64G chassis through X-port connectors, one on each BYA8X board.

BYNET V2 512 Node Switch (BYB64G chassis) – this V2 switch is actually composed of 4 BYB16G switch boards in the BYB64X chassis. This is effectively the Stage B expansion switch board. These boards interconnect through the BYB backpanel to provide 8 expansion ports. The expansion ports are used to interconnect BYA64GX chassis (through X-port connections on BYA8X switch boards). A maximum of 8 BYA64GX chassis can be interconnected to a maximum of 8 BYB64G chassis (one X-port connection from each BYA64 chassis to the expansion port on each BYB64 chassis), thus providing up to 512-node switching capacity. This chassis also include a Diagnostic Processor board.

BYNET Switches

BYA4P Switch - Version 1.1 switch
- used with 4800 systems
- PCI card that connects up to 4 SMPs

BYA4G Switch - Version 2 switch
- used with 485x systems
- PCI card that connects up to 4 SMPs

BYA4M Switch - Version 2.1 switch
- used with 4900 systems
- PCI card that connects up to 4 SMPs

BYA16G Switch (BYA16G) - Version 2 switch
- used with 52xx/5300 systems for up to 16 SMPs
- 3U chassis that resides in 52xx System Cabinet

BYA64GX Switch - Version 2 switch
- connects up to 64 52xx/5300 SMPs
- 12U chassis resides in BYNET V2 Switch Cabinet

BYB64G Expansion Switch - Version 2 switch
- connects up to 8 BYA64GX switches together (8 x 64 nodes = 512 nodes)
- 12U chassis resides in BYNET V2 Switch Cabinet

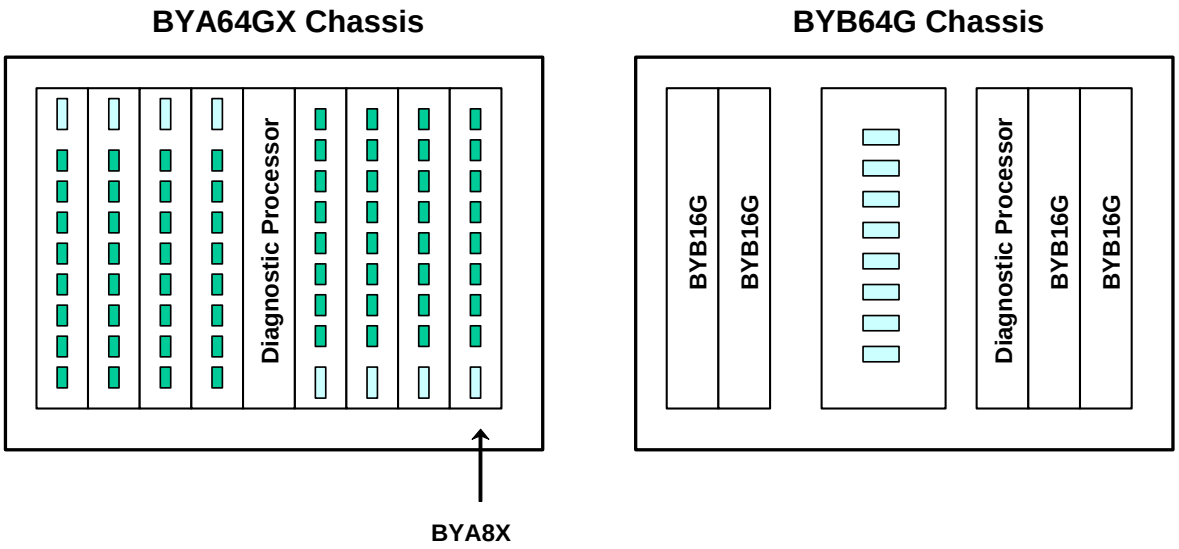
BYNET Switches (BYA64GX and BYB64G)

The facing page illustrates the purpose of BYA64GX and BYB64G switches. BYA switches connect SMPs. With more than 64 SMPs, multiple BYA switches are needed for one BYNET and the multiple BYA switches are connected together with BYB switches.

BYNET V2 Switches

Conceptual views of the BYA64GX chassis and the BYB64G chassis are shown on the facing page.

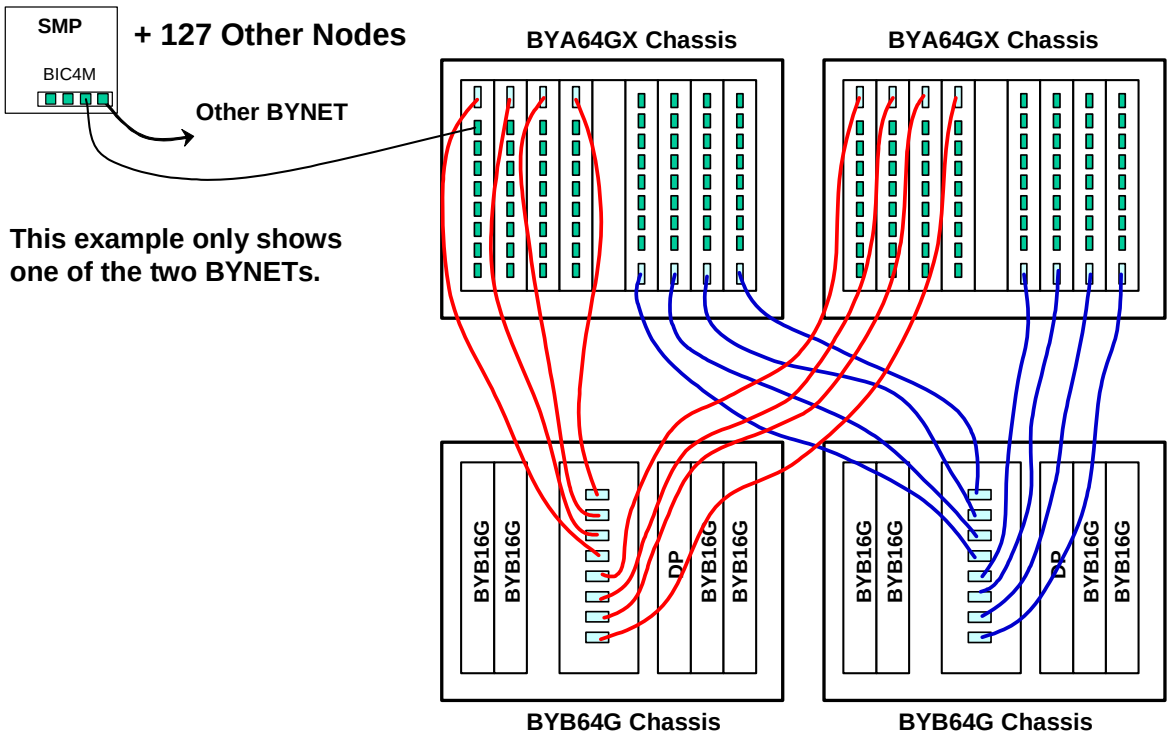
BYNET V2 Switches



Example of BYNET V2 Switches - 128 Nodes

A conceptual view of the BYNET switches for 128 nodes is shown on the facing page.

Example of BYNET Switches for 128 Nodes



BYNET™ Software

The **bynet** software package is required for the Teradata Database. This package contains the **blm** (BYNET Link Manager) and **bdl** (BYNET Data Link) drivers. This package provides for both BYNET protocol and TCP/IP communication across the BYNET.

BYNET™ Device Drivers

The **bynet** software package provides two primary drivers to access the BYNET.

- **blm** – BYNET Link Manager
- **bdl** – BYNET Data Link

BLM

The BLM software driver is a STREAMS-based UNIX driver and is responsible for managing the BIC adapters (physical device). The BLM driver places messages/data on the BIC adapters and is the UNIX interface to the BYNET. This driver is linked into the UNIX kernel.

The following software currently uses the BLM driver:

- Teradata Database software to directly access the BYNET.
- BDL software driver – interface between TCP/IP software and BLM.
- Can also be used by databases or applications other than Teradata Database.

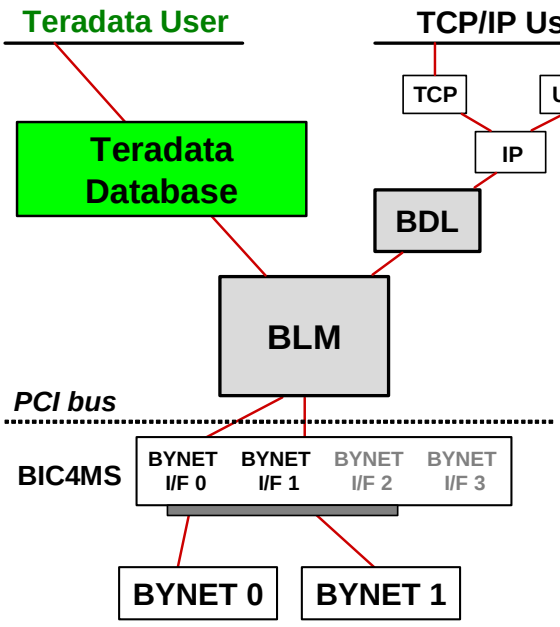
There are two **blmd** daemons running on an SMP node – one for each BYNET. The functions of these daemons include:

- Topology generation when UNIX initializes and when topology needs regenerating.
- Handle heartbeat messages.

BDL

The BDL software driver is also a STREAMS-based UNIX driver and its primary purpose is to allow TCP/IP over the BYNET.

BYNET™ Software



BYNET software package (bynet)

- **provides the blm and bdl drivers for the BYNET.**
  - **Provides features used by Teradata.**
- **Simple (UNIX) diagnostic utility is**

  # /usr/bin/bam

bam -s

Board	State	Nodes	Net #	Net name
BPCI 0	Online	2	0	BYA022-0
BPCI 1	Online	2	1	BYA022-1

Enterprise Storage Solutions

The chart below attempts to clarify the major releases of Engenio (LSI Logic) Disk Arrays.

Key for this chart:

NSC – NCR Storage Cabinet
WES – WorldMark Enterprise Storage
DS – Deskside or Pedestal
MP-RAS – UNIX MP-RAS
Win NT – Windows NT
Win 2000 – Windows 2000

Release	Key Feature	Disk Array Class/Model	O.S. Support	GCA
NSC 1.0	Modular Arrays (SCSI)	6285-1220 6285-1440	MP-RAS, Win NT, Win 2000	1998
NSC 2.0	Fibre Channel	6286-1220 Rack or Deskside	Win NT, Win 2000	1999
WES 3.0	Quad Array (SCSI)	6288-1440	MP-RAS	1999
WES 3.5	Quad Array (SCSI)	6288-1452	MP-RAS	2000
WES 4.0	Fibre Channel	6289-1440	Win 2000	07/2000
WES 5.0	Fibre Channel	6840-1440 6840-1456	MP-RAS	05/2002
Enterprise Storage (5.5)	Fibre Channel	6841-2456	MP-RAS, Win 2000	08/2003
Enterprise Storage (6.0)	Fibre Channel	6841-6456	MP-RAS, Win 2000	08/2003
Enterprise Storage (6.1)	Fibre Channel	6841-7456	MP-RAS	07/2004

Enterprise Storage Solutions

Engenio Disk Array Comparisons

Feature	6284	6841-6456	6841-7456
Interface	Fibre Channel 2 Gbit (end-to-end)	Fibre Channel 2 Gbit (end-to-end)	Fibre Channel 2 Gbit (end-to-end)
Teradata Use	SMP only	MPP	MPP
Operating Environments	UNIX MP-RAS Windows	Windows UNIX MP-RAS	UNIX MP-RAS
Disks	36 GB – 15K RPM 73 GB – 15K RPM	36 GB – 15K RPM 73 GB – 15K RPM	36 GB – 15K RPM 73 GB – 15K RPM
Node to Array Ratio (typical)	1 : 1 or 2	1 : 1 (36 GB) 2 : 1 (73 GB)	1 : 1 (36 GB) 2 : 1 (73 GB)
Maximum Teradata Disks (array/cabinet)	28	56/112	56/112
Floor Space (width)	Deskside	Rack - 1 Floor Tile	Rack - 1 Floor Tile

SMP Connectivity – Fibre Channel

The WorldMark Enterprise Storage (WES) 6840 Solution uses a Fibre Channel Arbitrated Loop (FCAL) interconnect scheme for the Teradata cliques. This scheme effectively uses point-to-point Fibre Channel cables from the host (SMP) to the disk array controller. This provides better performance, higher reliability, and easier cabling and configurations.

