SQL Server Inside Out

SQL Server Inside Out

Trainer's Guide

APTECH LIMITED

Contact E-mail: ov-support@onlinevarsity.com

First Edition - 2016

Aptech® Unleash your potential

Aptech™ COMPUTER EDUCATION Unleash your potential

Preface

The book 'SQL Server Inside Out Trainer's Guide' serves understanding on features and functionalities of Microsoft SQL Server for storing and managing data. The faculty/trainer should teach the concepts in the theory class using the slides. This Trainer's Guide will provide guidance on the flow of the session and also provide tips and additional examples wherever necessary. The trainer can ask questions to make the session interactive and also to test the understanding of the students.

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 Certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

We will be glad to receive your suggestions.

Design Team

“Practice is the best of all instructors.

Table of Contents

Sessions

“The future depends on what we do in the present.”

Session 1 – RDBMS Concepts

1.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth.
Prepare a question or two which will be a key point to relate the current session objectives.

1.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the concept of data and database
- Describe the approaches to data management
- Define a Database Management System (DBMS) and list its benefits
- Explain the different database models
- Define and explain RDBMS
- Describe entities and tables and list the characteristics of tables
- List the differences between a DBMS and an RDBMS

1.1.2 Teaching Skills

To teach this session, you should be well-versed with the concepts related to databases and database management systems, explore various database models, and introduce the concept of an RDBMS.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slide and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

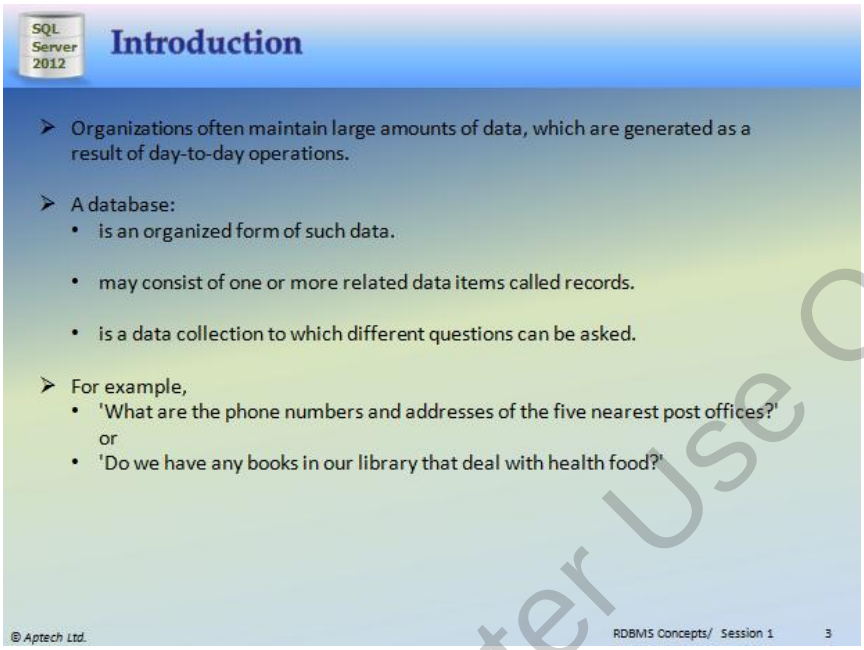
In-Class Activities:
Follow the order given here during In-Class activities.

Overview of the Session:
Give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces RDMS concepts. They will learn about the concepts related to databases and database management systems, explore various database models, and will be introduced to the concept of an RDBMS.

1.2 In-Class Explanations

Introduction to RDBMS

Slide 3

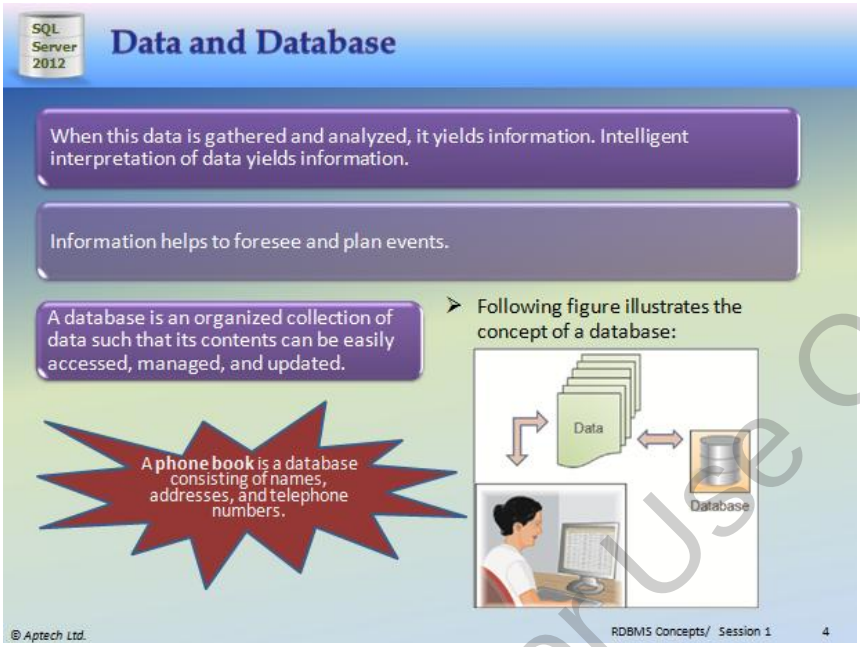


Using slide 3, explain the need of database and its management.

Mention that organizations often maintain large amounts of data, which are generated as a result of day-to-day operations. A database is an organized form of such data. It may consist of one or more related data items called records. Think of a database as a data collection to which different questions can be asked. For example, 'What are the phone numbers and addresses of the five nearest post offices?' or 'Do we have any books in our library that deal with health food? If so, on which shelves are they located?' or 'Show me the personnel records and sales figures of five best-performing sales people for the current quarter, but their address details need not be shown'.

Data and Database

Slide 4



Using slide 4, explain the data and database.

Data means information and it is the most important component in any work that is done. In day-to-day activities, either existing data is used or more data is generated. When this data is gathered and analyzed, it yields information. It can be any information such as information about the vehicle, sports, airways, and so on. For example, a sport magazine journalist (who is a soccer enthusiast) gathers the score (data) of Germany's performance in 10 world cup matches. These scores constitute data. When this data is compared with the data of 10 world cup matches played by Brazil, the journalist can obtain information as to which country has a team that plays better soccer.

Information helps to foresee and plan events. Intelligent interpretation of data yields information. In the world of business, to be able to predict an event and plan for it could save time and money. Consider an example, where a car manufacturing company is planning its annual purchase of certain parts of the car, which has to be imported since it is not locally available. If data of the purchase of these parts for the last five years is available, the company heads can actually compile information about the total amount of parts imported. Based on these findings, a production plan can be prepared. Therefore, information is a key-planning factor.

A database is a collection of data. Some like to think of a database as an organized mechanism that has the capability of storing information. This information can be retrieved by the user in an effective and efficient manner.

A phone book is a database. The data contained consists of individuals' names, addresses, and telephone numbers. These listings are in alphabetical order or indexed. This allows the

user to reference a particular local resident with ease. Ultimately, this data is stored in a database somewhere on a computer. As people move to different cities or states, entries may have to be added or removed from the phone book. Likewise, entries will have to be modified for people changing names, addresses, or telephone numbers, and so on.

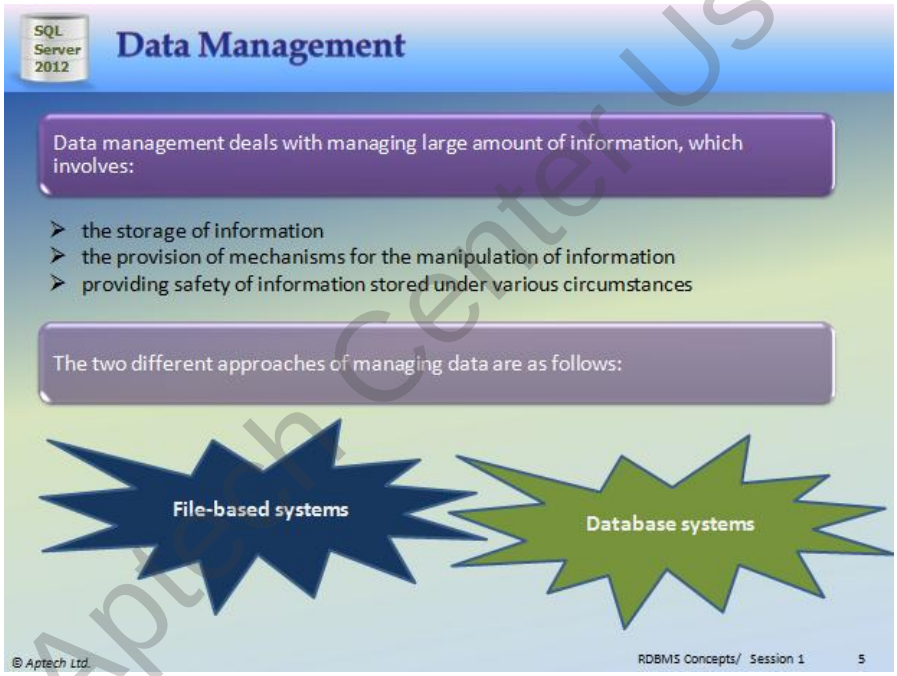
In-Class Question:

What is a database?

Answer:
A database is a collection of data.

Data Management

Slide 5

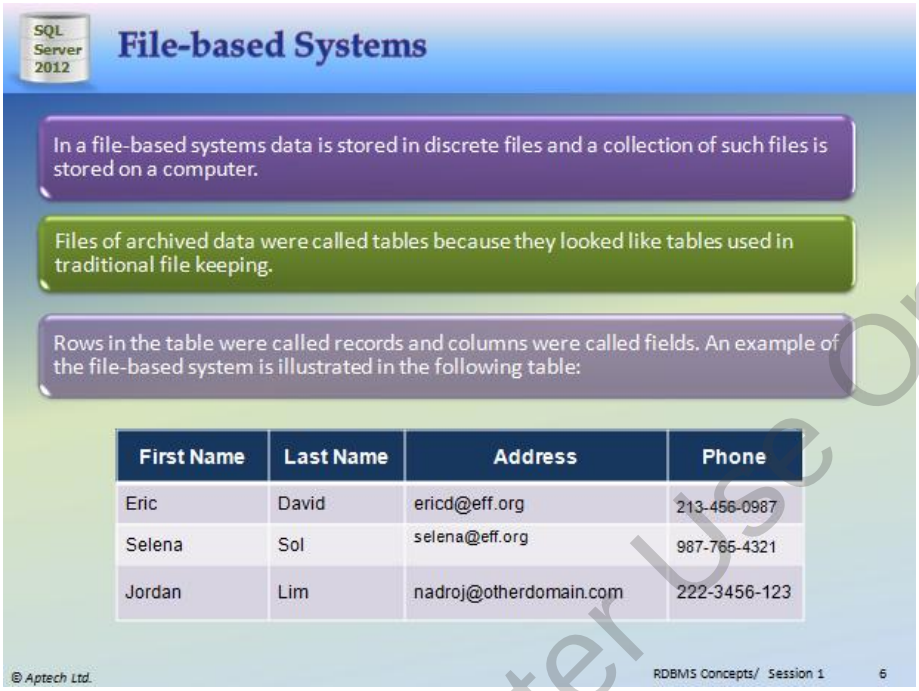


Using slide 5, explain data management.

Data management deals with managing large amount of information, which involves both the storage of information and the provision of mechanisms for the manipulation of information. In addition, the system should also provide the safety of the information stored under various circumstances, such as multiple user access and so on. The two different approaches of managing data are file-based systems and database systems.

File-based Systems

Slide 6



Using slide 6, explain File-based systems.

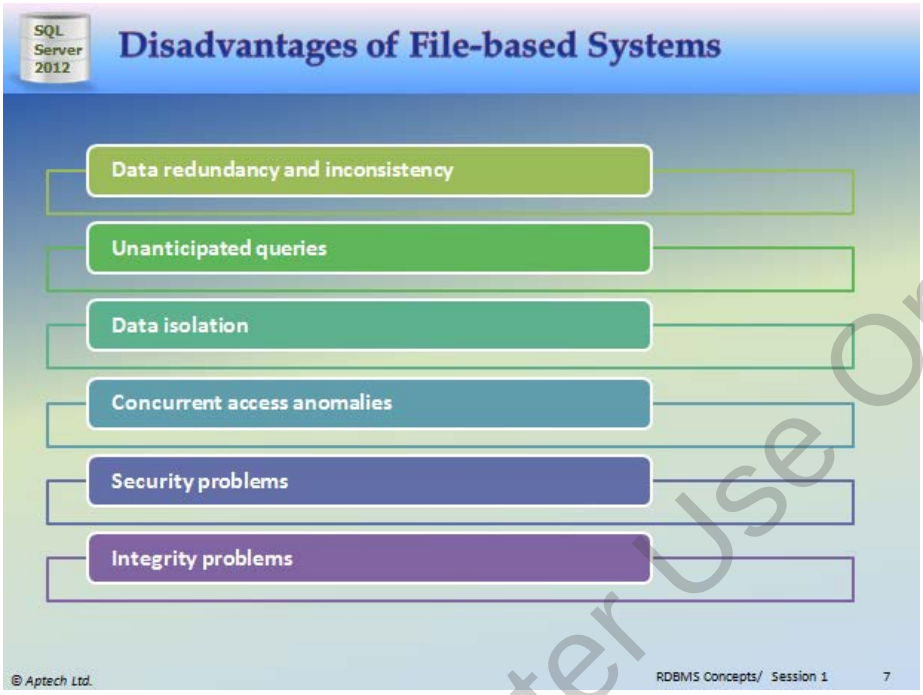
Mention storage of large amounts of data has always been a matter of huge concern. In early days, file-based systems were used. In this system, data was stored in discrete files and a collection of such files was stored on a computer. These could be accessed by a computer operator. Files of archived data were called tables because they looked like tables used in traditional file keeping. Rows in the table were called records and columns were called fields.

Conventionally, before the database systems evolved, data in software systems was stored in flat files.

Give an example of the file-based system. Mention that file based systems were developed as better alternatives to paper based filing systems.

Disadvantages of File-based Systems

Slide 7



Using slide 7, explain the disadvantages of file-based systems in detail.

In a file-based system, different programs in the same application may be interacting with different private data files. There is no system enforcing any standardized control on the organization and structure of these data files.

- **Data redundancy and inconsistency**
  Since data resides in different private data files, there are chances of redundancy and resulting inconsistency. For example, a customer can have a savings account as well as a mortgage loan. Here, the customer details may be duplicated since the programs for the two functions store their corresponding data in two different data files. This gives rise to redundancy in the customer's data. Since the same data is stored in two files, inconsistency arises if a change made in the data of one file is not reflected in the other.

- **Unanticipated queries**
  In a file-based system, handling sudden/ad-hoc queries can be difficult, since it requires changes in the existing programs. For example, the bank officer needs to generate a list of all the customers who have an account balance of $20,000 or more. The bank officer has two choices: either obtain the list of all customers and have the needed information extracted manually, or hire a system programmer to design the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and several days later, the officer needs to trim that list to include only those customers who have opened their account one year ago. As the program to generate such a list does not exist, it leads to a difficulty in accessing the data.

- **Data isolation**
  Data are scattered in various files, and files may be in a different format. Though data used by different programs in the application may be related, they reside as isolated data files.

- **Concurrent access anomalies**
  In large multi-user systems, the same file or record may need to be accessed by multiple users simultaneously. Handling this in a file-based system is difficult.

- **Security problems**
  In data-intensive applications, security of data is a major concern. Users should be given access only to required data and not to the whole database.

  For example, in a banking system, payroll personnel need to view only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. Since application programs are added to the system in an ad-hoc manner, it is difficult to enforce such security constraints. In a file-based system, this can be handled only by additional programming in each application.

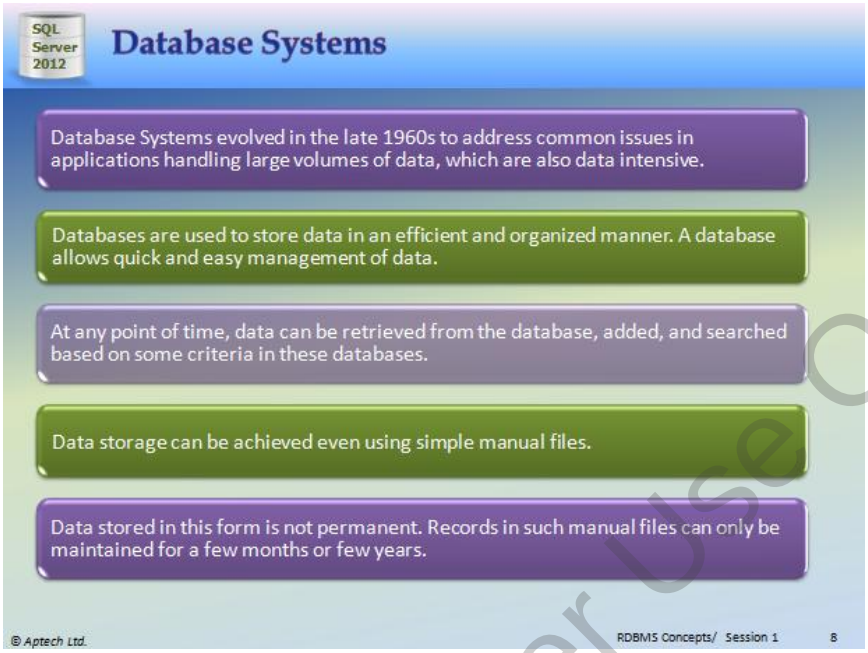
- **Integrity problems**

  In any application, there will be certain data integrity rules, which need to be maintained. These could be in the form of certain conditions/constraints on the elements of the data records. In the savings bank application, one such integrity rule could be 'Customer ID, which is the unique identifier for a customer record, should not be empty'. There can be several such integrity rules. In a file-based system, all these rules need to be explicitly programmed in the application program.

  Though all these are common issues of concern to any data-intensive application, each application had to handle all these problems on its own. The application programmer needs to bother not only about implementing the application business rules but also, about handling these common issues.

Database Systems

Slide 8



Using slide 8 explain, the Database system.

Database Systems evolved in the late 1960s to address common issues in applications handling large volumes of data, which are also data intensive. Some of these issues could be traced back to the disadvantages of File-based systems.

Databases are used to store data in an efficient and organized manner. A database allows quick and easy management of data. For example, a company may maintain details of its employees in various databases. At any point of time, data can be retrieved from the database, new data can be added into the databases and data can be searched based on some criteria in these databases.

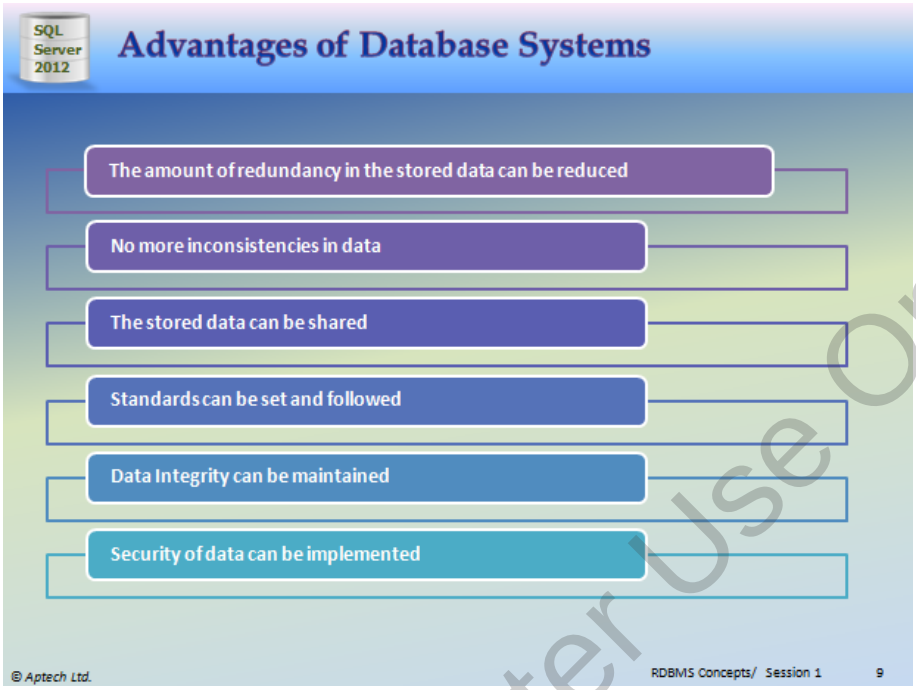
Data storage can be achieved even using simple manual files. For instance, a college has to maintain information about teachers, students, subjects, and examinations.

Details of the teachers can be maintained in a Staff Register and details of the students could be entered in a Student Register and so forth. However, data stored in this form is not permanent. Records in such manual files can only be maintained for a few months or few years. The registers or files are bulky, consume a lot of space, and hence, cannot be kept for many years.

Instead of this, if the same data was stored using database system, it could be more permanent and long-lasting.

Advantages of Database Systems

Slide 9



Using slide 9, explain the advantages of Database systems.

Mention information or data can be permanently stored in the form of computerized databases. A database system is advantageous because it provides a centralized control over the data.

Some of the benefits of using such a centralized database system are as follows:

- **The amount of redundancy in the stored data can be reduced**
  In an organization, several departments often store the same data. Maintaining a centralized database helps the same data to be accessed by many departments. Thus, duplication of data or 'data redundancy' can be reduced.

- **No more inconsistencies in data**
  When data is duplicated across several departments, any modifications to the data have to be reflected across all departments. Sometimes, this can lead to inconsistency in the data. As there is a central database, it is possible for one person to take up the task of updating the data on a regular basis. Consider that Mr. Larry Finner, an employee of an organization is promoted as a Senior Manager from Manager. In such a case, there is just one record in the database that needs to be changed. As a result, data inconsistency is reduced.

- **The stored data can be shared**
  A central database can be located on a server, which can be shared by several users. In this way, all users can access the common and updated information all the time.

- **Standards can be set and followed**
  A central control ensures that a certain standard in the representation of data can be set and followed. For example, the name of an employee has to be represented as 'Mr. Larry Finner'. This representation can be broken down into the following components:
  - A title (Mr.)
  - First name (Larry)
  - Last name (Finner)

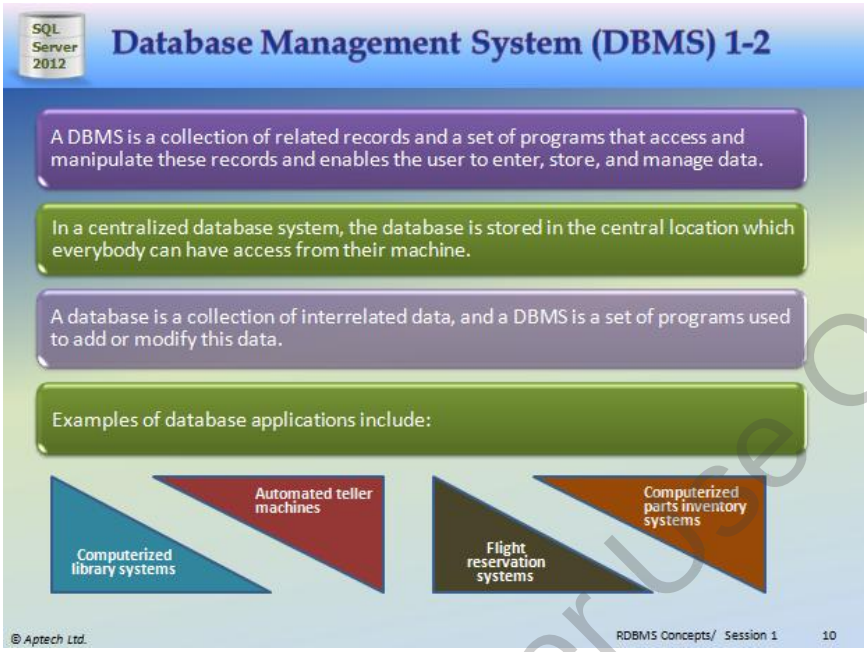
  It is certain that all the names stored in the database will follow the same format if the standards are set in this manner.

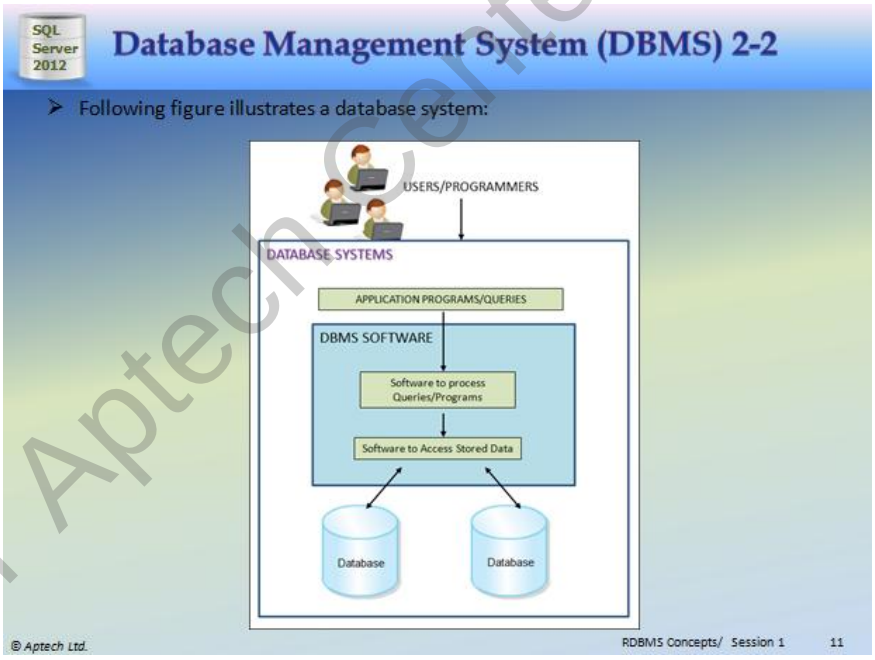
- **Data Integrity can be maintained**
  Data integrity refers to the accuracy of data in the database. For example, when an employee resigns and leaves the organization, consider that the Accounts department has updated its database and the HR department has not updated its records. The data in the company's records is hence inaccurate. Centralized control of the database helps in avoiding these errors. It is certain that if a record is deleted from one table, its linked record in the other table is also deleted.

- **Security of data can be implemented**
  In a central database system, the privilege of modifying the database is not given to everyone. This right is given only to one person who has full control over the database. This person is called as Database Administrator or DBA. The DBA can implement security by placing restrictions on the data. Based on the permissions granted to them, the users can add, modify, or query data.

DBMS Concept

Slides 10 and 11





Using slides 10 and 11, explain the DBMS concept.

A DBMS can be defined as a collection of related records and a set of programs that access and manipulate these records. A DBMS enables the user to enter, store, and manage data. The main problem with the earlier DBMS packages was that the data was stored in the flat file format. So, the information about different objects was maintained separately in different physical files. Hence, the relations between these objects, if any, had to be

maintained in a separate physical file. Thus, a single package would consist of too many files and vast functionalities to integrate them into a single system.

A solution to these problems came in the form of a centralized database system. In a centralized database system, the database is stored in the central location. Everybody can have access to the data stored in a central location from their machine. For example, a large central database system would contain all the data pertaining to the employees. The Accounts and the HR department would access the data required using suitable programs. These programs or the entire application would reside on individual computer terminals. A Database is a collection of interrelated data, and a DBMS is a set of programs used to add or modify this data. Thus, a DBMS is a set of software programs that allow databases to be defined, constructed, and manipulated.

A DBMS provides an environment that is both convenient and efficient to use when there is a large volume of data and many transactions to be processed. Different categories of DBMS can be used, ranging from small systems that run on personal computers to huge systems that run on mainframes.

Examples of database applications include the following:

- Computerized library systems
- Automated teller machines
- Flight reservation systems
- Computerized parts inventory systems

From a technical standpoint, DBMS products can differ widely. Different DBMS support different query languages, although there is a semi-standardized query language called Structured Query Language (SQL). Sophisticated languages for managing database systems are called Fourth Generation Language (4GLs).

The information from a database can be presented in a variety of formats. Most DBMS include a report writer program that enables the user to output data in the form of a report. Many DBMS also include a graphic component that enables the user to output information in the form of graphs and charts.

It is not necessary to use general-purpose DBMS for implementing a computerized database. The users can write their own set of programs to create and maintain the database, in effect creating their own special-purpose DBMS software. The database and the software together are called a database system. The end user accesses the database system through application programs and queries. The DBMS software enables the user to process the queries and programs placed by the end user. The software accesses the data from the database. The figure given in slide 11 illustrates a database system.

Mention DBMS can be thought of as a *file manager* that manages data in databases rather than files in file systems. It is usually an inherent part of a database product.
Microsoft's SQL Server is an example of a DBMS which serves database requests from multiple (client) users.

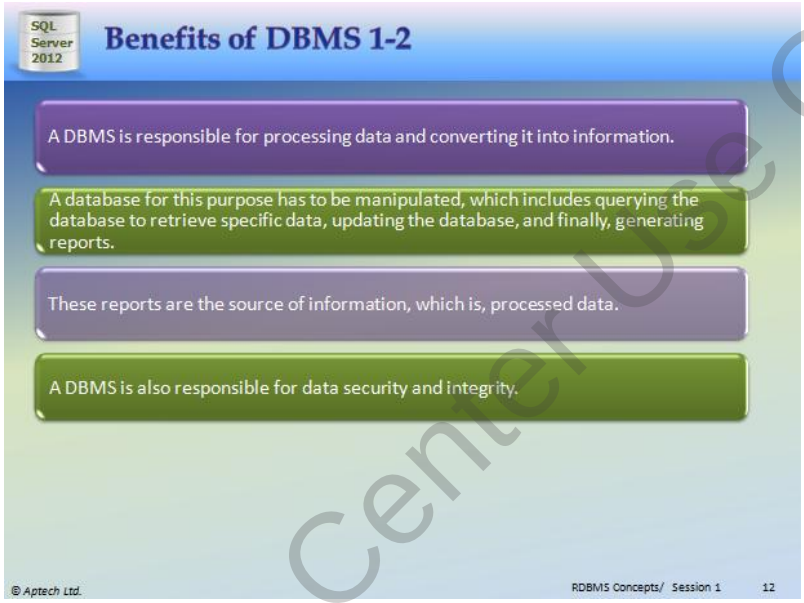
In-Class Question:

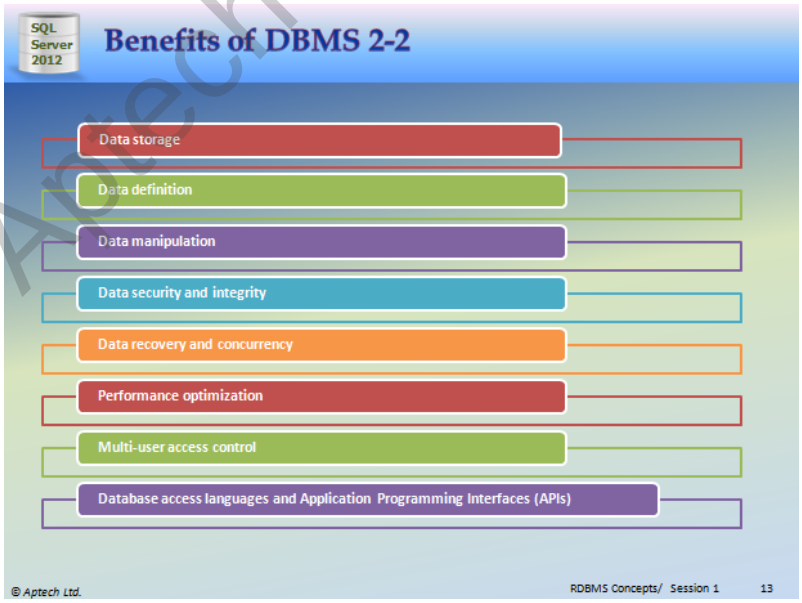
What is a DBMS?

Answer:
A DBMS can be defined as a collection of related records and a set of programs that access and manipulate these records.

Benefits of DBMS

Slides 12 and 13





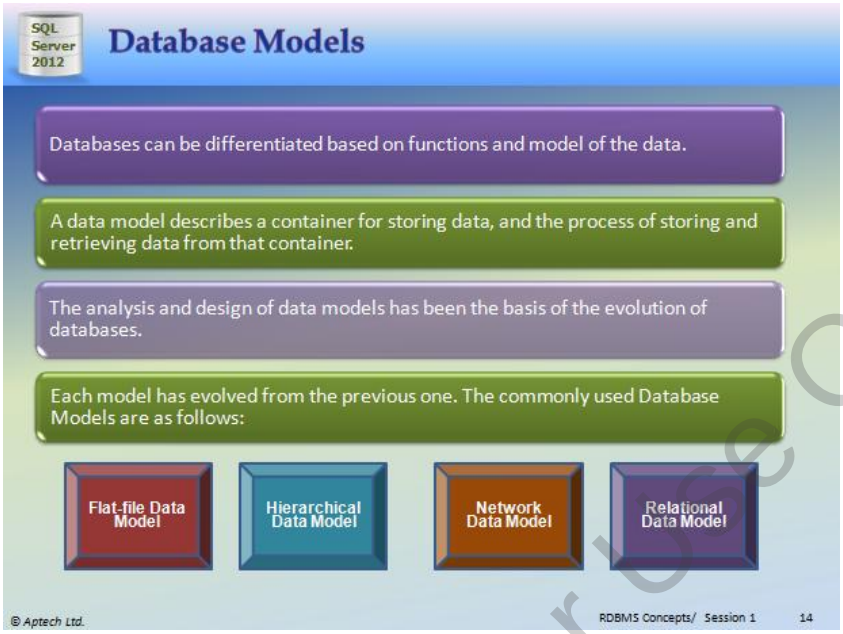
Using slides 12 and 13, explain the benefits of DBMS.
A DBMS is responsible for processing data and converting it into information. For this purpose, the database has to be manipulated, which includes querying the database to retrieve specific data, updating the database, and finally, generating reports.

These reports are the source of information, which is, processed data. A DBMS is also responsible for data security and integrity. The benefits of a typical DBMS are as follows:

- **Data storage**: The programs required for physically storing data, handled by a DBMS, is done by creating complex data structures, and the process is called data storage management.
- **Data definition**: A DBMS provides functions to define the structure of the data in the application. These include defining and modifying the record structure, the type and size of fields, and the various constraints/conditions to be satisfied by the data in each field.
- **Data manipulation**: Once the data structure is defined, data needs to be inserted, modified, or deleted. The functions, which perform these operations, are also part of a DBMS. These functions can handle planned and unplanned data manipulation needs. Planned queries are those, which form part of the application. Unplanned queries are ad-hoc queries, which are performed on a need basis.
- **Data security and integrity**: Data security is of utmost importance when there are multiple users accessing the database. It is required for keeping a check over data access by users. The security rules specify, which user has access to the database, what data elements the user has access to, and the data operations that the user can perform. Data in the database should contain as few errors as possible. For example, the employee number for adding a new employee should not be left blank. Telephone number should contain only numbers. Such checks are taken care of by a DBMS. Thus, the DBMS contains functions, which handle the security and integrity of data in the application. These can be easily invoked by the application and hence, the application programmer need not code these functions in the programs.
- **Data recovery and concurrency**: Recovery of data after a system failure and concurrent access of records by multiple users are also handled by a DBMS.
- **Performance**: Optimizing the performance of the queries is one of the important functions of a DBMS. Hence, the DBMS has a set of programs forming the Query Optimizer, which evaluates the different implementations of a query and chooses the best among them.
- **Multi-user access control**: At any point of time, more than one user can access the same data. A DBMS takes care of the sharing of data among multiple users, and maintains data integrity.
- **Database access languages and Application Programming Interfaces (APIs)**: The query language of a DBMS implements data access. SQL is the most commonly used query language. A query language is a non-procedural language, where the user needs to request what is required and need not specify how it is to be done. Some procedural languages such as C, Visual Basic, Pascal, and others provide data access to programmers.

Database Models

Slide 14



Using slide 14, explain the Database Model.

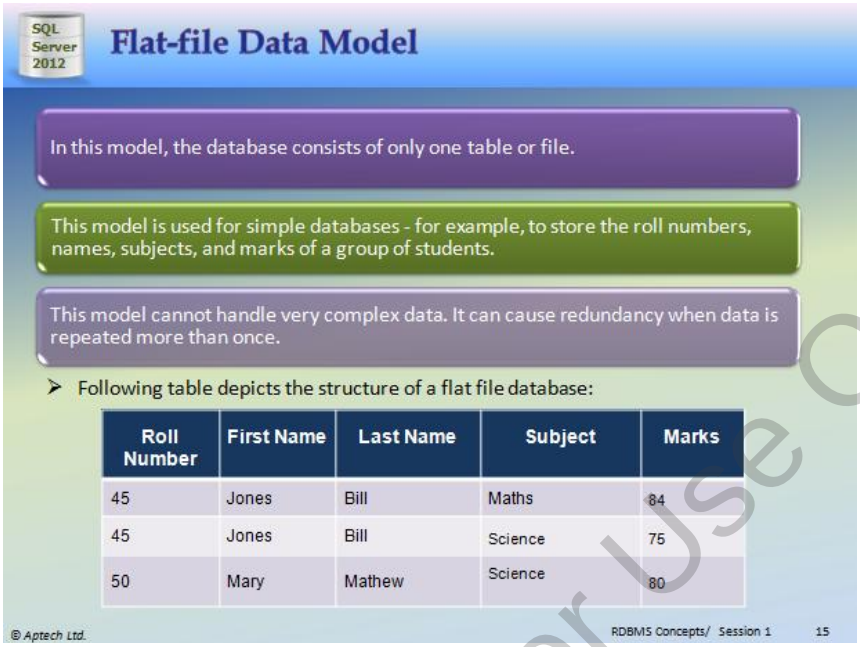
Databases can be differentiated based on functions and model of the data. A data model describes a container for storing data, and the process of storing and retrieving data from that container. The analysis and design of data models has been the basis of the evolution of databases. Each model has evolved from the previous one.

Database Models are briefly discussed in the further sections.

Also, there are object-oriented model and object relational model.

Flat-file Data Model

Slide 15



Using slide 15, explain the flat-file data model.

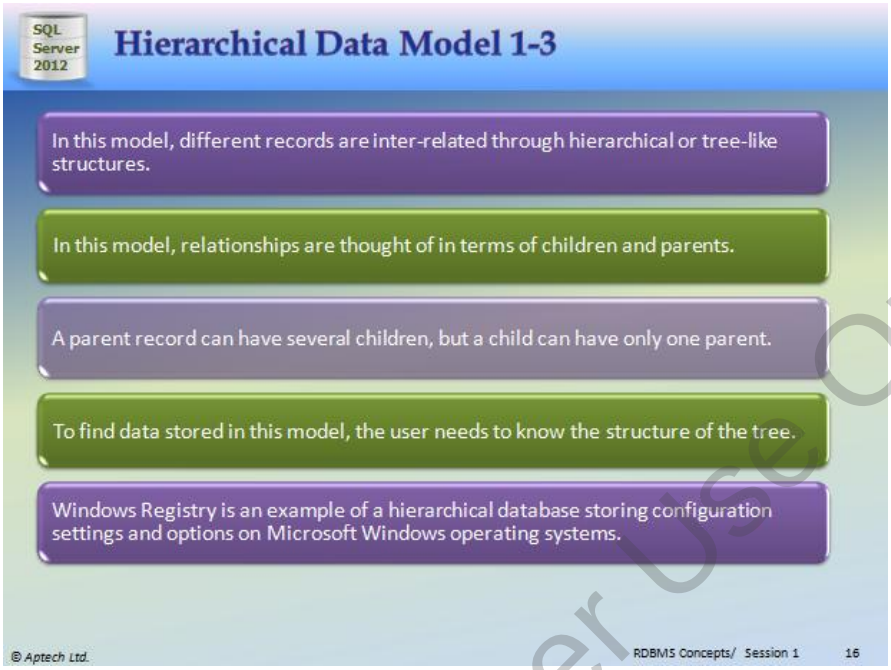
In this model, the database consists of only one table or file. This model is used for simple databases - for example, to store the roll numbers, names, subjects, and marks of a group of students. This model cannot handle very complex data. It can cause redundancy when data is repeated more than once.