6841-2456 Fibre Channel Disk Array

Cabinet characteristics include:

- 2U Server Management Chassis
- Three 2U UPS with Dual AC Distribution Boxes
- Support for two arrays for a total of 112 drives and 4.1 TB raw data capacity using 36 GB disk drives

Fibre Channel Disk Array characteristics include:

- Dual hot swappable redundant 4884 Fibre Channel RAID controllers with 1 GB of cache and an Intel 550 MHz Celeron microprocessor.
- Each 6841-2456 disk array has support for up to 56 hot swappable 2 Gbit Fibre Channel 36GB 15K rpm disk drives (Seagate Cheetah drives).
- Quad Fibre Channel Host Adapter (2 Gb/sec) point-to-point connectivity between processing node and disk arrays.
- Support for RAID 1 and RAID 5.

Performance and availability features include:

- Quad Ported Fibre Channel Host Adapters
- Dual ported Fibre Channel drives for higher availability and reliability
- Quad Modular Fibre Channel RAID Controllers are designed to match the I/O demands of 2+ GHz nodes.
- Fibre optic point-to-point connections between node and array provide for higher performance and greater distances between node and array.

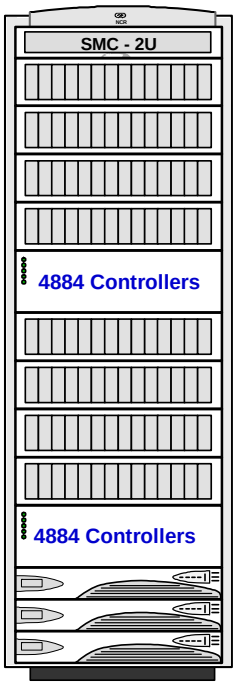
6841-2456 Fibre Channel Disk Array

Cabinet characteristics:

- Support for two arrays for a total of 112 drives.

Fibre Channel Disk Array characteristics:

- Utilizes new drive tray that provides 2 Gbit access to disks. I/O path is 2 Gbit from hosts to array controllers to disk drives.
- Dual hot swappable redundant 4884 Fibre Channel RAID controllers
  - I/O maximum throughput (per array) is approximately 250 MB/sec.
- Utilizes 2 Gbit Fibre Channel 36 GB 15K rpm disk drives.
- Typical configuration usage within a clique:
  - Typically used in coexistence environments
  - 4 nodes sharing 4 arrays
- Support for RAID 1 and RAID 5.



6841-2456

LSI Logic – Models 1000, 2000, & 3000

The Enterprise Storage 7.0 (NS 7.0) solution utilizes a newly designed cabinet (model 6700). The benefits include:

- New rack frame with integrated cable channel in rear
- New rack to be implemented across all product lines.
- Cable harnesses, like 54XX, fit to connector location, labeled & color coded
- RoHS (Restriction of Hazardous Substances) compliant – standard required by European Union starting July 1, 2006

The disk arrays that will be placed in the 6700 storage cabinet are 6843 disk arrays. Characteristics and benefits of the 6843 array include:

- Higher performance controllers that support 4Gb FC end-to-end
- 16 drive slots per tray
- DAP error protection thru drives (DAP-1 and DAP-3 protection)
- Up to 9 trays per array (144 drives) – up to 29% increase in storage density
- RoHS compliant

Note: DAP – Data Availability Protection. The 6843 provides DAP-1 (Intra-Controller Protection) and DAP-3 (Controller to Disks and Back) protection. DAP-3 is new with the 6843.

6700 Cabinet Models

6700-1000 Characteristics include:

- 1 Controller Module with 3 drive trays (6843-1000) – RAID 1 only
- This array can be expanded with two additional 3-drive tray modules
- The array in this cabinet will have 3, 6, 9 drive trays
- 2 Ethernet Switches, 4 UPS's, no SM3G, and 50 Amp AC Power

6700-2000 Characteristics include:

- 1 Controller Module with 4 drive trays (6843-2000) – RAID 1 or 5
- This array can be expanded with one additional 4-drive tray module
- The array in this cabinet will have 4 or 8 drive trays
- 2 Ethernet Switches, 4 UPS's, optional SM3G, and 50 Amp AC Power

6700-3000 Characteristics ("Lite" version) include:

- 1 Controller Module with 3 drive trays (6843-1000) – RAID 1 only
- This array can be expanded with one additional 3-drive tray module
- The array in this cabinet will have 3 or 6 drive trays
- 2 Ethernet Switches, 3 UPS's, optional SM3G, and 30 Amp AC Power

LSI Logic – Models 1000, 2000, & 3000

The Storage Cabinet (model 6700) has three models supporting the 6843 Disk Array.

Common characteristics:

- New cabinet has one controller module (dual array controllers) that support 4 Gbit Fibre Channel speeds.
- Drive trays support up to 16 disks – 73 GB and 146 GB drives.

Array Cabinet 6700-1000 (RAID 1 only)

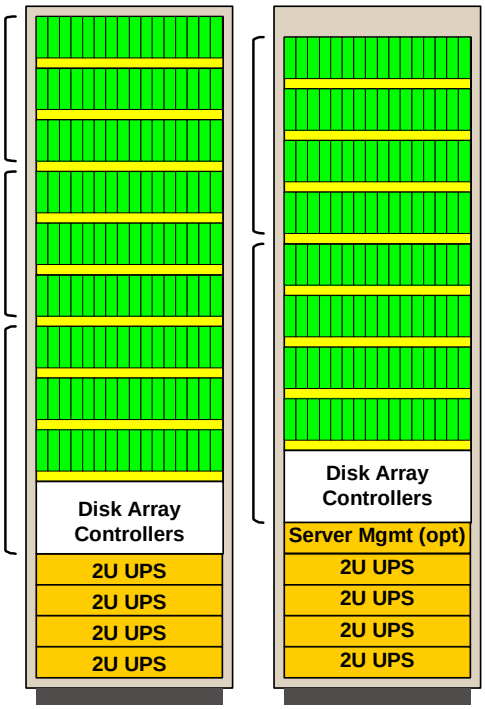
- Supports 3, 6, or 9 drive trays – maximum of 144 disks
- No Server Management Chassis (for 54xx/5500 systems)

Array Cabinet 6700-2000 (RAID 1 or RAID 5)

- Supports 4 or 8 drive trays – maximum of 128 disks
- Optional Server Management Chassis

Array Cabinet 6700-3000 (RAID 1 only)

- Supports 3 or 6 drive trays – maximum of 96 disks
- Optional Server Management Chassis



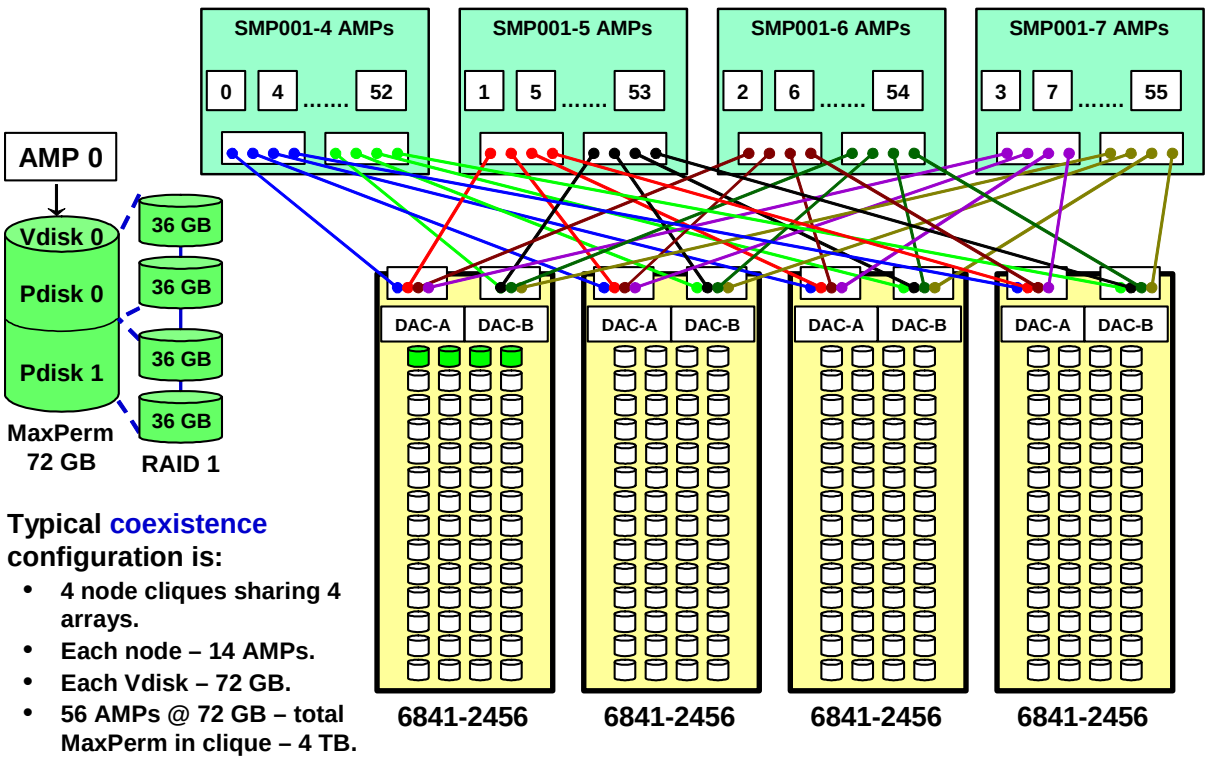
6700-1000 6700-2000

5380 and 6841-2456 Disk Arrays

The facing page contains an example of a 4-node clique sharing **four** 6841-2456 disk arrays.

Each SMP has 2 Quad Fibre Channel adapters (2 Gbit/sec.). Fibre Channel cables are point-to-point connections.

5380 and 6841-2456 Disk Arrays



Typical coexistence configuration is:

- 4 node cliques sharing 4 arrays.
- Each node – 14 AMPs.
- Each Vdisk – 72 GB.
- 56 AMPs @ 72 GB – total MaxPerm in clique – 4 TB.

Server Management (2nd Generation)

To communicate with processing nodes and system components, the AWS is connected to the system via two Ethernet LANs.

- SLAN – Service (or System) Ethernet LAN known as the SLAN; connects AWS to the CMIC (Chassis Management Interface Controller) in the Server Management Chassis of each cabinet

- PvtLAN – Private Ethernet LAN - connects AWS directly to SMPs.

Service LAN (SLAN)

The Service or System LAN (SLAN) is a private Ethernet LAN between the AWS and CMICs in a 48xx/52xx and 4900/5300 systems. The CMIC (Chassis Management Interface Controller) is the intelligent component (actually an Intel processor) of the SMC. The CMIC provides network connectivity for the SMC to the SLAN.

The MLAN (Management LAN) is an internal LAN that connects components to the CMICs and also connects the SMC in a storage rack to the SMC in a processor rack.

- System Events
- Console and Diagnostic connections

The new recommended name for the SLAN is the Service LAN.

Private LAN (PvtLAN)

This Ethernet LAN is used to directly connect the AWS to each processing node (SMP).

- All SMPs are connected using 10baseT (Twisted Pair).

- Up to 100 Mbps full duplex to AWS.

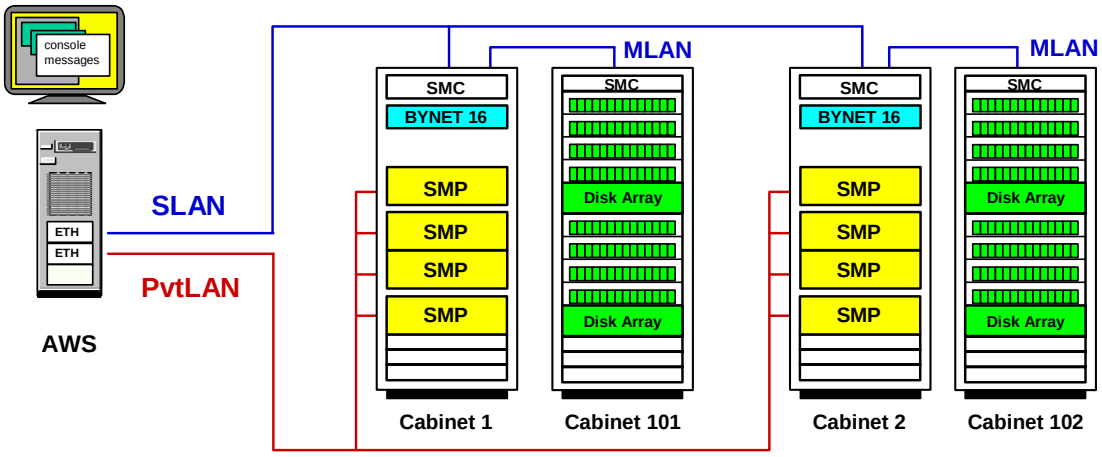
- The PvtLAN uses hubs and switches to allow connection to all nodes.

The PvtLAN is also connected to the disk array controllers in the 6841 (and future) Fibre Channel disk arrays.

Server Management (2nd Generation)

AWS (UNIX or Windows) is connected to the system via two Ethernet LANs.

- **SLAN (Service or System Ethernet LAN) – connects AWS to the CMIC in the SMC of each processor cabinet.**
  - **Used for hardware functions; console connections, power control, etc.**
  - **CMIC is connected to components within rack via "serial" type connections.**
- **PvtLAN – Private Ethernet LAN - connects AWS directly to SMPs.**
  - **Used for software functions; LAN connections, Teradata Database Window, etc.**



Server Management with AWS

The AWS provides a single operational view of a Teradata Large System (e.g., 2500, 54xx, 55xx), and the environment to configure, monitor, and manage the system. The AWS is a UNIX-based or Windows-based processor with a user-friendly graphical interface.

The AWS effectively becomes a central console for MPP systems.

The AWS is one part of the Server Management subsystem that provides monitoring and management capabilities of MPP systems.

1st Generation Server Management (3600) – Server Management (SM) processing, storage and display occurred on AWS.

2nd Generation Server Management (5100, 48xx/52xx, 49xxx/53xx) – most SM processing occurs on CMICs and Management Boards. The AWS still provides all the storage and display.

3rd Generation Server Management (SM3G) – new with 2500, 54xx, and 55xx – all SM processing and storage occurs on the 2U management nodes (effectively CMICs). These CMICs also enable customizable displays (a.k.a., Service Nodes or Panels). The AWS provides the on-site management views.

Server Management (3rd Generation) – SM3G

The 54xx/55xx platform uses a new Server Management architecture, SM3G (Third Generation Server Management). The Third Generation Server Management Chassis manages the communication interface between the AWS and other chassis located within the cabinet.

The Server Management subsystem in a 54xx/55xx uses industry standard parts, **2U Server Management Node** (similar to a processing node and referred to as a CMIC node) and **Ethernet switches** to implement an Ethernet based Server Management solution. This new Server Management is referred to a Third Generation Server Management (SM3G).

SM3G provides all of features/functionality of the legacy platform sever management. Note: the acronym SM3G should be used for 3rd Generation Server Management because AT&T already has a product named 3GSM.

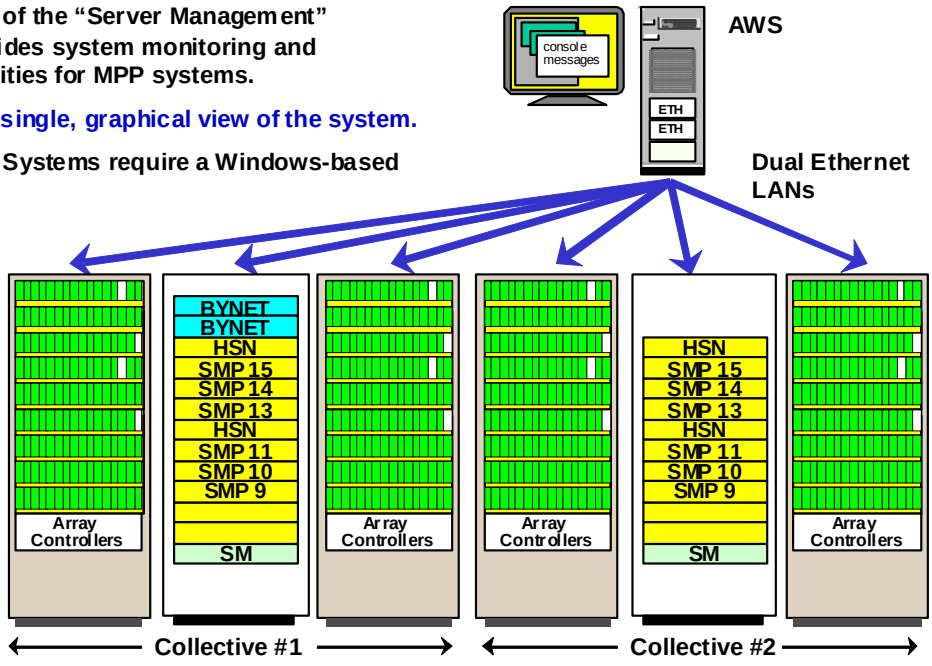
Industry Standard Ethernet Interfaces

One of the reasons for the new Server Management subsystem is to better adhere to industry standards. Ethernet-based management is now the industry standard for chassis vendors.

Server Management with AWS

The AWS is a standalone Windows-based workstation that is the primary operations interface for MPP systems.

- The AWS is one part of the "Server Management" subsystem that provides system monitoring and management capabilities for MPP systems.
- The AWS provides a single, graphical view of the system.
- 15xx/25xx/54xx/55xx Systems require a Windows-based AWS.



AWS provides GUI to ...

- connect to SMPs
- connect to DB Window
- power on/off/reset
- manage faults
- obtains h/w or s/w status information

Server Management (3rd Generation) – SM3G

The 5400 platform uses a new Server Management architecture, SM3G (Third Generation Server Management). The Third Generation Server Management Chassis manages the communication interface between the AWS and other chassis located within the cabinet.

The Server Management subsystem in a 5400 uses industry standard parts, **2U Server Management Node** (similar to a processing node and referred to as a CMIC node) and **Ethernet switches** to implement an Ethernet based Server Management solution. This new Server Management is referred to a Third Generation Server Management (SM3G).

SM3G provides all of features/functionality of the legacy platform sever management. Note: the acronym SM3G should be used for 3rd Generation Server Management because AT&T already has a product named 3GSM.

Industry Standard Ethernet Interfaces

One of the reasons for the new Server Management subsystem is to better adhere to industry standards. Ethernet-based management is now the industry standard for chassis vendors.

Node Interface (including CMICs)

The nodes – processing and sever management (CMICs) – have two Ethernet ports on the system board; these two ports provide the connections to the Primary and Secondary Server management networks. Through BIOS Setup, console redirection is enabled and sent over the Primary network.

BYNET Interfaces

Each BYNET chassis (BYA32G, BYA64GX, and BYC64G) has a single Ethernet port on the Diagnostic Processor (DP) interface that connects to Primary management network.

The BYOX and BYCLK chassis do not connect to the server management network; instead they are managed from the BYA64GX and BYC64G chassis.

Engenio Disk Array Controllers Interface

Each Disk array controller module has two Ethernet ports, one for controller A (Primary Network) the second for controller B (Secondary Network).

Fibre Channel Switch

Each QLogic Fibre Channel Switch (FCS) has a single Ethernet interface that is connected to the Primary management network.

UPS Interface

Each UPS has a Dual Ethernet UPS Interface (DEUI) plug-in adapter that replaces the legacy RS-232 port. The primary and secondary management networks connect to this adapter.

Server Management (3rd Generation) – SM3G

NCR 5400 System Monitoring and Management is provided through Ethernet-based server management – called Server Management 3rd Generation (SM3G).

- Server management functions and software execute on the Server Management node (also referred to as the CMIC node).
- Redundant Ethernet connections to most chassis provide the management path.

Advantages of using an Ethernet-based server management subsystem are:

- Enables improvements for remote service applications such as AWS
  - Option of customized "Service Node" applications (future)
- Industry standard for chassis vendors (Ethernet instead of serial)
- Reduced number of SMC in system (storage and BYNET cabinets don't need a SMC)

Collective – a collective is the collection of chassis controlled by a CMIC.

- Generally consists of a node cabinet and its associated disk array cabinets.
- There is a CMIC controlling each collective in the system and an optional fail-over CMIC node for systems with only one collective.
- In systems with more than one collective, the CMIC nodes from different collectives will provide fail-over for each other. Every node cabinet will have CMIC (SM chassis).

SM3G Architecture Description

SM3G provides chassis level management for the following chassis:

MP-RAS Nodes	Bynet V3
CMIC Chassis	2u UPS
Windows Nodes	Engenio Disk Array
Fibre Channel Switch	

Server Management Networks

The Server Management Network consists of two physical networks; Primary Network (PN) and Secondary Network (SN).

Primary Network

The purpose of the Primary Network is to provide direct communication between the AWS(s), CMIC(s) and all rack mounted chassis. For co-existence with Legacy Systems, the SLAN will connect to the Primary Network.

Secondary Network

The purpose of the Secondary Network is to provide backup communications between those chassis that support a second Ethernet port. For co-existence with Legacy Systems, the PLAN will connect to the Secondary Network.

CMICS and Collectives

The SM3G CMIC (CMIC4) will has similar functionality to the legacy CMIC (CMIC3) but the CMIC hardware is an industry standard Intel based computing node (same as the 2U processing node). The CIMC4 will utilize all new code.

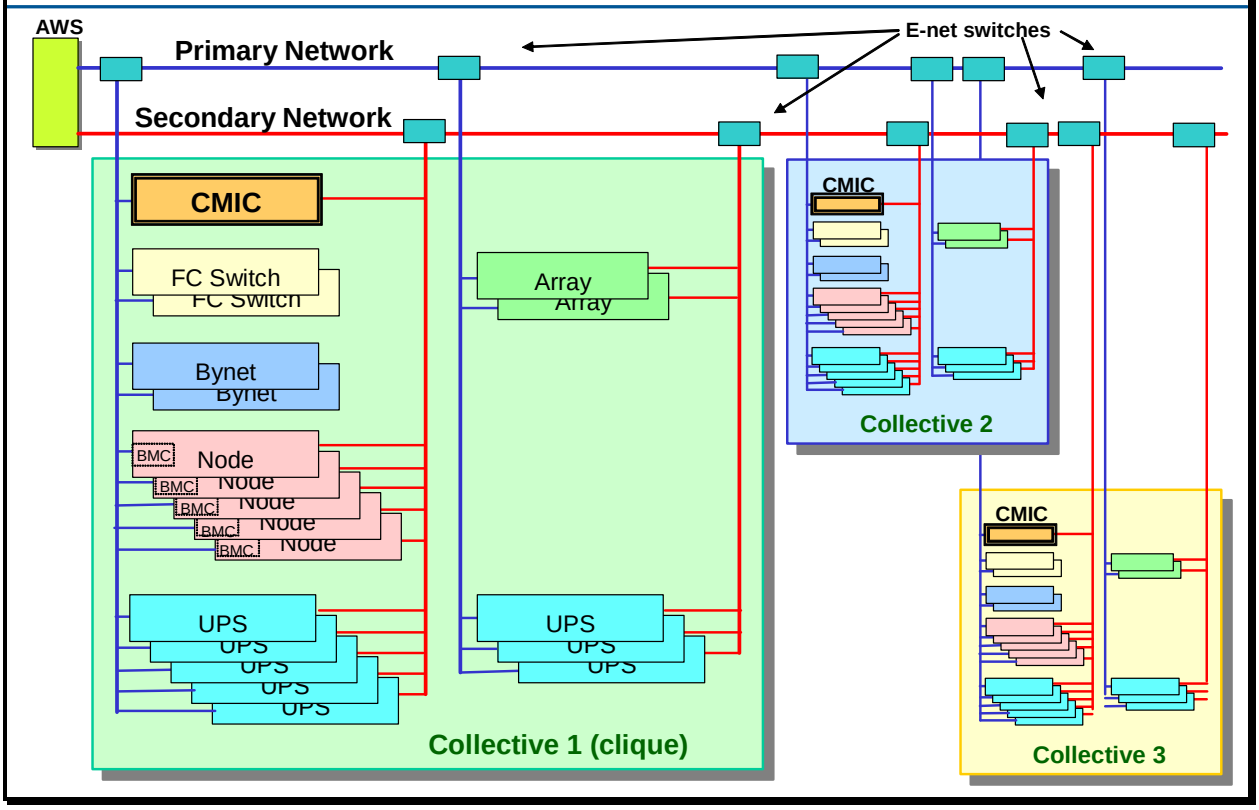
A **collective** is the collection of chassis controlled by a CMIC. Generally it includes a node cabinet and its associated disk array cabinets which can include Bynet chassis resident in node cabinets.