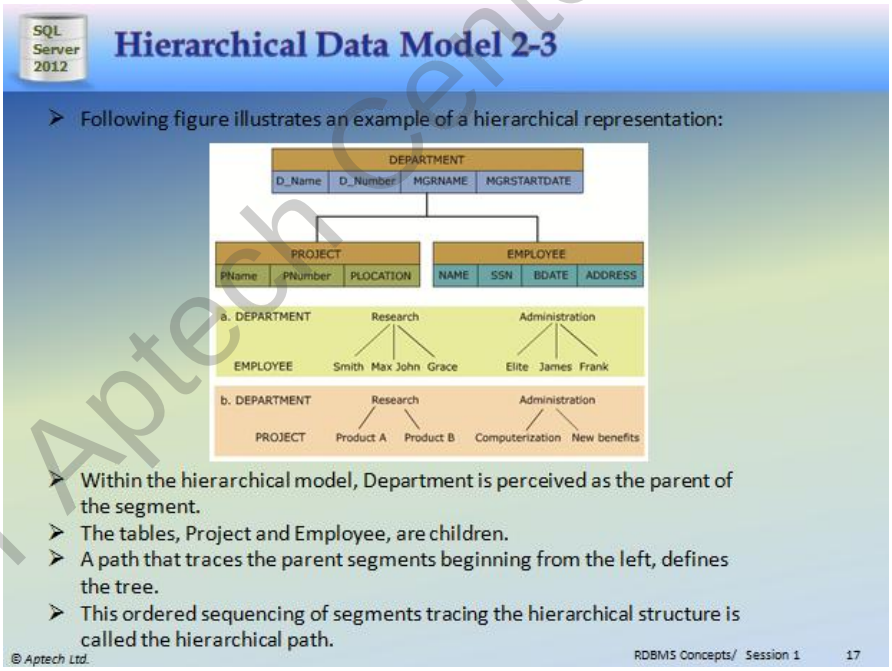
Explain the structure of a flat file database using table.

Mention a flat file database cannot contain multiple tables like a relational database. Most database programs such as Microsoft Access and FileMaker Pro can import flat file databases and use them in a larger relational database.

Hierarchical Data Model

Slides 16 to 18



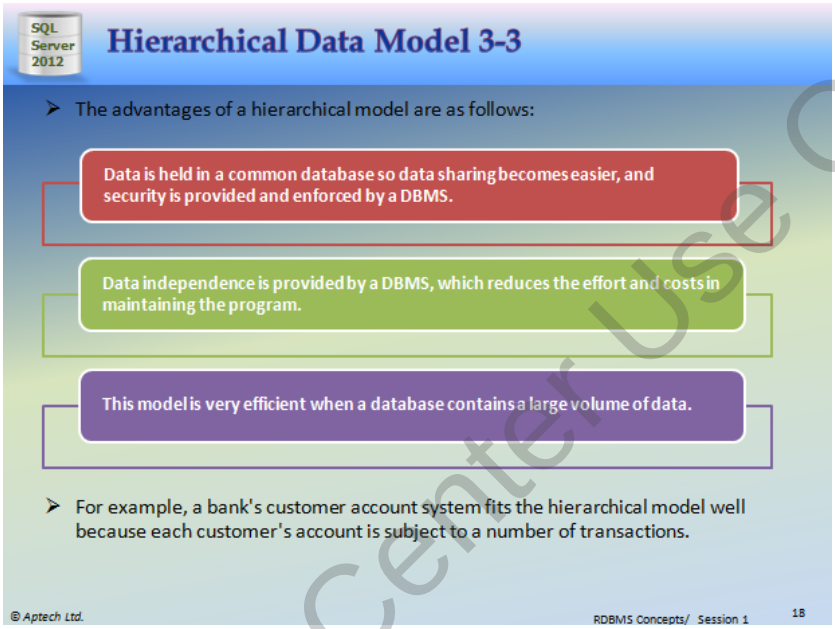


Using slides 16 and 17, explain the Hierarchical Data Model.

In the Hierarchical Model, different records are inter-related through hierarchical or tree-like structures. In this model, relationships are thought of in terms of children and parents. A parent record can have several children, but a child can have only one parent. To find data stored in this model, the user needs to know the structure of the tree.

The Windows Registry is an example of a hierarchical database storing configuration settings and options on Microsoft Windows operating systems. Figure shown in slide 17 illustrates an example of a hierarchical representation.

Within the hierarchical model, Department is perceived as the parent of the segment. The tables, Project and Employee, are children. A path that traces the parent segments beginning from the left, defines the tree. This ordered sequencing of segments tracing the hierarchical structure is called the hierarchical path. It is clear from the figure that in a single department, there can be many employees and a department can have many projects.



Using slide 18, explain the advantages of hierarchical model.

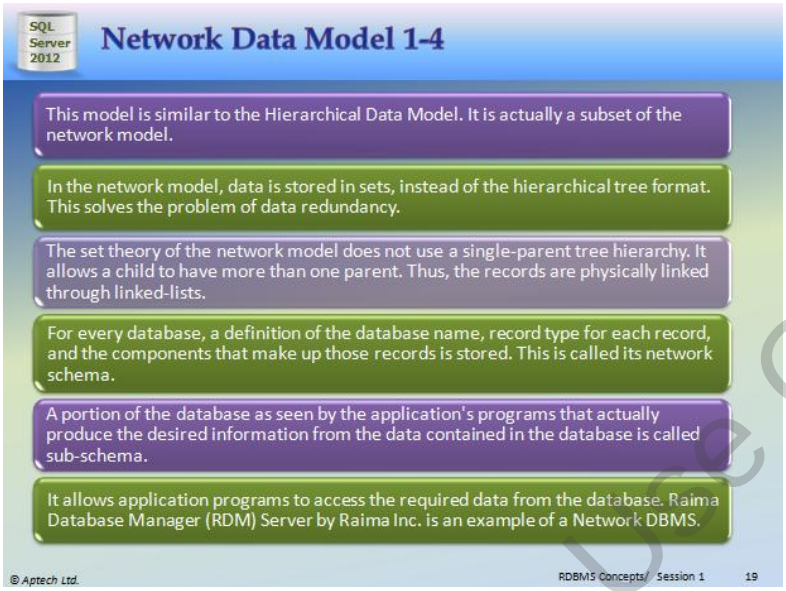
The advantages of a hierarchical model are as follows:

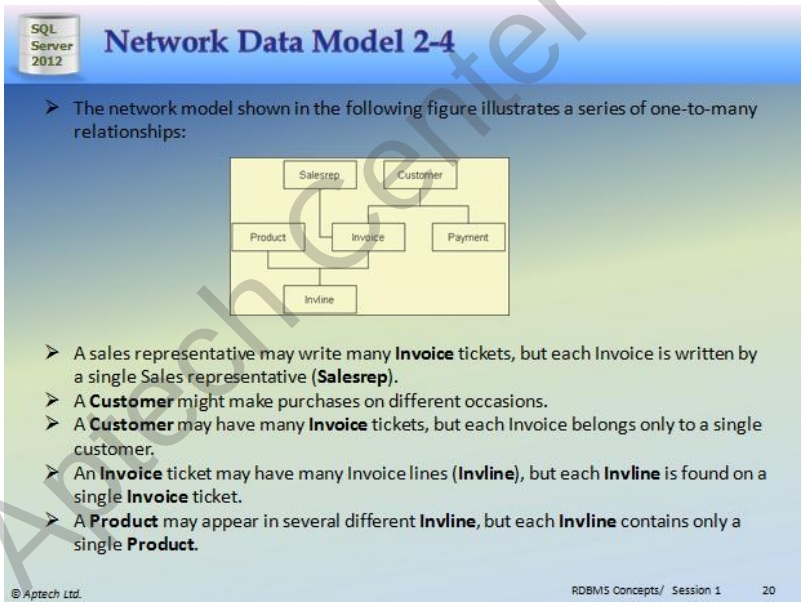
- Data is held in a common database so data sharing becomes easier, and security is provided and enforced by a DBMS.
- Data independence is provided by a DBMS, which reduces the effort and costs in maintaining the program.

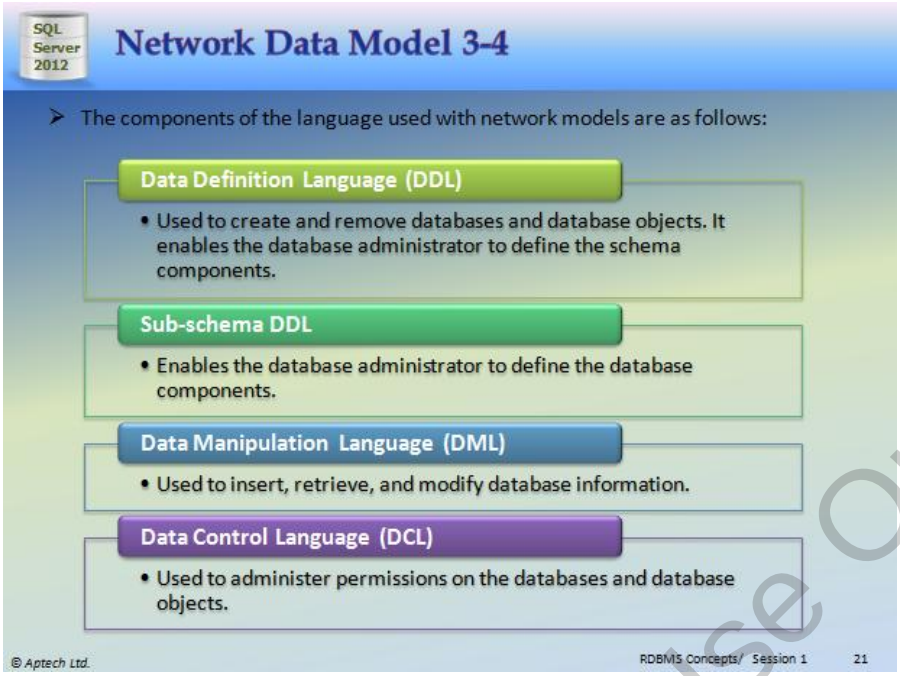
This model is very efficient when a database contains a large volume of data. For example, a bank's customer account system fits the hierarchical model well because each customer's account is subject to a number of transactions.

Network Data Model

Slides 19 to 22







Using slides 19 to 21, explain Network Data Model.

Mention this model is similar to the Hierarchical Data Model. The hierarchical model is actually a subset of the network model. However, instead of using a single-parent tree hierarchy, the network model uses set theory to provide a tree-like hierarchy with the exception that child tables were allowed to have more than one parent.

In the network model, data is stored in sets, instead of the hierarchical tree format. This solves the problem of data redundancy. The set theory of the network model does not use a single-parent tree hierarchy. It allows a child to have more than one parent. Thus, the records are physically linked through linked-lists. Integrated Database Management System (IDMS) from Computer Associates International Inc. and Raima Database Manager (RDM) Server by Raima Inc. are examples of a Network DBMS.

The network model together with the hierarchical data model was a major data model for implementing numerous commercial DBMS. The network model structures and language constructs were defined by Conference on Data Systems Language (CODASYL).

For every database, a definition of the database name, record type for each record, and the components that make up those records is stored. This is called its network schema. A portion of the database as seen by the application's programs that actually produce the desired information from the data contained in the database is called sub-schema. It allows application programs to access the required data from the database.

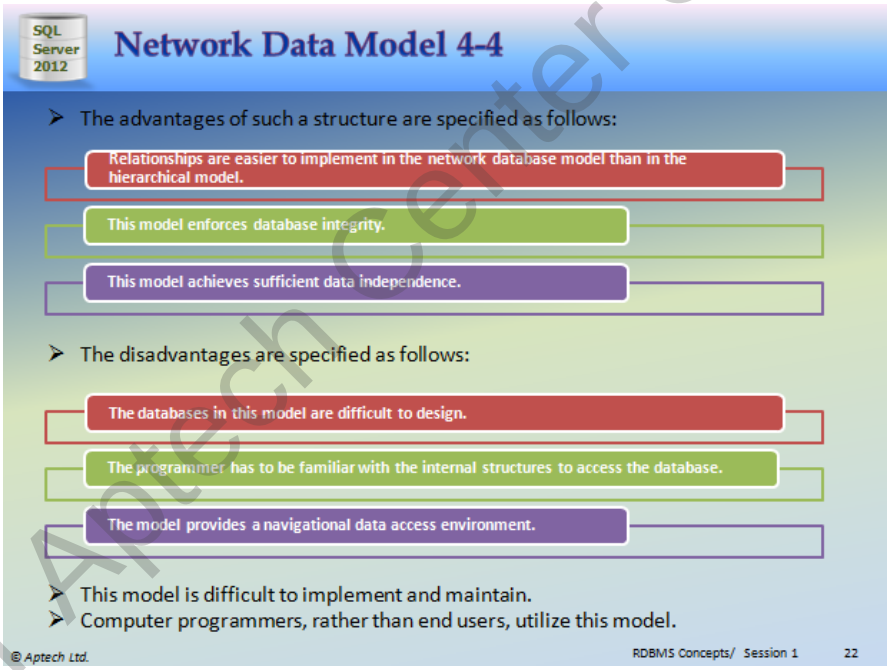
The network model as shown in figure in slide 20 illustrates a series of one-to-many relationships, as follows:

- A sales representative might have written many Invoice tickets, but each Invoice is written by a single Sales representative (Salesrep).

- A Customer might have made purchases on different occasions. A Customer may have many Invoice tickets, but each Invoice belongs only to a single customer.
- An Invoice ticket may have many Invoice lines (Invline), but each Invline is found on a single Invoice ticket.
- A Product may appear in several different Invline, but each Invline contains only a single Product.

The components of the language used with network models are as follows:

1. A Data Definition Language (DDL) that is used to create and remove databases and database objects. It enables the database administrator to define the schema components.
2. A sub-schema DDL that enables the database administrator to define the database components.
3. A Data Manipulation Language (DML), which is used to insert, retrieve, and modify database information. All database users use these commands during the routine operation of the database.
4. Data Control Language (DCL) is used to administer permissions on the databases and database objects.



Using slide 22, explain the advantages and disadvantages of network model.

The advantages of such a structure are specified as follows:

- The relationships are easier to implement in the network database model than in the hierarchical model.
- This model enforces database integrity.
- This model achieves sufficient data independence.

The disadvantages are specified as follows:

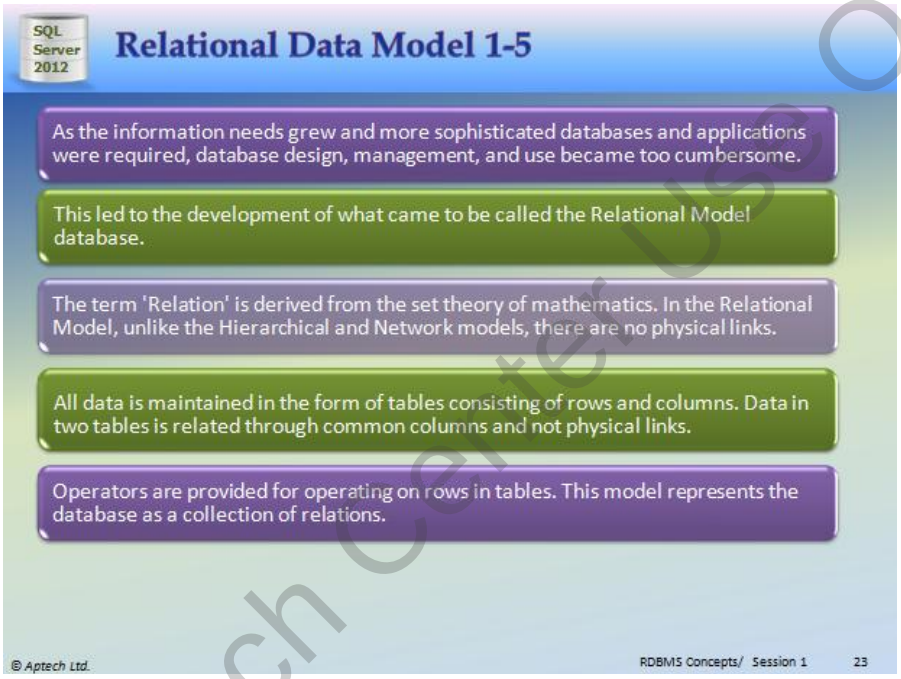
- The databases in this model are difficult to design.

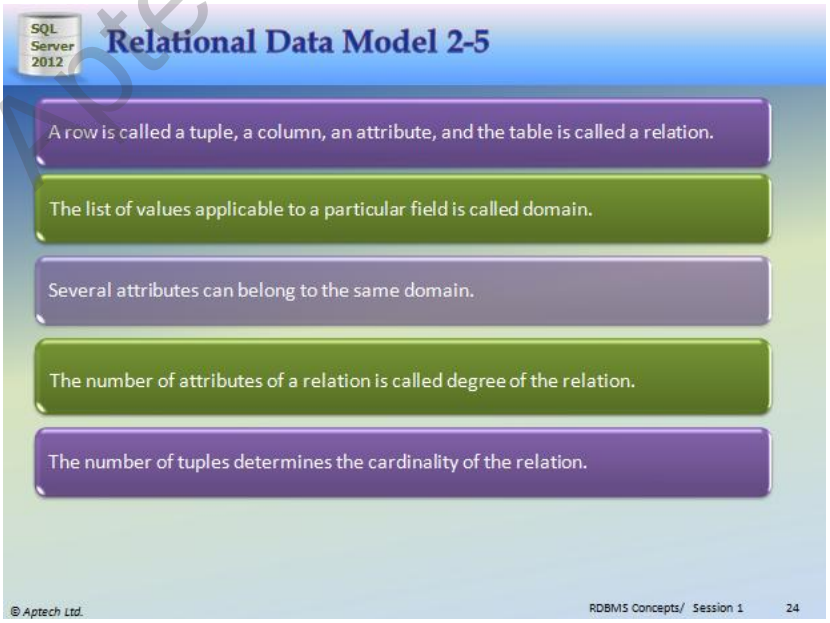
- The programmer has to be very familiar with the internal structures to access the database.
- The model provides a navigational data access environment. Hence, to move from A to E in the sequence A-B-C-D-E, the user has to move through B, C, and D to get to E.

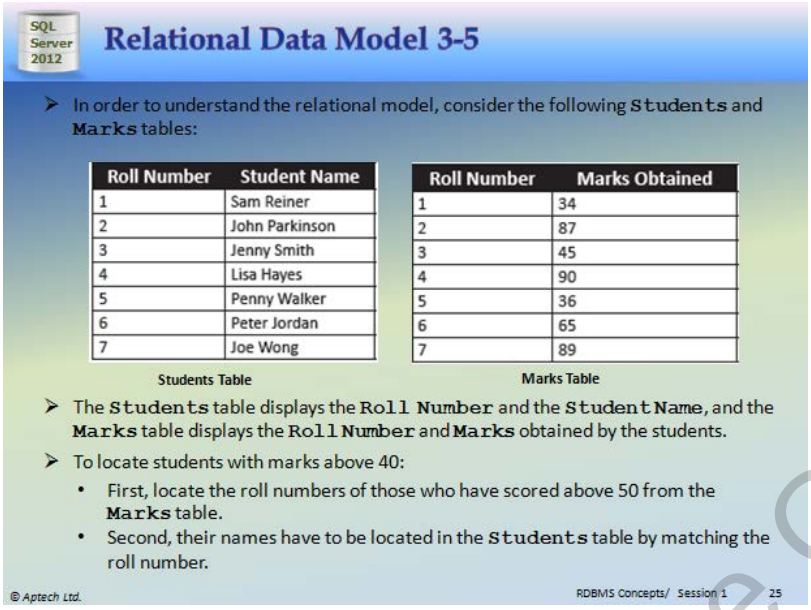
This model is difficult to implement and maintain. Computer programmers, rather than end users, utilize this model.

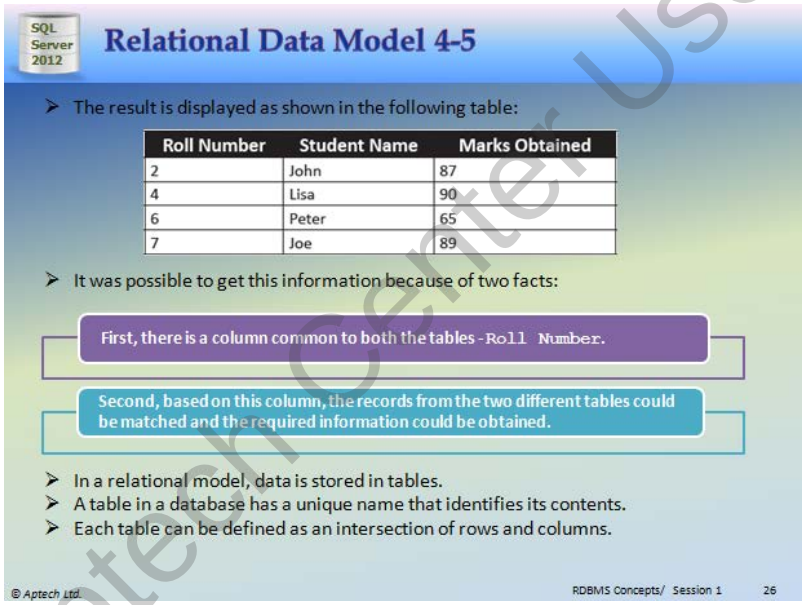
Relational Data Model

Slides 23 to 27









Using slides 23 to 26, explain the Relational Data Model.

As the information needs grew and more sophisticated databases and applications were required, database design, management, and use became too cumbersome. The lack of query facility took a lot of time of the programmers to produce even the simplest reports. This led to the development of what came to be called the Relational Model database.

The term 'Relation' is derived from the set theory of mathematics. In the Relational Model, unlike the Hierarchical and Network models, there are no physical links. All data is maintained in the form of tables consisting of rows and columns. Data in two tables is related through common columns and not physical links. Operators are provided for operating on rows in tables.

The popular relational DBMSs are Oracle, Sybase, DB2, Microsoft SQL Server, and so on.

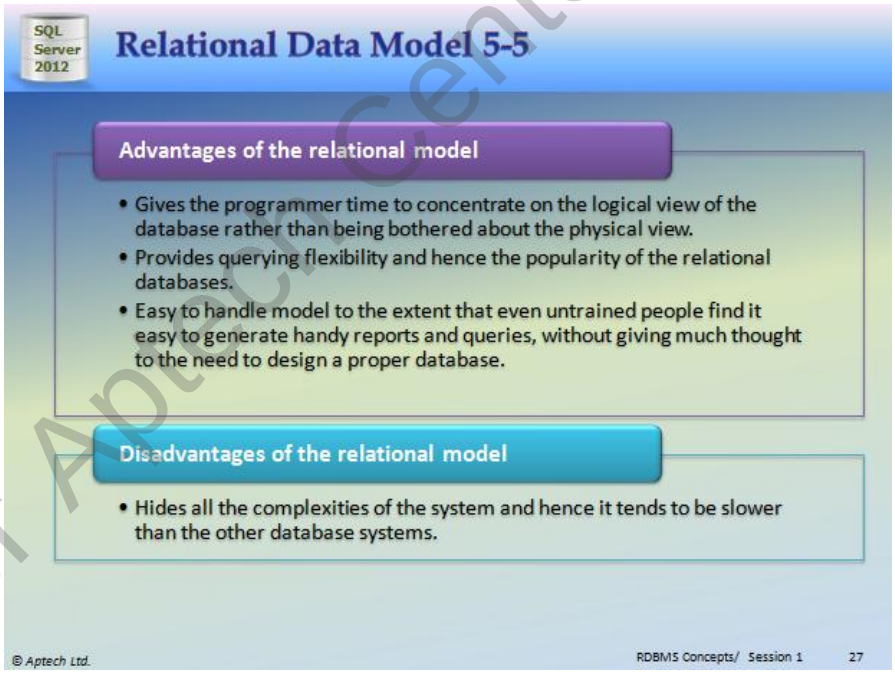
This model represents the database as a collection of relations. In this model's terminology, a row is called a tuple, a column, an attribute, and the table is called a relation. The list of values applicable to a particular field is called domain. It is possible for several attributes to have the same domain. The number of attributes of a relation is called degree of the relation. The number of tuples determines the cardinality of the relation.

In order to understand the relational model, consider tables on slide 25.

The Students table displays the Roll Number and the Student Name, and the Marks table displays the Roll Number and Marks obtained by the students. Now, two steps need to be carried out for students who have scored above 50. First, locate the roll numbers of those who have scored above 50 from the Marks table. Second, their names have to be located in the Students table by matching the roll number. The result will be as shown in the table on slide 26.

It was possible to get this information because of two facts: First, there is a column common to both the tables - Roll Number. Second, based on this column, the records from the two different tables could be matched and the required information could be obtained.

In a relational model, data is stored in tables. A table in a database has a unique name that identifies its contents. Each table can be defined as an intersection of rows and columns.



Using slide 27, explain advantages and disadvantages of relational model.

Advantages of the relational model

The relational database model gives the programmer time to concentrate on the logical view of the database rather than being bothered about the physical view. One of the reasons for the popularity of the relational databases is the querying flexibility. Most of the relational databases use Structured Query Language (SQL). An RDBMS uses SQL to translate

the user query into the technical code required to retrieve the requested data. Relational model is so easy to handle that even untrained people find it easy to generate handy reports and queries, without giving much thought to the need to design a proper database.

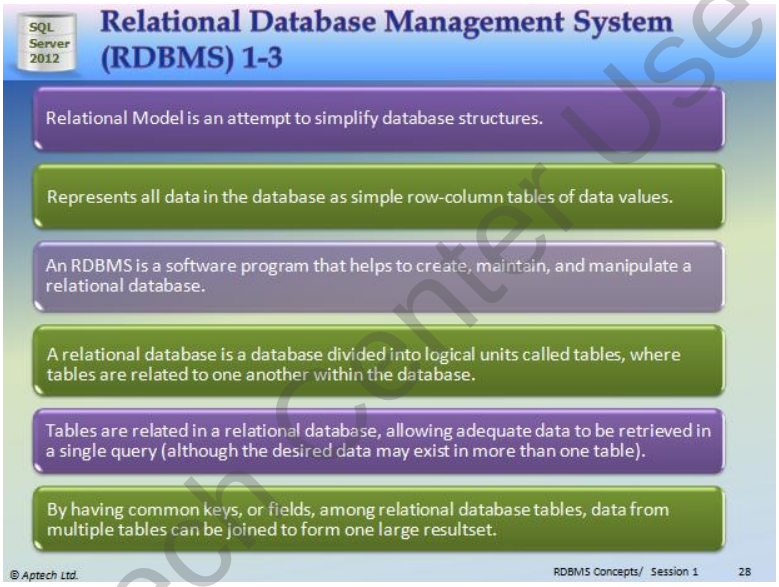
Disadvantages of the relational model

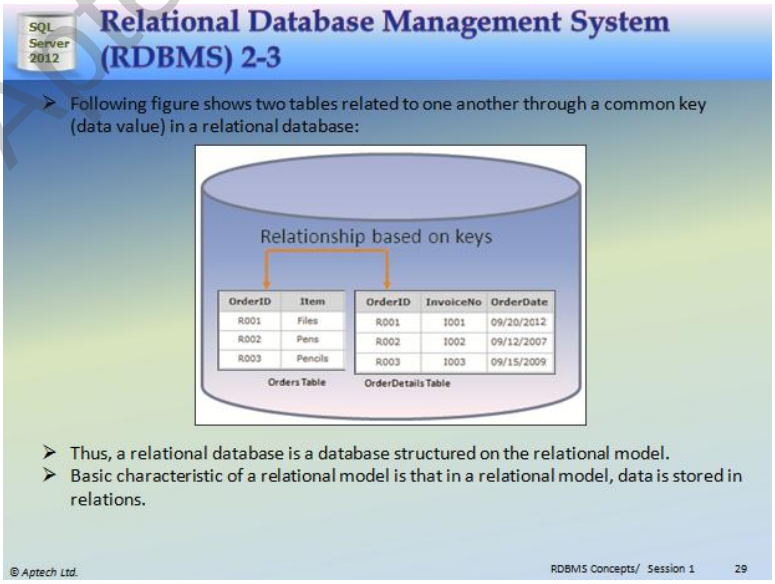
Though the model hides all the complexities of the system, it tends to be slower than the other database systems.

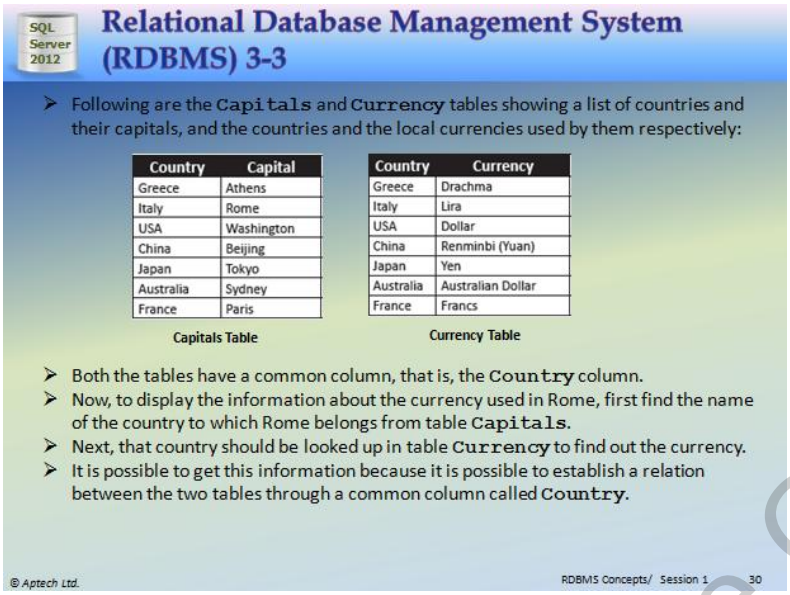
As compared to all other models, the relational data model is the most popular and widely used.

RDBMS Concept

Slides 28 to 30







Using slides 28 to 30, explain the RDBMS concept.

The Relational Model is an attempt to simplify database structures. It represents all data in the database as simple row-column tables of data values. An RDBMS is a software program that helps to create, maintain, and manipulate a relational database. A relational database is a database divided into logical units called tables, where tables are related to one another within the database.

Tables are related in a relational database, allowing adequate data to be retrieved in a single query (although the desired data may exist in more than one table). By having common keys, or fields, among relational database tables, data from multiple tables can be joined to form one large resultset.

In slide 29, figure shows two tables related to one another through a common key (data value) in a relational database.

Thus, a relational database is a database structured on the relational model. The basic characteristic of a relational model is that in a relational model, data is stored in relations. To understand relations, consider the following example.

The Capitals table shown in table displays a list of countries and their capitals, and the Currency table displays the countries and the local currencies used by them.
Both the tables have a common column, that is, the Country column. Now, if the user wants to display the information about the currency used in Rome, first find the name of the country to which Rome belongs. This information can be retrieved from table. Next, that country should be looked up in table to find out the currency.

It is possible to get this information because it is possible to establish a relation between the two tables through a common column called Country.

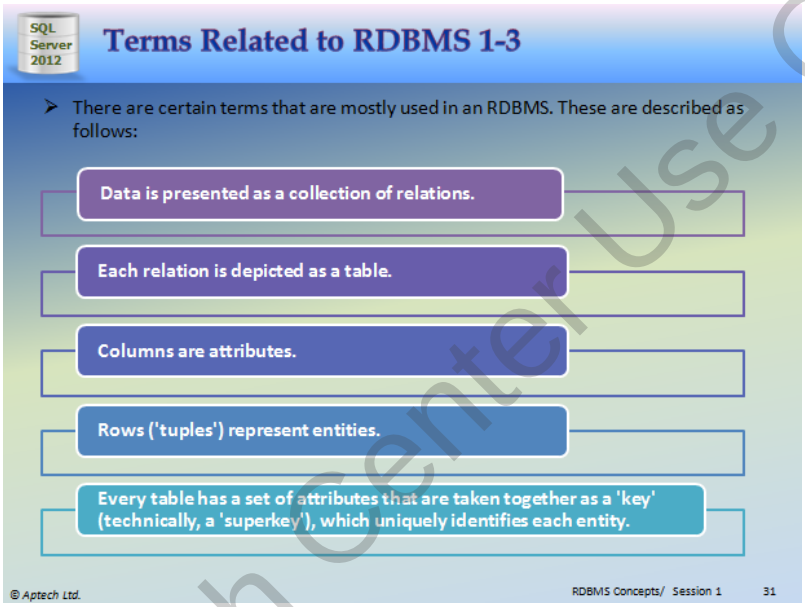
In-Class Question:

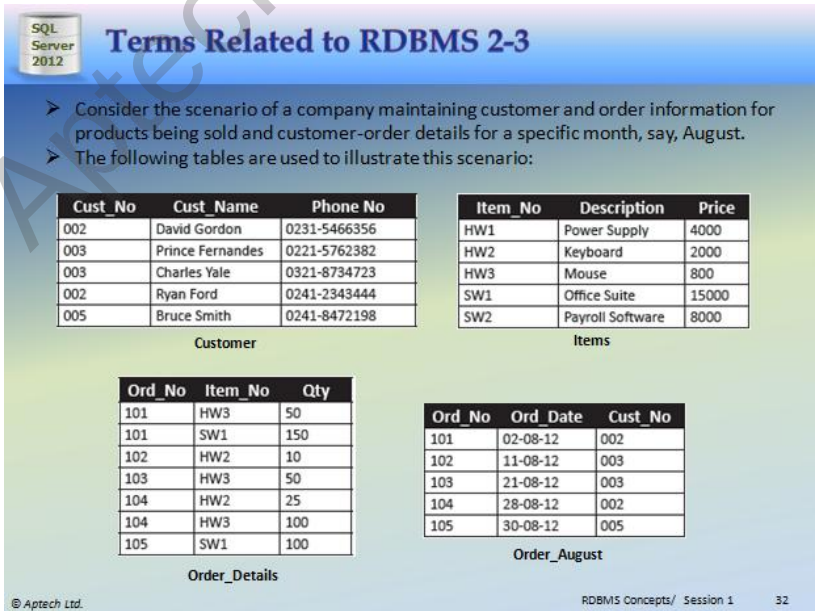
What is a RDBMS?

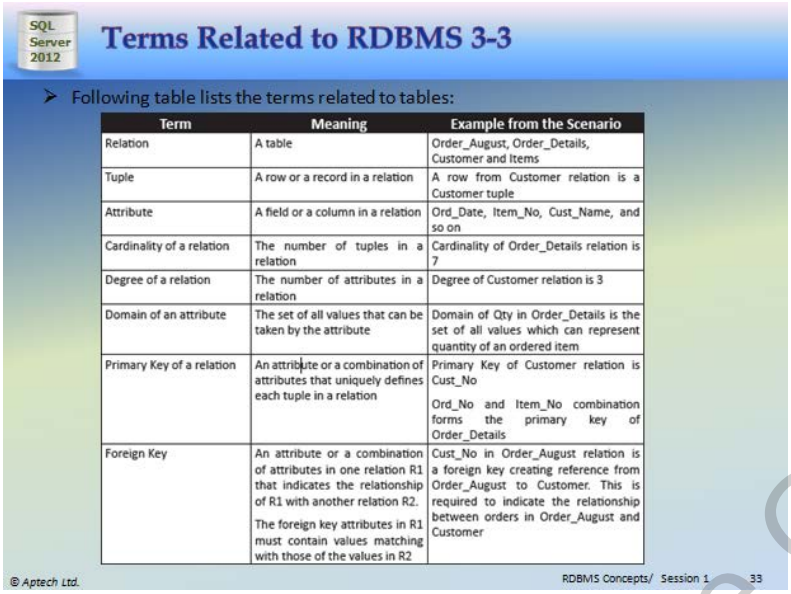
Answer:
An RDBMS is a software program that helps to create, maintain, and manipulate a relational database.

Terms Related to RDBMS

Slides 31 to 33







Using slides 31 to 33, explain some terms related to RDBMS.

For example, a company might have an Employee table with a row for each employee. What attributes might be interesting for such a table? This will depend on the application and the type of use the data will be put to, and is determined at database design time.

Consider the scenario of a company maintaining customer and order information for products being sold and customer-order details for a specific month, such as, August. The tables are used to illustrate this scenario. These tables depict tuples and attributes in the form of rows and columns. Various terms related to these tables are given in table. Explain the terms on slide in detail by using the tables.

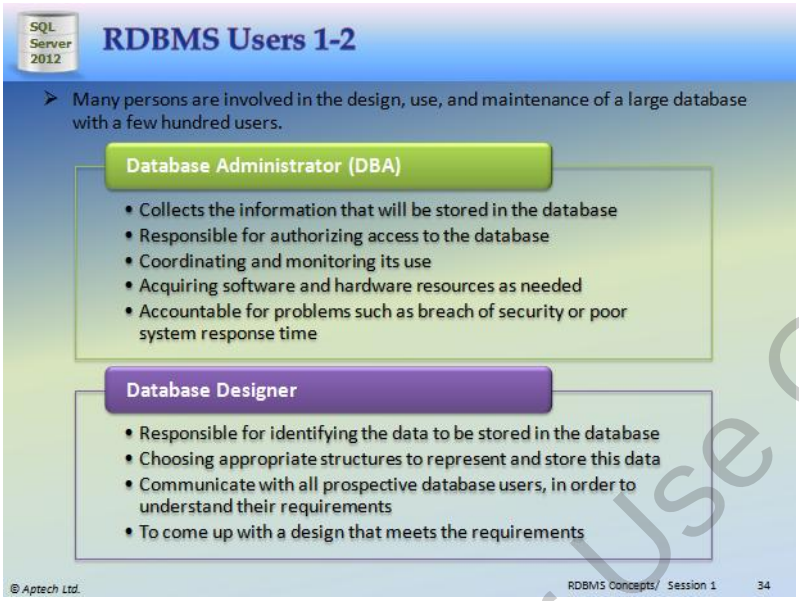
In-Class Question:

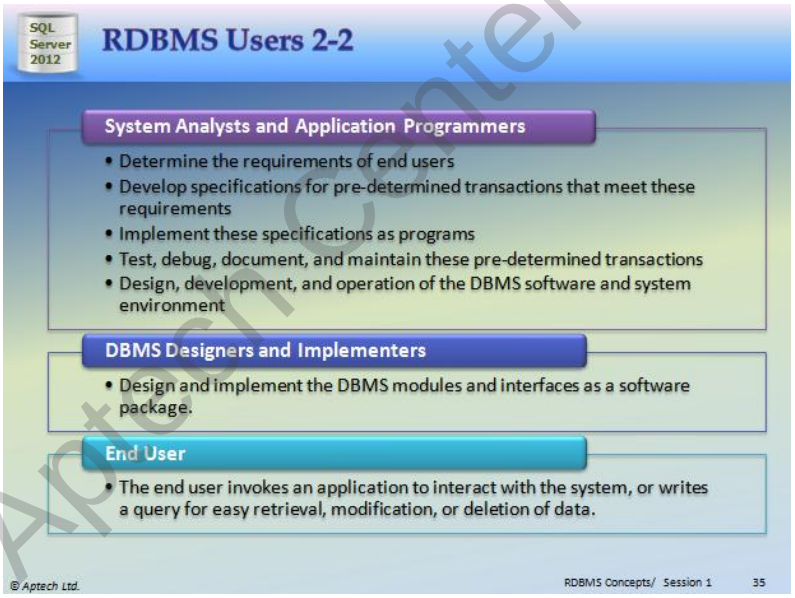
What is a foreign key?