There is a CMIC controlling each collective in the system and an optional fail-over CMIC for systems with only one collective. In systems that have more than one collective the CMICs from different collectives will provide fail-over for each other. CMICs always reside in the node cabinet.

Some key SM3G functional changes include:

- Redistribute some AWS functionality (other than display related) to the CMIC
- Nodes take over AC fail operation
- Employ CMIC failover

SM3G Architecture



SM3G Chassis Interfaces

Node Interface (including CMICs)

The nodes (processing and CMICs) have two Ethernet ports on the system board, these two ports proved the connections to the Primary and Secondary Server management networks. Through BIOS Setup, Console redirection is enabled and sent over the Primary network.

UPS Interface

Each UPS has a Dual Ethernet UPS Interface (DEUI) plug-in adapter that replaces the legacy RS-232 port. The primary and secondary management networks connect to this adapter.

BYNET Interfaces

The BYNET chassis (BYA16Gx, BYA64GX, and BYC64) has a single Ethernet port on the Diagnostic Processor (DP) interface that connects to Primary management network.

There is no MLAN port on the SM3G supported chassis. The CMB interface on the BYA64 and BYC64 chassis was modified to provide status through the DP's Ethernet port.

The BYOX and BYCLK chassis do not connect to the server management network, instead they are managed from the BYA64 and BYC64 chassis.

Engenio Disk Array Controllers Interface

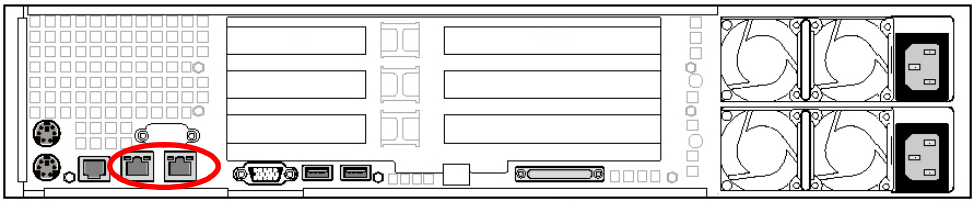
Each Disk array controller module has two Ethernet ports, one for controller A (Primary Network) the second for controller B (Secondary Network).

Fibre Channel Switch

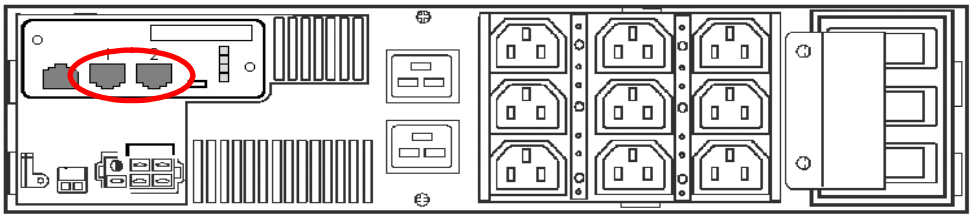
The QLogic Fibre Channel Switches (FCS) have a single Ethernet interface that is connected to the Primary management network.

SM3G Chassis Interfaces

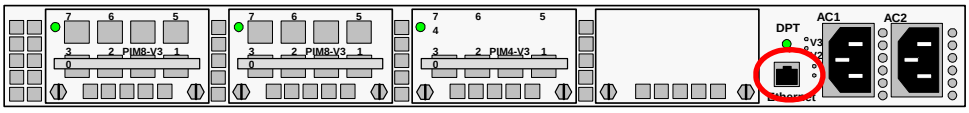
Nodes (Processing & CMIC)



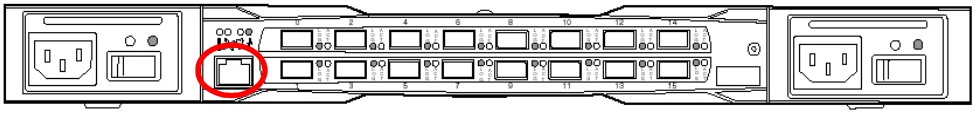
UPS



BYNET (BYA16GX)



Fibre Channel Switch



SM3G Fault Tolerance

Primary Network Switch Failures

AWS, CMIC, and Chassis communications will occur through the secondary network (as possible) if the primary network fails

The following functions are supported on both networks, therefore they will continue to function if either network fails:

CMIC heartbeats / failover
Node and CMIC:
    Orderly Reset
Software Event Forwarding
    Telnet / Remote Desktop Consoles
UPS Status, Events, Control including:
    AC Fail processing - SMP and CMIC against UPS's in Node Cabinets
    Enhanced AC Fail processing - CMIC against UPS's in Disk/Bynet Cabinets
LSI Disk Array Status, Events, Control, and Console
    Assuming Controller-B is Dual Active

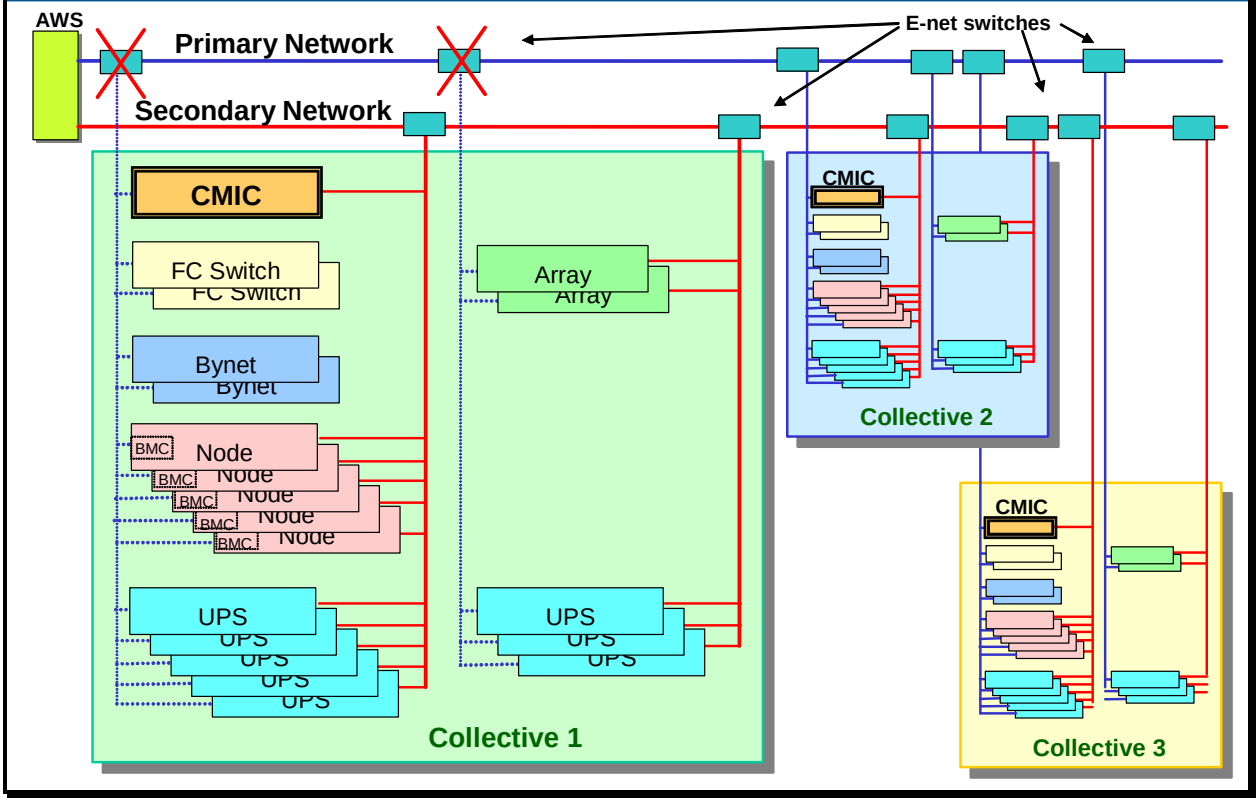
The following functions will NOT be supported if the primary network fails::

Node and CMIC Hardware Events and Status
  Out-of-Band Commands:
        Power On, Hard Power Off, Hard Reset, NMI, View SEL
Out-of-Band Consoles (console redirection is enabled on primary only)
Node and CMIC Orderly Shutdown (would not be able to power them back on)
Fibre Channel Switch Status, Events, Control, and console (no Secondary network connection
Bynet V3 Status, Events, Control, and Console (no Secondary-network connection)
Power Cable Verification (will lose communications when Secondary-network cabinet switch powers off)
Hardware configuration changes (bootp clients/servers)

Secondary Network Switch Failures

There will be no impact if the secondary network fails.

SM3G Fault Tolerance



Notes

TERADATA
Raising Intelligence

Appendix T: Table of Contents

This Appendix contains the combined table of contents for each module in this course.

Notes

Table of Contents

Module 1 – Overview

What is Teradata? 1-4
Teradata – A Brief History 1-6
What is a Data Warehouse? 1-8
What is Active Data Warehousing? 1-10
What is a Relational Database? 1-12
Answering Questions with a Relational Database 1-14
Teradata Database Competitive Advantages 1-16
Module 1: Review Questions 1-18

Module 2 – Teradata Basics

Major Components of Teradata 2-4
Teradata Storage Architecture 2-6
Teradata Retrieval Architecture 2-8
Multiple Tables on Multiple AMPs 2-10
Linear Growth and Expandability 2-12
Teradata Objects 2-14
The Data Dictionary Directory (DD/D) 2-16
Structure Query Language (SQL) 2-18
CREATE TABLE – Example of DDL 2-20
Views 2-22
Multi-Table Views 2-24
Macros 2-26
HELP Commands 2-28
SHOW Command 2-30
EXPLAIN Facility 2-32
Summary 2-34
Module 2: Review Questions 2-36

Module 3 – Teradata Database Architecture

Teradata and MPP Systems 3-4
Teradata Functional Overview 3-6
Channel-Attached Client Software Overview 3-8
Network-Attached Client Software Overview 3-10
The Parsing Engine 3-12
Message Passing Layer 3-14
The Access Module Processor (AMP) 3-16
Teradata Parallelism 3-18
Module 3: Review Questions 3-20

Module 4 – Teradata Databases and Users

A Teradata Database 4-4
A Teradata User 4-6
Database – User Comparison 4-8
The Hierarchy of Databases and Users 4-10
Example of a System Hierarchy 4-12
Permanent Space 4-14
Spool Space 4-16
Temporary Space 4-18
Creating Tables 4-20
Data Types 4-22
Access Rights and Privileges 4-24
Module 4: Review Questions 4-26

Module 5 – Primary Index Access and Mechanics

Primary Keys and Primary Indexes 5-4
Distribution of Rows 5-6
Specifying a Primary Index 5-8
Primary Index Values 5-10
Accessing Via a Unique Primary Index 5-12
Accessing Via a Non-Unique Primary Index 5-14
Row Distribution Using a Unique Primary Index (UPI) – Case 1 5-16
Row Distribution Using a Non-Unique Primary Index (NUPI) – Case 2 5-18
Row Distribution Using a Highly Non-Unique Primary Index (NUPI) – Case 3 5-20
Which AMP has the Row? 5-22
Hashing Down to the AMPs 5-24
A Hashing Example 5-26
The Hash Map 5-28
Hash Maps for Different Systems 5-30
Identifying Rows 5-32
The Row ID 5-34
Storing Rows (1 of 2) 5-36
Locating a Row on an AMP Using a PI 5-40
Module 5: Review Questions 5-42

Module 6 – Secondary Indexes and Table Scans

Secondary Indexes 6-4
Choosing a Secondary Index 6-6
Unique Secondary Index (USI) Access 6-8
Non-Unique Secondary Index (NUSI) Access 6-10
Comparison of Primary and Secondary Indexes 6-12
Full Table Scans 6-14
Module 6: Review Questions 6-16

Module 7 – Teradata System Architecture

Teradata Database Releases 7-4
Teradata Database Architecture 7-6
Teradata Database – Multiple Nodes 7-8
MPP Systems 7-10
Example of 2+1 Node Teradata System 7-12
Teradata Cliques 7-14
BYNET 7-16
BYNET Communication Protocols 7-18
Vproc Inter-process Communication 7-20
Examples of Teradata Database Systems 7-22
6650 Cabinets 7-24
What makes Teradata's MPP Platforms Special? 7-26
Summary 7-28
Module 7: Review Exercises 7-30

Module 8 – Data Protection

Data Protection Features 8-4
Disk Arrays 8-6
RAID Technologies 8-8
RAID 1 – Mirroring 8-10
RAID 1 Summary 8-12
Cliques 8-14
Teradata Vproc Migration 8-16
Hot Standby Nodes (HSN) 8-18
Performance Degradation with Node Failure 8-20
Fallback 8-22
Fallback Clusters 8-24
Fallback and RAID Protection 8-26
Fallback and RAID 1 Example 8-28
Fallback vs. non-Fallback Tables Summary 8-38
Clusters and Cliques 8-40
Locks 8-42
Locking Modifier 8-44
Rules of Locking 8-46
Access Locks 8-48
Transient Journal 8-50
Recovery Journal for Down AMPs 8-52
Permanent Journal 8-54
Archiving and Recovering Data 8-56
Module 8: Review Questions 8-58

Module 9 – Introduction to Teradata Systems

Teradata Systems 9-4
SMP Architecture 9-6
Hyper-Threading and Multi-Core CPUs 9-8
Comparing Performance of Servers 9-10
Cabinet or Rack Pictures 9-12
Teradata 6650 Systems 9-14
Teradata 6650 Cabinets 9-16
Adding SSD to a 6650 (Future) 9-18
Teradata 6650 Configuration Examples 9-20
Teradata 6690 Systems 9-22
Teradata 6690 Cabinets 9-24
Teradata Extended Nodes 9-26
Making Sense of the Different Platforms 9-28
Linux Coexistence Combinations 9-30
Teradata Appliance Introduction 9-32
Teradata 2650/2690 Appliances 9-34
Teradata 2650/2690 Cabinets 9-36
Appliance Configuration Examples 9-38
What is the BYNET™? 9-40
BYNET 32 Switches 9-42
BYNET 64 Switches 9-44
BYNET Expansion Switches 9-46
BYNET Expansion to 1024 Nodes 9-46
Server Management with SWS 9-48
Node Naming Conventions 9-50
Summary 9-52
Module 9: Review Questions 9-54

Module 10 – How Teradata uses MPP Systems

Teradata and the Processing Node 10-4
Memory and the Teradata Database 10-6
SMP Memory – Summary 10-8
O.S. Managed Memory and FSG Cache 10-10
WAL – Write Ahead Logic 10-12
WAL Concepts 10-14
Linux Vproc Number Assignment 10-16
Disk Arrays from a O.S. Perspective 10-18
Logical Units and Partitions 10-20
Teradata and Disk Arrays 10-22
Teradata 6650 (2+1) Logical View 10-24
Teradata 6650 (3+1) Logical View 10-26
Example of 1.2 TB Vdisk (pre-TVS) 10-28
Teradata File System Concepts 10-30
Teradata Vdisk Size Limits 10-30
Teradata 13.10 Large Cylinder Support 10-32
Full Cylinder Read 10-34
Summary 10-36
Module 10: Review Questions 10-38

Module 11 – Teradata Virtual Storage

Teradata Virtual Storage 11-4
Teradata Virtual Storage Concepts 11-6
Teradata Virtual Storage Terminology 11-8
TVS Operational Modes 11-10
Expanding Data Storage Concepts 11-12
Multi-Temperature Concepts 11-14
Storage Performance vs. Data Temperature 11-16
Teradata with Hybrid Storage 11-18
What Goes Where? 11-20
Result of Data Migration 11-22
Teradata with Hybrid Storage 11-24
Multi-Temperature Data Example 11-26
Teradata 6690 Cabinets 11-28
Multi-Temperature Data Example 11-30
Summary 11-32
Module 11: Review Questions 11-34

Module 12 – Physical Database Design Overview

The Stages of Database Development 12-4
Example of Data Model – ER Diagram 12-6
Customer Service Logical Model 12-8
Relational Terms Review 12-10
Domains 12-12
Attributes 12-14
Entities and Relationships 12-16
Decomposable Data 12-18
Normal Forms 12-20
Normalization 12-22
Normalization Example 12-24
Denormalizations 12-34
Derived Data 12-36
Pre-Joins 12-38
System Assigned Primary Keys 12-40
Exercise 1: Choose Indexes 12-40
Tables Index Selection 12-42
Database Design Components 12-44
Extended Logical Data Model 12-46
Physical Data Model 12-48
The Principles of Index Selection 12-50
Transactions and Parallel Processing 12-52
Module 12: Review Questions 12-54

Module 13 – Data Distribution and Hashing

Data Distribution 13-4
Hashing 13-6
Hash Related Expressions 13-8
Hashing – Numeric Data Types 13-10
Multi-Column Hashing 13-12
Additional Hash Examples 13-16
Using Hash Functions to View Distribution 13-18
Primary Index Hash Mapping 13-20
Hash Maps 13-22
Primary Hash Map 13-24
Hash Maps for Different Systems 13-26
Fallback Hash Map 13-28
Reconfiguration 13-30
Row Retrieval via PI Value – Overview 13-32
Names and Object IDs 13-34
Table ID 13-36
Row ID 13-38
AMP File System – Locating a Row via PI 13-40
Teradata File System Overview 13-42
Master Index Format 13-44
Cylinder Index Format 13-46
Data Block Layout 13-48
Example of Locating a Row – Master Index 13-50
Example of Locating a Row – Cylinder Index 13-52
Example of Locating a Row – Data Block 13-54
Accessing the Row within the Data Block 13-56
AMP Read I/O Summary 13-58
Module 13: Review Questions 13-60

Module 14 – File System Writes

AMP Write I/O 14-4
New Row INSERT – Part 1 14-6
New Row INSERT – Part 2 14-8
New Row INSERT – Part 2 (cont.) 14-10
New Row INSERT – Part 3 14-12
New Row INSERT – Part 4 14-14
Blocking in Teradata 14-16
Block Size and Filling Cylinders 14-18
Variable Block Sizes 14-20
Block Splits (INSERT and UPDATE) 14-22
Space Fragmentation 14-24
Cylinder Full 14-26
Mini-Cylpack 14-28
Space Utilization 14-30
Merge Datablocks (13.10 Feature) 14-32
File System Write Summary 14-36
Module 14: Review Questions 14-38

Module 15 – SQL Assistant

SQL Assistant 15-4
Defining a Data Source 15-6
Defining a Data Source (cont.) 15-8
Defining a Data Source (cont.) 15-10
Connecting to a Data Source 15-12
Main Window 15-14
Database Explorer Tree 15-16
Creating and Executing a Query 15-18
Dragging Object Names to the Query Window 15-20
Query Options 15-22
Viewing Query Results 15-24
Formatting Answersets 15-26
Using Query Builder 15-28
History Window 15-30
General Options 15-32
Connecting to Multiple Data Sources 15-34
Additional Options 15-36
Importing/Exporting Large Object Files 15-38
Importing/Exporting Large Object Files 15-40
To Import a LOB into Teradata 15-40
Selecting from a Table with a LOB 15-42
Displaying a JPG within SQL Assistant 15-44
Teradata SQL Assistant Summary 15-46
Module 15: Review Questions 15-48
Lab Exercise 15-1 15-50

Module 16 – Analyze Primary Index Criteria

Primary Index Choice Criteria 16-4
Primary Index Defaults 16-6
CREATE TABLE – Indexing Rules 16-8
Order of Preference Exercise 16-10
Primary Index Characteristics 16-12
Multi-Column Primary Indexes 16-14
Primary Index Considerations 16-16
PKs and Duplicate Rows 16-18
NUPI Duplicate Row Check 16-20
Primary Index Demographics 16-22
Column Distribution Demographics for a PI Candidate 16-24
SQL to View Data Demographics 16-26
Example of Using Data Demographic SQL 16-28
TableSize View 16-32
SQL to View Data Distribution 16-34
E-R Diagram for Exercises 16-36
Exercise 2 – Sample 16-38
Exercise 2 – Choosing PI Candidates 16-40
What is a NoPI Table? 16-52
Reasons to Consider Using NoPI Tables 16-54
Creating a Table without a PI 16-56
How is a NoPI Table Implemented? 16-58
NoPI Random Generator 16-60
The Row ID for a NoPI Table 16-62
Multiple NoPI Tables at the AMP Level 16-66
Loading Data into a NoPI Table 16-68
NoPI Options 16-70
Summary 16-72
Module 16: Review Questions 16-74
Module 16: Review Questions (cont.) 16-76
Lab Exercise 16-1 16-78
Lab Exercise 16-2 16-82

Module 17 – Partitioned Primary Indexes

Partitioning a Table 17-4
How is Partitioning Implemented? 17-6
Logical Example of NPPI versus PPI 17-8
Primary Index Access (NPPI) 17-10
Primary Index Access (PPI) 17-12
Why Partition a Table? 17-14
Advantages/Disadvantages of Partitioning 17-16
PPI Considerations 17-18
How to Define a PPI 17-20
Partitioning with CASE_N and RANGE_N 17-22
Partitioning with RANGE_N – Example 1 17-24
Partitioning with RANGE_N – Example 2 17-34
Partitioning – Example 3 17-36
Special Partitions with CASE_N and RANGE_N 17-38
Special Partition Examples 17-40
Partitioning with CASE_N – Example 4 17-42
SQL Use of PARTITION Key Word 17-44
SQL Use of CASE_N 17-46
Using ALTER TABLE with PPI Tables 17-48
ALTER TABLE – Example 5 17-50
ALTER TABLE – Example 5 (cont.) 17-52
ALTER TABLE TO CURRENT 17-54
ALTER TABLE TO CURRENT – Example 7 17-56
PPI Enhancements 17-58
Multi-level PPI Concepts 17-60
Multi-level PPI Concepts (cont.) 17-62
Multi-level Partitioning – Example 8 17-64
Multi-level Partitioning – Example 8 (cont.) 17-66
How is the MLPPI Partition # Calculated? 17-68
Character PPI 17-70
Character PPI – Example 9 17-72
Summary 17-74
Module 17: Review Questions 17-76
Lab Exercise 17-1 17-80

Module 18 – Teradata Columnar

Teradata Columnar 18-4
Teradata Columnar Benefits 18-6
No Primary Index Table DDL 18-8
The No Primary Index Table 18-10
Column Partition Table DDL (without Auto-Compression) 18-12
Column Partition Container (No Automatic Compression) 18-14
The Column Partition Table (without Auto-Compression) 18-16
CP Table Query #1 (without Auto-Compression) 18-18
CP Table Query #1 (without Auto-Compression) 18-20
Column Partition Table DDL (with Auto-Compression) 18-22
Auto-Compression for CP Tables 18-24
Auto-Compression Techniques for CP Tables 18-26
User-Defined Compression Techniques 18-28
Column Partition Container (Automatic Compression) 18-30
The Column Partition Table (with Auto-Compression) 18-32
CP Table Query #2 (with Auto-Compression) 18-34
CP Table with Row Partitioning DDL 18-36
The Column Partition Table (with Row Partitioning) 18-38
CP Table with Multi-Column Container DDL 18-40
The CP Table with Multi-Column Container 18-42
CP Table Hybrid Row & Column Store DDL 18-44
The CP Table (with Hybrid Row & Column Store) 18-46
Populating a CP Table 18-48
DELETE Considerations 18-50
UPDATE Considerations 18-52
CP Table Restrictions 18-54
Summary 18-56
Module 18: Review Questions 18-58
Lab Exercise 18-1 18-60

Module 19 – Secondary Index Usage

Secondary Indexes 19-4
Defining Secondary Indexes 19-6
Secondary Index Subtables 19-8
USI Subtable General Row Layout 19-10
USI Hash Mapping 19-12
NUSI Subtable General Row Layout 19-14
NUSI Hash Mapping 19-16
Table Access – A Complete Example 19-18
Secondary Index Considerations 19-20
Single NUSI Access (Between, Less Than, or Greater Than) 19-22
Dual NUSI Access 19-24
NUSI Bit Mapping 19-26
Value-Ordered NUSIs 19-28
Value-Ordered NUSIs (cont.) 19-30
Covering Indexes 19-32
Covering Indexes (cont.) 19-34
NUSI vs. Full Table Scan (FTS) 19-36
Full Table Scans – Sync Scans 19-38
Module 19: Review Questions 19-40