Answer:
An attribute or a combination of attributes in one relation R1 that indicates the relationship of R1 with another relation R2 is termed as a foreign key.

RDBMS Users

Slides 34 and 35





Using slides 34 and 35, explain the RDBMS users.

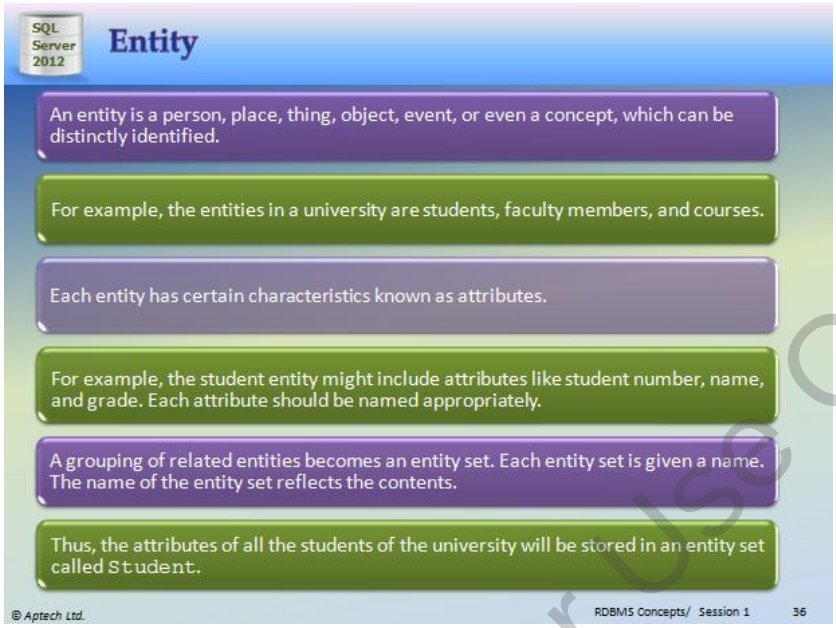
The primary goal of a database system is to provide an environment for retrieving information from and storing new information into the database.

For a small personal database, one person typically defines the constructs and manipulates the database. However, many persons are involved in the design, use, and maintenance of a large database with a few hundred users.

Explain in detail the different RDBMS users.

Entity

Slide 36



Using slide 36, explain the entity.

Mention components of an RDBMS are entities and tables, which will be explained in this section.

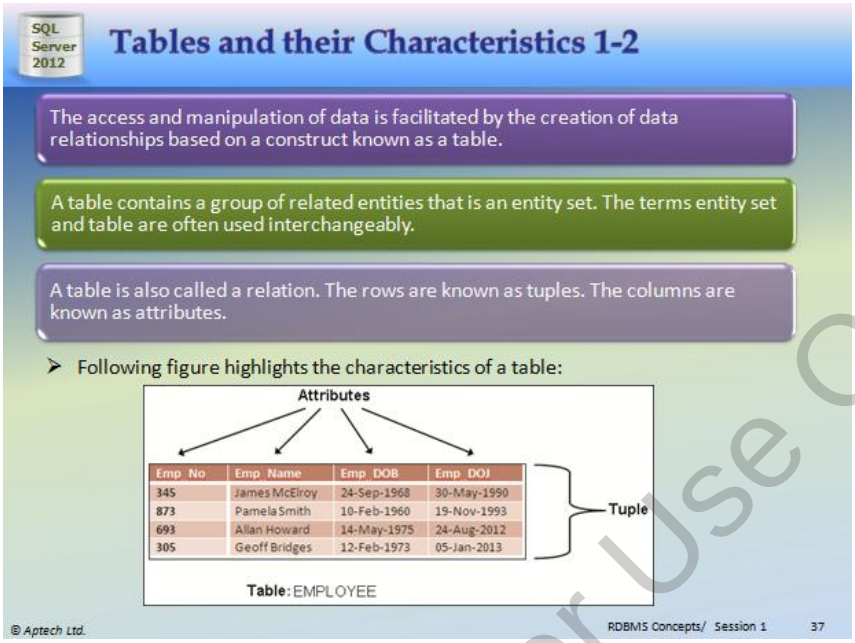
An entity is a person, place, thing, object, event, or even a concept, which can be distinctly identified. For example, the entities in a university are students, faculty members, and courses.

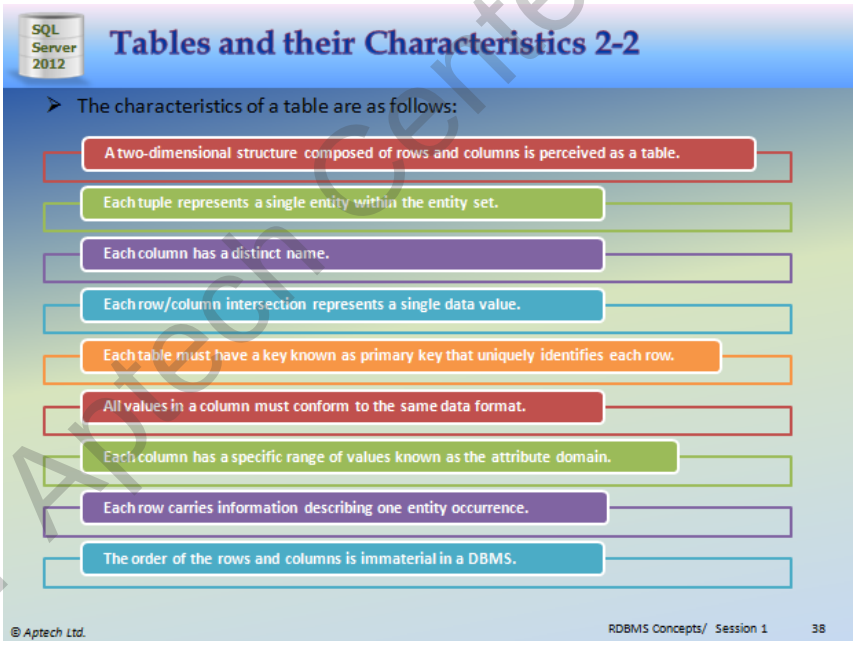
Each entity has certain characteristics known as attributes. For example, the student entity might include attributes such as student number, name, and grade. Each attribute should be named appropriately.

A grouping of related entities becomes an entity set. Each entity set is given a name. The name of the entity set reflects the contents. Thus, the attributes of all the students of the university will be stored in an entity set called Student.

Tables and their Characteristics

Slides 37 and 38



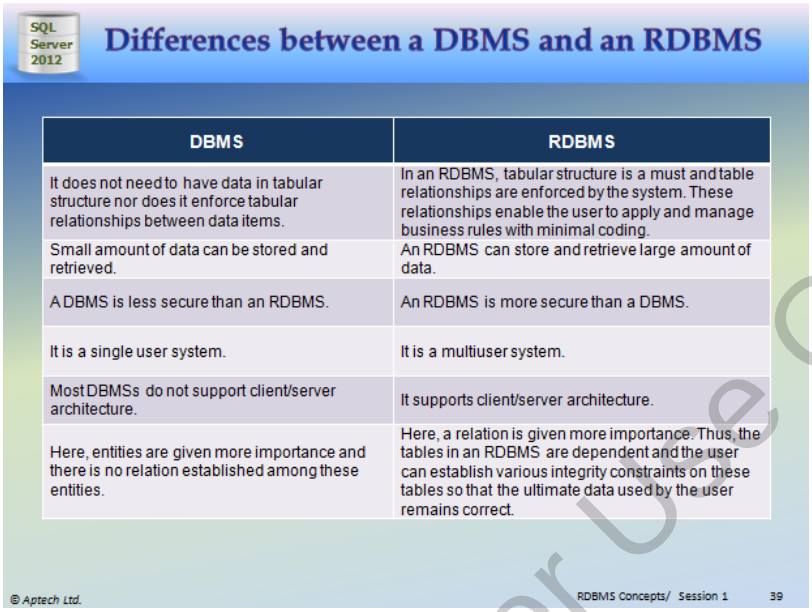


Using slides 37 and 38, explain the tables and their characteristics.

Mention the access and manipulation of data is facilitated by the creation of data relationships based on a construct known as a table. A table contains a group of related entities that is an entity set. The terms entity set and table are often used interchangeably. A table is also called a relation. The rows are known as tuples. The columns are known as attributes. Figure highlights the characteristics of a table. Explain the characteristics of the table.

Difference between a DBMS and an RDBMS

Slide 39

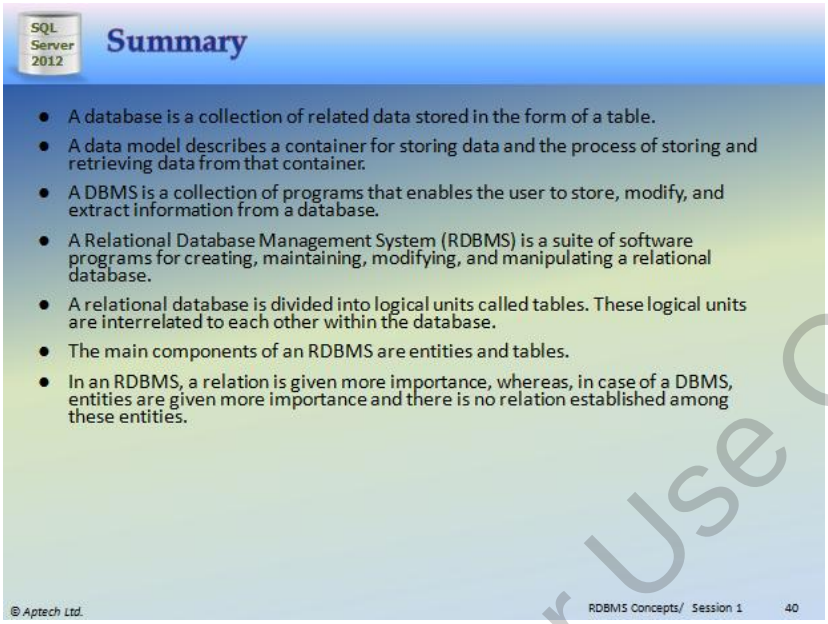


Using slide 39, explain the difference between a DBMS and an RDBMS in details.

Mention in an RDBMS, a relation is given more importance. Thus, the tables in an RDBMS are dependent and the user can establish various integrity constraints on these tables so that the ultimate data used by the user remains correct. In case of a DBMS, entities are given more importance and there is no relation established among these entities.

Summarize Session

Slide 40



Using slide 40, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- A database is a collection of related data stored in the form of a table.
- A data model describes a container for storing data and the process of storing and retrieving data from that container.
- A DBMS is a collection of programs that enables the user to store, modify, and extract information from a database.
- A Relational Database Management System (RDBMS) is a suite of software programs for creating, maintaining, modifying, and manipulating a relational database.
- A relational database is divided into logical units called tables. These logical units are interrelated to each other within the database.
- The main components of an RDBMS are entities and tables.
- In an RDBMS, a relation is given more importance, whereas, in case of a DBMS, entities are given more importance and there is no relation established among these entities.

1.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the E-R Model and Normalization that are offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 2 – Entity-Relationship (E-R) Model and Normalization

2.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

2.1.1 Objectives

By the end of this session, the learners will be able to:

- Define and describe data modeling
- Identify and describe the components of the E-R model
- Identify the relationships that can be formed between entities
- Explain E-R diagrams and their use
- Describe an E-R diagram, the symbols used for drawing, and show the various relationships
- Describe the various Normal Forms
- Outline the uses of different Relational Operators

2.1.2 Teaching Skills

To teach this session, you should be well-versed with concept about Data Modeling, the E-R model, its components, symbols, diagrams, and relationships. The session also covers the concept of Data Normalization, and Relational Operators.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slide and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

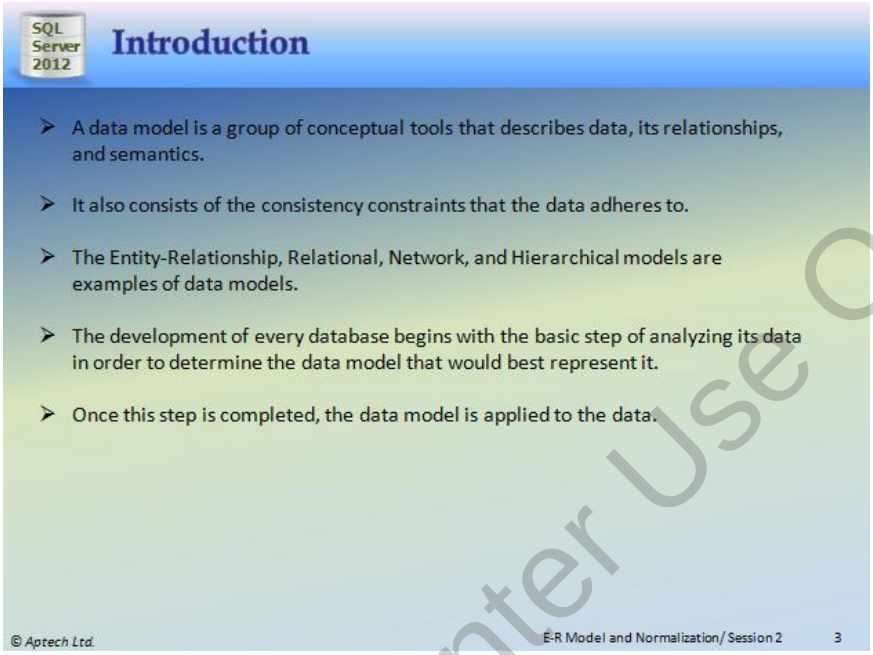
Overview of the Session:

Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces E-R model and normalization. They will learn about Data Modeling, the E-R model, its components, symbols, diagrams, and relationships. They will also know about Data Normalization, and Relational Operators.

2.2 In-Class Explanations

Introduction to Data Model

Slide 3

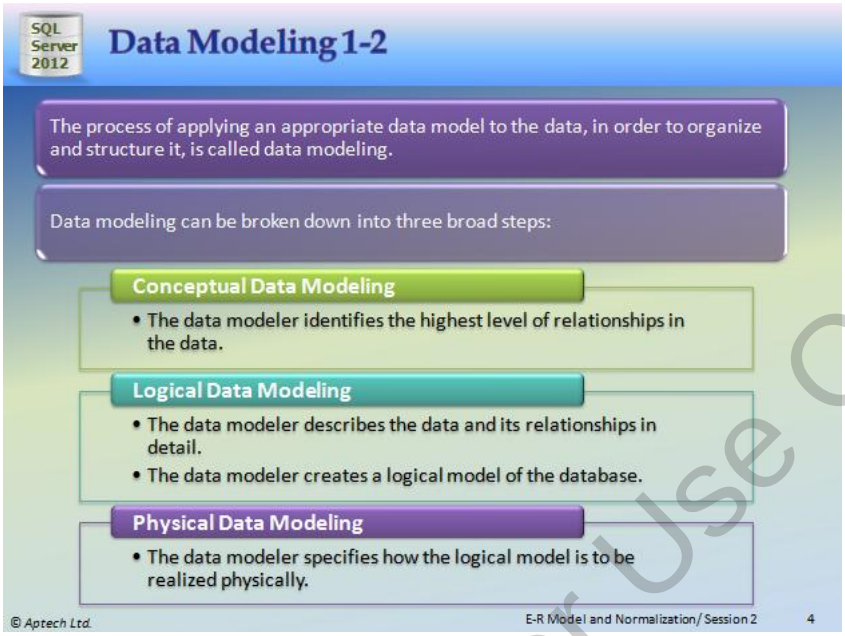


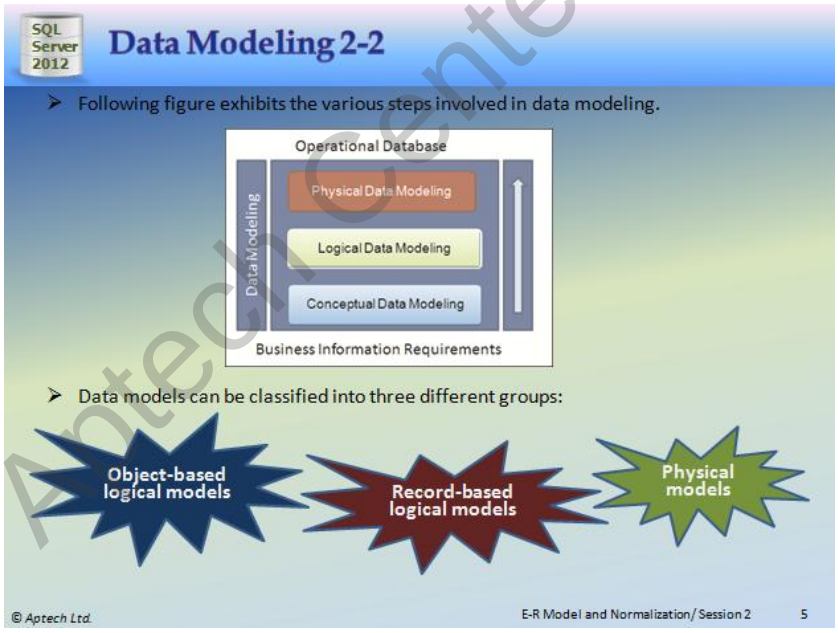
Using slide 3, explain the need of data model.

A data model is a group of conceptual tools that describes data, its relationships, and semantics. It also consists of the consistency constraints that the data adheres to. The Entity-Relationship, Relational, Network, and Hierarchical models are examples of data models. The development of every database begins with the basic step of analyzing its data in order to determine the data model that would best represent it. Once this step is completed, the data model is applied to the data.

Data Modeling

Slides 4 and 5





Using slides 4 and 5, explain the data modeling.

The process of applying an appropriate data model to the data, in order to organize and structure it, is called data modeling.

Data modeling is as essential to database development as are planning and designing to any project development. Building a database without a data model is similar to developing a project without its plans and design. Data models help database developers to define the

relational tables, primary and foreign keys, stored procedures, and triggers required in the database.

Data modeling can be broken down into the following three broad steps:

- Conceptual Data Modeling: The data modeler identifies the highest level of relationships in the data.
- Logical Data Modeling: The data modeler describes the data and its relationships in detail. The data modeler creates a logical model of the database.
- Physical Data Modeling: The data modeler specifies how the logical model is to be realized physically.

Figure exhibits the various steps involved in data modeling.

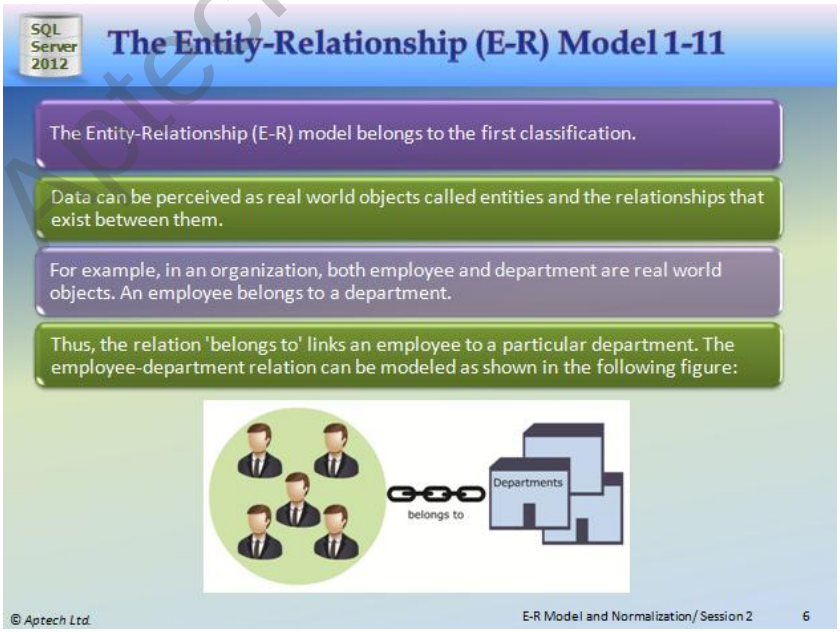
We should first start with the conceptual data model then move on to the logical data model and finally the physical data model. In a data warehousing project, sometimes the conceptual data model and the logical data model are considered as a single deliverable.

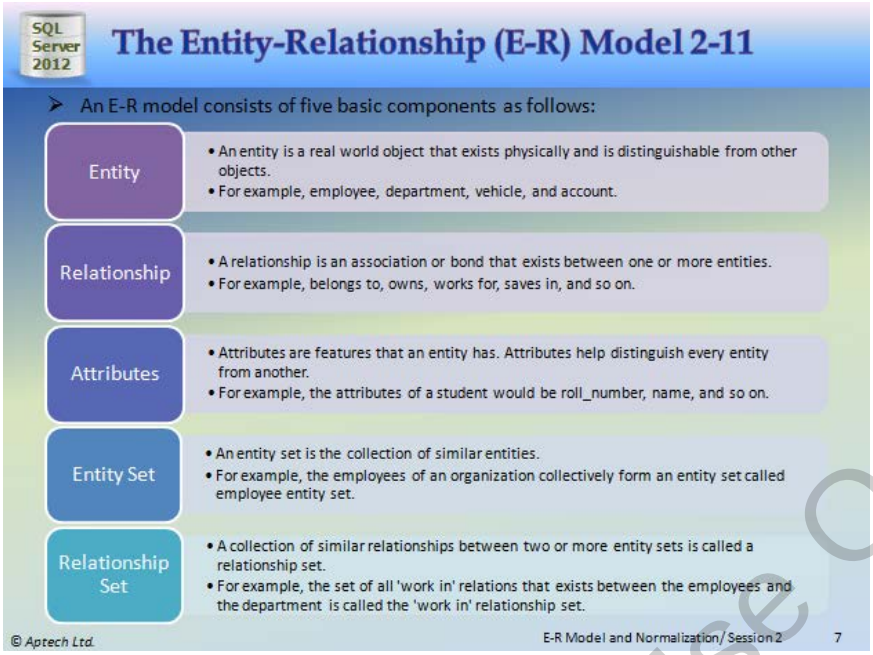
In-Class Question:

What is data modeling?

Answer:
The process of applying an appropriate data model to the data, in order to organize and structure it, is called data modeling.

Slides 6 to 16
Let us understand the E-R model.





Using slides 6 and 7, explain the E-R model.

Data models can be classified into three different groups:

- Object-based logical models
- Record-based logical models
- Physical models

Mention Entity-Relationship (E-R) model belongs to the first classification. The model is based on a simple idea. Data can be perceived as real-world objects called entities and the relationships that exist between them. For example, the data about employees working for an organization can be perceived as a collection of employees and a collection of the various departments that form the organization. Both employee and department are real-world objects. An employee belongs to a department. Thus, the relation 'belongs to' links an employee to a particular department.

The employee-department relation can be modeled as shown in figure in slide 7.

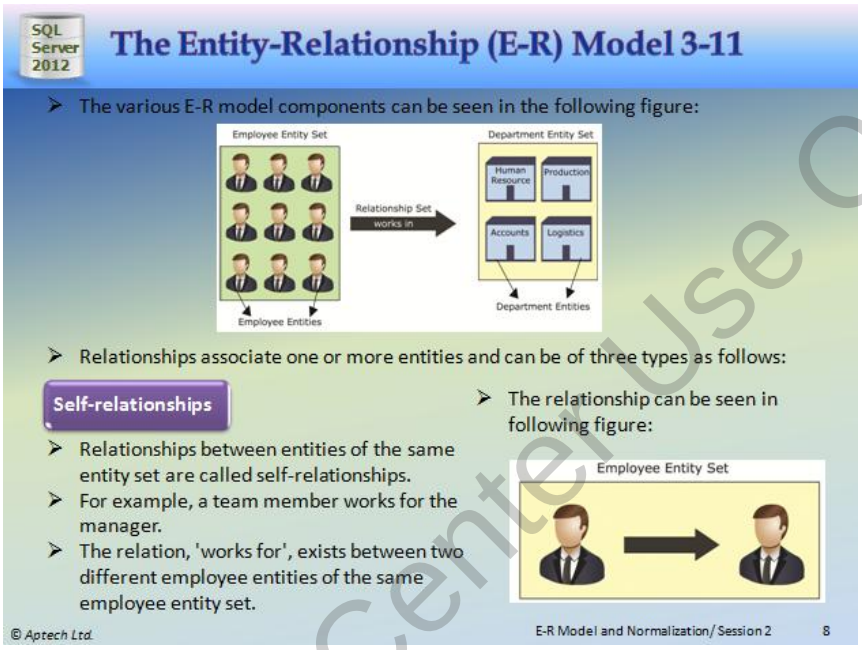
Using slide 7, explain the components of the E-R model.

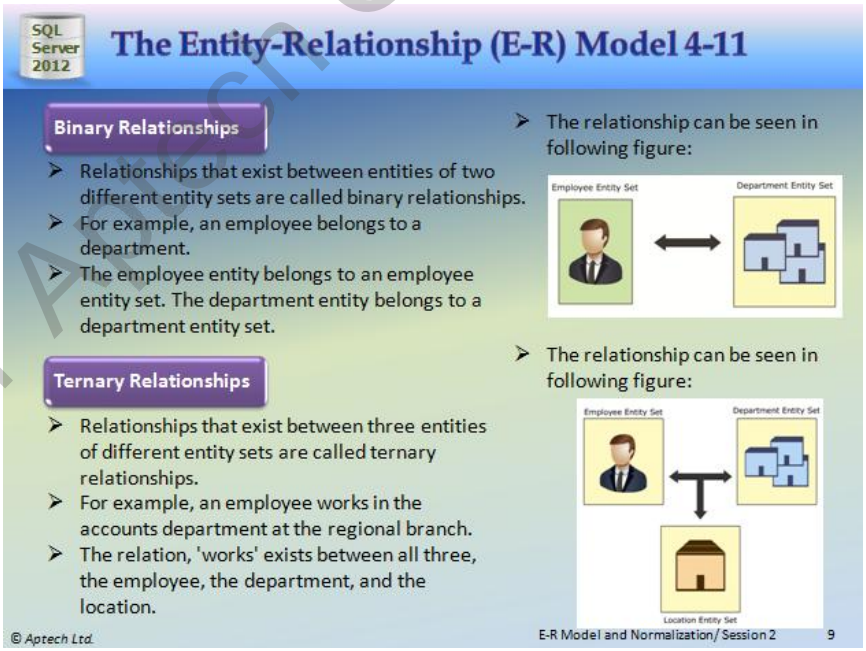
An E-R model consists of five basic components. They are as follows:

- **Entity**: An entity is a real-world object that exists physically and is distinguishable from other objects. For example, employee, department, student, customer, vehicle, and account are entities.
- **Relationship**: A relationship is an association or bond that exists between one or more entities. For example, belongs to, owns, works for, saves in, purchased, and so on.
- **Attributes**: Attributes are features that an entity has. Attributes help distinguish every entity from another. For example, the attributes of a student would be roll_number, name, stream, semester, and so on. The attributes of a car would be registration_number, model, manufacturer, color, price, owner, and so on.

- **Entity Set**: An entity set is the collection of similar entities. For example, the employees of an organization collectively form an entity set called employee entity set.
- **Relationship Set**: A collection of similar relationships between two or more entity sets is called a relationship set. For example, employees work in a particular department. The set of all 'work in' relations that exists between the employees and the department is called the 'work in' relationship set.

The various E-R model components can be seen in figure on slide 8.





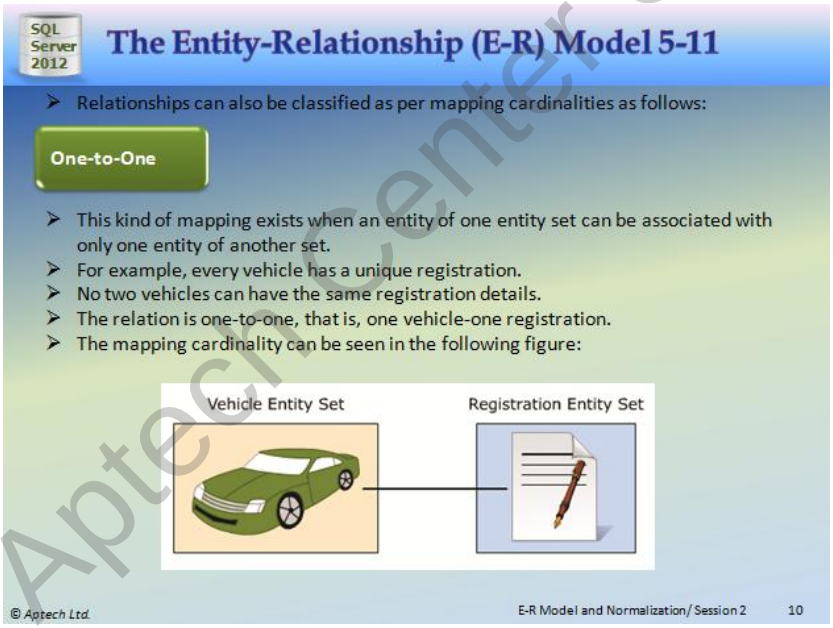
Using slides 8 and 9, explain the concept of relationship and its types.

Relationships associate one or more entities and can be of three types. They are as follows:

- **Self-relationships**: Relationships between entities of the same entity set are called self-relationships. For example, a manager and his team member, both belong to the employee entity set. The team member works for the manager. Thus, the relation, 'works for', exists between two different employee entities of the same employee entity set. The relationship can be seen in figure on slide 8.

- **Binary relationships**: Relationships that exist between entities of two different entity sets are called binary relationships. For example, an employee belongs to a department. The relation exists between two different entities, which belong to two different entity sets. The employee entity belongs to an employee entity set. The department entity belongs to a department entity set. The relationship can be seen in figure on slide 9.

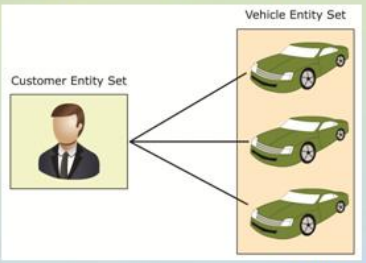
- **Ternary relationships**: Relationships that exist between three entities of different entity sets are called ternary relationships. For example, an employee works in the accounts department at the regional branch. The relation, 'works' exists between all three, the employee, the department, and the location. The relationship can be seen in figure on slide 9.



The Entity-Relationship (E-R) Model 6-11

One-to-Many

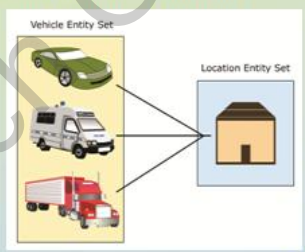
- This kind of mapping exists when an entity of one set can be associated with more than one entity of another entity set.
- For example, a customer can have more than one vehicle.
- Therefore, the mapping is a one to many mapping, that is, one customer - one or more vehicles.
- The mapping cardinality can be seen in the following figure:

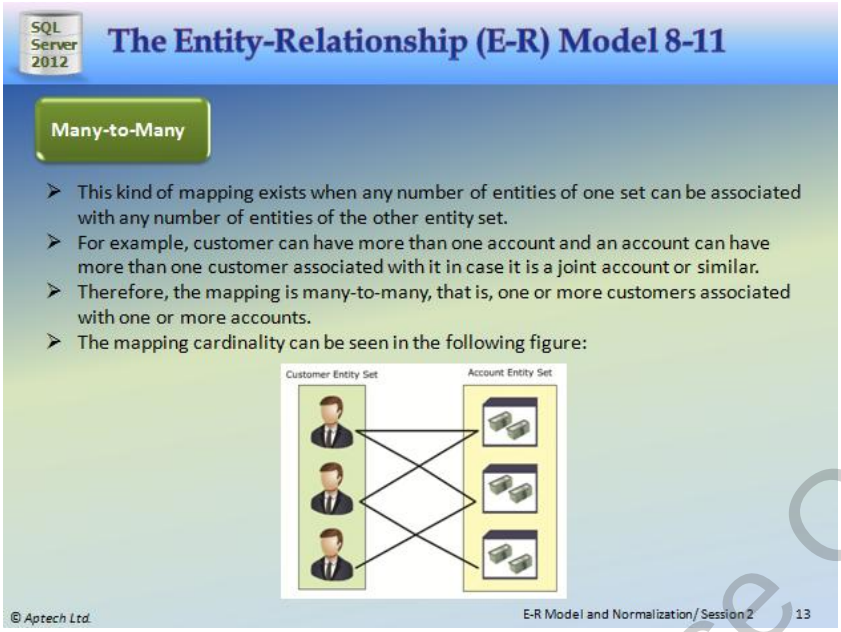


The Entity-Relationship (E-R) Model 7-11

Many-to-One

- This kind of mapping exists when many entities of one set is associated with an entity of another set.
- This association is done irrespective of whether the latter entity is already associated to other or more entities of the former entity set.
- For example, every vehicle has only one manufacturing company but the same company or coalition can manufacture more than one kind of vehicle.
- The mapping can be seen in the following figure:





Using slides 10 to 13, explain the cardinality and their mappings.

Mention cardinality or fundamental principle of one data table with respect to another is a critical aspect in data modeling. In order to explain how each table links together, relationship of one to the other table must be precise and exact between each other. Relationships can also be classified as per mapping cardinalities. The different mapping cardinalities are as follows:

- **One-to-One**: This kind of mapping exists when an entity of one entity set can be associated with only one entity of another set. Consider the relationship between a vehicle and its registration. Every vehicle has a unique registration. No two vehicles can have the same registration details. The relation is one-to-one, that is, one vehicle-one registration. The mapping cardinality can be seen in figure on slide 10.

- **One-to-Many**: This kind of mapping exists when an entity of one set can be associated with more than one entity of another entity set. Consider the relation between a customer and the customer's vehicles. A customer can have more than one vehicle. Therefore, the mapping is a one-to-many mapping, that is, one customer - one or more vehicles. The mapping cardinality can be seen in figure on slide 11.

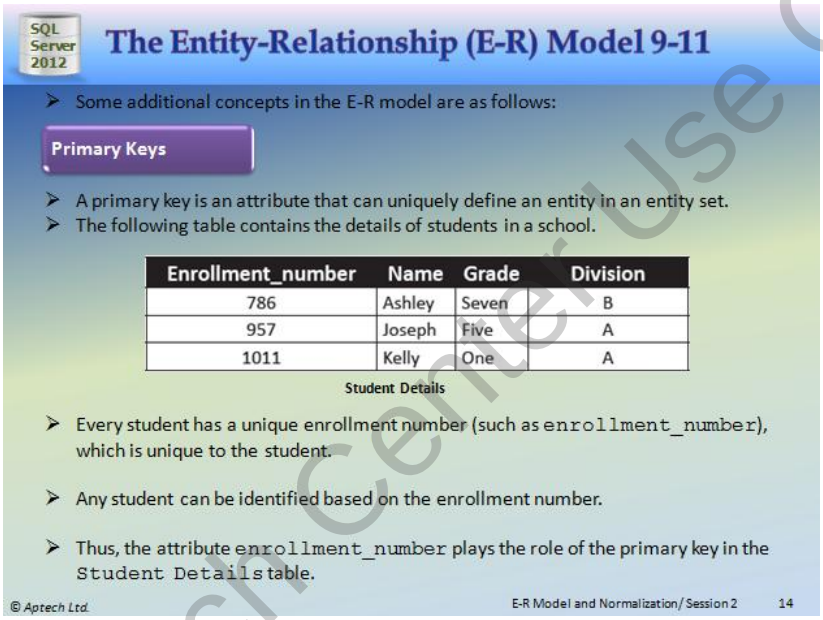
- **Many-to-One:** This kind of mapping exists when many entities of one set is associated with an entity of another set. This association is done irrespective of whether the latter entity is already associated to other or more entities of the former entity set. Consider the relation between a vehicle and its manufacturer. Every vehicle has only one manufacturing company or coalition associated to it under the relation, 'manufactured by', but the same company or coalition can manufacture more than one kind of vehicle. The mapping can be seen in figure on slide 12.

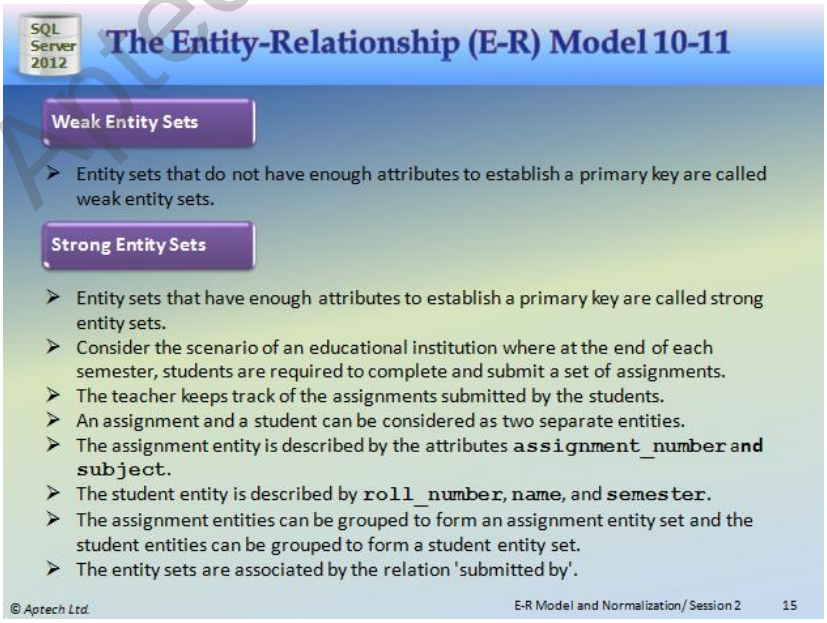
- **Many-to-Many**: This kind of mapping exists when any number of entities of one set can be associated with any number of entities of the other entity set. Consider the relation

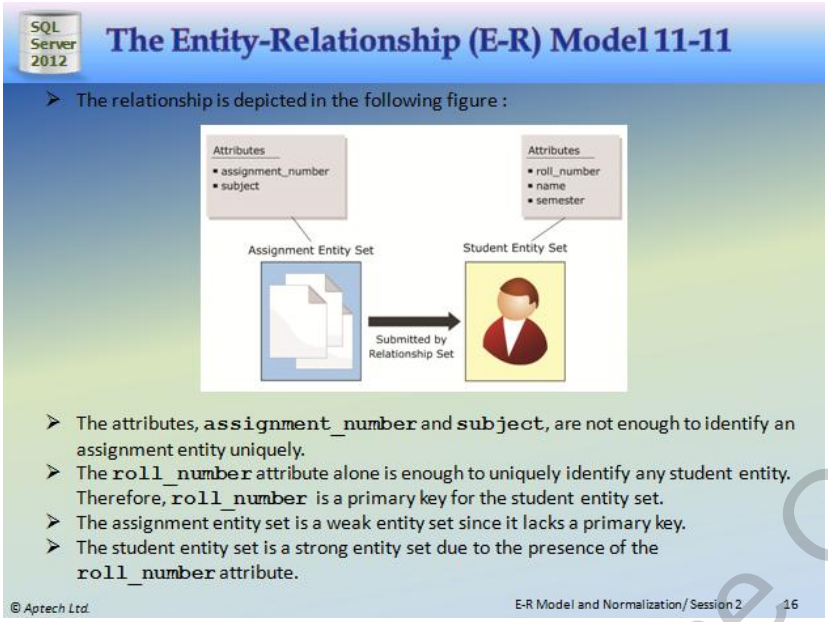
between a bank's customer and the customer's accounts. A customer can have more than one account and an account can have more than one customer associated with it in case it is a joint account or similar. Therefore, the mapping is many-to-many, that is, one or more customers associated with one or more accounts. The mapping cardinality can be seen in figure on slide 13.