Module 20 – Analyze Secondary Index Criteria

Accessing Rows 20-4
Row Selection 20-6
Secondary Index Considerations 20-8
Composite Secondary Indexes 20-10
Secondary Index Candidate Guidelines 20-12
Exercise 3 – Sample 20-14
Exercise 3 – Choosing SI Candidates 20-16
Change Rating 20-28
Value and Range Access 20-30
Exercise 4 – Sample 20-32
Exercise 4 – Eliminating Index Candidates 20-34
Module 20: Review Questions 20-46

Module 21 – Access Considerations and Constraints

Access Method Comparison 21-4
Optimizer Access Scenarios 21-6
Data Conversions 21-8
Storing Numeric Data 21-10
Data Conversion Example 21-12
Matching Data Types 21-14
Counting I/O Operations 21-16
Transient Journal I/O 21-18
INSERT and DELETE Operations 21-20
UPDATE Operations 21-22
Primary Index Value UPDATE 21-24
Additional I/O 21-26
Table Level Attributes 21-28
Column Level Constraints 21-30
Table Level Constraints 21-32
CHECK Constraints 21-32
Example: Department Table with Constraints 21-34
Example (13.0) – SHOW Department Table 21-36
Example (13.10) – SHOW Department Table 21-38
Altering Table Constraints 21-40
Identity Column – Overview 21-42
Identity Column – Implementation 21-44
Identity Column – Example 1 21-46
Identity Column – Example 2 21-48
Identity Column – Considerations 21-50
Module 21: Review Questions 21-52

Module 22 – Referential Integrity

Referential Integrity 22-4
Parent-Child Relationships 22-6
Three Types of Referential Constraints 22-8
Reference Index Subtables 22-10
Reference Index Example – Add REFERENCES 22-12
Fixing Referential Integrity Problems 22-20
Batch Referential Integrity 22-22
Soft Referential Integrity 22-24
Referential Integrity Example 22-26
Referential Integrity Example (cont.) 22-28
Join Optimization with RI 22-30
Summary 22-34
Module 22: Review Questions 22-36

Module 23 – Sizing

General Row Format 23-4
Presence Bits 23-6
NULL and COMPRESS 23-8
Multi-Value Compression 23-10
Implementing Multi-Value Compression 23-12
ALTER TABLE and Compression 23-14
Algorithmic Compression Example 23-18
Detailed Row Format 23-20
Multi-Value Compression vs. VARCHAR 23-22
Teradata Compression Comparison 23-24
Sizing a Data Row Considerations 23-26
Teradata Data Types 23-28
INTEGER Data Types 23-30
DECIMAL and FLOAT Data Types 23-32
NUMBER Data Type (14.0) 23-34
DATE and TIME Data Types 23-36
CHARACTER Data Types 23-38
Character Sets 23-40
BYTE Data Types 23-42
Large Object Data Types 23-44
Variable Column Offsets 23-46
Row Size Calculation Form 23-48
Example: Sizing a Row 23-50
Row Size Exercise 23-54
Sizing Tables and Indexes 23-56
Table Headers 23-58
Sizing a Data Table 23-60
Table Sizing Exercise 23-62
Estimating the Size of a USI Subtable 23-64
Estimating the Size of a NUSI Subtable 23-66
Estimating the Size of a Reference Index Subtable 23-68
Index Sizing Exercise 23-70
Other Sizing Techniques 23-72
Empirical Sizing 23-74
Collect Demographics Command 23-76
Collect Demographics Example 23-78
Spool Space 23-80
Release of Spool 23-82
System Sizing Exercise 23-84
Sizing Summary 23-86
Module 23: Review Questions 23-88
Lab Exercise 23-1 23-90
Lab Exercise 23-2 23-92
Lab Exercise 23-3 (optional) 23-96

Module 24 – Parser

Internal, Channel and LAN Parcels 24-4
Request Parcel 24-6
The Data Parcel 24-8
SQL Parser Overview 24-10
Software Cache 24-12
Request-To-Steps Cache 24-14
Dictionary Cache 24-20
Syntaxer 24-22
Resolver 24-24
Security 24-26
Optimizer 24-28
Generator 24-30
Apply 24-32
Dispatcher 24-32
SQL Parser Review 24-34
Parser Summary 24-36
Module 24: Review Questions 24-38

Module 25 – Optimizer and Collecting Statistics

Teradata Optimizer....25-4
Optimizer – Cost Based vs. Rule Based....25-6
Optimizer Statistics....25-8
Optimizer's Search for Statistics....25-10
Optimizer – Random AMP Samples....25-12
Random AMP Sampling – How it Works....25-14
Example of an Optimizer Estimate without Collected Statistics....25-16
Statistics....25-18
Statistics Data – What is Collected?....25-20
Statistics Data – What is Collected? (cont.)....25-22
Statistics Data – What is Collected? (cont.)....25-24
Statistics Example....25-26
Statistics Example (cont.)....25-28
COLLECT STATISTICS Command....25-30
Collecting Statistics....25-32
Refresh or Re-Collect Statistics....25-34
COLLECT STATISTICS Command....25-36
COLLECT STATISTICS on a Data Sample....25-38
Collecting Statistics (14.0 Examples)....25-40
Viewing Statistics....25-42
Optimizer's use of Statistics with Uneven NUSI....25-44
Collecting Statistics on PARTITION....25-46
Copying STATISTICS....25-48
Statistics Extrapolation....25-50
Teradata 13.0 Enhancements....25-52
Teradata 14.0 Enhancements....25-54
Teradata Statistics Wizard....25-56
Teradata Statistics Wizard – Main Window....25-58
Teradata Statistics Wizard – Interval Statistics....25-60
Collect, Re-Collect, or Drop Statistics....25-62
Recommendation Options....25-64
Statistics Recommendations....25-66
Statistics Summary....25-68
Module 25: Review Questions....25-70

Module 26 – The EXPLAIN Facility

The EXPLAIN Facility 26-4
EXPLAIN Facility Output 26-6
Example 1 – EXPLAIN of a Simple SELECT 26-8
Example 2 – EXPLAIN of a SELECT (FTS) 26-10
EXPLAIN Terminology 26-12
Pseudo Table Locks 26-16
Understanding Row and Time Estimates (Part 1) 26-18
Parallel Steps 26-22
Example 3 – EXPLAIN with Parallel Steps 26-24
Example 4 – EXPLAIN of a SELECT (BMSMS) 26-26
Example 5 – EXPLAIN of Create Table 26-28
EXPLAINing Macros 26-30
EXPLAIN Terminology for PPI Tables 26-32
Example 6 – Partition Elimination with a PPI Table 26-34
Example 7 – Primary Index Access of PPI Table 26-36
Example 8 – Dynamic Partition Elimination 26-38
Example 9 – CURRENT_DATE Improvements 26-40
EXPLAIN Summary 26-42
Module 26: Review Questions 26-44

Module 27 – Visual Explain

Teradata Visual Explain 27-4
Visual Explain – Connect to Teradata 27-6
Setting up the Environment 27-8
Placing Plans into QCD 27-10
Creating a Plan using "Execute SQL" Option 27-12
Open Execution Plans 27-14
Open Execution Plans (cont.) 27-16
Open Execution Plans (cont.) 27-18
Visual Explain of a Merge Join 27-20
Visual Explain Options 27-22
Visual Explain – Comparing Multiple Plans 27-24
Visual Explain – Example of Comparing 2 Plans 27-26
Granting Access Rights on a QCD 27-28
Visual Explain Summary 27-30
Module 27: Review Questions 27-32
Lab Exercise 27-1 27-34
Lab Exercise 27-2 27-36

Module 28 – Join Processing Analysis

SELECT Statement ANSI Join Syntax 28-4
Example of ANSI and Teradata JOIN Syntax 28-6
LEFT Outer Join Example 28-8
RIGHT Outer Join Example 28-10
FULL Outer Join Example 28-12
Join Processing 28-14
Optimizer Minimizes Spool Usage 28-16
Row Selection 28-18
Join Redistribution 28-20
Duplicating a Table in Spool 28-24
General Join Distribution Strategies 28-26
Merge Join 28-28
Merge Join Strategy 28-30
Merge Join – Matching Primary Indexes 28-32
Merge Join – Row Redistribution 28-34
Merge Join – Duplicate the Smaller Table 28-36
Nested Joins 28-38
Product Join 28-40
Cartesian Product 28-42
Product Join – Duplicate the Smaller Table 28-44
Hash Join 28-46
Exclusion Joins 28-48
Exclusion Join Example 28-50
Inclusion Joins 28-52
n-Table Joins 28-54
Join Considerations with PPI 28-56
NPPI to PPI Join – Few Partitions 28-58
NPPI to PPI Join – Many Partitions 28-60
NPPI to PPI Join – Sliding Window 28-62
NPPI to PPI Join – Sliding Window (cont.) 28-64
NPPI to PPI Join – Hash Ordered Spool File Join 28-66
PPI to PPI Join – Rowkey-Based Join 28-68
PPI to PPI Join – Unmatched Partitions 28-70
Additional Join Options with PPI 28-72
Join Processing Summary 28-74
Module 28: Review Questions 28-76

Module 29 – Explains of Joins and Index Choices

Join Diagramming 29-4
Nested Join 29-6
Merge Join (Matching Primary Indexes) 29-8
Hash Join 29-10
Visual Explain (Hash Join) 29-12
Merge Join (Joining a Table to Itself) 29-14
Three-Table Join 29-16
Product Join 29-20
A “UNION” Solution 29-24
Cartesian Product Join 29-28
Exclusion Join 29-30
Example of PPI to PPI Join 29-32
Join ACCESS 29-34
Exercise 5 – Sample 29-36
Exercise 5 – Making Final Index Choices 29-38
Teradata Index Wizard 29-50
Teradata Index Wizard – Main Window 29-52
Teradata Index Wizard – Index Analysis 29-54
Teradata Index Wizard – Index Analysis Results 29-56
Teradata Index Wizard – Partition Analysis 29-58
Teradata Index Wizard – Partition Analysis Results 29-60
Teradata Index Wizard – Reports 29-62
Teradata Index Wizard – Validation 29-64
Teradata Index Wizard – Validation (View Graph) 29-66
Teradata Index Wizard – Creation 29-68
Summary – Index Choice Guidelines 29-70
Module 29: Review Questions 29-72

Module 30 – Additional Index Choices

Additional Index Choices....30-4
Join Indexes....30-6
Options for Join Indexes....30-8
Join Index Considerations....30-10
Join Index Example – Customer and Order Tables....30-12
Compressed Multi-Table Join Index....30-14
Non-Compressed Multi-Table Join Index....30-16
Compressed and Non-Compressed Join Indexes....30-18
Example 1 – Does a Join Index Help?....30-20
Example 2 – Does a Join Index Help?....30-22
Example 3 – Partitioning a Join Index....30-24
Join Index – Single Table....30-26
Join Index – Single Table (cont.)....30-28
Creating a Join Index – Single Table....30-30
Example 4 – Does the Join Index Help?....30-32
Why use Aggregate Join Indexes?....30-34
Aggregate Join Index Properties....30-36
Aggregation without an Aggregate Index....30-38
Creating an Aggregate Join Index....30-40
Aggregation with an Aggregate Index....30-42
Sparse Join Indexes....30-44
Creating a Sparse Join Index....30-46
Creating a Sparse Join Index on a Partitioned Table....30-48
ALTERing a Join Index to CURRENT....30-50
Partitioning a Sparse Join Index....30-52
Global (Join) Indexes....30-54
Global Index – Multiple Tables....30-56
Global Index as a “Hashed NUSI”....30-58
Creating a Global Index (“Hashed NUSI”)....30-60
Example: Using a Global Index as a Hashed NUSI....30-62
Repeating Row Ids in Global Index....30-64
Hash Indexes....30-66
Hash Index – Example....30-68
Hash Index – Example (cont.)....30-70
Summary....30-72
Module 30: Review Questions....30-74
Lab Exercise 30-1....30-76

Module 31 – Miscellaneous SQL Features

Teradata SQL 31-4
Teradata SQL and ANSI Differences 31-6
SQL Session Modes 31-8
Transaction Modes – Teradata 31-10
Transaction Modes – ANSI 31-12
Duplicate Rows 31-14
Transaction Mode Examples 31-16
Multi-Statement Requests 31-18
CASE Sensitivity Issues 31-20
Setting the SQL Flagger 31-22
SQLFLAG Example 31-24
HELP SESSION Command 31-26
Why a System Calendar? 31-28
Calendar View Layout 31-30
One Row in the Calendar 31-32
Using the Calendar 31-34
Temporary Table Choices 31-36
Derived Tables Revisited 31-38
Volatile Tables 31-40
Volatile Table Restrictions 31-42
Global Temporary Tables 31-44
Creating Global Temporary Tables 31-46
Teradata 12.0 – Major Features 31-48
Teradata 13.0 – Major Features 31-50
Teradata 13.10 – Major Features 31-52
Teradata 14.0 – Major Features 31-54
Teradata Limits (Different Releases) 31-56
Module 31: Review Questions 31-58

Module 32 – Introduction to Application Utilities

Application Utilities 32-4
Application Utilities Environments 32-6
Application Development 32-8
Transferring Large Amounts of Data 32-10
INSERT/SELECT: The Fast Path 32-12
Multi-Statement Insert/Select Example 32-14
DELETE (ALL): The Fast Path 32-16
AXSMOD, INMOD, and OUTMOD Routines 32-18
Teradata Parallel Transporter 32-20
Teradata Parallel Transporter Operators 32-22
Referential Integrity and Load Utility Issues 32-24
Maximum Number of Load Jobs 32-26
Maximum Number of Load Jobs (cont.) 32-28
Application Utility Checklist 32-30
Application Utility Summary 32-32
Module 32: Review Questions 32-34

Module 33 – BTEQ

BTEQ 33-4
Using BTEQ Conditional Logic 33-6
BTEQ Error Handling 33-8
BTEQ EXPORT – Example 1 33-10
4 Types of BTEQ .EXPORT 33-12
BTEQ Data Modes 33-14
BTEQ EXPORT – Example 2 33-16
BTEQ EXPORT – Example 3 33-18
Indicator Variables 33-20
Determining the Logical Record Length with Fixed Length Columns 33-22
Determining the Logical Record Length with Variable Length Columns 33-24
Determining the Logical Record Length with .EXPORT INDICDATA 33-26
.IMPORT (for Network-Attached Systems) 33-28
.IMPORT (for Channel-Attached Systems) 33-30
.PACK 33-32
.REPEAT 33-32
BTEQ IMPORT – Example 1 33-34
BTEQ IMPORT – Example 2 33-36
BTEQ IMPORT – Example 3 33-38
Multiple Sessions 33-40
.SET SESSIONS 33-42
Parallel Processing Using Multiple Sessions to Access Individual Rows 33-44
When Do Multiple Sessions Make Sense? 33-46
Application Utility Checklist 33-48
Module 33: Review Questions 33-50
Lab Exercise 33-1 33-52
Lab Exercise 33-2 33-56

Module 34 – FastLoad

FastLoad 34-4
FastLoad Phase 1 34-6
FastLoad Phase 2 34-8
A Sample FastLoad Script 34-10
Converting the Data 34-12
Data Conversion Chart 34-14
NULLIF 34-16
FastLoading Zoned Decimals and Time Stamps 34-18
FastLoad BEGIN LOADING Statement 34-20
BEGIN LOADING Statement 34-20
FastLoad Error Tables 34-22
Error Recovery 34-24
CHECKPOINT Option 34-26
END LOADING Statement 34-28
RECORD Statement 34-30
INSERT Statement 34-32
Staged Loading of Multiple Data Files 34-34
FastLoad Fails to Complete 34-36
Restarting FastLoad (Output) 34-38
Restarting FastLoad – Summary 34-40
Additional FastLoad Commands 34-42
FastLoad with Additional Options 34-44
Invoking FastLoad 34-46
INMOD 34-48
Application Utility Checklist 34-50
Summary 34-52
Module 34: Review Questions 34-54
Lab Exercise 34-1 34-56
Lab Exercise 34-2 34-58

Module 35 – The Support Environment

Support Environment 35-4
Setting Up the Support Environment 35-6
Invoking Utilities 35-8
Support Environment Commands 35-10
Initializing the Log Table 35-12
Initialization and Wrap Up Commands 35-14
User-defined and System Variables 35-16
.ACCEPT – Environment or File Variable 35-18
.DISPLAY and .ROUTE Commands 35-20
Example: Using Variables in a Script 35-22
Working with Control Logic 35-24
Support Environment Example – Input 35-26
Support Environment Example – Output 35-28
Teradata SQL Support 35-30
Script – Example Input 35-32
Script – Example Output 35-34
Summary 35-36
Module 35: Review Questions 35-38
Lab Exercise 35-1 (optional) 35-40

Module 36 – FastExport

FastExport 36-4
.BEGIN and .END EXPORT 36-6
.EXPORT 36-8
A FastExport Script 36-10
The SELECT Request 36-12
Impact of Requesting Sorted Output 36-14
The SORT Procedure 36-16
Multiple Exports in one FastExport Job 36-18
Invoking FastExport 36-20
FastExport and Variable Input 36-22
A FastExport Script with ACCEPT 36-24
A FastExport Script with LAYOUT 36-26
Application Utility Checklist 36-28
Summary 36-30
Module 36: Review Questions 36-32
Lab Exercise 36-1 36-34
Lab Exercise 36-2 36-36

Module 37 – MultiLoad Part 1

What is MultiLoad? 37-4
MultiLoad Limitations 37-6
How MultiLoad Works 37-8
Advantages of MultiLoad 37-10
Basic MultiLoad Statements 37-12
Sample MultiLoad IMPORT Task 37-14
IMPORT Task 37-16
5 Phases of IMPORT Task 37-18
Phase 1: Preliminary 37-20
Phase 2: DML Transaction 37-22
Phase 3: Acquisition 37-24
Phase 3: Acquisition – a Closer Look 37-26
Phase 4: Application 37-28
Phase 4: Application – a Closer Look 37-30
Phase 5: Cleanup 37-32
Sample MultiLoad DELETE Tasks 37-34
DELETE Task Differences from IMPORT Task 37-36
A Closer Look at DELETE Task Application Phase 37-38
MultiLoad Locks 37-40
Restarting MultiLoad 37-42
RELEASE MLOAD Statement 37-44
Invoking MultiLoad 37-46
Application Utility Checklist 37-48
Summary 37-50
Module 37: Review Questions 37-52
Lab Exercise 37-1 37-54

Module 38 – MultiLoad Part 2

New Accounts Application – Description 38-4
.BEGIN IMPORT Task Commands 38-12
Work Tables 38-14
Error Tables 38-16
ERRLIMIT 38-18
CHECKPOINT 38-20
More .BEGIN Parameters 38-22
More .BEGIN Parameters: AMPCHECK 38-24
DELETE Task Commands 38-26
.LAYOUT and .TABLE 38-28
.LAYOUT Parameters — CONTINUEIF 38-30
.LAYOUT Parameters — INDICATORS 38-32
.FIELD and .FILLER 38-34
.LAYOUT Command — Examples 38-36
Redefining the Input – Example 38-38
The .DML Command Options 38-40
MultiLoad Statistics 38-44
INMOD 38-46
Summary 38-48
Module 38: Review Questions 38-50
Lab Exercise 38-1 38-52
Lab Exercise 38-2 38-54

Module 39 – TPump

TPump 39-4
TPump Limitations 39-6
.BEGIN LOAD Statement 39-8
TPump Specific Parameters 39-10
.BEGIN LOAD – PACK and RATE 39-12
.BEGIN LOAD – SERIALIZE OFF 39-14
.BEGIN LOAD – SERIALIZE ON 39-16
.BEGIN LOAD – ROBUST ON 39-18
Sample TPump Script (1 of 2) 39-20
Sample TPump Script (2 of 2) 39-22
TPump Compared with MultiLoad 39-24
Additional TPump Statements 39-26
Invoking TPump 39-28
TPump Statistics 39-30
TPump Monitor 39-32
INMOD 39-34
Application Utility Checklist 39-36
Summary 39-38
Module 38: Review Questions 39-40
Lab Exercise 39-1 39-42

Module 40 – Choosing a Utility

Solution 1: Update or Delete vs. Insert/Select 40-4
Solution 2: SQL Update vs. MultiLoad or TPump 40-6
Solution 3: SQL Update vs. FastLoad 40-8
Utility Considerations 40-10
Various Ways of Performing an Update 40-12
Choosing a Utility Exercise 40-14

Module 41 – Database Administration and Building the Database Environment

Database Administration 41-4
Initial Teradata Database 41-6
Administrative User 41-8
Owners, Parents and Children 41-10
Creating New Users and Databases 41-12
Transfer of Ownership 41-14
DELETE/DROP Statements 41-16
Teradata Administrator – New System 41-18
Teradata Administrator – Hierarchy 41-20
Summary 41-22
Module 41: Review Questions 41-24

Module 42 – The Data Dictionary

Data Dictionary / Directory....42-4
Fallback Protected Data Dictionary Tables....42-6
Non-Hashed Data Dictionary Tables....42-8
Updating Data Dictionary Tables....42-10
Supplied Data Dictionary Views....42-12
Restricted Views....42-14
Suffix Options with Views....42-16
Selecting Information about Created Objects....42-18
Children View....42-20
Databases View....42-22
Users View....42-24
Tables View....42-26
Columns View....42-28
Indices View....42-30
IndexConstraints View....42-34
ShowTblChecks View....42-36
ShowColChecks View....42-38
Triggers View....42-40
AllTempTables View....42-42
Referential Integrity Views....42-44
Using the DBC.Tables View....42-46
Referential Integrity States....42-48
DBC.All_RI_Children View....42-50
DBC.Databases2 View....42-52
Time Stamps in Data Dictionary....42-54
Teradata Administrator – List Columns of a View....42-56
Teradata Administrator – Object Options....42-58
Summary....42-60
Module 42: Review Questions....42-62
Lab Exercise 42-1....42-64
Lab Exercise 42-2 (optional)....42-70

Module 43 – Space Allocation and Usage

Permanent Space Terminology 43-4
Spool Space Terminology 43-6
Temporary Space Terminology 43-8
Resetting Peak Values 43-10
Assigning Perm and Spool Limits 43-12
Giving One User to Another 43-14
Teradata Administrator – Move Space 43-16
Reserving Space for Spool 43-18
Views for Space Allocation Reporting 43-20
DiskSpace View 43-22
TableSize View 43-24
AllSpace View 43-26
DataBaseSpace Table 43-28
Different Views — Different Results 43-30
Additional Utilities to View Space Utilization 43-32
Teradata Administrator – Database Menu Options 43-34
Teradata Administrator – Object Menu Options 43-36
Transient Journal Space 43-38
Ferret Utility 43-40
Ferret SHOWSPACE Command 43-42
Ferret SHOWBLOCKS 43-44
Module 43: Review Questions 43-48

Module 44 – Users, Accounts, and Accounting

Creating New Users & Databases 44-4
CREATE DATABASE Statement 44-6
CREATE USER Statement 44-8
CREATE USER and the Data Dictionary 44-10
CREATE USER and the Data Dictionary (cont.) 44-12
MODIFY USER Statement 44-14
Teradata Administrator – Tools Menu > Create Options 44-16
Creating and Using Account IDs 44-18
Dynamically Changing an Account ID 44-20
Account Priorities 44-22
Account String Expansion 44-24
ASE Accounting Example 44-26
System Accounting Views 44-28
AccountInfo View 44-30
AMPUsage View 44-32
AMPUsage View – Example 44-34
Users, Accounts & Accounting Summary 44-36
Module 44: Review Questions 44-38
Lab Exercise 44-1 44-40

Module 45 – Profiles

Profiles 45-4
Example of Simplifying User Management 45-6
Implementing Profiles 45-8
Impact of Profiles on Users 45-10
CREATE/MODIFY PROFILE Statement 45-12
Password Attributes (CREATE/MODIFY PROFILE) 45-14
Teradata Password Control 45-16
Teradata Password Control (cont.) 45-18
Teradata Password Control Options 45-20
CREATE PROFILE Example 45-22
Teradata Administrator CREATE PROFILE Example 45-24
CREATE PROFILE Example (cont.) 45-26
DROP PROFILE Statement 45-28
ProfileInfo View 45-30
Miscellaneous SQL Functions 45-32
Summary 45-34
Module 45: Review Questions 45-36
Lab Exercise 45-1 45-38