Also, mention about participation constraints.

- **Partial participation:** Not all entities are involved in the relationship. Partial participation is represented by single line.
- **Total Participation:** Each entity in the entity is involved in the relationship. Total participation is represented by double lines.







Using slides 14 to 16, explain some additional concepts of E-R model.

Some additional concepts in the E-R model are as follows:

- **Primary keys**: A primary key is an attribute that can uniquely define an entity in an entity set. Consider table containing the details of students in a school.
  In a school, every student has a unique enrolment number (such as enrolment_number in table), which is unique to the student. Any student can be identified based on the enrolment number. Thus, the attribute enrolment_number plays the role of the primary key in the Student Details table.

- **Weak entity sets**: Entity sets that do not have enough attributes to establish a primary key are called weak entity sets.

  Mention that weak entity set depends on some other entities primary key.

- **Strong entity sets**: Entity sets that have enough attributes to establish a primary key are called strong entity sets.

  Consider the scenario of an educational institution where at the end of each semester, students are required to complete and submit a set of assignments. The teacher keeps track of the assignments submitted by the students. Now, an assignment and a student can be considered as two separate entities. The assignment entity is described by the attributes assignment_number and subject. The student entity is described by roll_number, name, and semester. The assignment entities can be grouped to form an assignment entity set and the student entities can be grouped to form a student entity set. The entity sets are associated by the relation 'submitted by'. This relation is depicted in figure on slide 15.

Explain the figure on slide 15. The attributes, assignment_number and subject, are not enough to identify an assignment entity uniquely. The roll_number attribute alone is enough to uniquely identify any student entity. Therefore, roll_number is a primary key for the student entity set. The assignment entity set is a weak entity set since it lacks a primary key. The student entity set is a strong entity set due to the presence of the roll_number attribute.

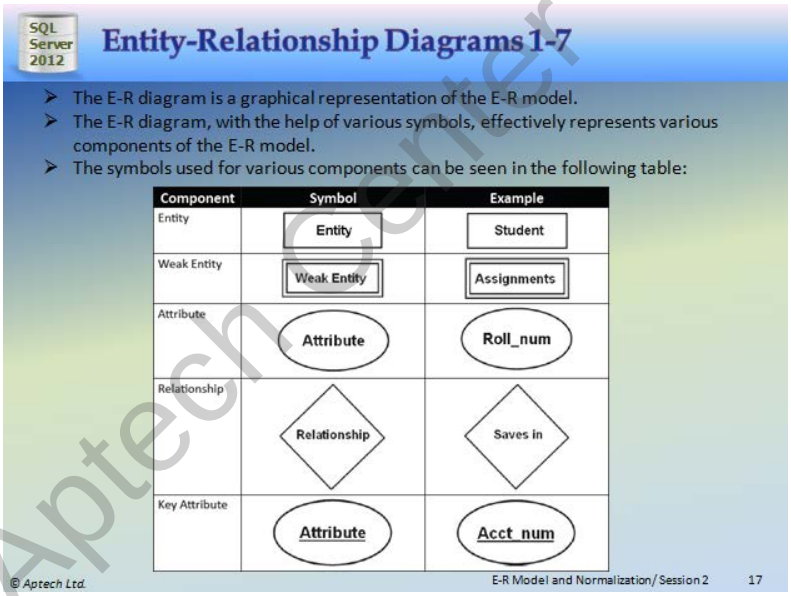
In-Class Question:

What is a weak entity?

Answer:
Entity sets that do not have enough attributes to establish a primary key are called weak entity sets.

Entity-Relationship Diagrams

Slides 17 to 23



Using slide 17, explain the E-R diagram.

The E-R diagram is a graphical representation of the E-R model. The E-R diagram, with the help of various symbols, effectively represents various components of the E-R model. Explain the symbols used for various components in E-R diagram using slide 17.

Entity-Relationship Diagrams 2-7

- Attributes in the E-R model can be further classified as:

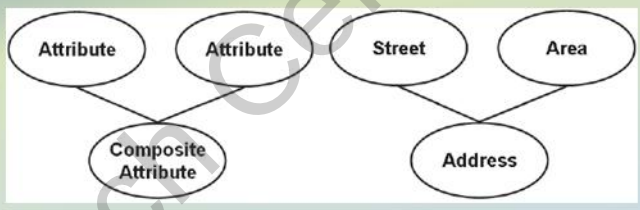
Multi-valued

- A multi-valued attribute is illustrated with a double-line ellipse, which has more than one value for at least one instance of its entity.
- This attribute may have upper and lower bounds specified for any individual entity value.
- The telephone attribute of an individual may have one or more values, that is, an individual can have one or more telephone numbers.
- Hence, the telephone attribute is a multi-valued attribute.
- The symbol and example of a multi-valued attribute can be seen in the following figure:

Entity-Relationship Diagrams 3-7

Composite

- A composite attribute may itself contain two or more attributes, which represent basic attributes having independent meanings of their own.
- The address attribute is usually a composite attribute, composed of attributes such as street, area, and so on.
- The symbol and example of a composite attribute can be seen in the following figure:



Entity-Relationship Diagrams 4-7

Derived

- Derived attributes are attributes whose value is entirely dependent on another attribute and are indicated by dashed ellipses.
- The age attribute of a person is the best example for derived attributes.
- For a particular person entity, the age of a person can be determined from the current date and the person's birth date.
- The symbol and example of a derived attribute can be seen in the following figure:

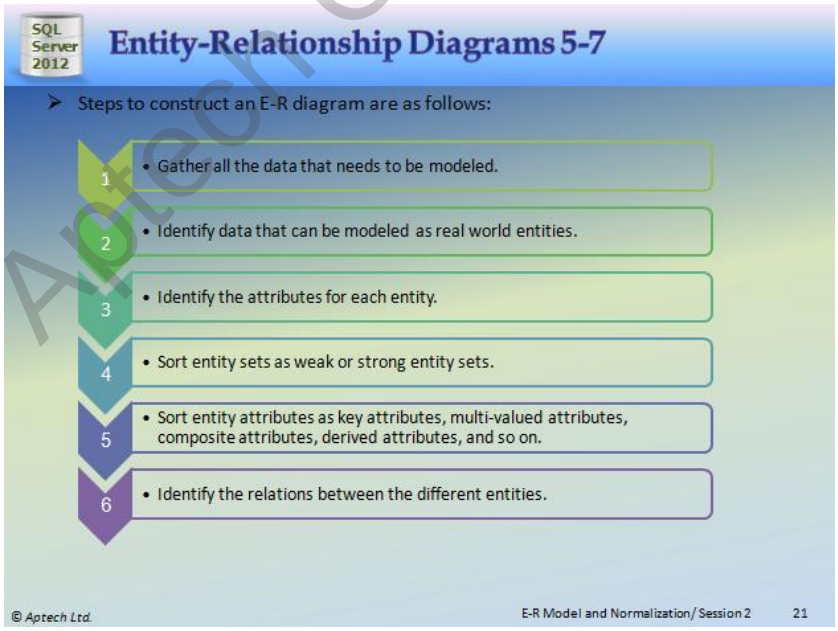
Using slides 18 to 20, explain the attributes in E-R diagram in detail.

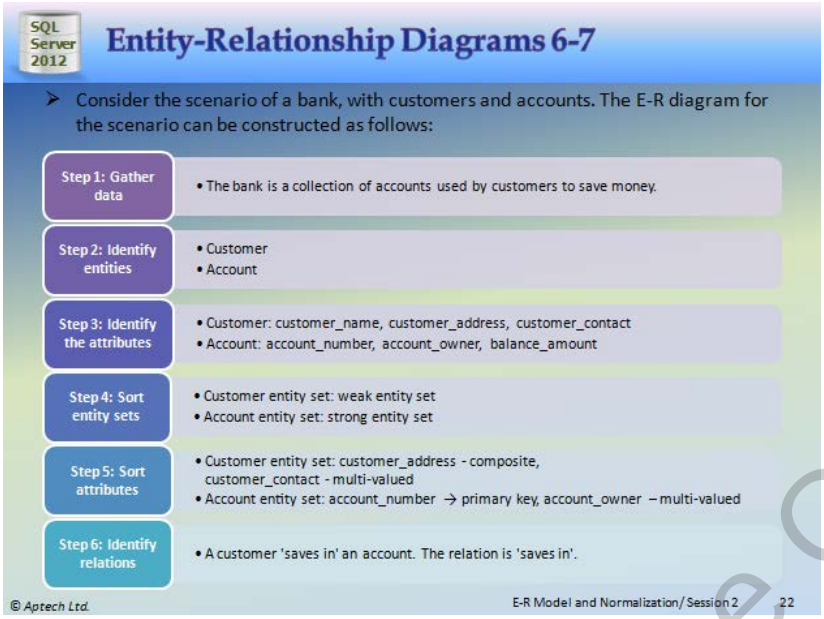
Attributes in the E-R model can be further classified as follows:

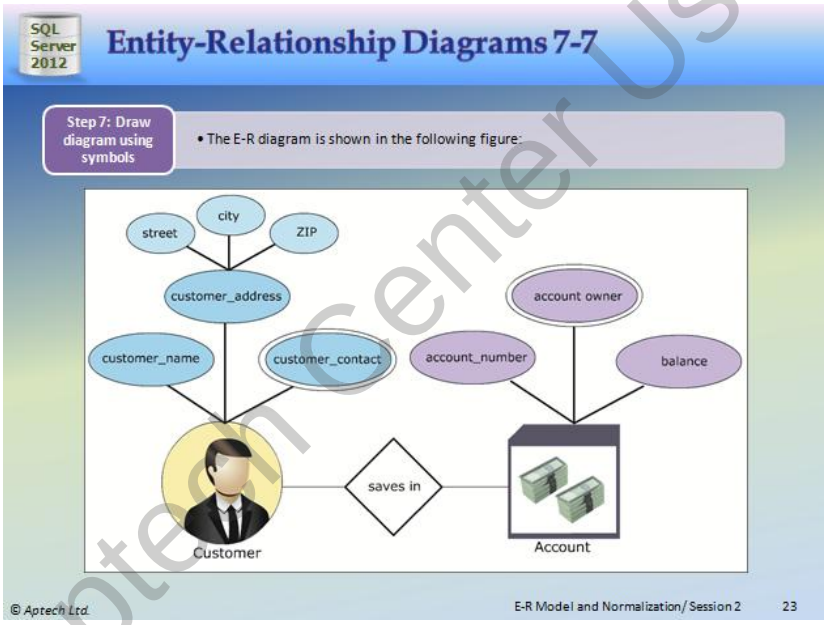
- **Multi-valued**: A multi-valued attribute is illustrated with a double-line ellipse, which has more than one value for at least one instance of its entity. This attribute may have upper and lower bounds specified for any individual entity value.
  The telephone attribute of an individual may have one or more values, that is, an individual can have one or more telephone numbers. Hence, the telephone attribute is a multi-valued attribute. The symbol and example of a multi-valued attribute can be seen in figure.
- **Composite**: A composite attribute may itself contain two or more attributes, which represent basic attributes having independent meanings of their own. The address attribute is usually a composite attribute, composed of attributes such as street, area, and so on. The symbol and example of a composite attribute can be seen in figure.
- **Derived**: Derived attributes are attributes whose value is entirely dependent on another attribute and are indicated by dashed ellipses. The age attribute of a person is the best example for derived attributes. For a particular person entity, the age of a person can be determined from the current date and the person's birth date. The symbol and example of a derived attribute can be seen in figure.

These attribute types can come together in a way such as:

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes







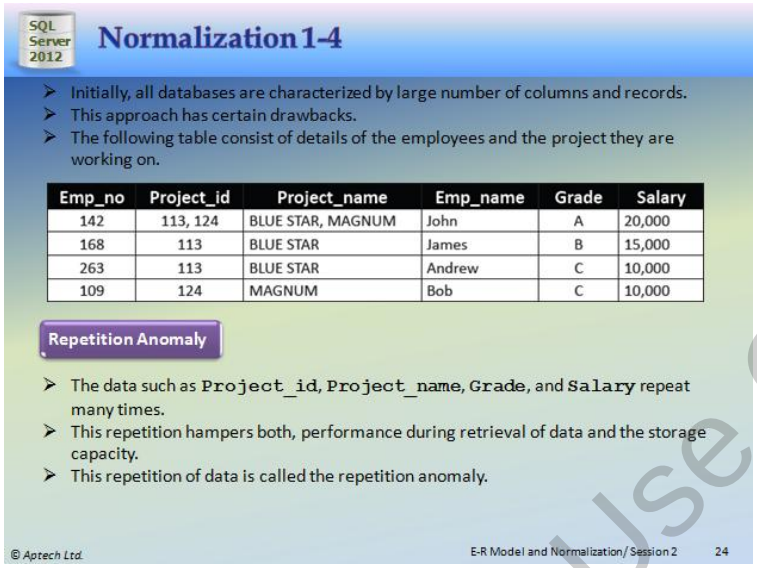
Using slide 21, explain the steps involved in construction of the E-R diagram.
Using different symbols draw the entities, their attributes, and their relationships. Use appropriate symbols while drawing attributes.

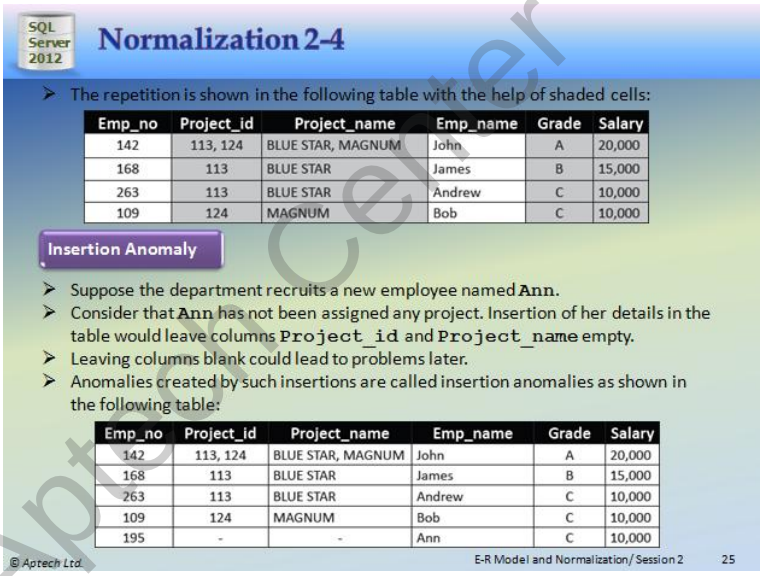
Explain the scenario of a bank, with customers and accounts and tell the student to identify the entities, relationships and attributes. Then using slide 22, explain the components in the scenario.

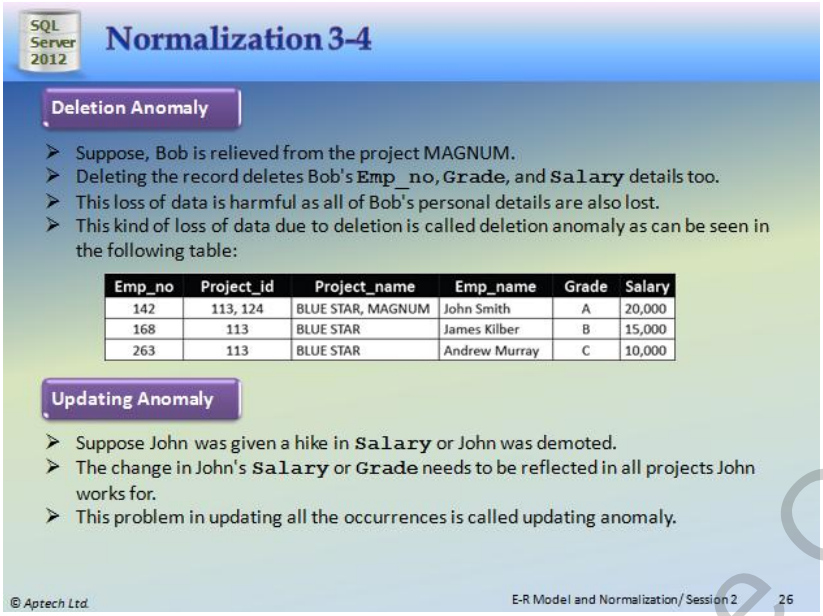
Let student draw the diagram first for the example then use slide 23 to explain the exact diagram.

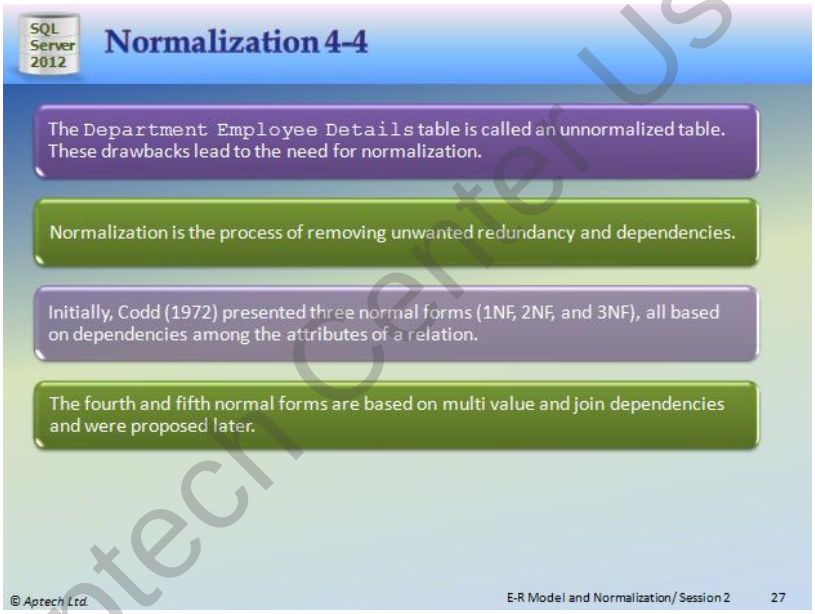
Normalization

Slides 24 to 27









Using slides 24 to 27, explain the concept of Normalization.

Mention that all databases are characterized by large number of columns and records. This approach has certain drawbacks. Consider the following details of the employees in a department. Table consists of the employee details as well as the details of the project they are working on.

For understanding normalization, explain the anomalies in the table.

The various anomalies which lead to normalization are as under:

- **Repetition anomaly**: The data such as Project_id, Project_name, Grade, and Salary repeat many times. This repetition hampers both, performance during retrieval of data and the storage capacity. This repetition of data is called the repetition anomaly. The repetition is shown in the table with the help of shaded cells.

- **Insertion anomaly**: Suppose the department recruits a new employee named Ann. Now, consider that Ann has not been assigned any project. Insertion of her details in the table would leave columns Project_id and Project_name empty. Leaving columns blank could lead to problems later. Anomalies created by such insertions are called insertion anomalies. The anomaly can be seen in table.

- **Deletion anomaly:** Suppose, Bob is relieved from the project MAGNUM. Deleting the record deletes Bob's Emp_no, Grade, and Salary details too. This loss of data is harmful as all of Bob's personal details are also lost as seen in the table. This kind of loss of data due to deletion is called deletion anomaly. The anomaly can be seen in table.

- **Updating anomaly**: Suppose John was given a hike in Salary or John was demoted. The change in John's Salary or Grade needs to be reflected in all projects John works for. This problem in updating all the occurrences is called updating anomaly.

The Department Employee Details table is called an unnormalized table. These drawbacks lead to the need for normalization.

Normalization is the process of removing unwanted redundancy and dependencies.
Initially, Codd (1972) presented three normal forms (1NF, 2NF, and 3NF), all based on dependencies among the attributes of a relation. The fourth and fifth normal forms are based on multi-value and join dependencies and were proposed later.

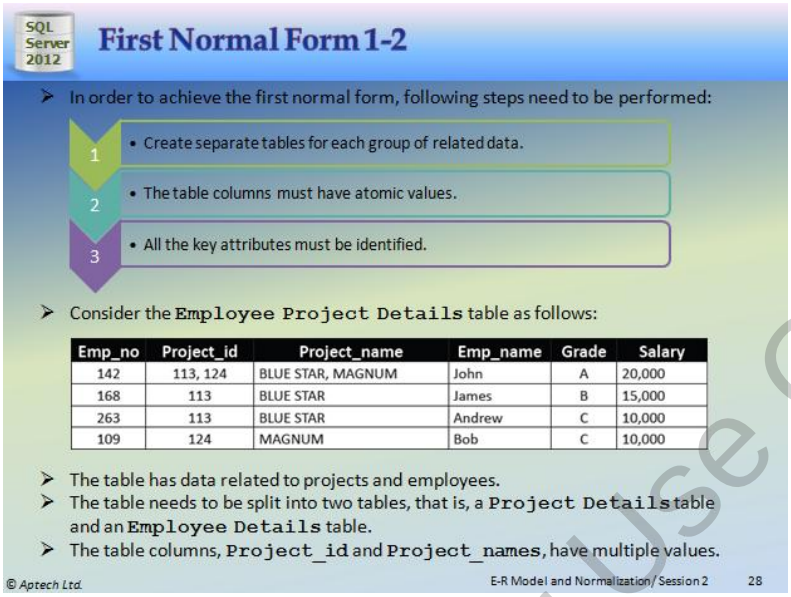
In-Class Question:

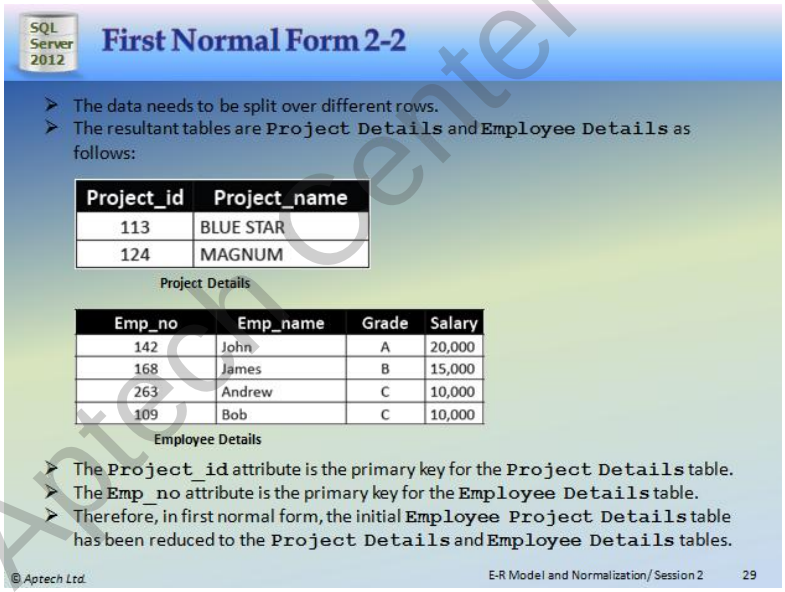
 What is normalization?

Answer:
Normalization is the process of removing unwanted redundancy and dependencies.

First Normal Form

Slides 28 and 29





Using slides 28 and 29, explain the First normal form.

In order to achieve the first normal form, following steps need to be performed:

- Create separate tables for each group of related data
- The table columns must have atomic values
- All the key attributes must be identified

Consider the Employee Project Details table as shown in slide 28.

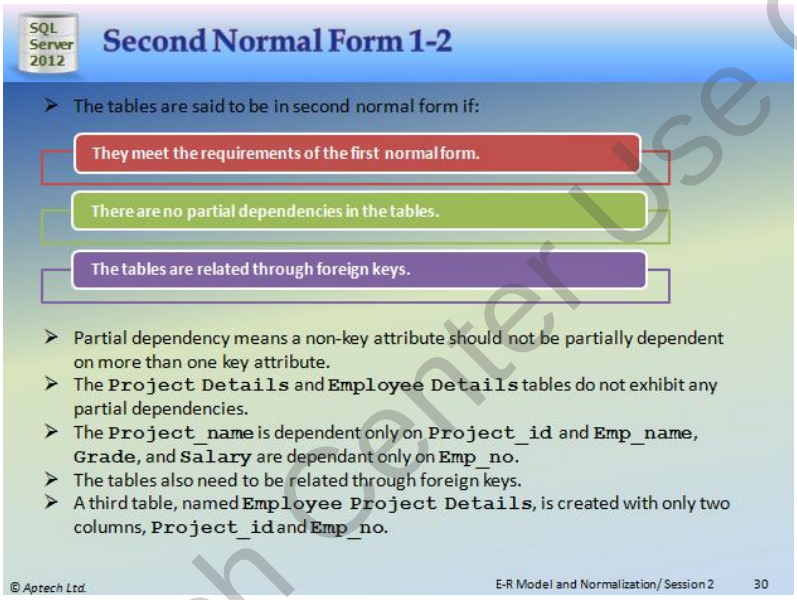
The table has data related to projects and employees. The table needs to be split into two tables, that is, a Project Details table and an Employee Details table. The table columns,

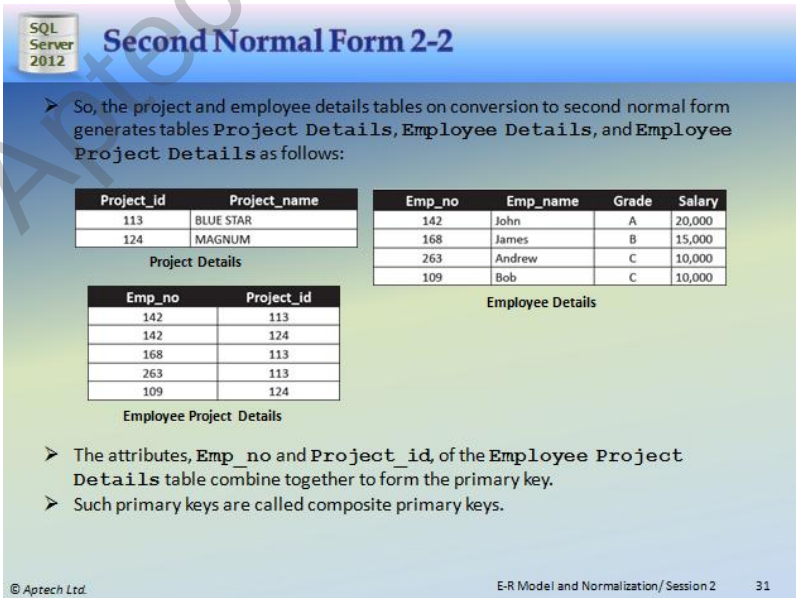
Project_id and Project_names, have multiple values. The data needs to be split over different rows. The resultant tables are Project Details and Employee Details as shown in tables in slide 29.

The Project_id attribute is the primary key for the Project Details table. The Emp_no attribute is the primary key for the Employee Details table. Therefore, in first normal form, the initial Employee Project Details table has been reduced to the Project Details and Employee Details tables.

Second Normal Form

Slides 30 and 31





Using slides 30 and 31, explain the second normal form.

The tables are said to be in second normal form if:

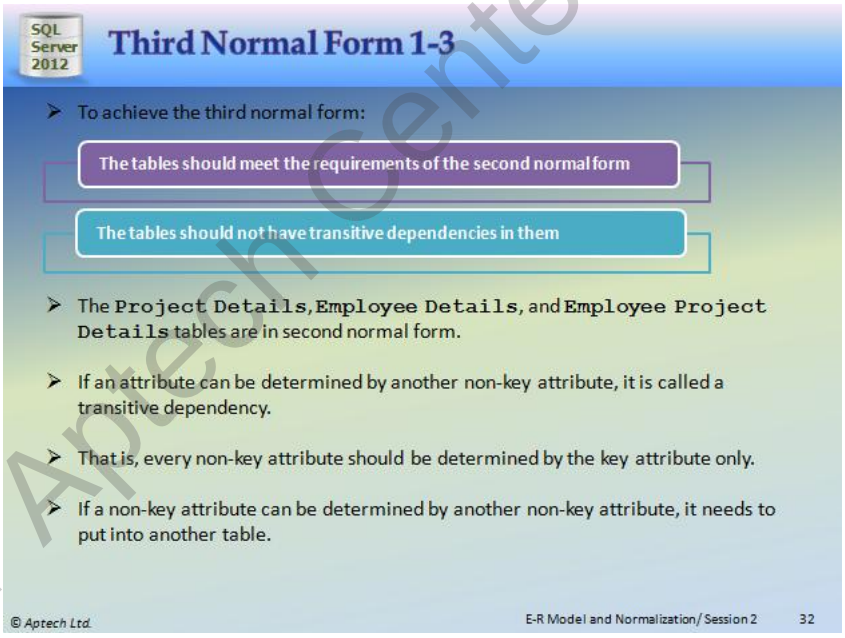
- They meet the requirements of the first normal form
- There are no partial dependencies in the tables
- The tables are related through foreign keys

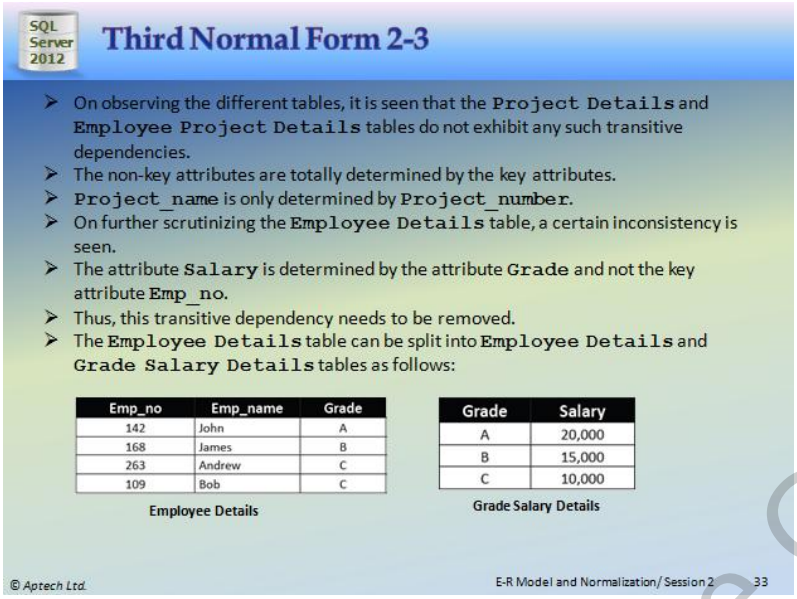
Partial dependency means a non-key attribute should not be partially dependent on more than one key attribute. The Project Details and Employee Details tables do not exhibit any partial dependencies. The Project_name is dependent only on Project_id and Emp_name, Grade, and Salary are dependent only on Emp_no. The tables also need to be related through foreign keys. A third table, named Employee Project Details, is created with only two columns, Project_id and Emp_no.

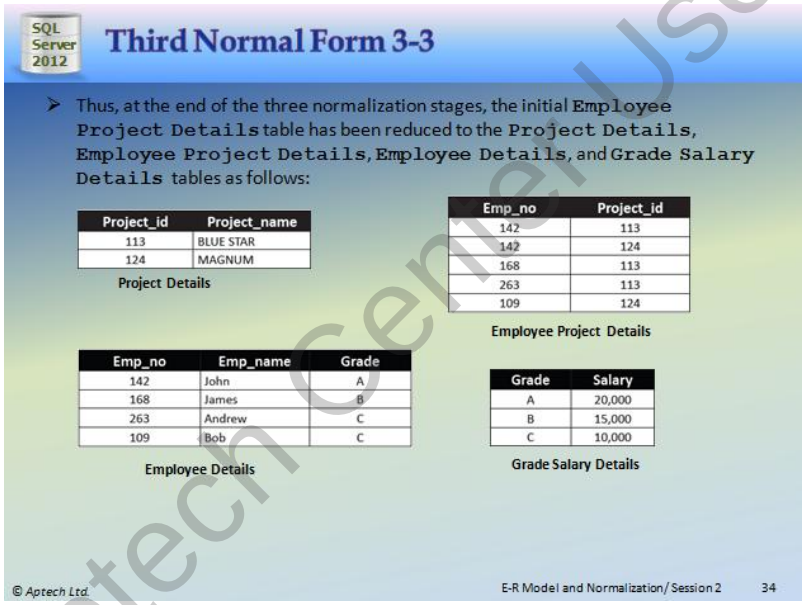
So, the project and employee details tables on conversion to second normal form generates tables Project Details, Employee Details, and Employee Project Details as shown in tables. The attributes, Emp_no and Project_id, of the Employee Project Details table combine together to form the primary key. Such primary keys are called composite primary keys.

Third Normal Form

Slides 32 to 34







Using slides 32 to 34, explain the third normal form.

To achieve the third normal form:

- The tables should meet the requirements of the second normal form
- The tables should not have transitive dependencies in them

The Project Details, Employee Details, and Employee Project Details tables are in second normal form. If an attribute can be determined by another non-key attribute, it is called a transitive dependency. To make it simpler, every non-key attribute should be determined by the key attribute only. If a non-key attribute can be determined by another non-key attribute, it needs to put into another table.

On observing the different tables, it is seen that the Project Details and Employee Project Details tables do not exhibit any such transitive dependencies. The non-key attributes are totally determined by the key attributes. Project_name is only determined by

Project_number. On further scrutinizing the Employee Details table, a certain inconsistency is seen. The attribute Salary is determined by the attribute Grade and not the key attribute Emp_no. Thus, this transitive dependency needs to be removed.

The Employee Details table can be split into the Employee Details and Grade Salary Details tables as shown in tables on slide 34.

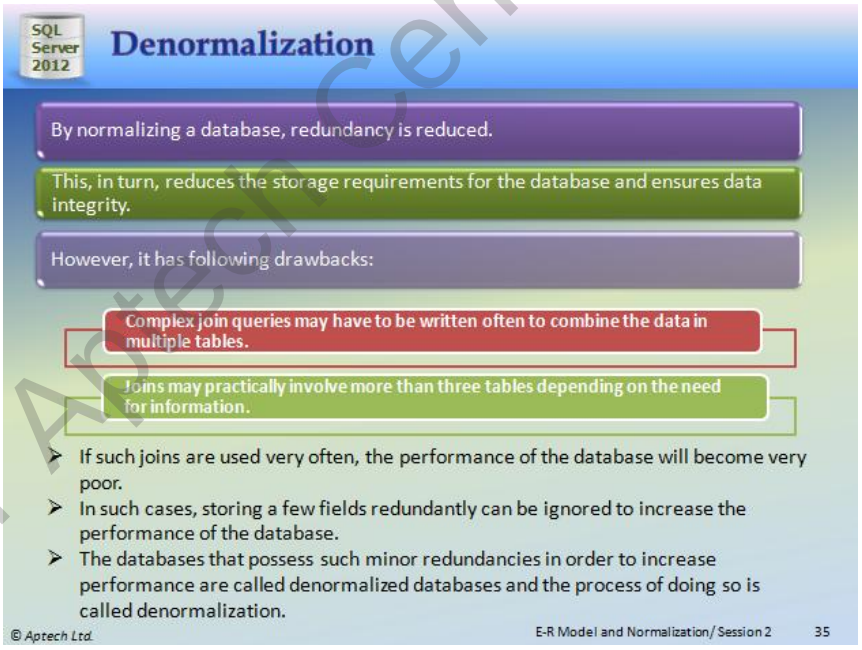
Thus, at the end of the three normalization stages, the initial Employee Project Details table has been reduced to the Project Details, Employee Project Details, Employee Details, and Grade Salary Details tables as shown in tables in slide 34.

Also mention, Boyce-Codd normal form.

A database table is in BCNF if and only if there are no non-trivial functional dependencies of attributes on anything other than a superset of a candidate key. At first glance it would seem that BCNF and 3NF is the same thing. However, in some rare cases it does happen that a 3NF table is not BCNF-compliant. This may happen in tables with two or more overlapping composite candidate keys.

Denormalization

Slide 35



Using slide 35, explain the denormalization.

By normalizing a database, redundancy is reduced. This, in turn, reduces the storage requirements for the database and ensures data integrity. However, it has some drawbacks. They are as follows:

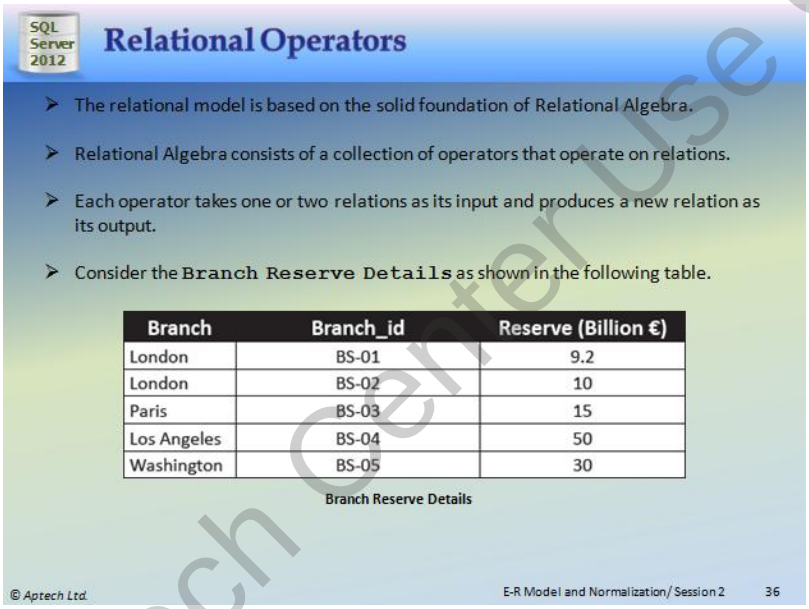
- Complex join queries may have to be written often to combine the data in multiple tables.

- Joins may practically involve more than three tables depending on the need for information.

Mention, if such joins are used very often, the performance of the database will become very poor. The CPU time required to solve such queries will be very large too. In such cases, storing a few fields redundantly can be ignored to increase the performance of the database. The databases that possess such minor redundancies in order to increase performance are called denormalized databases and the process of doing so is called denormalization.

Relational Operators

Slide 36



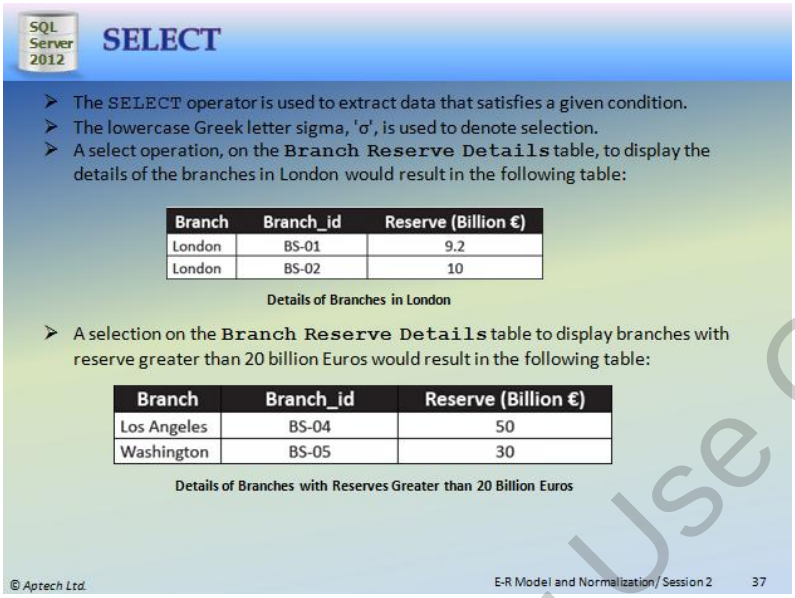
Using slide 36, explain the relational operators.