Module 46 – Access Rights

Privileges/Access Rights 46-4
Access Rights Mechanisms 46-6
CREATE TABLE – Automatic Rights 46-8
CREATE USER – Automatic Rights 46-10
Implicit, Automatic, and Explicit Rights 46-12
GRANT Command 46-14
Granting Rights at Database Level 46-16
GRANT Rights at the Table or Column Level 46-18
REVOKE Command 46-20
Revoking Non-Existent Rights 46-22
Removing a Level in the Hierarchy 46-24
Inheriting Access Rights 46-26
The GIVE Statement and Access Rights 46-28
Access Rights and Views 46-30
Access Rights and Nested Views 46-32
System Views for Access Rights 46-34
AllRights and UserRights Views 46-36
UserGrantedRights View 46-38
Teradata Administrator – Grant/Revoke Rights 46-40
Teradata Administrator – Rights on DB/User 46-42
Access Rights Summary 46-44
Module 46: Review Questions 46-46

Module 47 – Roles

What are Roles? 47-4
Advantages of Roles 47-4
Access Rights without Roles 47-6
Access Rights Issues (prior to Roles) 47-6
Access Rights Using a Role 47-8
Implementing Roles 47-10
Current or Active Roles 47-12
Nesting of Roles 47-14
Example of Using “Nested Roles” 47-16
Access Rights Validation and Roles 47-18
SQL Statements to Support Roles 47-20
GRANT Command (SQL Form) 47-22
REVOKE Command (SQL Form) 47-24
GRANT and REVOKE Commands (Role Form) 47-26
System Hierarchy (used in following examples) 47-28
Example of Using Roles 47-30
Example of Using Roles (cont.) 47-32
Example of Using Roles (cont.) 47-34
Roles for Directory-Based Users 47-36
Roles for Proxy Users 47-38
RoleInfo View 47-40
RoleMembers View 47-42
DBC.AccessRights and “Rights” Views 47-44
AllRoleRights and UserRoleRights Views 47-46
Steps to Implementing Roles 47-48
Summary 47-50
Module 47: Review Questions 47-52
Lab Exercise 47-1 47-54
Lab Exercise 47-2 47-58

Module 48 – Access Control

System Access Control Levels....48-4
Teradata Access Control Mechanisms....48-6
Teradata Password Encryption....48-8
Password Security Features....48-10
Teradata Connectivity....48-12
Host Logon Processing....48-14
Objects used in Host Logon Processing....48-16
GRANT/REVOKE LOGON Statements....48-18
GRANT/REVOKE LOGON Example....48-20
Session Related Views....48-22
LogonRules View....48-24
LogOnOff View....48-26
SessionInfo View....48-28
Additional Utilities to View Sessions....48-30
Viewpoint – Query Monitor....48-32
Teradata Manager Sessions....48-34
Remote Console – Viewpoint....48-36
Structure the System....48-38
A Recommended Access Rights Structure....48-40
A Recommended Structure Using Roles....48-42
A Recommended System Hierarchy....48-44
System Access Controls Summary....48-46
Module 48: Review Questions....48-48

Module 49 – Access and Query Logging

Access and Query Logging 49-4
Access Logging 49-6
Objects used in Access Logging 49-8
BEGIN LOGGING Statement 49-10
END LOGGING Statement 49-12
Setting up Access Logging 49-14
Access Log Views 49-16
AccLogRules View 49-18
BEGIN LOGGING – Example 49-20
AccessLog View 49-22
AccessLog View – Example 49-24
END LOGGING – Example 49-26
Teradata Administrator – Tools Menu > Access Logging 49-28
Query Logging (DBQL) Concepts 49-30
Objects used in Defining Rules for DBQL 49-32
Objects used in DBQL (cont.) 49-34
BEGIN QUERY LOGGING Statement 49-36
BEGIN QUERY LOGGING WITH … (cont.) 49-38
BEGIN QUERY LOGGING LIMIT … (cont.) 49-40
BEGIN QUERY LOGGING Examples 49-42
BEGIN QUERY LOGGING Examples (cont.) 49-44
BEGIN QUERY LOGGING Examples (cont.) 49-46
END QUERY LOGGING Statement 49-48
DBQLRules View 49-50
QryLog View – Example 49-52
QryLogSummary View – Example 49-54
Teradata Administrator – Tools Menu > Query Logging 49-56
Access and Query Logging Summary 49-58
Module 49: Review Questions 49-60
Lab Exercise 49-1 49-62
Lab Exercise 49-2 49-66

Module 50 – Priority Scheduler

Levels of Workload Management 50-4
Priority Scheduler Facility 50-6
Priority Scheduler Architecture 50-8
Priority Scheduler Architecture with TDWM Workloads 50-10
Priority Scheduler Concepts 50-12
Resource Partitions and Performance Groups 50-14
Relative Weights 50-16
Performance Periods and Milestones 50-18
CPU Usage Limits with Priority Scheduler 50-20
Use of Performance Groups 50-22
Getting Started with Priority Scheduler 50-24
Schmon Utility 50-26
Schmon Example 50-28
Priority Scheduler Administrator 50-34
Summary 50-36
Module 50: Review Questions 50-38

Module 51 – Workload Management

Levels of Workload Management 51-4
What is TASM? 51-6
TASM Capabilities 51-8
Query Management Architecture 51-10
Query Management Architecture (cont.) 51-12
TDWM Example 51-14
Workload Designer Example 51-16
Filters and Throttles for Query Management 51-18
Object Access and Query Resource Filters 51-20
Object and Load Utility Throttles 51-22
Workload Definitions 51-24
Example of Using Workloads 51-26
Creating Workloads 51-28
WD – Classification Criteria 51-30
Specify Exception Criteria 51-32
Example – Exception Handling 51-34
Teradata Workload Analyzer 51-38
Summary 51-40
Module 51: Review Questions 51-42

Module 52 – Teradata Viewpoint

What is Teradata Viewpoint? 52-4
Viewpoint Portal and Portlets 52-6
Logging onto Viewpoint 52-8
Example of Initial Session and Portlets 52-10
Viewpoint Portlet Controls 52-12
Viewpoint Rewind 52-14
Query Monitor 52-16
Viewpoint Query Monitor Detail View 52-18
My Queries 52-20
Viewpoint – Remote Console 52-22
Viewpoint Alert Viewer 52-24
Viewpoint SQL Scratchpad 52-26
Viewpoint SQL Scratchpad Object Browser 52-28
Viewpoint SQL Scratchpad Saved/History 52-30
Summary 52-32
Module 52: Review Questions 52-34

Module 53 – Performance Monitoring – ResUsage

Performance Monitoring Tools 53-4
Why Collect Performance Data? 53-6
Resource Usage Data 53-8
Collection Costs 53-8
Filling the ResUsage Tables 53-10
Specifying ResUsage Tables and Logging Rates 53-12
Resource Usage Tables 53-14
Resource Usage Views 53-16
Resource Usage Macros 53-18
Example Output from DBC.ResNode Macro 53-20
PM/API and Viewpoint 53-22
Teradata System Emulation Tool (Teradata SET) 53-24
Performance Monitoring Summary 53-26
Module 53: Review Questions 53-28

Module 54 – System Restarts

Types of Restarts 54-4
Scheduled Restarts 54-6
Restart Teradata from DB Window 54-8
Restart using the “tpareset” Command 54-10
PDE States 54-12
Unscheduled Restarts 54-14
Unscheduled Restarts (cont.) 54-16
Unscheduled Restarts (cont.) 54-18
TPA Reset – Crashdumps 54-20
Allocating Crashdumps Space 54-22
TPA Dump Maintenance 54-24
Linux Operating (Panic) Dumps 54-24
Module 54: Review Questions 54-26

Module 55 – System and Maintenance Utilities

Starting Teradata System Utilities 55-4
SMP and Database Window Utilities 55-6
Teradata Database Window 55-8
DBW Supervisor Window 55-10
DBS Control Utility 55-12
DBS Control Record – General Fields 55-14
DBS Control Record – General Fields 55-16
DBS Control Record – File System Fields 55-18
DBS Control Record – Performance Fields 55-20
Modifying DBS Control Parameters 55-22
Ferret – Defragment and Packdisk 55-24
Checking Data Integrity 55-26
Ferret – Scandisk Utility 55-28
Checktable Utility 55-30
Checktable – Levels of Checking 55-32
Checktable – Example 55-34
Table Rebuild Utility 55-36
Recovery Manager Utility 55-38
Recovery Manager Commands 55-40
Rcvmanager – List Status 55-42
Rcvmanager – List Locks 55-44
Rcvmanager – List Status (2nd Example) 55-46
Rcvmanager – List Rollback Tables 55-48
Rcvmanager – Cancel Rollback on Table 55-50
Showlocks Utility 55-52
Orphan or Phantom Spool Issues 55-54
Update Space Utility 55-56
Vprocmanager 55-58
Summary 55-60
Module 55: Review Questions 55-62

Module 56 – Permanent Journals

Automatic Data Protection Mechanisms (Review) 56-4
Permanent Journals 56-6
Location of Change Images 56-8
Assigning Tables to a Permanent Journal 56-10
Creating a Permanent Journal 56-12
Assigning a Permanent Journal 56-14
Before-Image Journals 56-16
After-Image Journals 56-18
Journal Subtables 56-20
Permanent Journal Statements 56-22
Recovery with Permanent Journals 56-24
Journals[x] View 56-26
Summary 56-28
Module 56: Review Questions 56-30

Module 57 – A Tale of Three Tables

Permanent Journal Scenario 57-4
Table X 57-6
Table Y 57-8
Table Z 57-10
Permanent Journals 57-12
Archive Policy 57-14
Archive Scenario 57-16
After Restart Processing Completes 57-18
After REBUILD and Restart of Teradata 57-20
Table X Recovery 57-22
Table Y Recovery 57-24
Table Z Recovery 57-26
After Recovery 57-28
Summary 57-30

Module 58 – Archiving Data

Archive and Recovery Utility (ARC) 58-4
Archive and Recovery Phases 58-6
Restore versus FastLoad 58-8
ARC 58-10
Session Control 58-12
Multiple Sessions 58-14
ARC Statements 58-16
ARCHIVE Statement 58-18
ARCHIVE Examples 58-20
Archiving Selected Partitions of PPI Table 58-24
ARCHIVE Partition Example 58-26
ANALYZE Statement 58-28
ANALYZE Output 58-30
Archive Objects 58-32
Archive Objects (cont.) 58-34
Archive Levels 58-36
Archive Options 58-38
ONLINE Archive Option 58-40
BakBone NetVault Example 58-42
Database DBC Archive 58-44
Summary 58-46
Module 58: Review Questions 58-48

Module 59 – Restoring Data

Understanding Restore Operations 59-4
Restore-Related Statements 59-6
The Restore Statement 59-8
Restoring Examples 59-10
RESTORE Example and Output 59-12
Restoring Selected Partitions of PPI Table 59-14
RESTORE Partition Example 59-16
COPY Statement 59-18
Copying Objects 59-20
Copying 59-22
BUILD Statement 59-24
RELEASE LOCK Statement 59-26
Revalidate References 59-28
Revalidate References Output 59-30
Recovery Control Data Dictionary Views 59-32
Association View 59-34
Events View 59-36
Restoring Data Summary 59-38
Module 59: Review Questions 59-40

Module 60 – Data Recovery Operations

Data Recovery Using Roll Operations 60-4
The CHECKPOINT Statement 60-6
CHECKPOINT WITH SAVE Statement 60-8
Using the ROLLBACK Command 60-10
The ROLLBACK Statement 60-12
ROLLFORWARD Statement 60-14
ROLLFORWARD Restrictions 60-16
The ROLLFORWARD Statement 60-18
DELETE JOURNAL Statement 60-20
Summary 60-22
Module 60: Review Questions 60-24

Module 61 – Teradata Factory Recap

Teradata Factory Review – Week 1 61-4
Teradata Factory Review – Week 2 61-6
Dictionary Tables to Maintain 61-8
Plan and Follow-up 61-10
Things You Never Have to do with Teradata 61-12
Things You Never Have to do with Teradata (cont.) 61-14
Teradata Differentiators 61-16
Teradata Certification Tests 61-18

Notes