The relational model is based on the solid foundation of Relational Algebra. Relational Algebra consists of a collection of operators that operate on relations. Each operator takes one or two relations as its input and produces a new relation as its output.

Consider the Branch Reserve Details table as shown in table.

SELECT Operator

Slide 37



Using slide 37, explain the SELECT operator.

The SELECT operator is used to extract data that satisfies a given condition. The lowercase Greek letter sigma, 'σ', is used to denote selection. A select operation, on the Branch Reserve Details table, to display the details of the branches in London would result in table. A selection on the Branch Reserve Details table to display branches with reserve greater than 20 billion Euros would result in table.

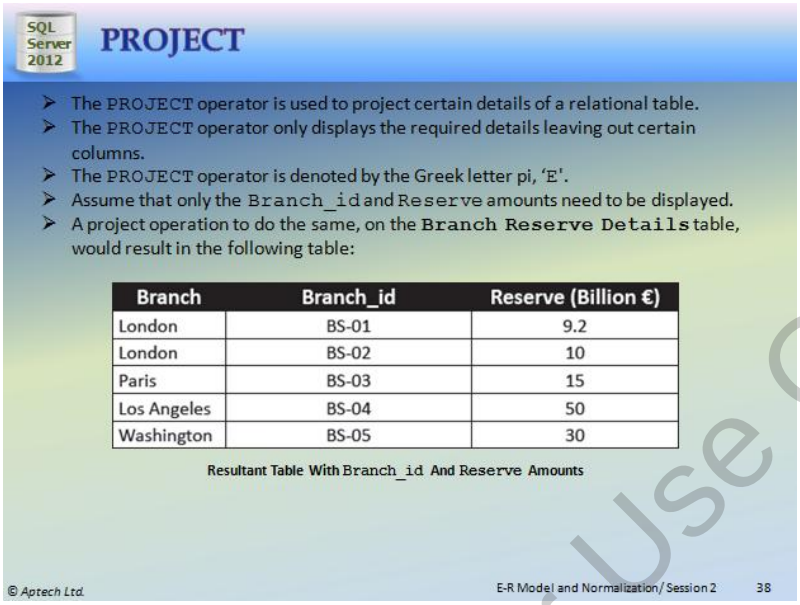
In-Class Question:

What is the use of SELECT operator?

Answer:
The SELECT operator is used to extract data that satisfies a given condition.

PROJECT Operator

Slide 38

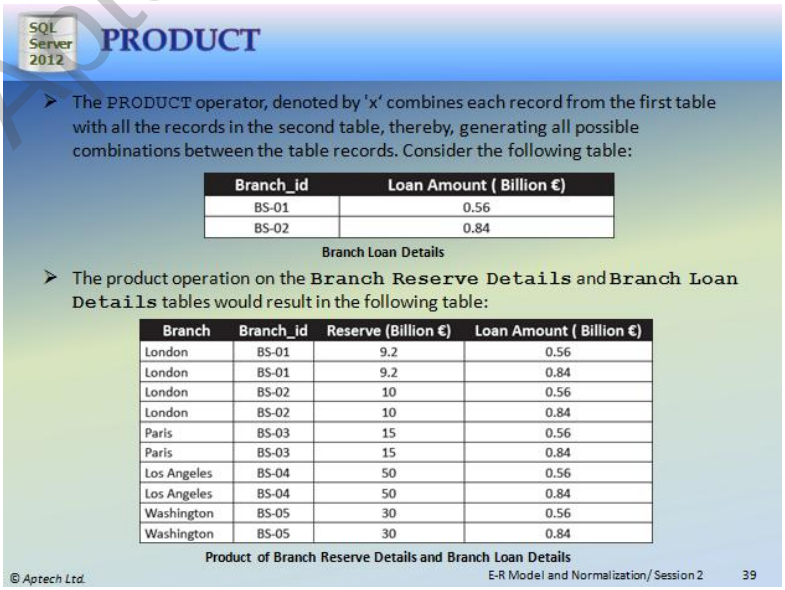


Using slide 38, explain the PROJECT operator.

The PROJECT operator is used to project certain details of a relational table. The PROJECT operator only displays the required details leaving out certain columns. The PROJECT operator is denoted by the Greek letter pi, 'ε'. Assume that only the Branch_id and Reserve amounts need to be displayed. A project operation to do the same, on the Branch Reserve Details table, would result in table.

PRODUCT Operator

Slide 39



Using slide 39, explain the PRODUCT operator.

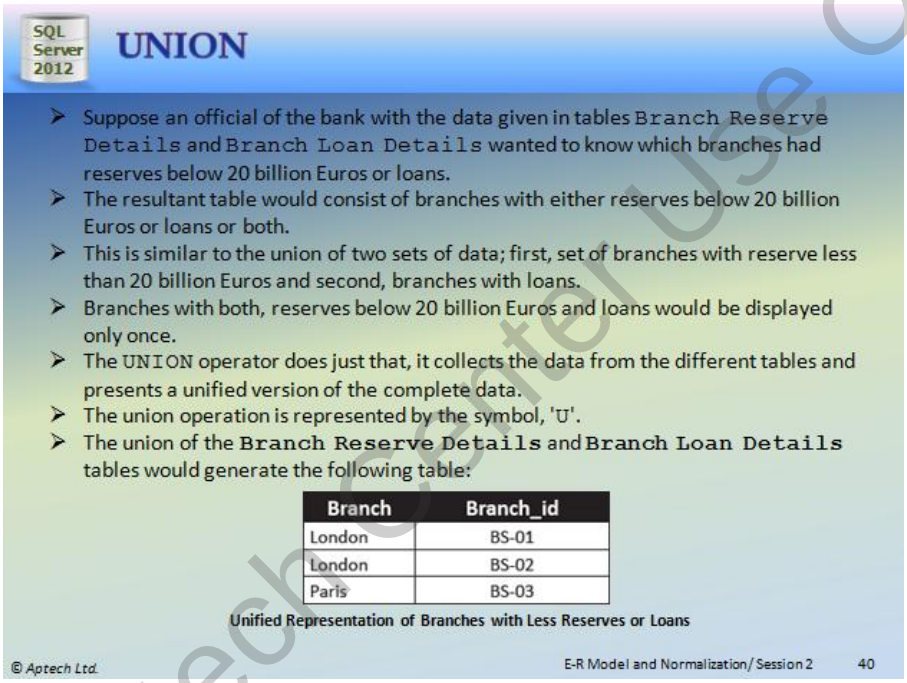
The PRODUCT operator, denoted by 'x' helps combine information from two relational tables.

The product operation on the Branch Reserve Details and Branch Loan Details tables would result in table.

The product operation combines each record from the first table with all the records in the second table, somewhat generating all possible combinations between the table records.

UNION Operator

Slide 40



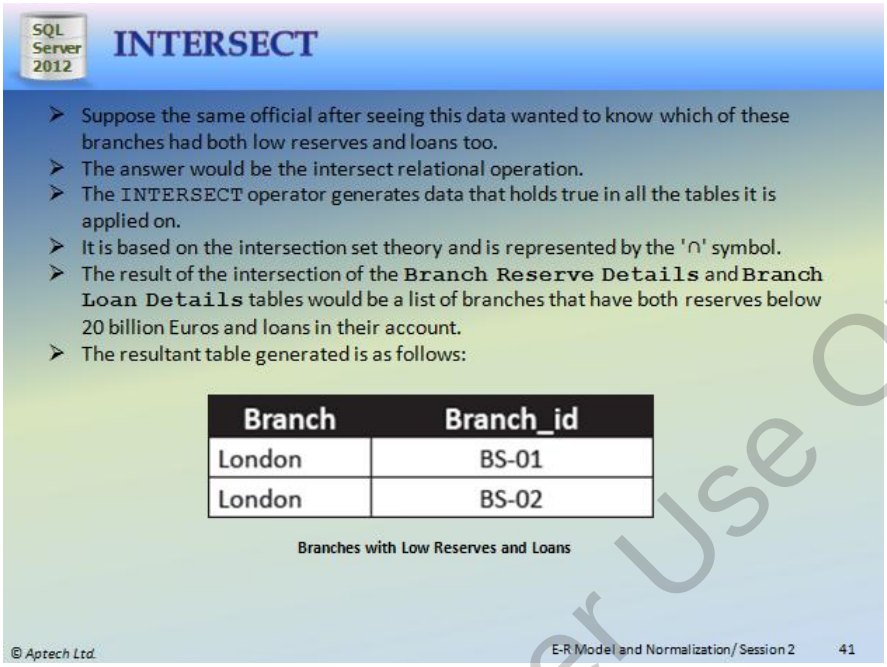
Using slide 40, explain the UNION operator.

Suppose an official of the bank with the data given in tables wanted to know which branches had reserves below 20 billion Euros or loans. The resultant table would consist of branches with either reserves below 20 billion Euros or loans or both.

This is similar to the union of two sets of data; first, set of branches with reserve less than 20 billion Euros and second, branches with loans. Branches with both, reserves below 20 billion Euros and loans would be displayed only once. The UNION operator does just that, it collects the data from the different tables and presents a unified version of the complete data. The union operation is represented by the symbol, 'U'. The union of the Branch Reserve Details and Branch Loan Details tables would generate table.

INTERSECT Operator

Slide 41

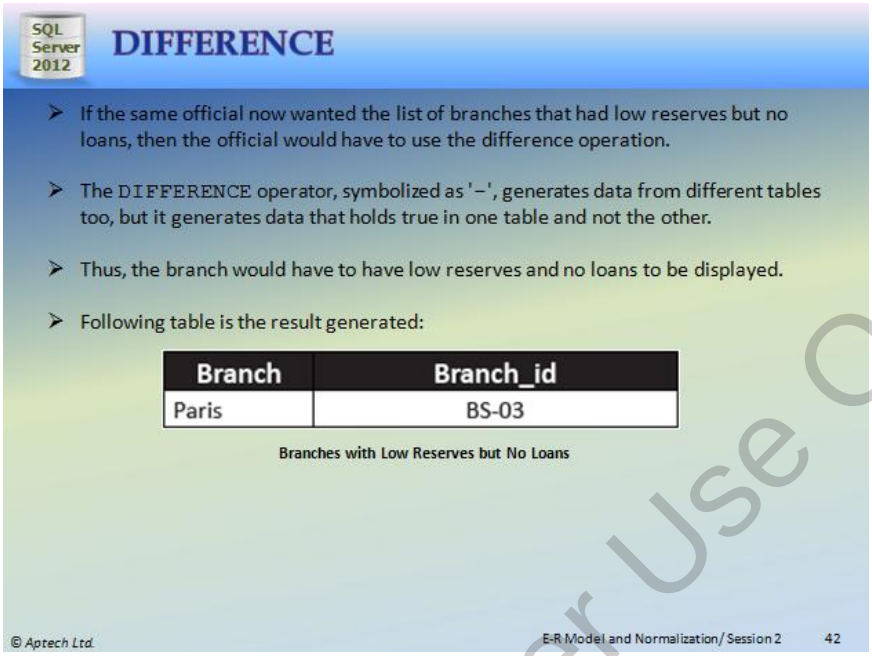


Using slide 41, explain the INTERSECT operator.

Suppose the same official after seeing this data wanted to know which of these branches had both low reserves and loans too. The answer would be the intersect relational operation. The INTERSECT operator generates data that holds true in all the tables it is applied on. It is based on the intersection set theory and is represented by the '∩' symbol. The result of the intersection of the Branch Reserve Details and Branch Loan Details tables would be a list of branches that have both reserves below 20 billion Euros and loans in their account. The resultant table generated is shown on slide 41.

DIFFERENCE Operator

Slide 42

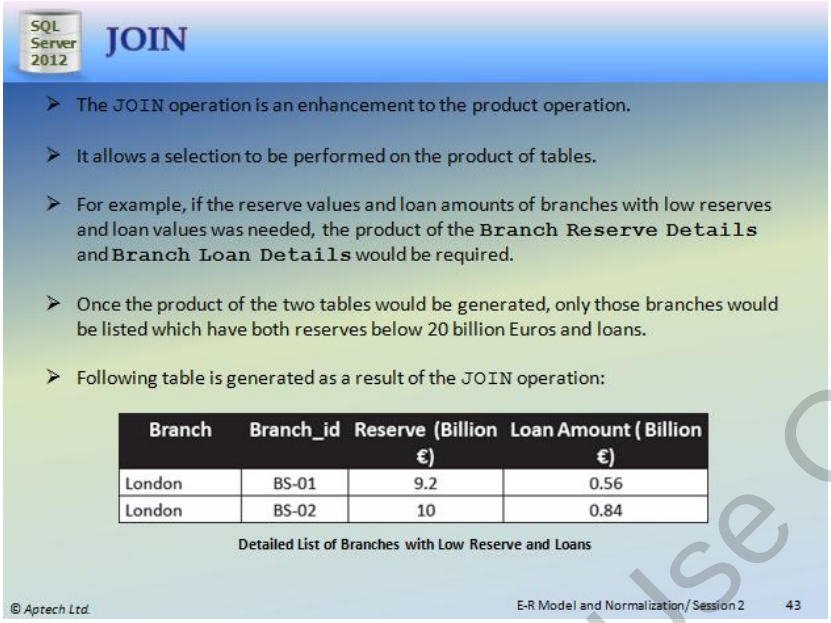


Using slide 42, explain the DIFFERENCE operator.

If the same official now wanted the list of branches that had low reserves but no loans, then the official would have to use the difference operation. The DIFFERENCE operator, symbolized as '-', generates data from different tables too, but it generates data that holds true in one table and not the other. Thus, the branch would have to have low reserves and no loans to be displayed. Table given in the slide is the result generated.

JOIN Operator

Slide 43

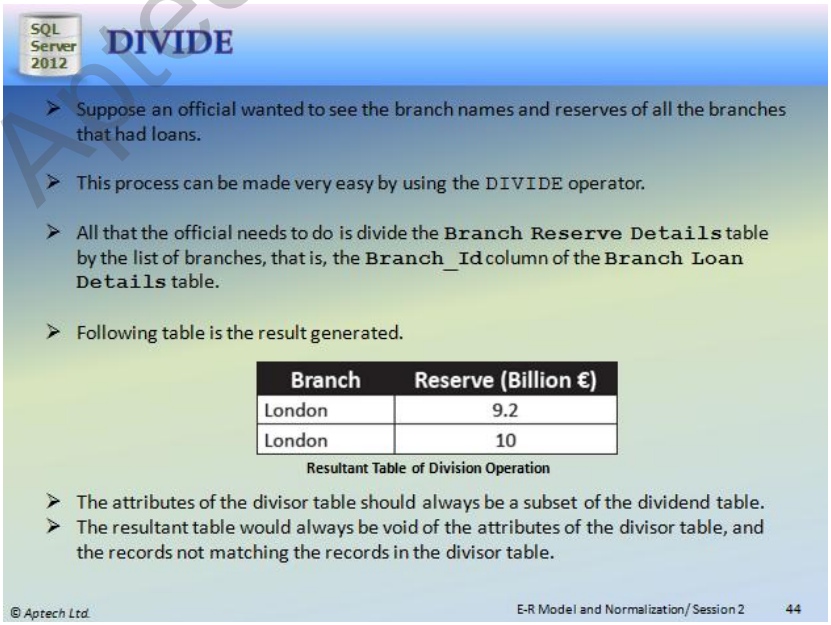


Using slide 43, explain the join operator.

The JOIN operation is an enhancement to the product operation. It allows a selection to be performed on the product of tables. For example, if the reserve values and loan amounts of branches with low reserves and loan values was needed, the product of the Branch Reserve Details and Branch Loan Details would be required. Once the product of tables would be generated, only those branches would be listed which have both reserves below 20 billion Euros and loans. Table is generated as a result of the JOIN operation.

DIVIDE Operator

Slide 44



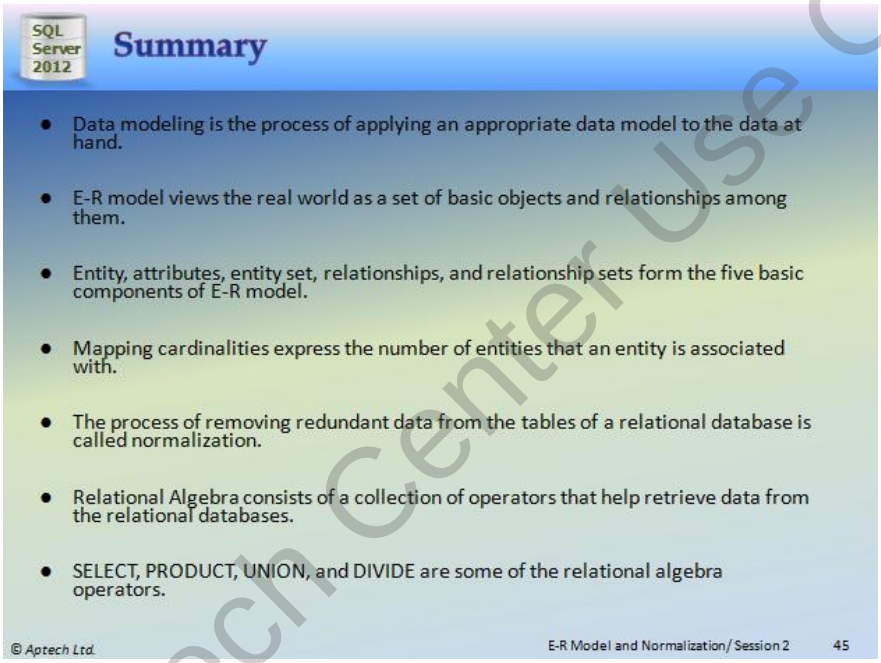
Using slide 44, explain the divide operator.

Suppose an official wanted to see the branch names and reserves of all the branches that had loans. This process can be made very easy by using the DIVIDE operator. All that the official needs to do is divide the Branch Reserve Details table by the list of branches, that is, the Branch Id column of the Branch Loan Details table. Table is the result generated.

Note that the attributes of the divisor table should always be a subset of the dividend table. The resultant table would always be void of the attributes of the divisor table and the records not matching in the divisor table.

Summarize Session

Slide 45



Using slide 45, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- Data modeling is the process of applying an appropriate data model to the data at hand.
- E-R model views the real world as a set of basic objects and relationships among them.
- Entity, attributes, entity set, relationships, and relationship sets form the five basic components of E-R model.
- Mapping cardinalities express the number of entities that an entity is associated with.
- The process of removing redundant data from the tables of a relational database is called normalization.
- Relational Algebra consists of a collection of operators that help retrieve data from the relational databases.

- SELECT, PRODUCT, UNION, and DIVIDE are some of the relational algebra operators.

2.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Introduction to SQL Server 2012 that is offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 3 – Introduction to SQL Server 2012

3.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

3.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe the basic architecture of SQL Server 2012
- List the various versions and editions of SQL Server
- Explain the role and structure of SQL Server databases
- List the new features of SQL Server 2012
- List the process of connecting to SQL Server Instances
- Explain script file creation and organization
- Explain the process to execute Transact-SQL queries

3.1.2 Teaching Skills

To teach this session, you should be well-versed with the basic architecture of SQL Server 2012 and lists the versions and editions of SQL Server. Also, aware yourself with the role and structure of SQL Server along with the new features added in SQL Server 2012.
The session also covers the process to connect to SQL Server instances, create and organize script files, and execute Transact-SQL queries.
You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

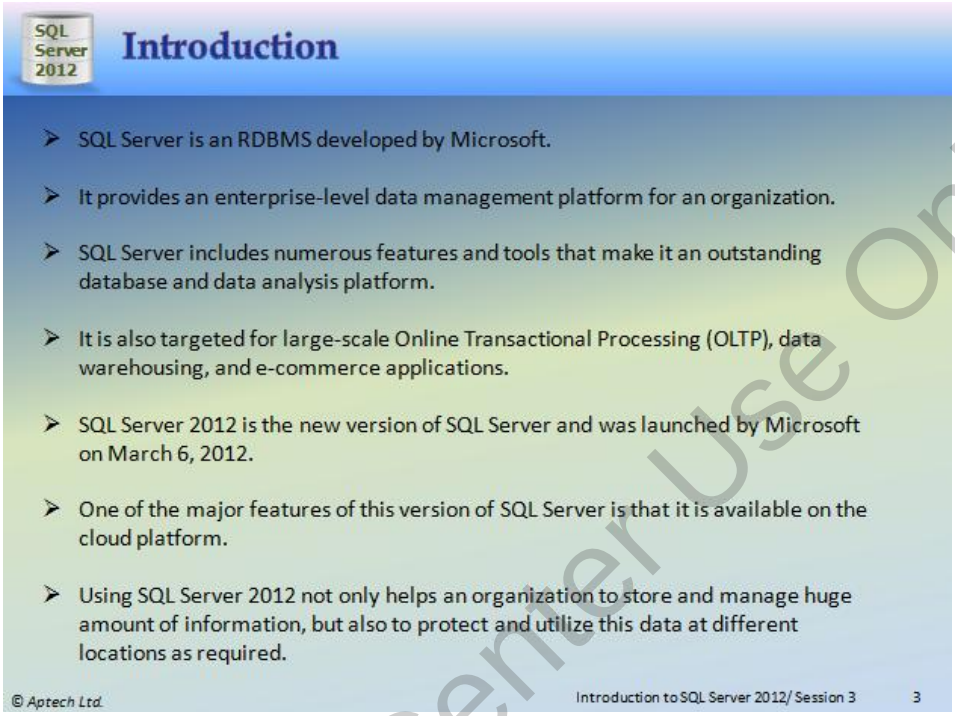
In-Class Activities:
Follow the order given here during In-Class activities.

Overview of the Session:
Then, give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces SQL Server 2012. They will learn about the basic architecture of SQL server 2012, the role and structure of SQL Server along with the new features added in SQL Server 2012. They will also know about the process to connect to SQL Server instances, create and organize script files, and execute Transact-SQL queries.

3.2 In-Class Explanations

Introduction

Slide 3

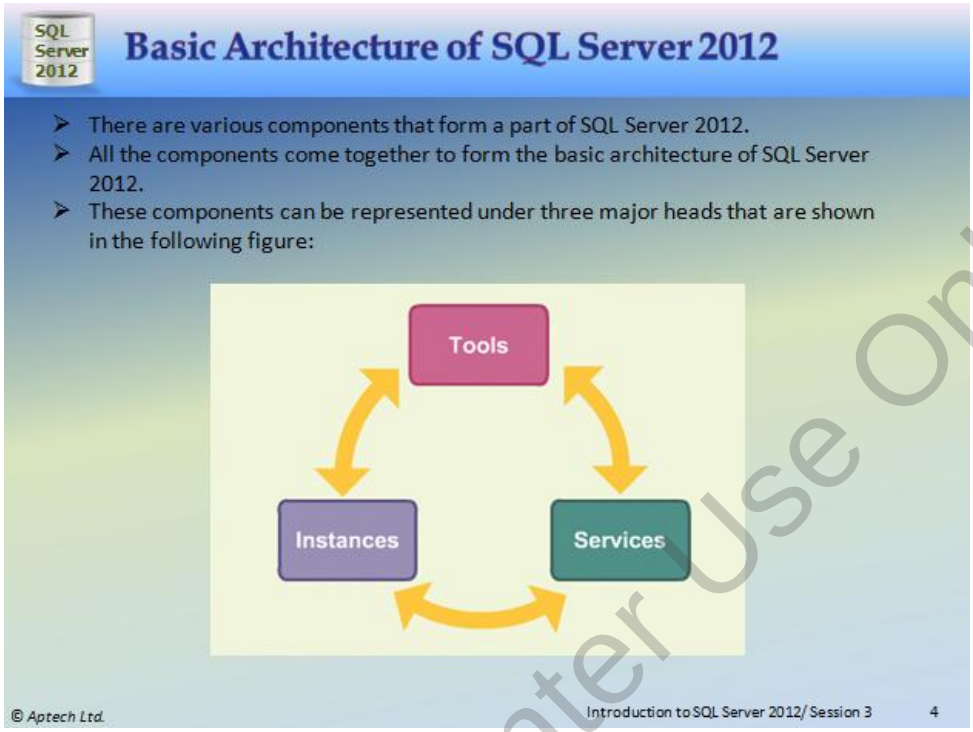


Using slide 3, explain the SQL server.
SQL Server is an RDBMS developed by Microsoft. It provides an enterprise-level data management platform for an organization. SQL Server includes numerous features and tools that make it an outstanding database and data analysis platform. It is also targeted for large-scale Online Transactional Processing (OLTP), data warehousing, and e-commerce applications.

SQL Server 2012 is the new version of SQL Server and was launched by Microsoft on March 6, 2012. One of the major features of this version of SQL Server is that it is available on the cloud platform. Using SQL Server 2012 not only helps an organization to store and manage huge amount of information, but also to protect and utilize this data at different locations as required.

Basic Architecture of SQL Server 2012

Slide 4

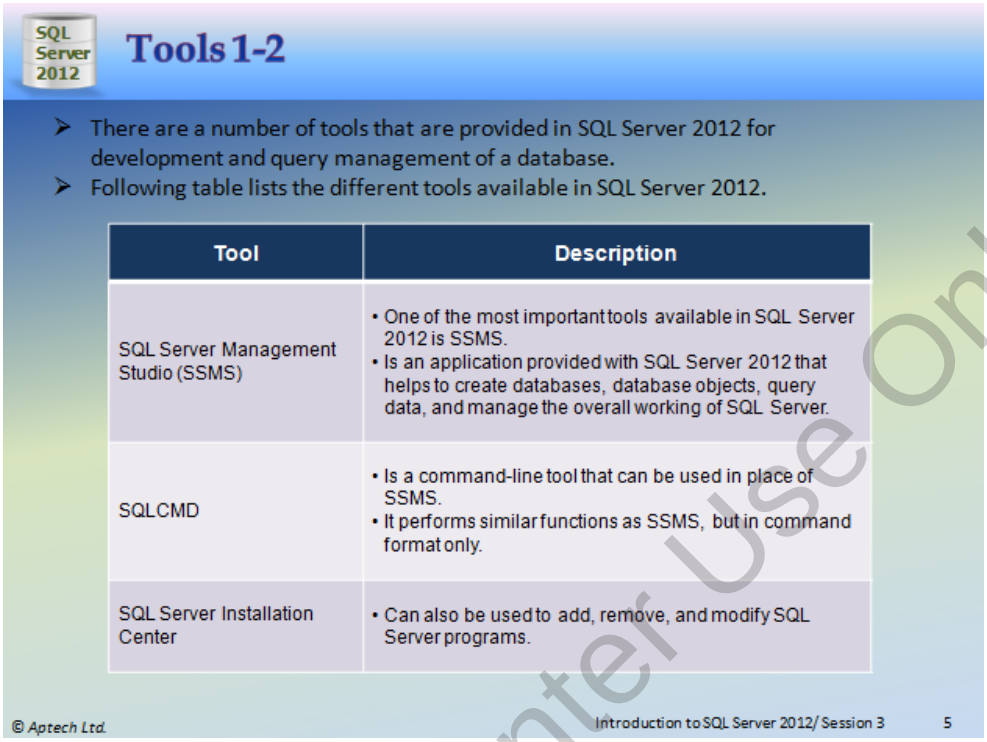


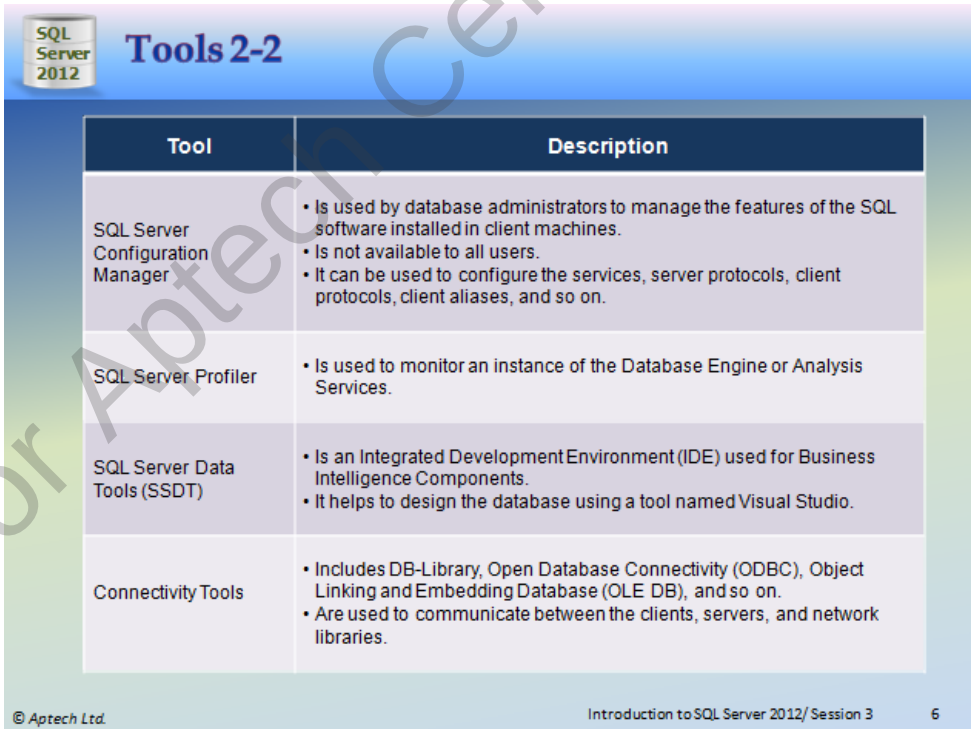
Using slide 4, explain the basic architecture of SQL Server 2012.

There are various components that form a part of SQL Server 2012. All the components come together to form the basic architecture of SQL Server 2012. These components can be represented under three major heads that are shown in the figure.

Tools

Slides 5 and 6





Using slides 5 and 6, explain the tools in SQL Server 2012.

There are a number of tools that are provided in SQL Server 2012 for development and query management of a database. The SQL Server Installation Center must be used to install SQL Server program features and tools. Features can also be modified or removed using the SQL Server Installation Center. Table lists the different tools available in SQL Server 2012

Explain the tools listed in the table.

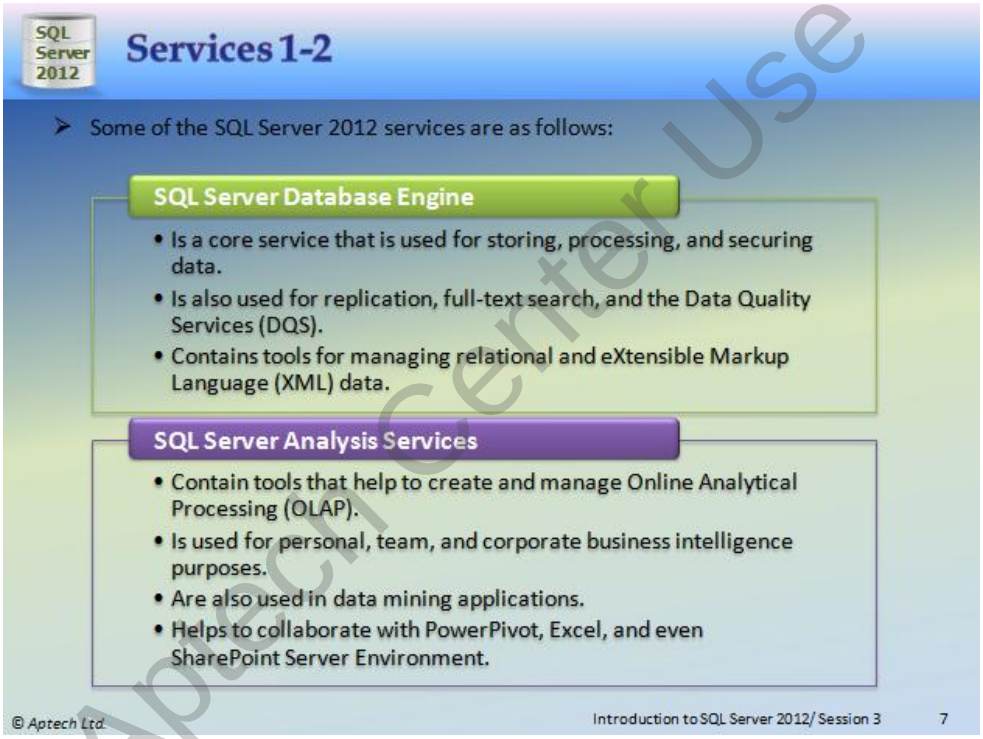
In-Class Question:

What is the use of SQL Profile Analyzer?

Answer:
SQL Server Profiler is used to monitor an instance of the Database Engine or Analysis Services.

Services

Slides 7 and 8



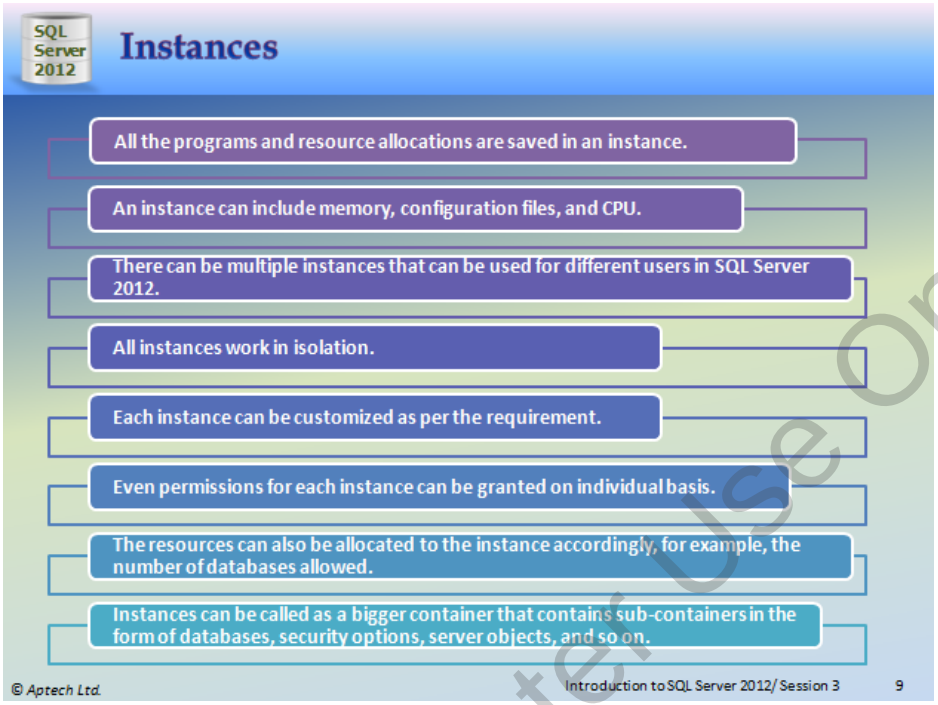
Using slides 7 and 8, explain the services in SQL Server 2012.

There are various services that are executed on a computer running SQL Server. These services run along with the other Windows services and can be viewed in the task manager. Some of the SQL Server 2012 services are as follows:

- **SQL Server Database Engine** - Database Engine is a core service that is used for storing, processing, and securing data. It is also used for replication, full-text search, and Data Quality Services (DQS). It contains tools for managing relational and eXtensible Markup Language (XML) data.
- **SQL Server Analysis Services** - Analysis Services contain tools that help to create and manage Online Analytical Processing (OLAP). This is used for personal, team, and corporate business intelligence purposes. Analysis services are also used in data mining applications. These services also help to collaborate with PowerPivot, Excel, and even SharePoint Server Environment.
- **SQL Server Reporting Services** - Reporting Services help to create, manage, publish, and deploy reports. These reports can be in tabular, matrix, graphical, or free-form format. Report applications can also be created using Reporting Services.
- **SQL Server Integration Services** - Integration Services are used for moving, copying, and transforming data using different graphical tools and programmable objects. The DQS component is also included in Integration Services. Integration services help to build high-performance data integration solutions.
- **SQL Server Master Data Services** - Master Data Services (MDS) are used for master data management. MDS is used for analysis, managing, and reporting information such as hierarchies, granular security, transactions, business rules, and so on.

Instances

Slide 9



Using slide 9, explain the instances.

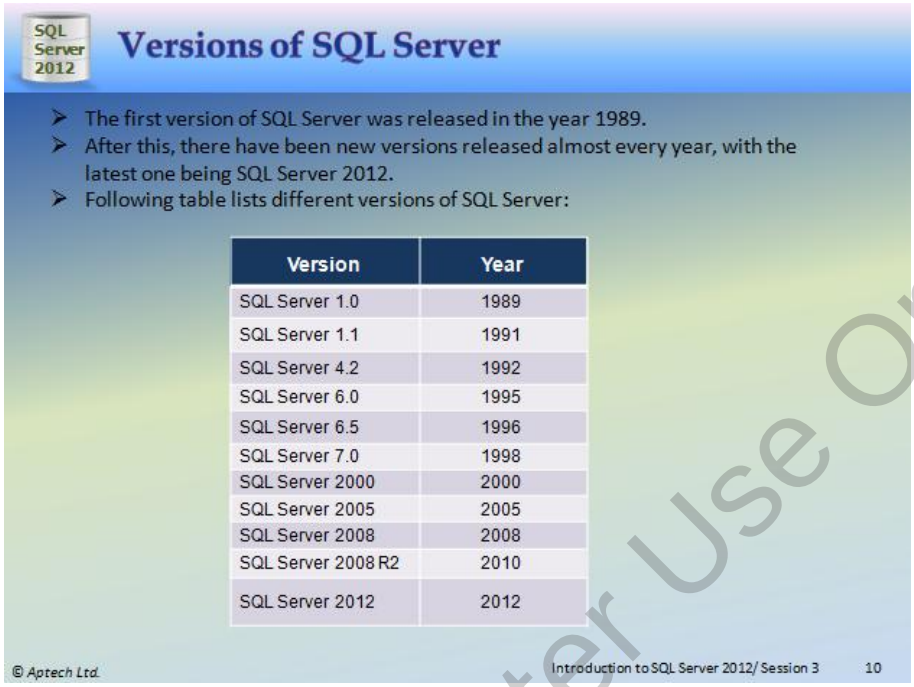
All the programs and resource allocations are saved in an instance. An instance can include memory, configuration files, and CPU. Multiple instances can be used for different users in SQL Server 2012. Even though many instances may be present on a single computer, they do not affect the working of other instances. This means that all instances work in isolation. Each instance can be customized as per the requirement. Even permissions for each instance can be granted on individual basis. The resources can also be allocated to the instance accordingly, for example, the number of databases allowed.

In other words, instances can be called as a bigger container that contains sub-containers in the form of databases, security options, server objects, and so on.

Mention, that creating separate SQL instances also cuts down on hardware and licensing costs. Administrators can use the same license for a single server. We don't have to buy another server and/or another license.

Versions of SQL Server

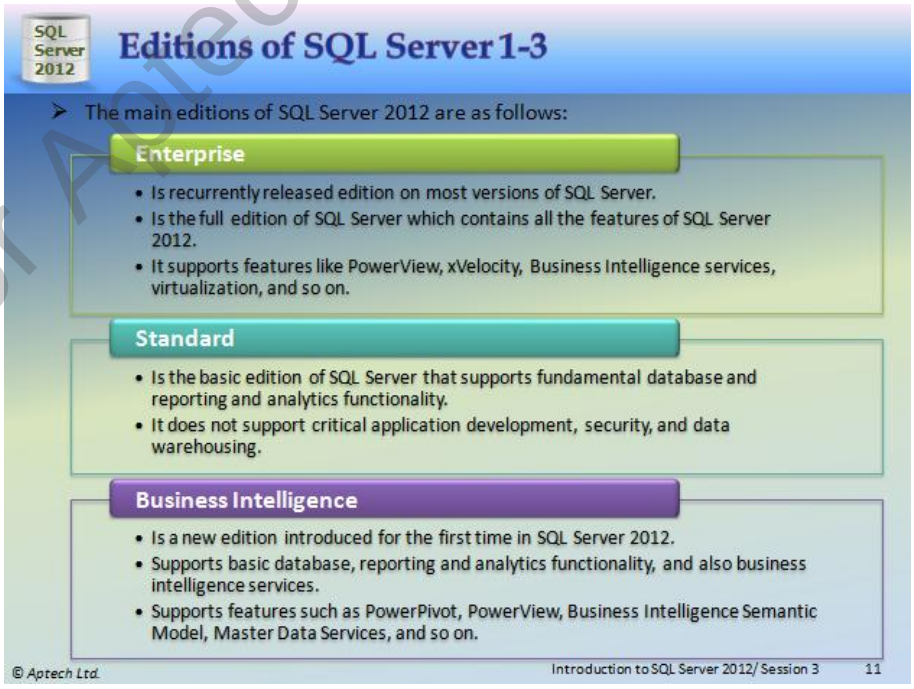
Slide 10

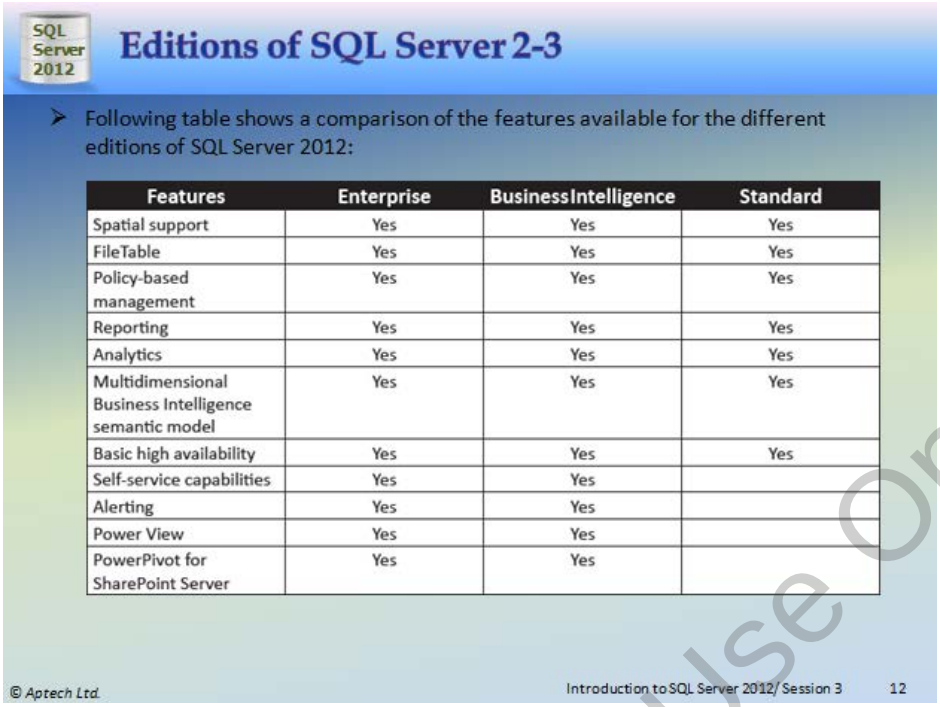


Using slide 10 explain, the versions of SQL server.
The first version of SQL Server was released in the year 1989. After this, there have been new versions released almost every year, with the latest one being SQL Server 2012. Table in the slide lists different versions of SQL Server.

Editions of SQL Server

Slides 11 to 13

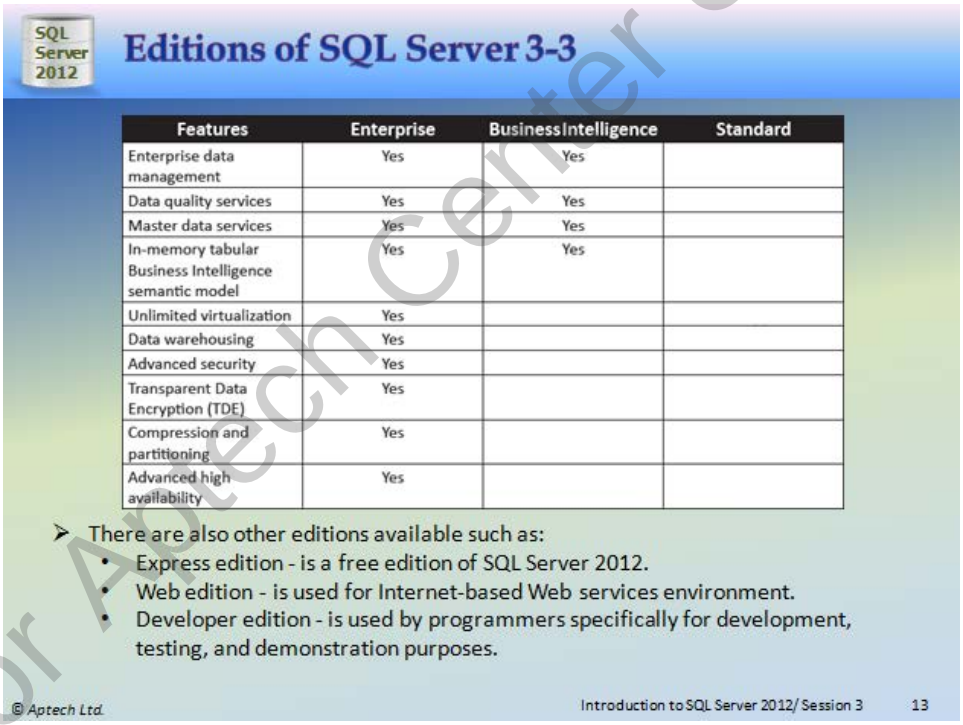




Editions of SQL Server 2-3

- Following table shows a comparison of the features available for the different editions of SQL Server 2012:

Features	Enterprise	Business Intelligence	Standard
Spatial support	Yes	Yes	Yes
FileTable	Yes	Yes	Yes
Policy-based management	Yes	Yes	Yes
Reporting	Yes	Yes	Yes
Analytics	Yes	Yes	Yes
Multidimensional Business Intelligence semantic model	Yes	Yes	Yes
Basic high availability	Yes	Yes	Yes
Self-service capabilities	Yes	Yes	
Alerting	Yes	Yes	
Power View	Yes	Yes	
PowerPivot for SharePoint Server	Yes	Yes	



Editions of SQL Server 3-3

Features	Enterprise	Business Intelligence	Standard
Enterprise data management	Yes	Yes	
Data quality services	Yes	Yes	
Master data services	Yes	Yes	
In-memory tabular Business Intelligence semantic model	Yes	Yes	
Unlimited virtualization	Yes		
Data warehousing	Yes		
Advanced security	Yes		
Transparent Data Encryption (TDE)	Yes		
Compression and partitioning	Yes		
Advanced high availability	Yes		

- There are also other editions available such as:
  - Express edition - is a free edition of SQL Server 2012.
  - Web edition - is used for Internet-based Web services environment.
  - Developer edition - is used by programmers specifically for development, testing, and demonstration purposes.

Using slides 11 to 13, explain the editions of SQL Server.

Based on database requirements, an organization can choose from any of the following three editions of SQL Server 2012 that have been released. These main editions of SQL Server 2012 are as follows:

- **Enterprise** – This is the edition that is recurrently released on most versions of SQL Server. This is the full edition of SQL Server which contains all the features of SQL Server 2012. The enterprise edition of SQL Server 2012 supports features such as Power View, xVelocity, Business Intelligence services, virtualization, and so on.

- **Standard** – The standard edition is the basic edition of SQL Server that supports fundamental database and reporting and analytics functionality. However, it does not support critical application development, security, and data warehousing.
- **Business Intelligence** – This is a new edition introduced for the first time in SQL Server 2012. This edition supports basic database, reporting and analytics functionality, and also business intelligence services. This edition supports features such as PowerPivot, PowerView, Business Intelligence Semantic Model, Master Data Services, and so on.

Table on slide 13 shows a comparison of the features available for the different editions of SQL Server 2012.
Other than these three editions, there are also other editions available such as Express edition, Web edition, and Developer edition. SQL Server 2012 Express is a free edition of SQL Server 2012. The Web edition is used for Internet-based Web services environment. The Developer edition is used by programmers specifically for development, testing, and demonstration purposes.

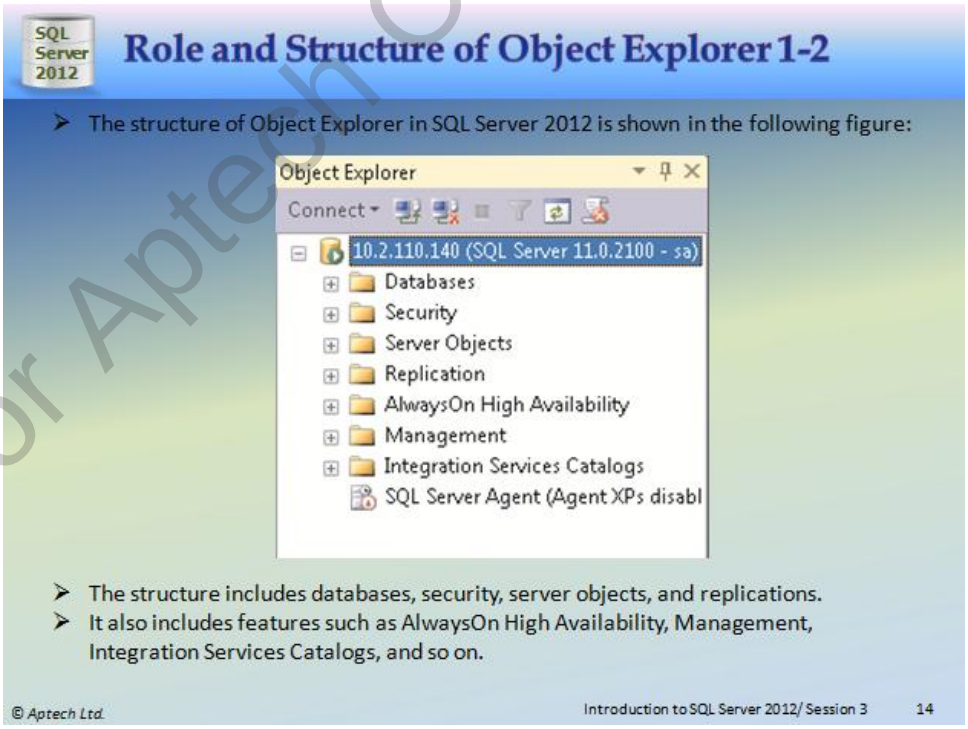
In-Class Question:

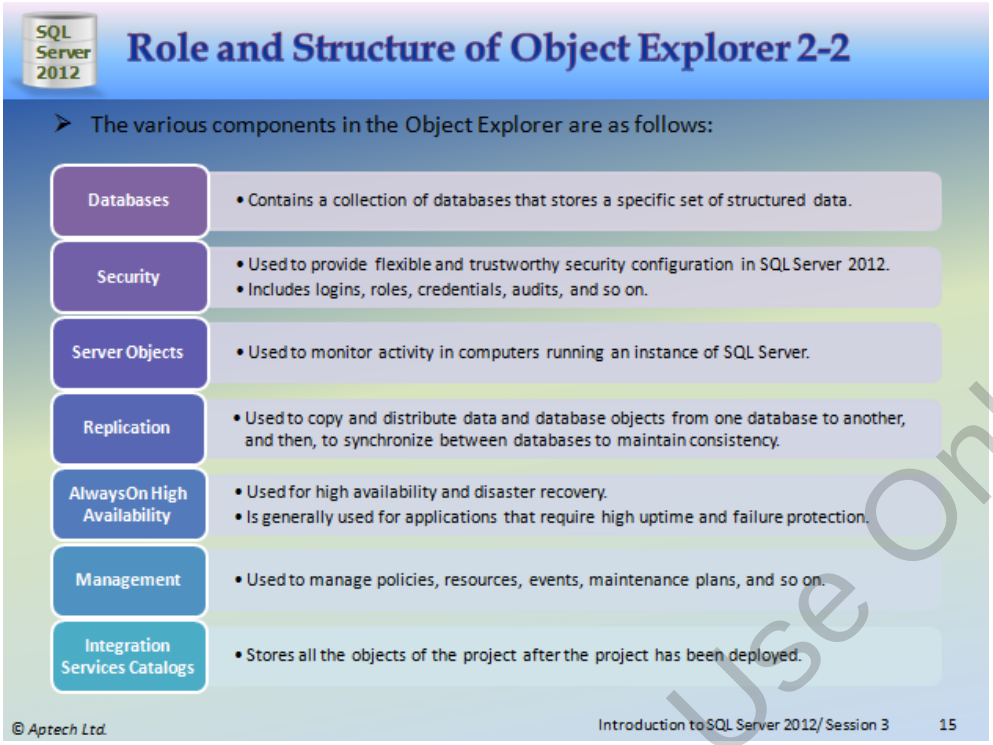
Which edition is introduced first time in SQL Server 2012?

Answer:
Business Intelligence edition is introduced for the first time in SQL Server 2012.

Role and Structure of Object Explorer

Slides 14 and 15

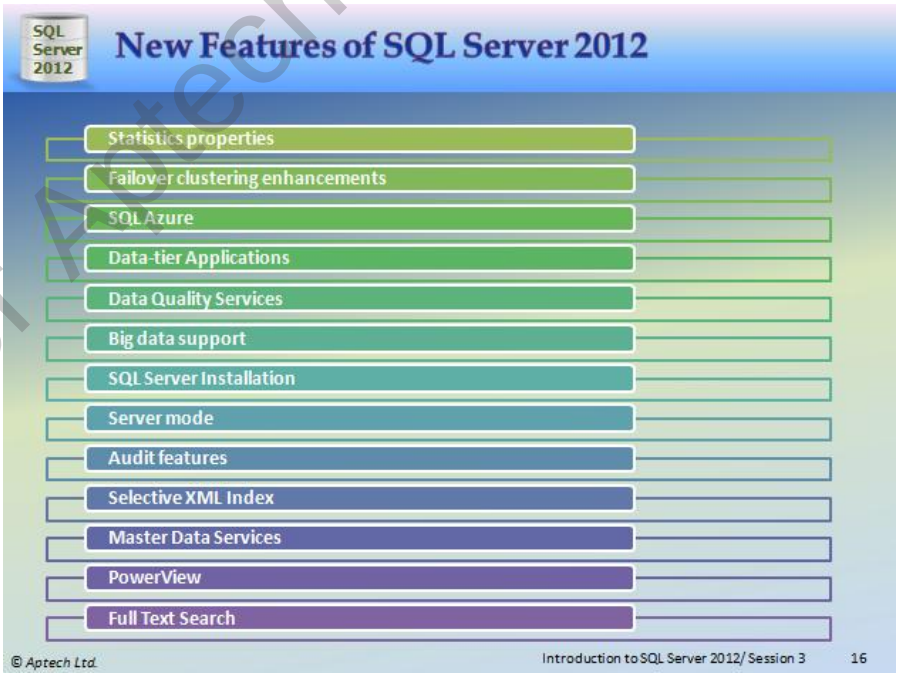




Using slides 14 and 15, explain the role and structure of object explorer.
The structure of Object Explorer in SQL Server 2012 is shown in figure on slide 14.
The structure includes databases, security, server objects, replications, and features such as AlwaysOn High Availability, Management, Integration Services Catalogs, and so on.
Explain the various components in the Object Explorer are shown on slide 15.

New Features of SQL Server 2012

Slide 16



Using slide 16, explain the new features of SQL server 2012.

The new features included in SQL Server 2012 are as follows:

- **Statistics properties** – Information about the statistics of objects can be viewed in SQL Server 2012 by using the sys.dm _db_stats_ properties function. Failover clustering enhancements – SQL Server 2012 provides multi-subnet failover clusters. It has also introduced indirect checkpoints and a flexible failover policy for cluster health detection. This has strengthened the existing disaster recovery solution in SQL Server 2012.
- **SQL Azure** – Microsoft SQL Azure is a cloud based relational database service that leverages existing SQL Server technologies. SQL Azure can be used to store and manage data using queries and other functions that are similar to SQL Server 2012. Reporting services and backup feature has been added in SQL Azure. Also, the database size in SQL Azure can now be increased upto 150 Giga Byte (GB).
- **Data-tier Applications** – A new version of the Data-tier Application (DAC) has been introduced in SQL Server 2012. A DAC is a logical database management entity defining SQL Server objects associated with a user's database. This DAC upgrade alters the existing database to match the schema to the new version of DAC.
- **Data Quality Services** – To maintain the integrity, conformity, consistency, accuracy, and validity of the data, the DQS has been integrated with SQL Server 2012. DQS uses techniques such as monitoring, data cleansing, matching and profiling, and so on to maintain the data quality and correctness.
- **Big data support** – Microsoft has announced a partnership with Cloudera to use Hadoop as the platform to support big data. Big data is a large collection of data under data sets that are divided for easier processing. The collection stored under big data can include information from social networking Websites, roadway traffic and signaling data, and so on. These kinds of data that are large and complex require specific applications such as Hadoop to process the data. SQL Server 2012 in collaboration with Hadoop would now be able to support big data.
- **SQL Server Installation** – The SQL Server Installation Center now includes SQL Server Data Tools (SSDT) and Server Message Block (SMB) file server. SSDT provides an IDE for building business intelligence solutions. SMB file server is a supported storage option that can be used for system databases and database engines.
- **Server mode** – The server mode concept has been added for Analysis Services installation. An Analysis Services instance has three server modes that are Multidimensional, Tabular, and SharePoint.
- **Audit features** – Customized audit specifications can be defined to write custom events in the audit log. New filtering features have also been added in SQL Server 2012.
- **Selective XML Index** – This is a new type of XML index that is introduced in SQL Server 2012. This new index has faster indexing process, improved scalability, and enhanced query performance.
- **Master Data Services** – This feature provides a central data hub to ensure consistency and integrity across different applications. In SQL Server 2012, an Excel add-in has been created to support master data when working with Excel. This add-in makes it easy to transfer and manage the master data in Excel. This data can be easily edited in Excel and it can also be published back to the database.
- **PowerView** – A new business intelligence toolkit named PowerView has been introduced in SQL Server 2012. This toolkit helps to create Business Intelligence reports for an entire organization. PowerView is an add-in of SQL Server 2012 that works in collaboration with Microsoft SharePoint Server 2010. This add-in helps to present and

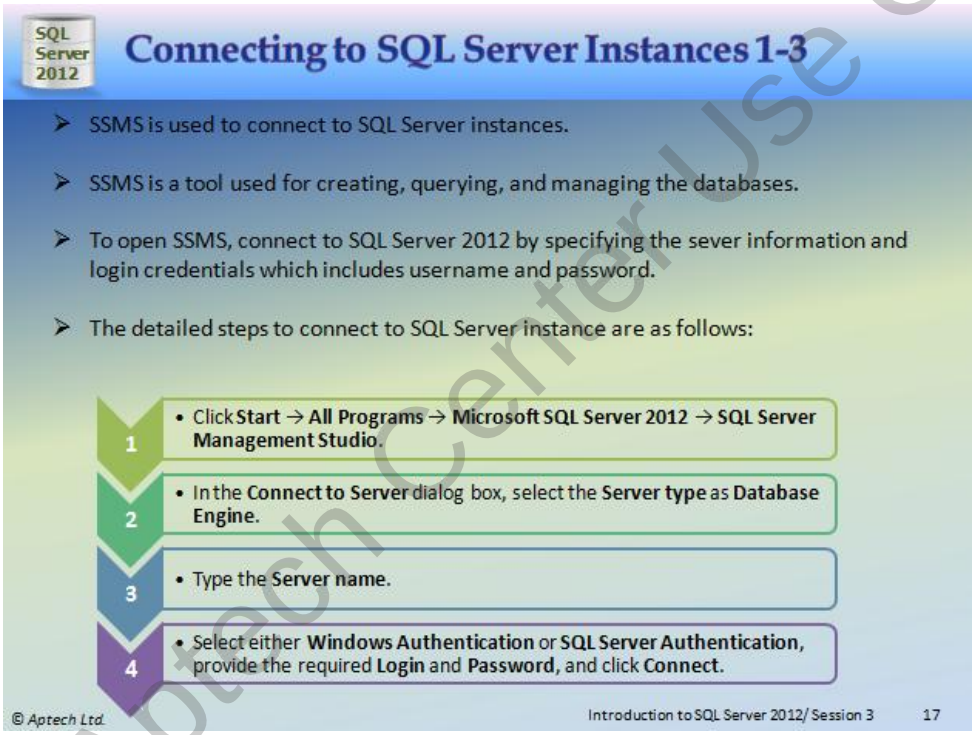
visualize SQL Server 2012 data in a compatible view on the SharePoint platform. PowerView is a business intelligence tool that can be used to make customer presentations by using models, animations, visualization, and so on.

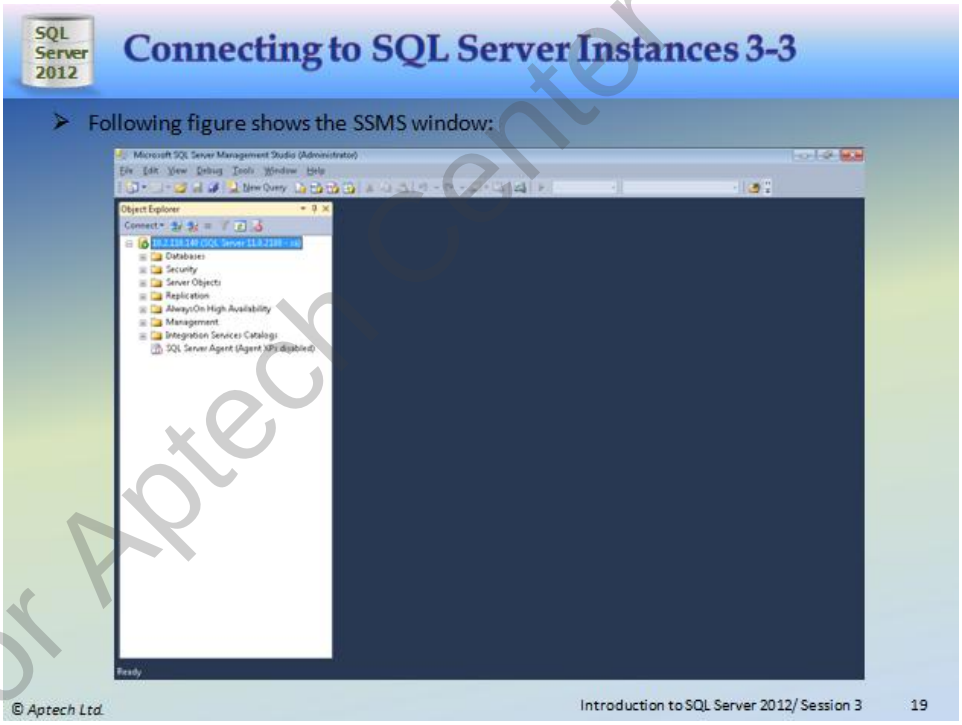
- **Full Text Search** - In SQL Server 2012, the data stored in extended properties and metadata is also searched and indexed. All the additional properties of a document are searched along with data present in the document. These additional properties include Name, Type, Folder path, Size, Date, and so on.

Column store indexes, contained database, pagination, and sequence objects are also some new features introduced in SQL server 2012.

Connect to SQL Server Instances

Slides 17 to 19





Using slide 17 to 19, explain how to connect to SQL Server instances.

SSMS is used to connect to SQL Server instances. SSMS is a tool used for creating, querying, and managing the databases. To open SSMS, connect to SQL Server 2012 by specifying the sever information and login credentials. The login credentials will include username and password. The detailed steps to connect to SQL Server instance are as follows:

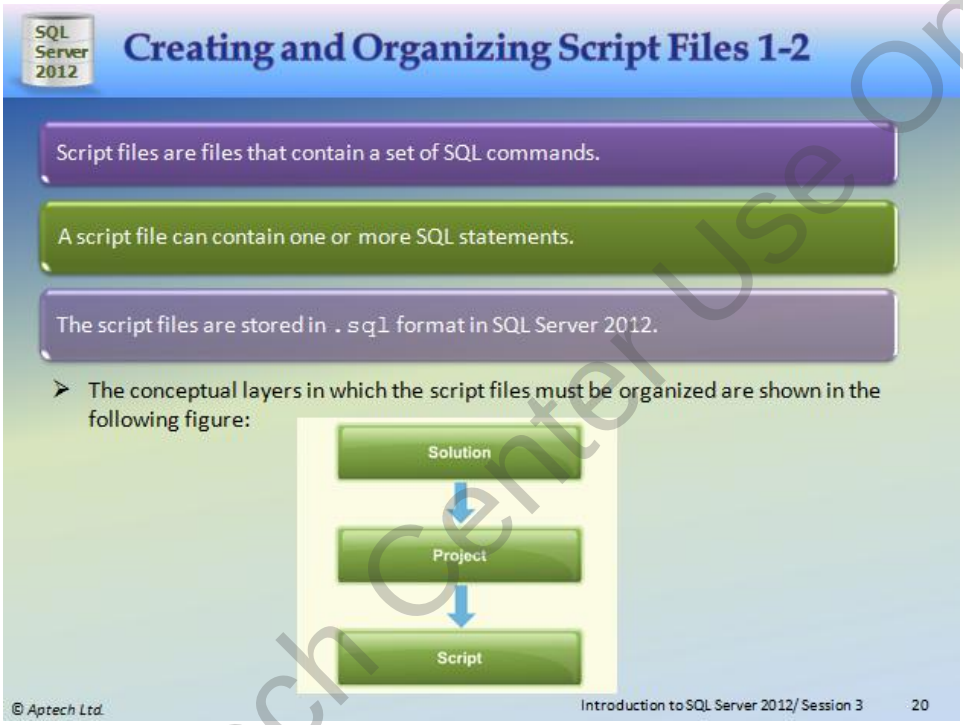
- Click Start → All Programs → Microsoft SQL Server 2012 → SQL Server Management Studio.
- In the Connect to Server dialog box, select the Server type as Database Engine.
- Type the Server name.

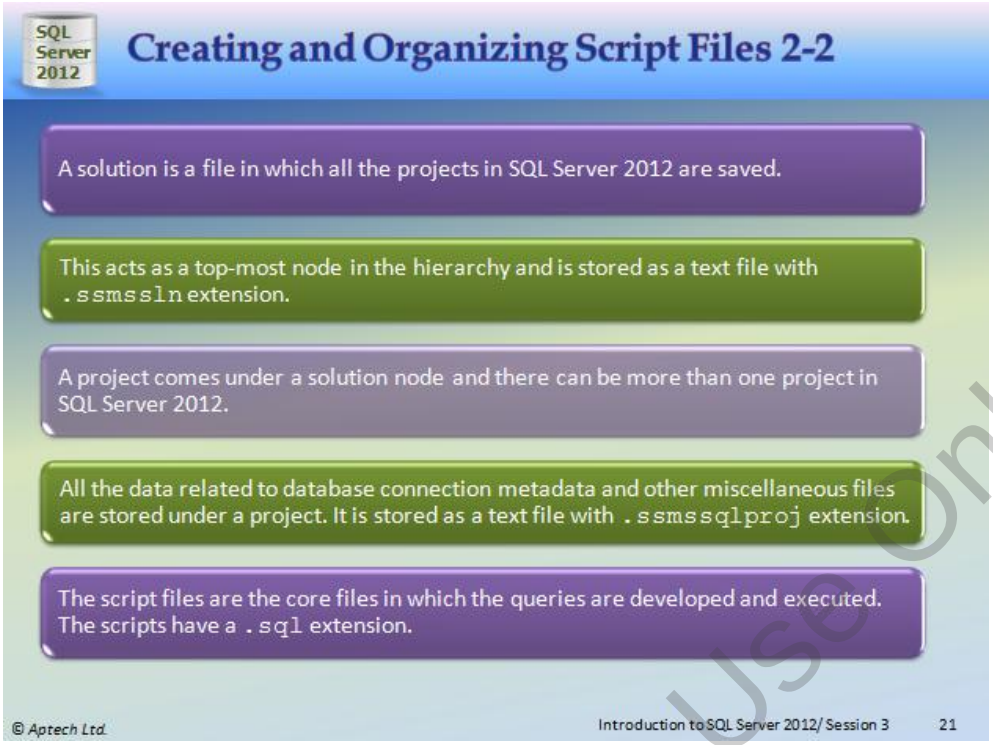
Select either Windows Authentication or SQL Server Authentication, provide the required Login and Password, and click Connect.

Figure on slide 18 shows the connect to server dialog box. Figure on slide 19 shows the SSMS window.

Creating and Organizing Script Files

Slides 20 and 21





Using slides 20 and 21, explain how to create and organize a script file.
Script files are files that contain a set of SQL commands. A script file can contain one or more SQL statements. The script files are stored in .sql format in SQL Server 2012.
The conceptual layers in which the script files must be organized are shown in the figure on slide 20.
A solution is a file in which all the projects in SQL Server 2012 are saved. This acts as a top-most node in the hierarchy. The solution file is stored as a text file with .ssmssln extension.
A project comes under a solution node. There can be more than one project in SQL Server 2012. All the data related to database connection metadata and other miscellaneous files are stored under a project. It is stored as a text file with .ssmssqlproj extension. The script files are the core files in which the queries are developed and executed. The scripts have a .sql extension.

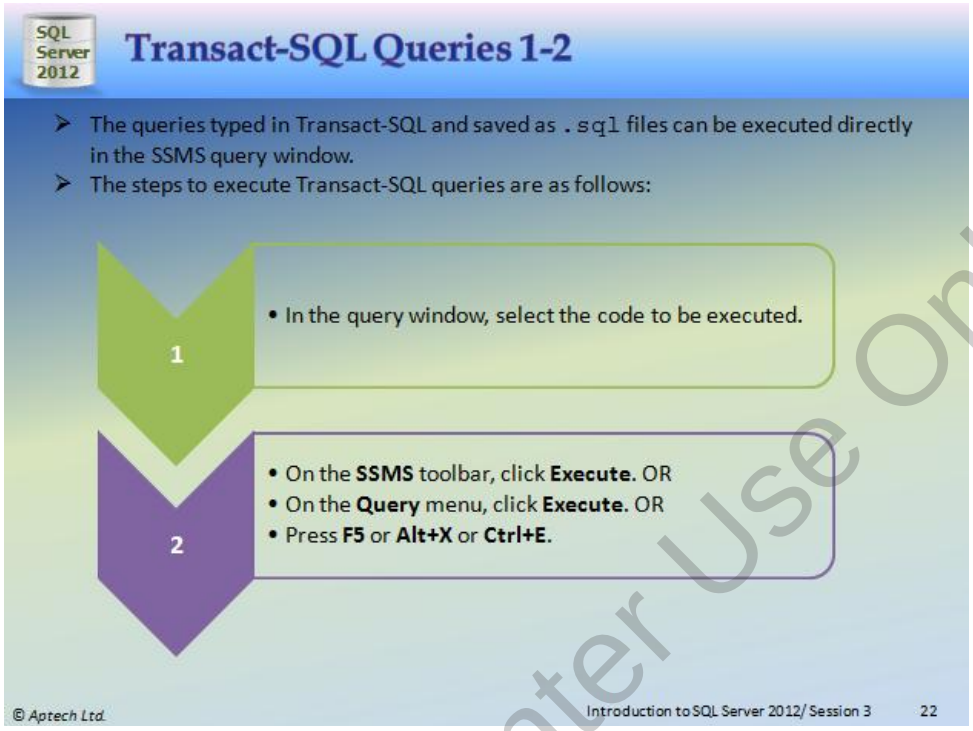
In-Class Question:

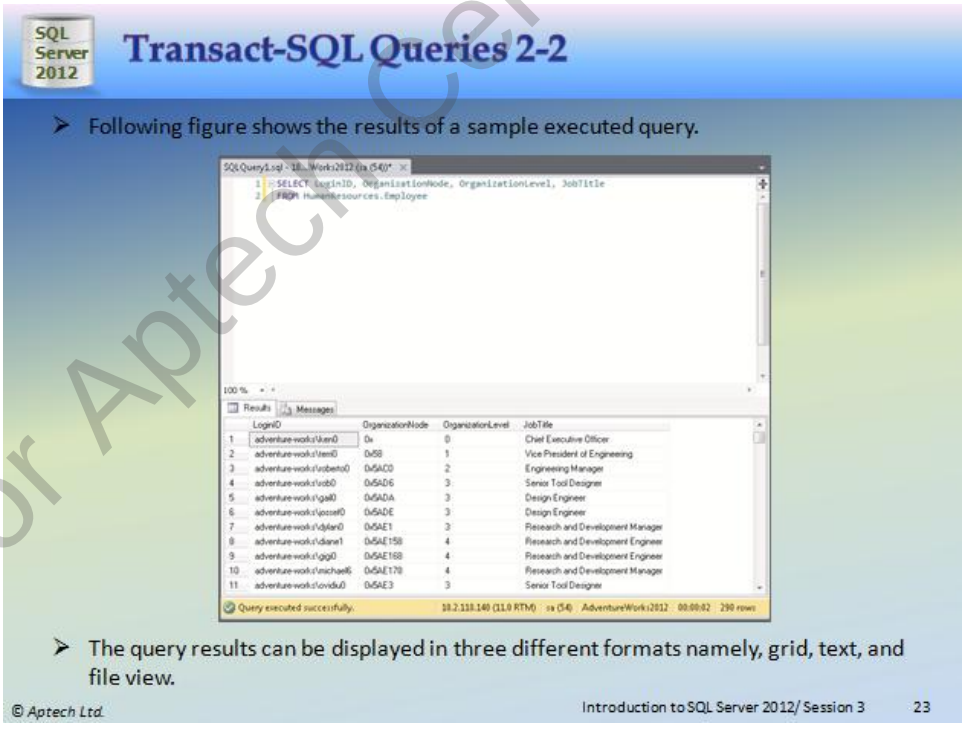
What is the extension of a script file?

Answer:
.sql is the extension for a script file.

Transact-SQL Queries

Slides 22 and 23





Using slides 22 and 23, explain the Transact-SQL queries.

The queries typed in Transact-SQL and saved as .sql files can be executed directly in the SSMS query window. The steps to execute Transact-SQL queries are as follows:

1. In the query window, select the code to be executed.
2. On the SSMS toolbar, click Execute. OR On the Query menu, click Execute. OR Press F5 OR Press Alt+X OR Press Ctrl+E.

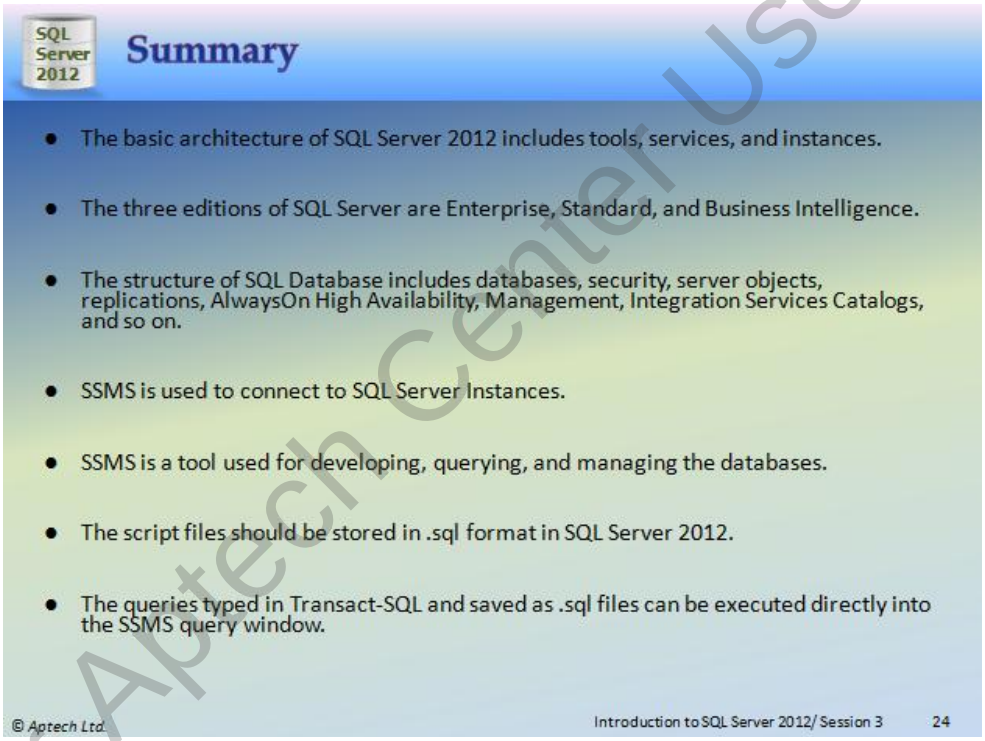
Figure on slide 23 shows the execution of a sample query.

The query results can be displayed in three different formats. The three formats available are grid, text, and file view.

The WITH RESULT SETS clause can also be used with a stored procedure, which returns multiple result sets and for each result set you can define the column name and data types for each column separately.

Summarize Session

Slide 24



Using slide 24, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- The basic architecture of SQL Server 2012 includes tools, services, and instances.
- The three editions of SQL Server are Enterprise, Standard, and Business Intelligence.
- The structure of SQL Database includes databases, security, server objects, replications, AlwaysOn High Availability, Management, Integration Services Catalogs, and so on.
- SSMS is used to connect to SQL Server Instances.
- SSMS is a tool used for developing, querying, and managing the databases.

- The script files should be stored in .sql format in SQL Server 2012.
- The queries typed in Transact-SQL and saved as .sql files can be executed directly into the SSMS query window.

3.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the SQL Azure topic offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 4 – SQL Azure

4.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

4.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain SQL Azure
- List the benefits of SQL Azure
- State the differences between SQL Azure and on-premises SQL Server
- List the steps to connect SQL Azure with SSMS

4.1.2 Teaching Skills

To teach this session, you should be well-versed with SQL Azure and its benefits. Also, the differences between SQL Azure and on-premises SQL Server should be known.
The session also covers the process to connect SQL Azure with SSMS.
You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

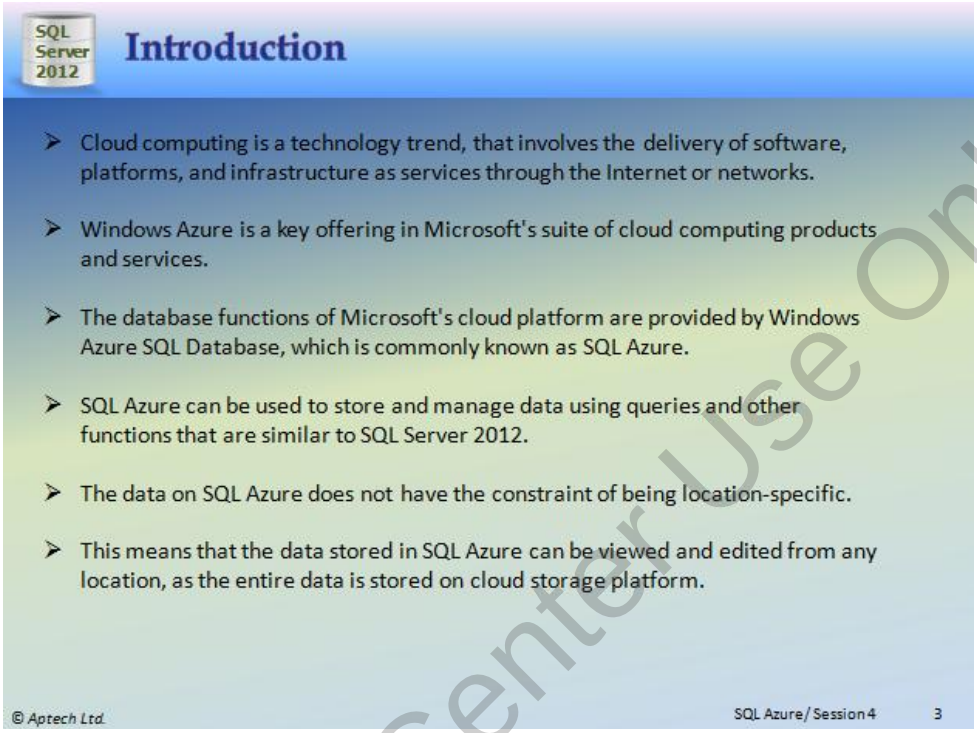
In-Class Activities:
Follow the order given here during In-Class activities.

Overview of the Session:
Then, give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces SQL Azure and its benefits. They will learn about the differences between SQL Azure and on-premises SQL Server. They will also know the process to connect SQL Azure with SSMS.

4.2 In-Class Explanations

Introduction

Slide 3

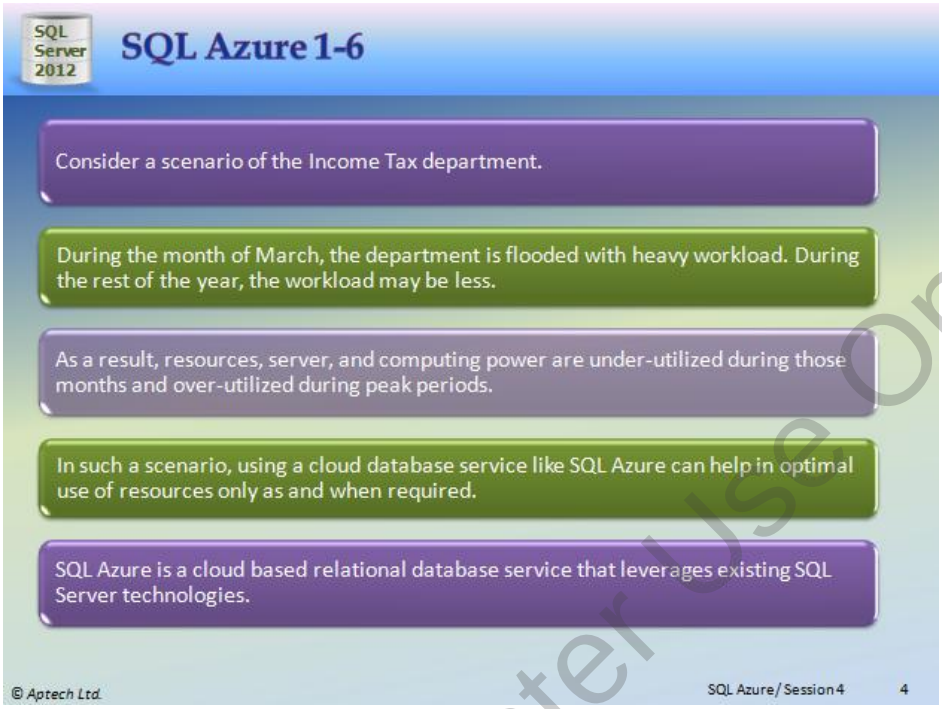


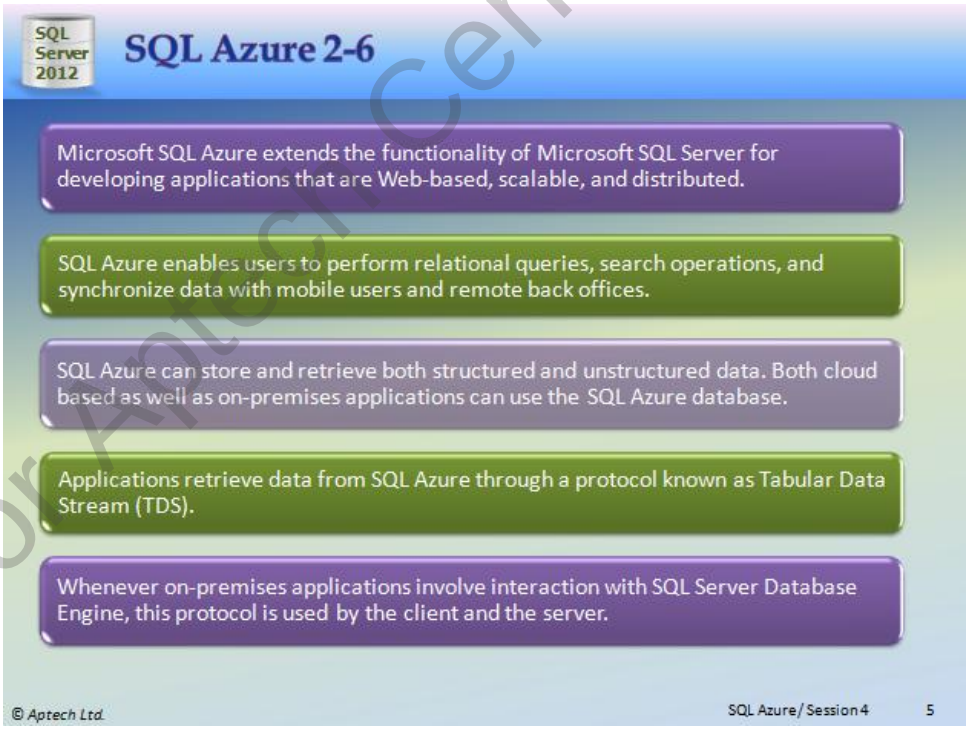
Using slide 3, explain the SQL Azure.
Cloud computing is a technology trend, that involves the delivery of software, platforms, and infrastructure as services through the Internet or networks. Windows Azure is a key offering in Microsoft's suite of cloud computing products and services. The database functions of Microsoft's cloud platform are provided by Windows Azure SQL Database, which is commonly known as SQL Azure.

SQL Azure can be used to store and manage data using queries and other functions that are similar to SQL Server 2012. The data on SQL Azure does not have the constraint of being location-specific. This means that the data stored in SQL Azure can be viewed and edited from any location, as the entire data is stored on cloud storage platform.

SQL Azure

Slides 4 to 9

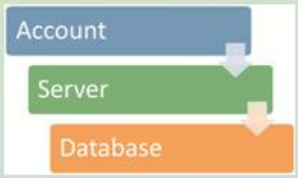




SQL Azure 3-6

- Following figure shows the simplified view of SQL Azure architecture:

- The process of SQL Azure operation is explained in the model as shown in the following figure:



SQL Azure 4-6

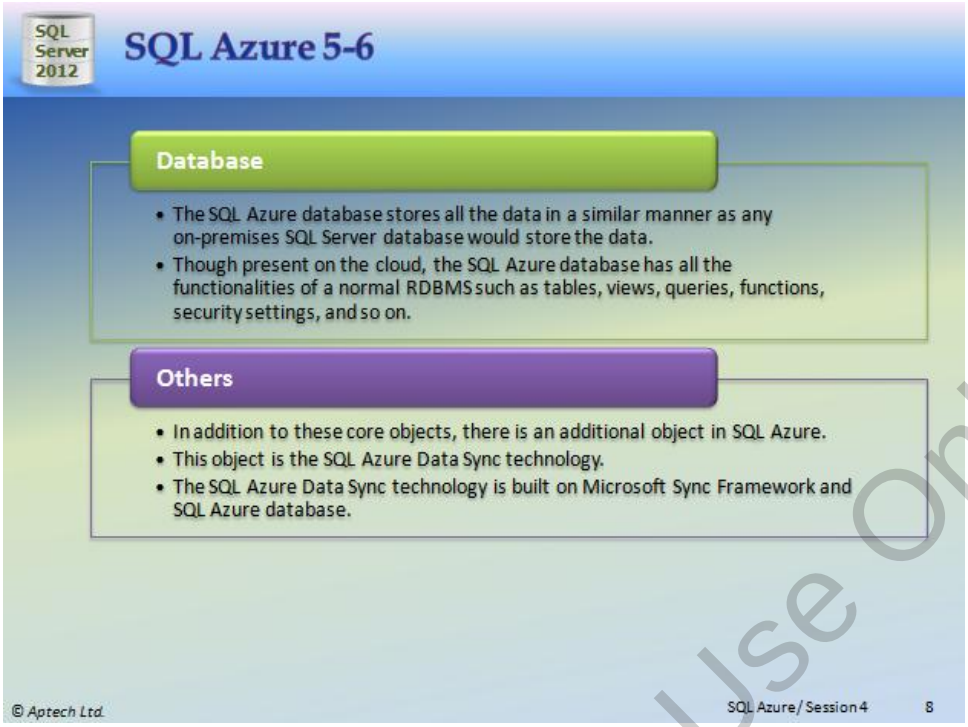
- The three core objects in the SQL Azure operation model are as follows:

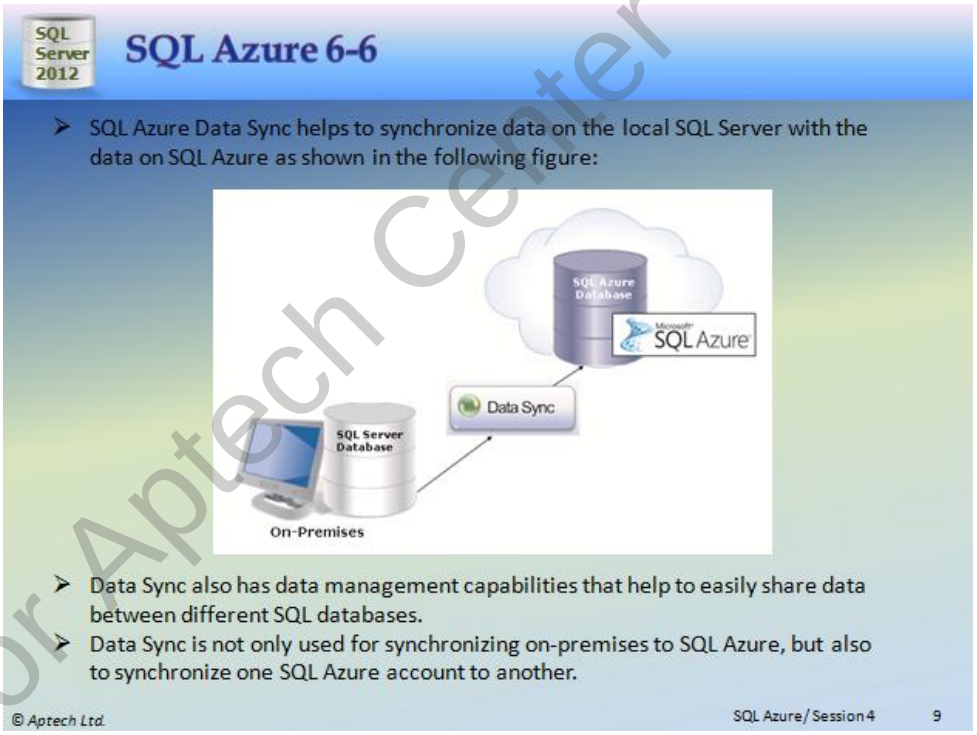
Account

- An SQL Azure account must first be created before adding servers that will help to store and manage the data.
- This account is created for billing purposes.
- The subscription for an account is recorded and metered and an individual is charged according to the usage.
- To create an account, the credentials need to be provided.
- After the user account is created, the requirements need to be provided for the SQL Azure database.
- This includes the number of databases required, database size, and so on.

Server

- The SQL Azure server is the object that helps to interact between the account and the database.
- After the account is registered, the databases are configured using the SQL Azure server.
- Other settings such as firewall settings and Domain Name System (DNS) assignment are also configured in the SQL Azure server.





Using slides 4 and 5, explain the basic architecture of SQL server 2012.

Consider a scenario of the Income Tax department. During the month of March, the department is flooded with heavy workload. During the rest of the year, the workload may be less. As a result, resources, server, and computing power are under-utilized during those months and over-utilized during peak periods. In such a scenario, using a cloud database service such as SQL Azure can help in optimal use of resources only as and when required. SQL Azure is a cloud based relational database service that leverages existing SQL Server technologies. Microsoft SQL Azure extends the functionality of Microsoft SQL Server for developing applications that are Web-based, scalable, and distributed.

SQL Azure enables users to perform relational queries, search operations, and synchronize data with mobile users and remote back offices. SQL Azure can store and retrieve both structured and unstructured data.

Both cloud based as well as on-premises applications can use the SQL Azure database. Applications retrieve data from SQL Azure through a protocol known as Tabular Data Stream (TDS). This protocol is not new to SQL Azure. Whenever on-premises applications involve interaction with SQL Server Database Engine, this protocol is used by the client and the server.

Figure on slide 6 shows the simplified view of SQL Azure architecture.
The process of SQL Azure operation is explained in the model as shown in figure on slide 6.
Using slides 7 and 8, explain the core objects in SQL Azure.
The three core objects in the SQL Azure operation model are as follows:

- **Account** – An SQL Azure account must first be created before adding servers that will help to store and manage the data. This account is created for billing purposes. The subscription for an account is recorded and metered and an individual is charged according to the usage. To create an account, the credentials need to be provided. After the user account is created, the requirements need to be provided for the SQL Azure database. This includes the number of databases required, database size, and so on.
- **Server** – The SQL Azure server is the object that helps to interact between the account and the database. After the account is registered, the databases are configured using the SQL Azure server. Other settings such as firewall settings and Domain Name System (DNS) assignment are also configured in the SQL Azure server.
- **Database** – The SQL Azure database stores all the data in a similar manner as any on-premises SQL Server database would store the data. Though present on the cloud, the SQL Azure database has all the functionalities of a normal RDBMS such as tables, views, queries, functions, security settings, and so on.

In addition to these core objects, there is an additional object in SQL Azure. This object is the SQL Azure Data Sync technology. The SQL Azure Data Sync technology is built on Microsoft Sync Framework and SQL Azure database.

SQL Azure Data Sync helps to synchronize data on the local SQL Server with the data on SQL Azure as shown in figure on slide 16.

Data Sync also has data management capabilities that help to share data easily between different SQL databases. Data Sync is not only used for synchronizing on-premises to SQL Azure, but also to synchronize one SQL Azure account to another.

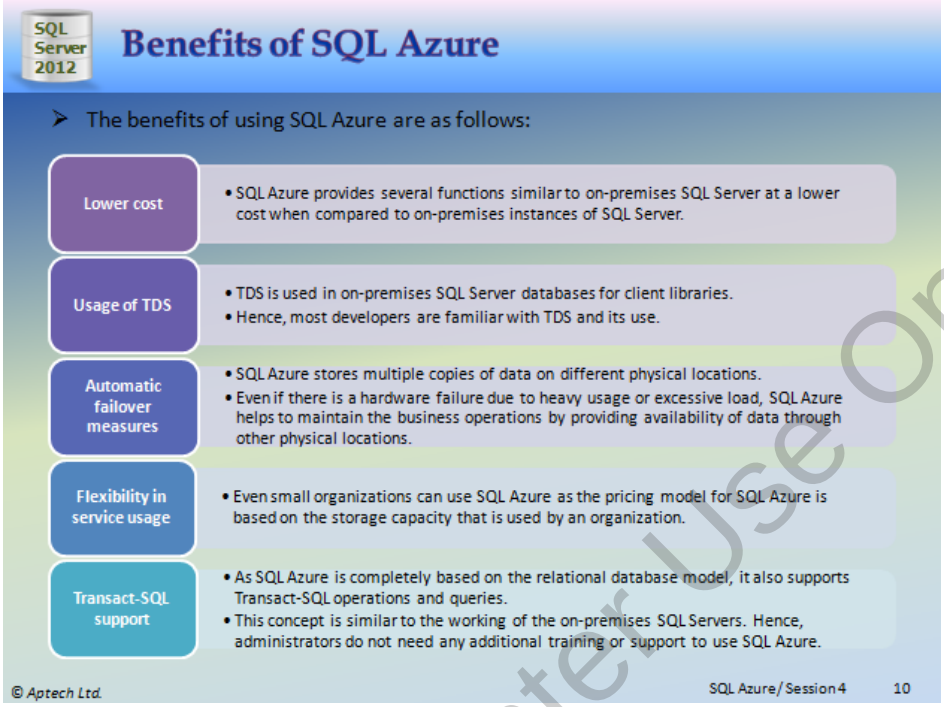
In-Class Question:

 What is SQL Azure?

Answer:
SQL Azure is a cloud based relational database service that leverages existing SQL Server technologies.

Benefits of SQL Azure

Slide 10

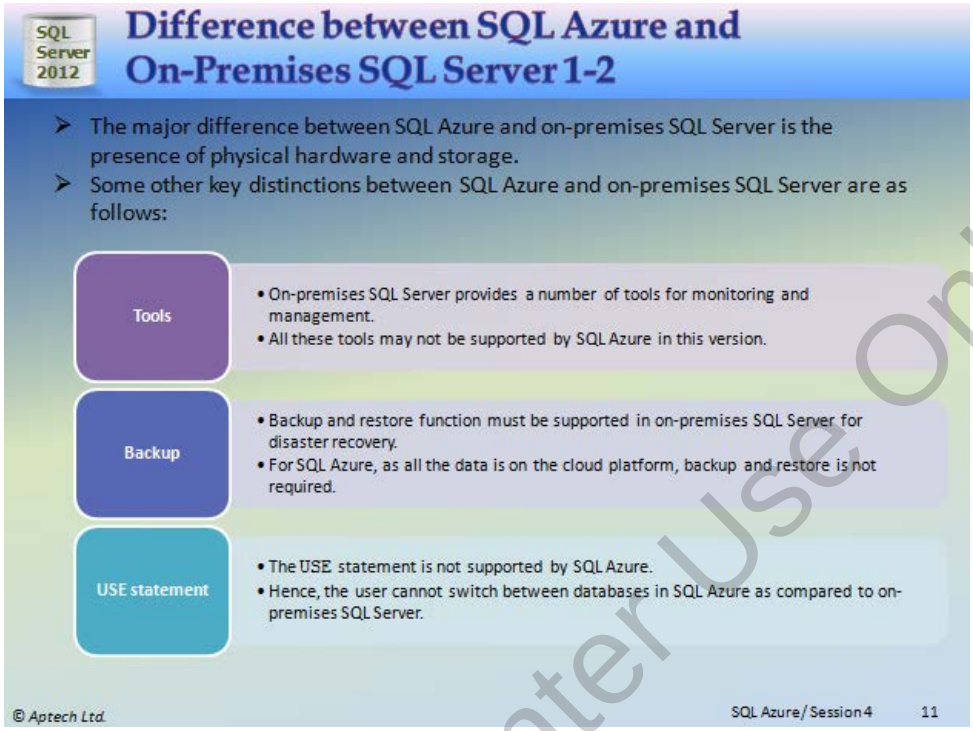


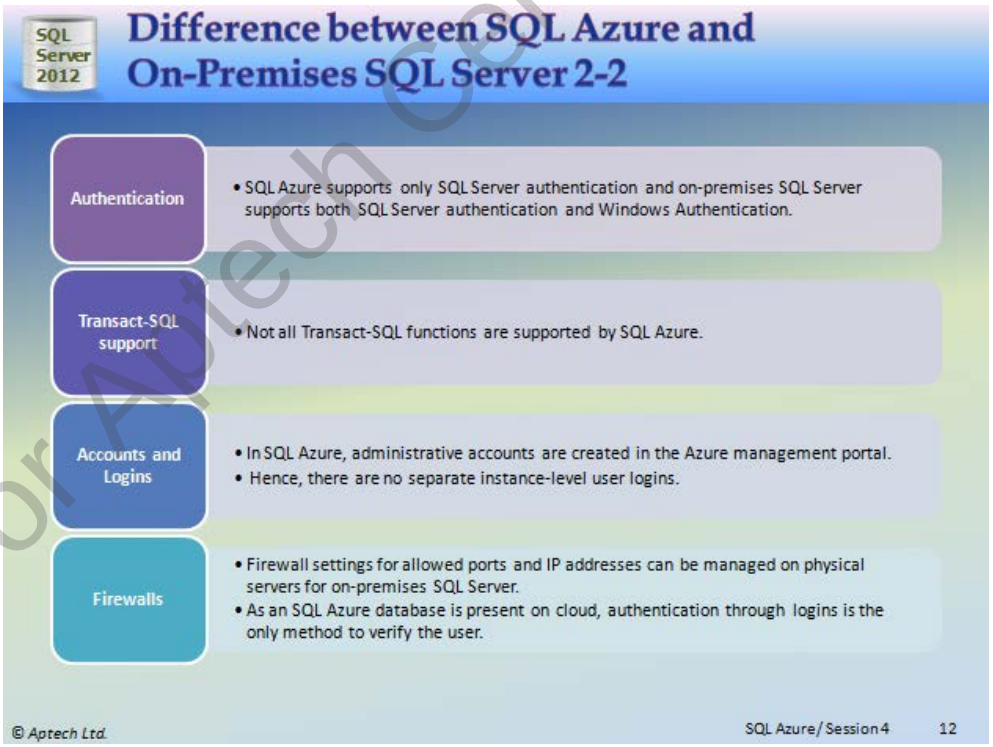
Using slide 10, explain the benefits of SQL Azure.
Explain the benefits of SQL Azure in details.

There are also many other benefits like multiple user access 24 hours a day 365 days a year, Azure SQL Databases User Level Access and Security, no expensive software to buy, data loss prevention by Microsoft's redundant backups, and failover protection and so on.

Difference between SQL Azure and On-Premises SQL Server

Slides 11 and 12





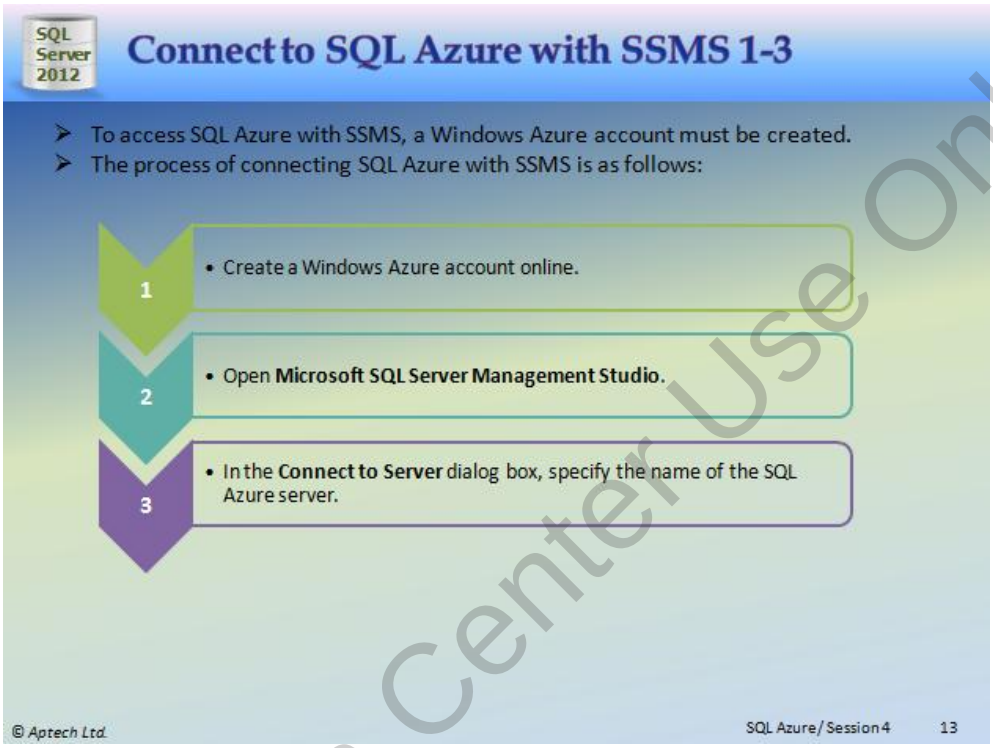
Using slides 11 and 12, explain the difference between SQL Azure and On-Premises SQL server.

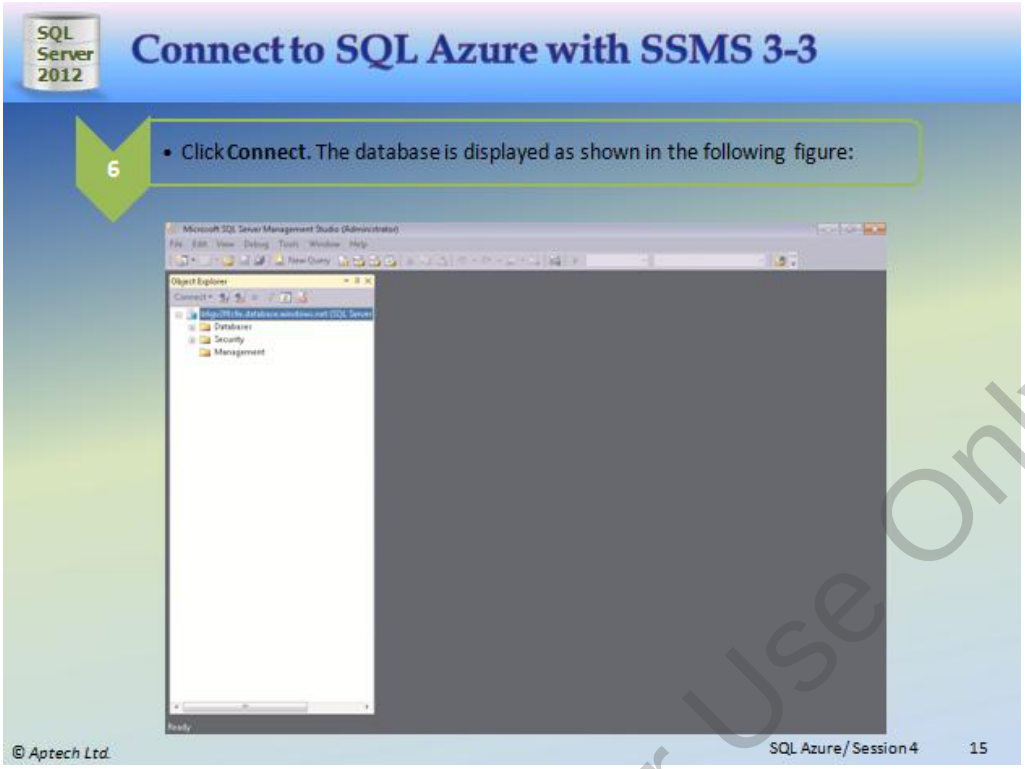
On-premises software is a type of software delivery model that is installed and operated from a customer's in-house server and computing infrastructure. The vendor also provides

after sales integration and support services but the security, availability, and overall management of it is the responsibility of customer.
Explain the difference between SQL Azure and On-Premises SQL server in details.

Connect to SQL Azure with SSMS

Slides 13 to 15





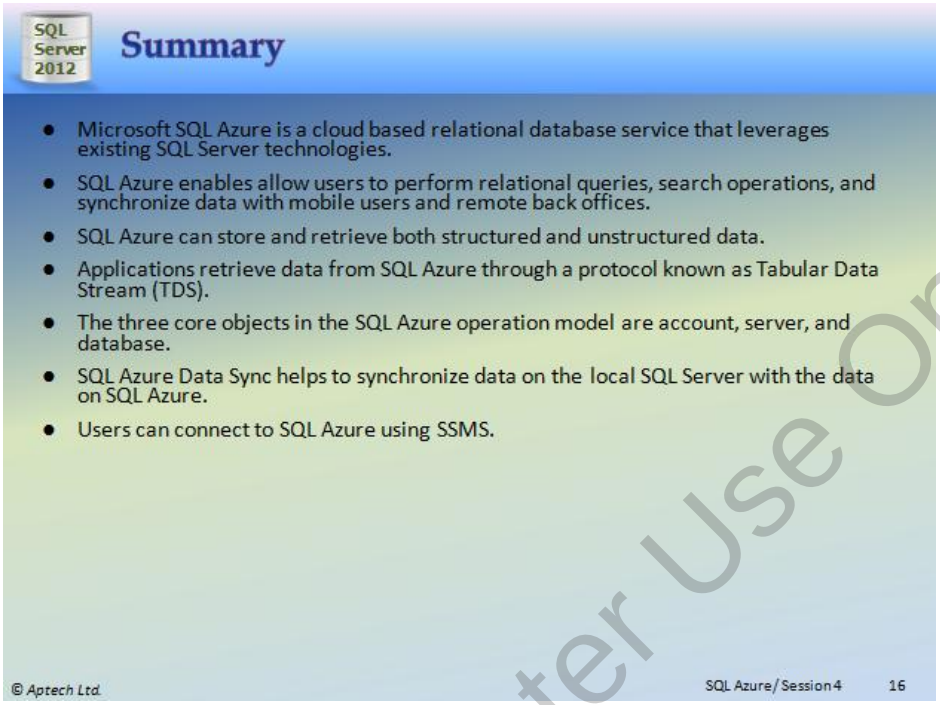
Using slides 13 to 15, explain the process of connecting to SQL Azure with SSMS.
To access SQL Azure with SSMS, a Windows Azure account must be created. The process of connecting SQL Azure with SSMS is as follows:

1. Create a Windows Azure account online.
2. Open Microsoft SQL Server Management Studio.
3. In the Connect to Server dialog box, specify the name of the SQL Azure server as shown in figure on slide 14. Each user account of SQL Azure has a specific Server name.
4. In the **Authentication** box, select **SQL Server Authentication**.
5. In the **Login** box, type the name of the SQL Azure administrator account, and the password.
6. Click Connect. The connection to the Database is successfully established as shown in figure on slide 15.

Mention that master database is the default database to which SQL Server connects to via SQL Azure. To connect to another database, on the **Connect to Server** box, click **Options** to reveal the **Connection Properties** tab and enter the name of the desired database in the **Connect to database** text box. After a connection to a user-defined database is established, a user cannot switch to other database without disconnecting and reconnecting to the next database. Users can switch from the master database to another database only through SSMS because the USE statement is not supported.

Summarize Session

Slide 16



Using slide 16, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- Microsoft SQL Azure is a cloud based relational database service that leverages existing SQL Server technologies.
- SQL Azure enables users to perform relational queries, search operations, and synchronize data with mobile users and remote back offices.
- SQL Azure can store and retrieve both structured and unstructured data.
- Applications retrieve data from SQL Azure through a protocol known as Tabular Data Stream (TDS).
- The three core objects in the SQL Azure operation model are account, server, and database.
- SQL Azure Data Sync helps to synchronize data on the local SQL Server with the data on SQL Azure.
- Users can connect to SQL Azure using SSMS.

4.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Transact-SQL topic offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 5 – Transact-SQL

5.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

5.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain Transact-SQL
- List the different categories of Transact-SQL statements
- Explain the various data types supported by Transact-SQL
- Explain Transact-SQL language elements
- Explain sets and predicate logic
- Describe the logical order of operators in the SELECT statement

5.1.2 Teaching Skills

To teach this session, you should be well-versed with Transact-SQL and the different categories of Transact-SQL statements. Also, the various data types and elements supported by Transact-SQL should be known.

The session also covers set theory, predicate logic, and the logical order of operators in the SELECT statement.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slide and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

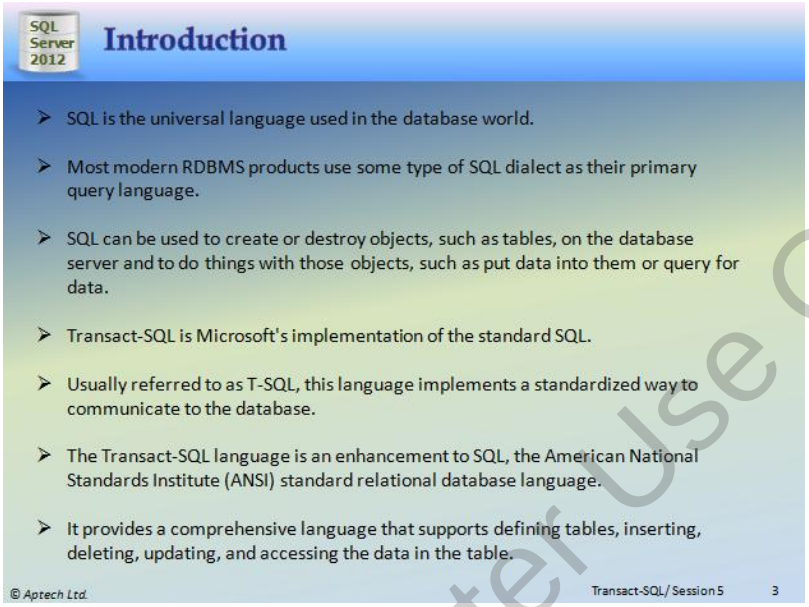
Overview of the Session:

Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces Transact-SQL and the different categories of Transact-SQL statements. They will learn about the various data types and elements supported by Transact-SQL. They will also know about set theory, predicate logic, and the logical order of operators in the SELECT statement.

5.2 In-Class Explanations

Introduction

Slide 3



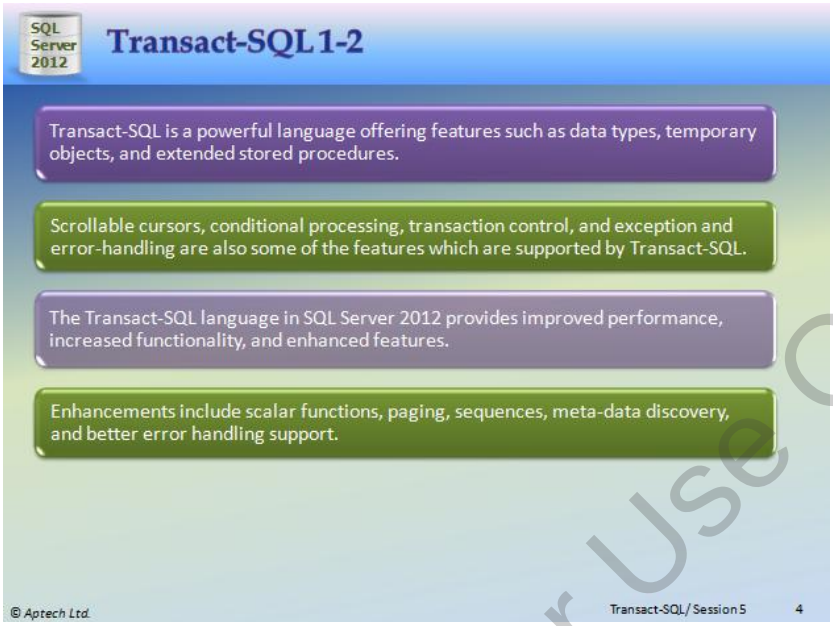
Using slide 3, explain the transact-SQL introduction.

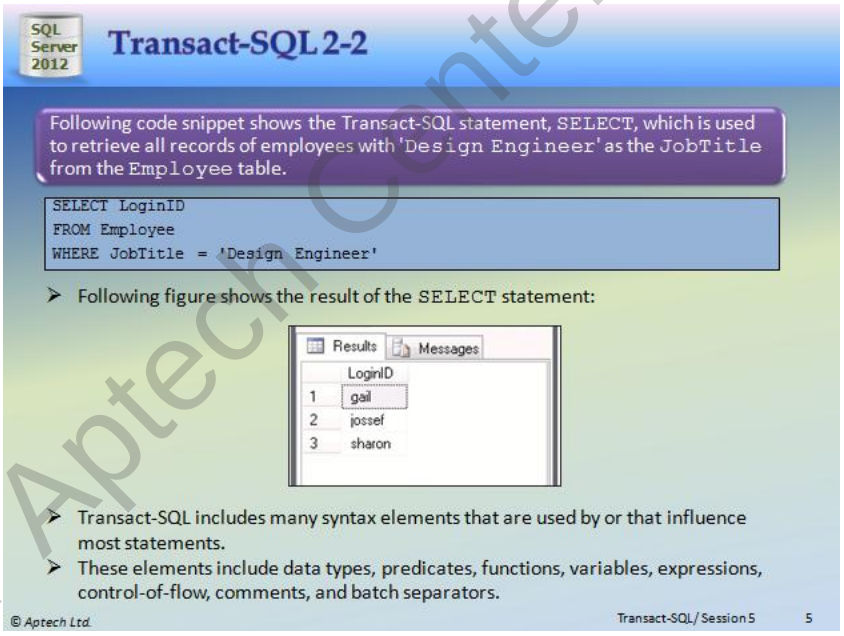
SQL is the universal language used in the database world. Most modern RDBMS products use some type of SQL dialect as their primary query language. SQL can be used to create or destroy objects, such as tables, on the database server and to do things with those objects, such as put data into them or query for data. Transact-SQL is Microsoft's implementation of the standard SQL. Usually referred to as T-SQL, this language implements a standardized way to communicate to the database.

The Transact-SQL language is an enhancement to SQL, the American National Standards Institute (ANSI) standard relational database language. It provides a comprehensive language that supports defining tables, inserting, deleting, updating, and accessing the data in the table.

Transact-SQL

Slides 4 and 5





Using slides 4 and 5, explain the transact-SQL.

Transact-SQL is a powerful language offering features such as data types, temporary objects, and extended stored procedures. Scrollable cursors, conditional processing, transaction control, and exception and error-handling are also some of the features which are supported by Transact-SQL.

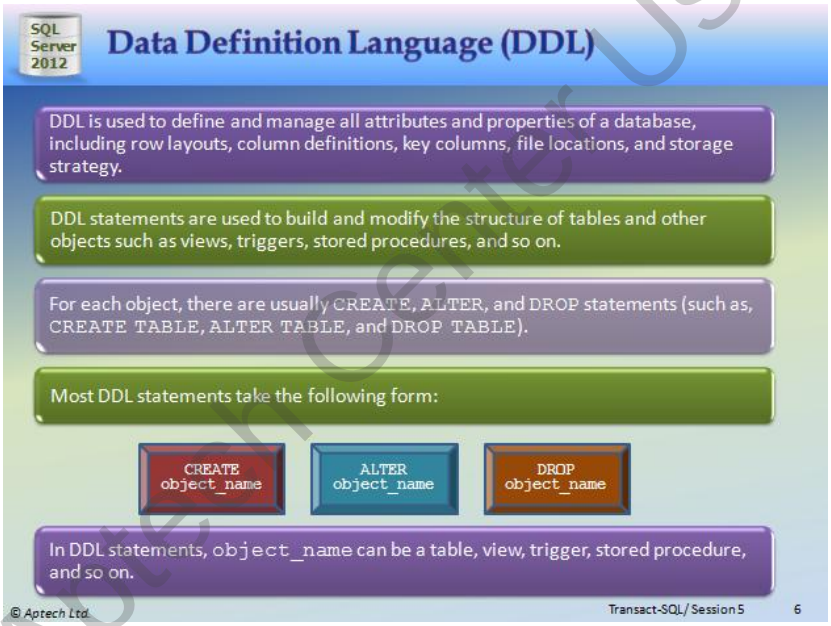
The Transact-SQL language in SQL Server 2012 provides improved performance, increased functionality, and enhanced features. Enhancements include scalar functions, paging, sequences, meta-data discovery, and better error handling support.

Explain Code Snippet shows the Transact-SQL statement, SELECT, which is used to retrieve all records of employees with 'Design Engineer' as the JobTitle from the Employee table. Figure shows the result that retrieves all records of employees with 'Design Engineer' as the JobTitle from the Employee table.

Transact-SQL includes many syntax elements that are used by or that influence most statements. These elements include data types, predicates, functions, variables, expressions, control-of-flow, comments, and batch separators.

Data Definition Language (DDL)

Slide 6



Using slide 6, explain the DDL.

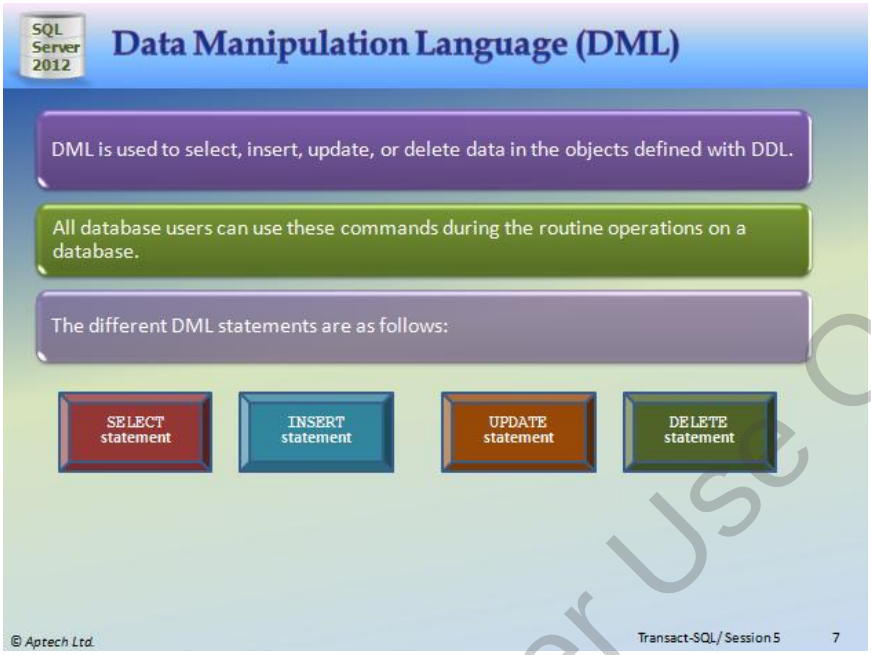
DDL, which is usually part of a DBMS, is used to define and manage all attributes and properties of a database, including row layouts, column definitions, key columns, file locations, and storage strategy. DDL statements are used to build and modify the structure of tables and other objects such as views, triggers, stored procedures, and so on. For each object, there are usually CREATE, ALTER, and DROP statements (such as, CREATE TABLE, ALTER TABLE, and DROP TABLE). Most DDL statements take the following forms:

- CREATE object_name
- ALTER object_name
- DROP object_name

In DDL statements, object_name can be a table, view, trigger, stored procedure, and so on.

Data Manipulation Language (DML)

Slide 7



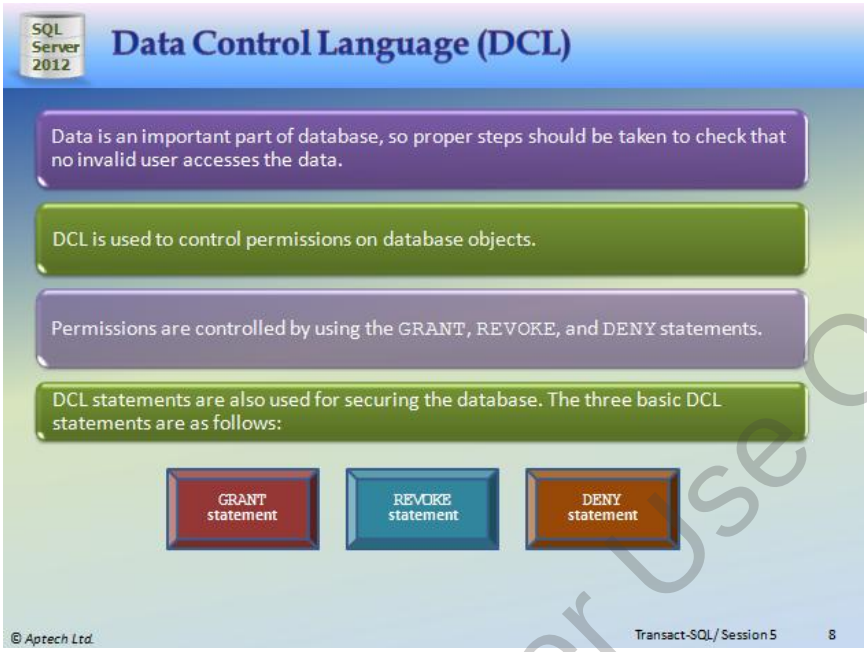
Using slide 7, explain the DML.

DML is used to select, insert, update, or delete data in the objects defined with DDL. All database users can use these commands during the routine operations on a database. The different DML statements are as follows:

- SELECT statement
- INSERT statement
- UPDATE statement
- DELETE statement

Data Control Language (DCL)

Slide 8



Using slide 8, explain the DCL.

Data is an important part of database, so proper steps should be taken to check that no invalid user accesses the data. Data control language is used to control permissions on database objects. Permissions are controlled by using the GRANT, REVOKE, and DENY statements. DCL statements are also used for securing the database. The three basic DCL statements are as follows:

- GRANT statement
- REVOKE statement
- DENY statement

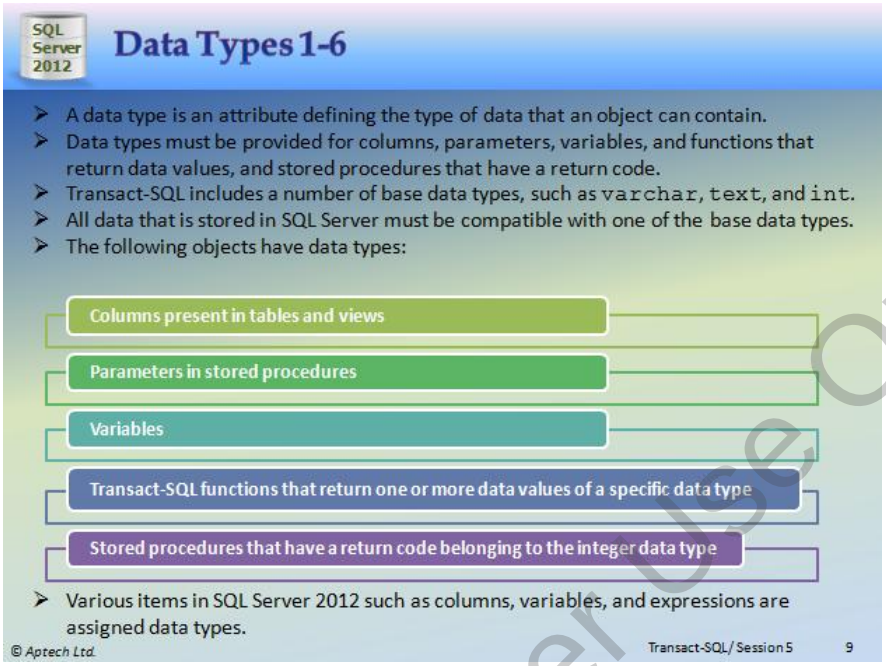
In-Class Question:

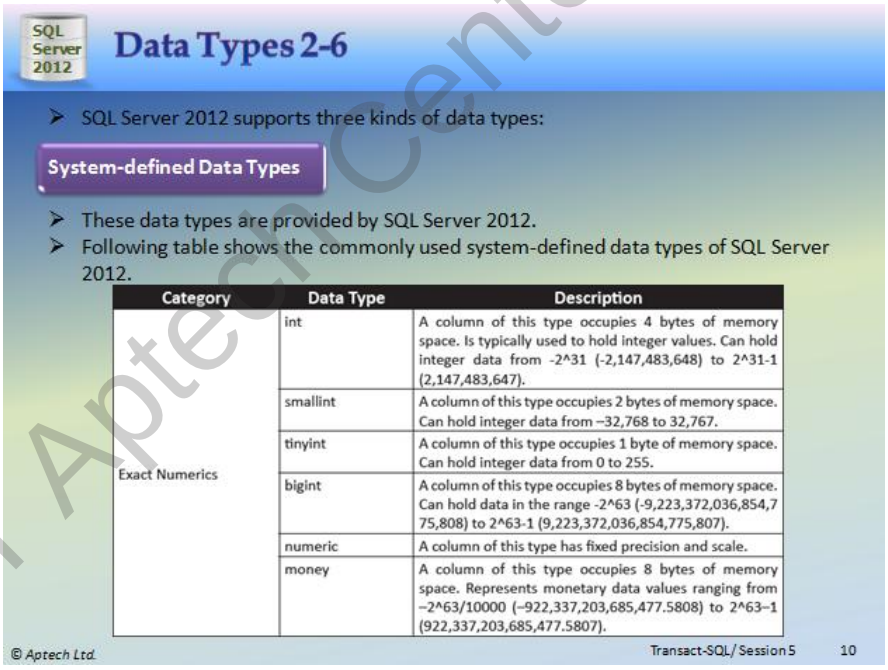
What is use of DCL?

Answer:
Permissions are controlled by using the GRANT, REVOKE, and DENY statements which are DCL statements.

Data Types in SQL

Slides 9 to 14





Data Types 3-6

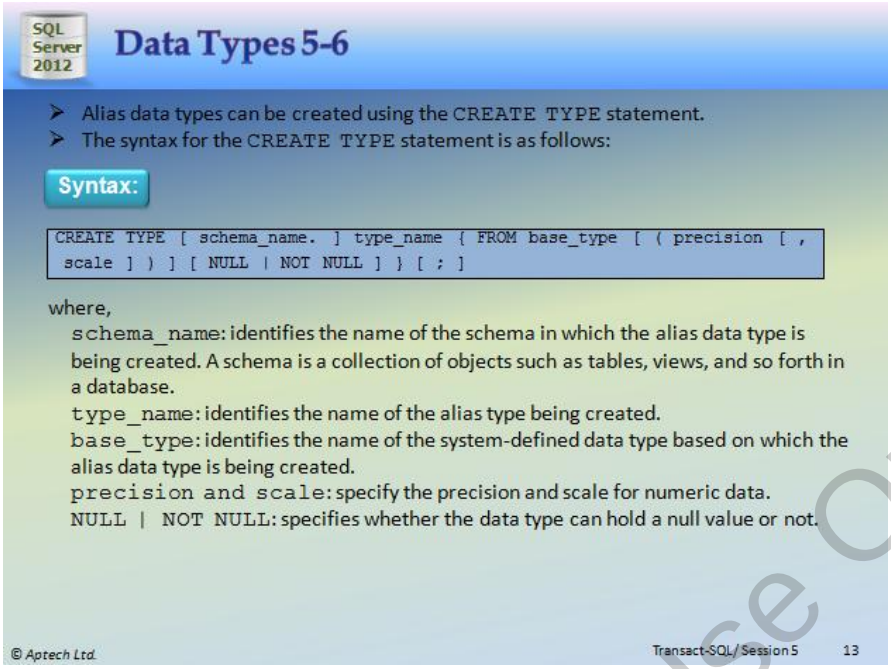
Category	Data Type	Description
Approximate Numerics	float	A column of this type occupies 8 bytes of memory space. Represents floating point number ranging from −1.79E +308 through 1.79E+308.
	real	A column of this type occupies 4 bytes of memory space. Represents floating precision number ranging from −3.40E+38 through 3.40E+38.
Date and Time	datetime	Represents date and time. Stored as two 4-byte integers.
	smalldatetime	Represents date and time.
Character String	char	Stores character data that is fixed-length and non-Unicode.
	varchar	Stores character data that is variable-length and non-Unicode with a maximum of 8,000 characters.
	text	Stores character data that is variable-length and non-Unicode with a maximum length of 2^31 - 1 (2,147,483,647) characters.
Unicode Types	nchar	Stores Unicode character data of fixed-length.
	nvarchar	Stores variable-length Unicode character data.

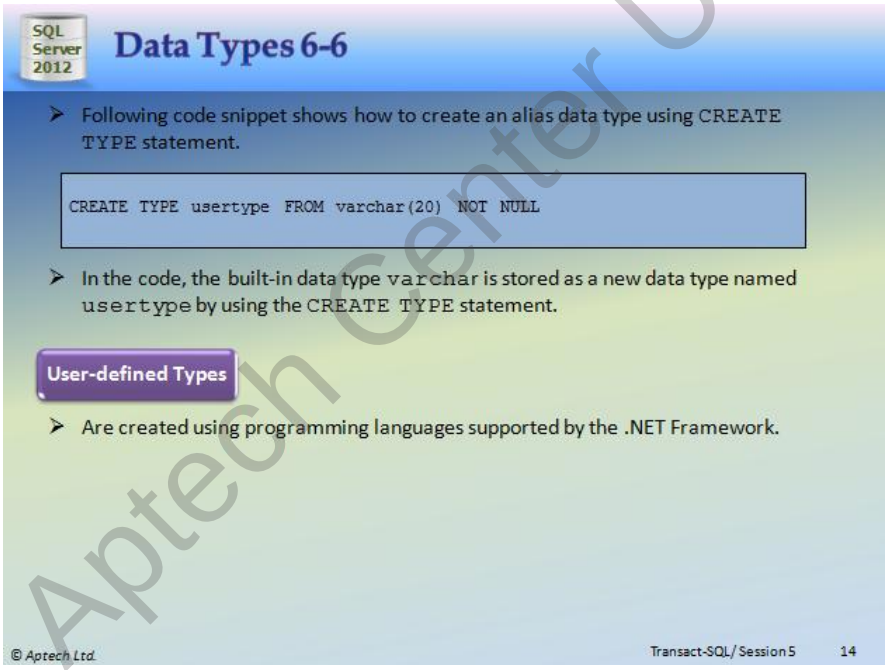
Data Types 4-6

Category	Data Type	Description
Other Data Types	timestamp	A column of this type occupies 8 bytes of memory space. Can hold automatically generated, unique binary numbers that are generated for a database.
	binary(n)	Stores fixed-length binary data with a maximum length of 8000 bytes.
	varbinary(n)	Stores variable-length binary data with a maximum length of 8000 bytes.
	image	Stores variable-length binary data with a maximum length of 2^30−1 (1,073,741,823) bytes.
	uniqueidentifier	A column of this type occupies 16 bytes of memory space. Also, stores a globally unique identifier (GUID).

Alias Data Types

- Are based on the system-supplied data types.
- Are used when more than one table stores the same type of data in a column and has similar characteristics such as length, nullability, and type.
- Can be created when it can be used commonly by all these tables.





Using slides 9 to 14, explain the data types in SQL server 2012.

A data type is an attribute defining the type of data that an object can contain. Data types must be provided for columns, parameters, variables, and functions that return data values, and stored procedures that have a return code. Transact-SQL includes a number of base data types, such as varchar, text, and int. All data that is stored in SQL Server must be compatible with one of the base data types.

Following objects have data types:

- Columns present in tables and views
- Parameters in stored procedures

- Variables
- Transact-SQL functions that return one or more data values of a specific data type
- Stored procedures that have a return code belonging to the integer data type

Various items in SQL Server 2012 such as columns, variables, and expressions are assigned data types. SQL Server 2012 supports three kinds of data types:

- **System-defined data types**: These data types are provided by SQL Server 2012. Table shows the commonly used system-defined data types of SQL Server 2012.

- **Alias data types**: These are based on the system-supplied data types. Alias data types are used when more than one table stores the same type of data in a column and has similar characteristics such as length, nullability, and type. In such cases, an alias data type can be created that can be used commonly by all these tables.

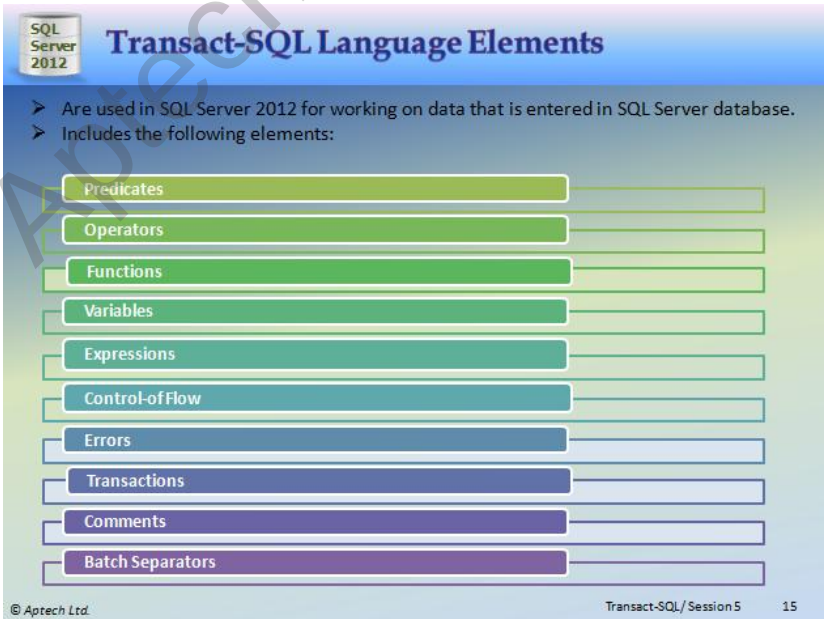
  Alias data types can be created using the CREATE TYPE statement. Explain the syntax.

- **User-defined types**: These are created using programming languages supported by the .NET Framework.

Mention varchar(max), nvarchar(max), and varbinary(max) are large value data types while text, ntext, image, varchar(max), nvarchar(max), varbinary(max), and xml are large object data types.

Transact-SQL Language Elements

Slide 15

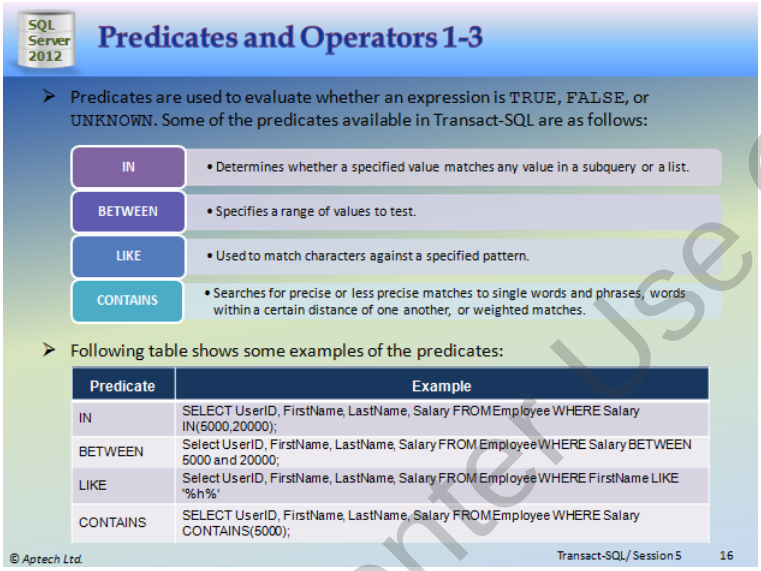


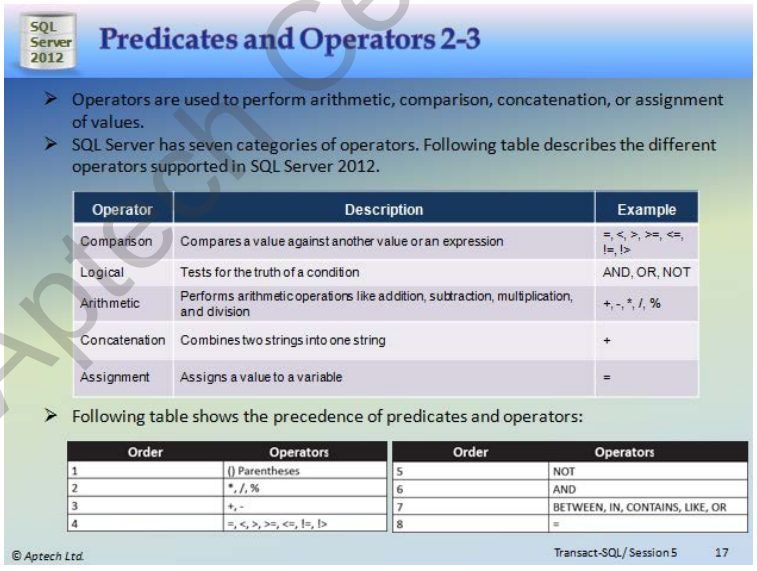
Using slide 15, explain the transact-SQL language elements.

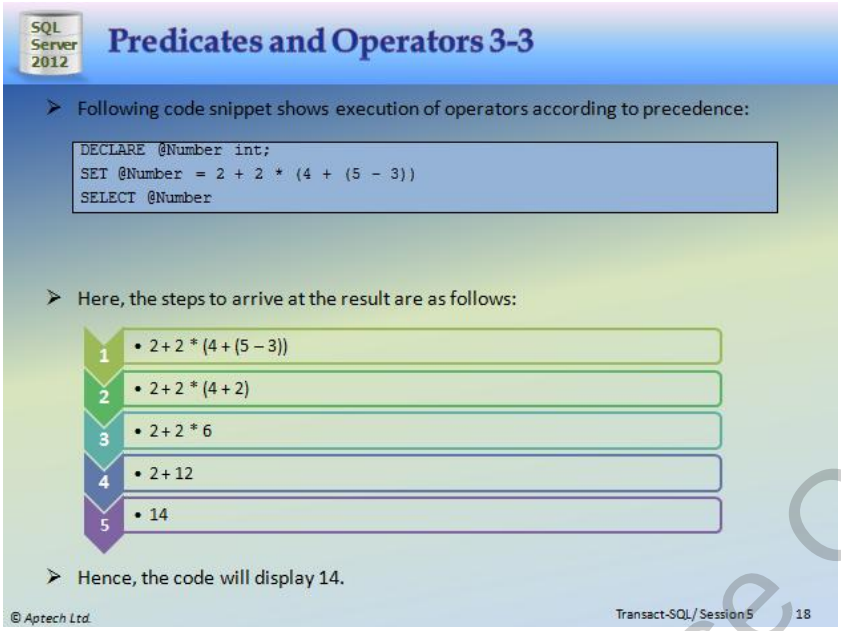
The Transact-SQL language elements are used in SQL Server 2012 for working on the data that is entered in SQL Server database. The Transact-SQL language elements includes predicates, operators, functions, variables, expressions, control-of-flow, errors, and transactions, comments, and batch separators.

Predicates and Operators

Slides 16 to 18







Using slides 16 to 18, explain the predicates and operators.

Predicates are used to evaluate whether an expression is TRUE, FALSE, or UNKNOWN. Some of the predicates available in Transact-SQL are as follows:

- **IN** - Determines whether a specified value matches any value in a subquery or a list.
- **BETWEEN** - Specifies a range of values to test.
- **LIKE** - Used to match characters against a specified pattern.
- **CONTAINS** - Searches for precise or less precise matches to single words and phrases, words within a certain distance of one another, or weighted matches.

Explain table which shows some examples of the predicates in slide 16.

Operators are used to perform arithmetic, comparison, concatenation, or assignment of values. For example, data can be tested to verify that the COUNTRY column for the customer data is populated (or has a NOT NULL value). In queries, anyone who can see the data in the table requiring an operator can perform operations. Appropriate permissions are required before data can be successfully changed. SQL Server has seven categories of operators.

Explain table which describes the different operators supported in SQL Server 2012 in slide 17.

Explain table which shows the precedence of predicates and operators in slide 17.

Explain Code Snippet which shows execution of operators according to precedence.

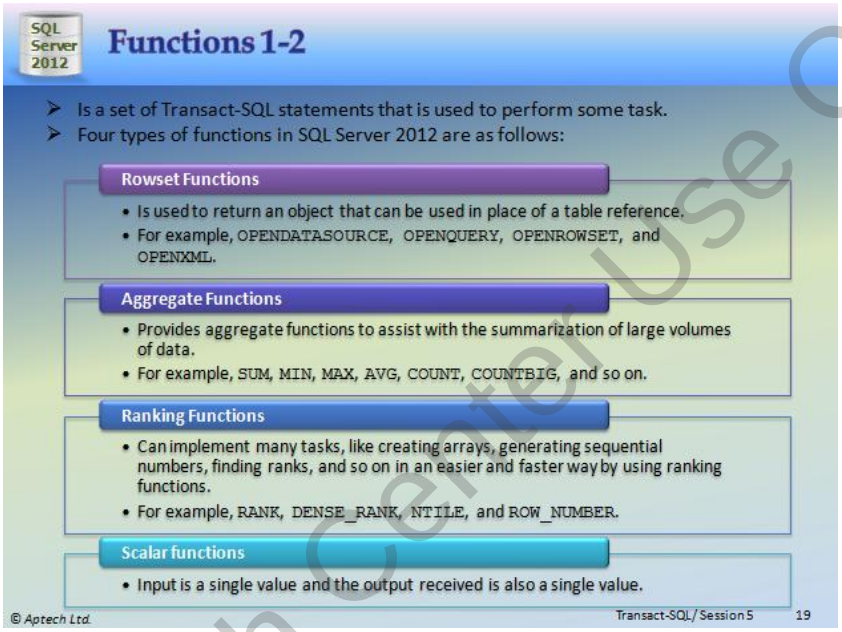
In-Class Question:

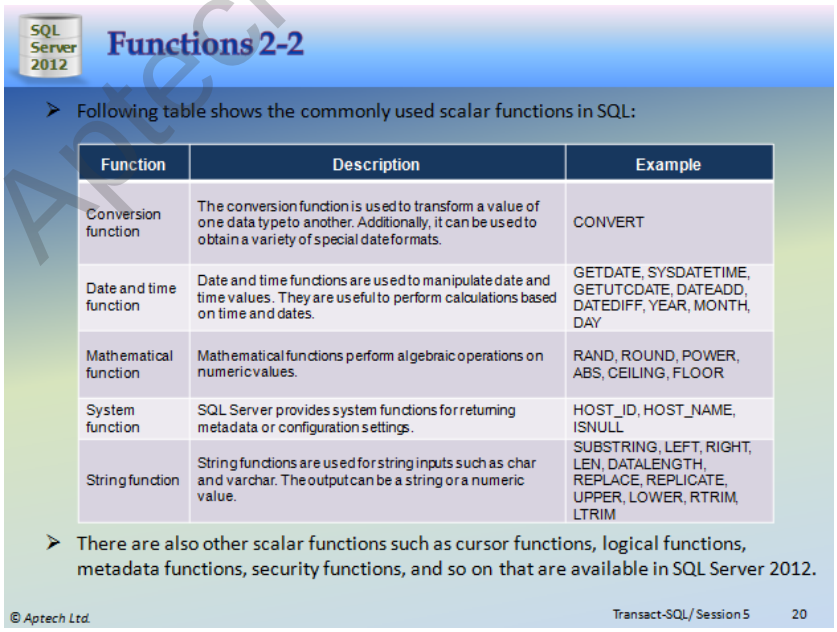
What is the IN clause used for?

Answer:
IN clause is used for specifying values which are to be compared in a subquery or a list.

Functions in SQL

Slides 19 and 20





Using slides 19 and 20, explain the functions in SQL.

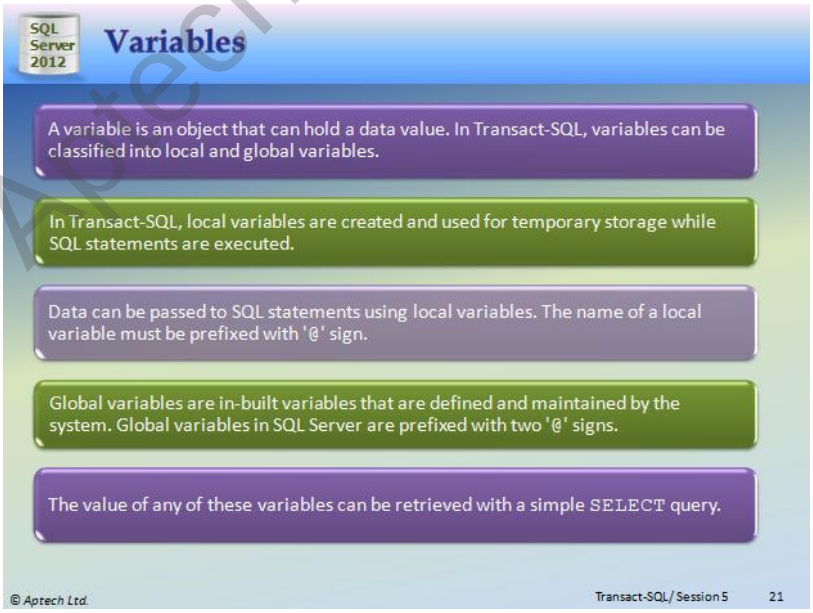
A function is a set of Transact-SQL statements that is used to perform some task. Transact-SQL includes a large number of functions. These functions can be useful when data is calculated or manipulated. In SQL, functions work on the data, or group of data, to return a required value. They can be used in a SELECT list, or anywhere in an expression. The four types of functions in SQL Server 2012 are as follows:

- **Rowset functions** - In Transact-SQL, the rowset function is used to return an object that can be used in place of a table reference. For example, OPENDATASOURCE, OPENQUERY, OPENROWSET, and OPENXML are rowset functions.
- **Aggregate functions** - Transact-SQL provides aggregate functions to assist with the summarization of large volumes of data. For example, SUM, MIN, MAX, AVG, COUNT, COUNTBIG, and so on are aggregate functions.
- **Ranking functions** - Many tasks, such as creating arrays, generating sequential numbers, finding ranks, and so on can be implemented in an easier and faster way by using ranking functions. For example, RANK, DENSE_RANK, NTILE, and ROW_NUMBER are ranking functions.
- **Scalar functions** - In scalar functions, the input is a single value and the output received is also a single value.

Explain the commonly used scalar functions in SQL as shown in table in slide20.
There are also other scalar functions such as cursor functions, logical functions, metadata functions, security functions, and so on that are available in SQL Server 2012.
There can also be user defined functions in SQL server which we will study in details in further session.

Variables

Slide 21



Using slide 21, explain the variables.
A variable is an object that can hold a data value. In Transact-SQL, variables can be classified into local and global variables.

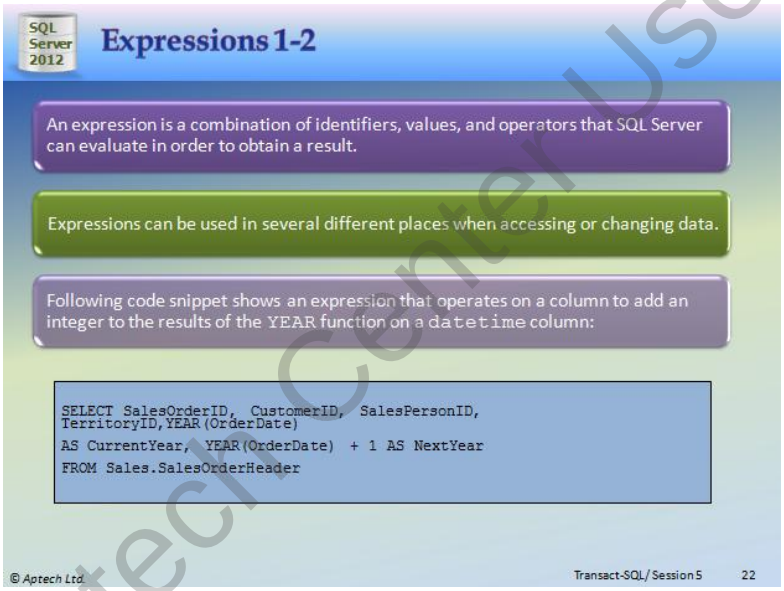
In Transact-SQL, local variables are created and used for temporary storage while SQL statements are executed. Data can be passed to SQL statements using local variables. The name of a local variable must be prefixed with '@' sign.

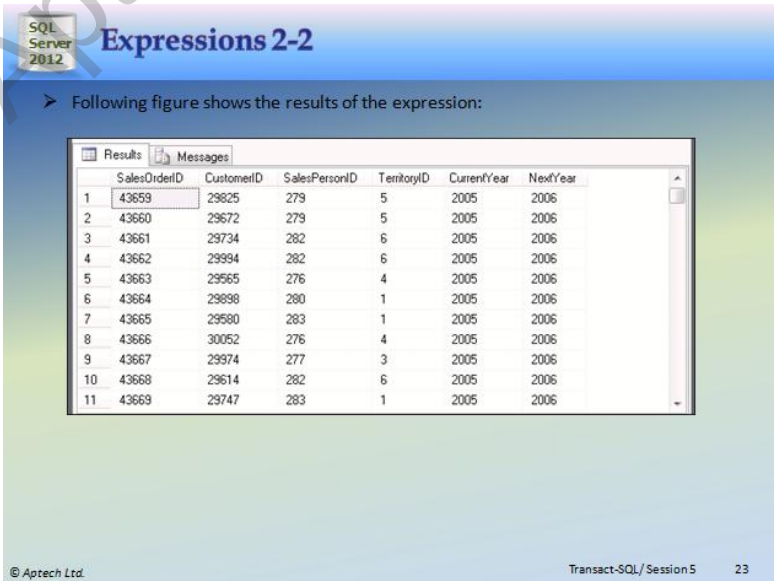
Global variables are in-built variables that are defined and maintained by the system. Global variables in SQL Server are prefixed with two '@' signs. The value of any of these variables can be retrieved with a simple SELECT query.

For example, @Rowcount is the number of rows affected or read while @row will be the local variable in SQL.

Expressions

Slides 22 and 23





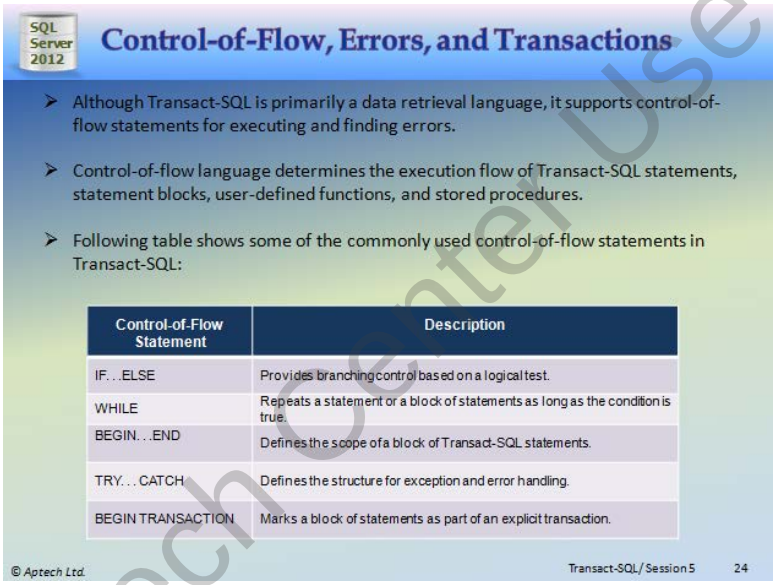
Using slides 22 and 23, explain the expressions.

An expression is a combination of identifiers, values, and operators that SQL Server can evaluate in order to obtain a result. Expressions can be used in several different places when accessing or changing data.

Explain Code Snippet which shows an expression that operates on a column to add an integer to the results of the YEAR function on a datetime column. Figure shows the results of the expression.

Control-of-Flow, Errors, and Transactions

Slide 24



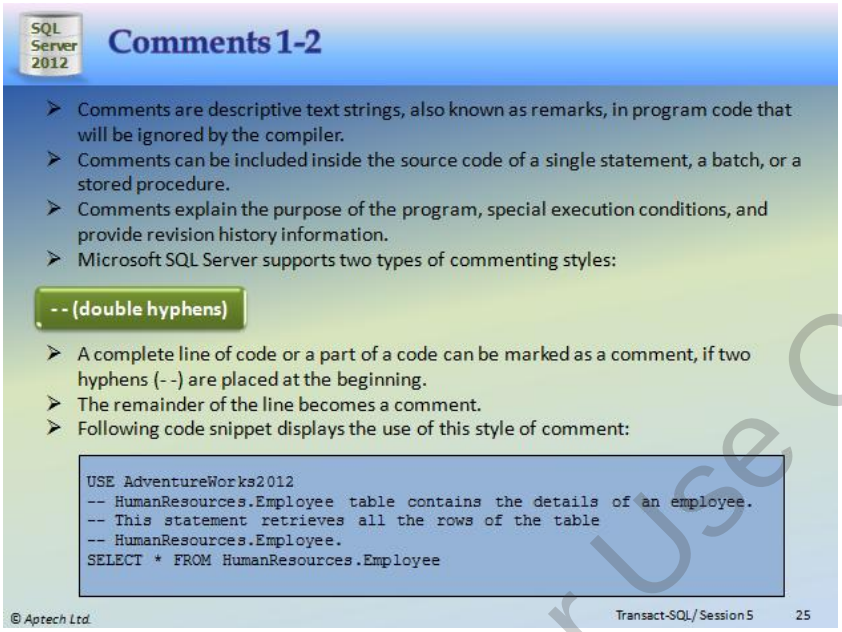
Using slide 24, explain the control-flow, errors, and transactions in SQL server.

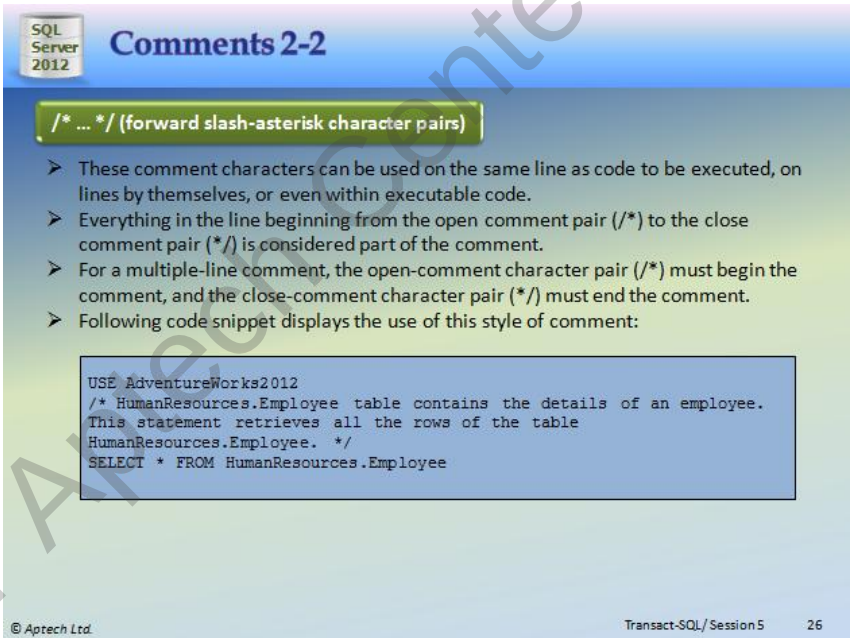
Although Transact-SQL is primarily a data retrieval language, it supports control-of-flow statements for executing and finding errors. Control-of-flow language determines the execution flow of Transact-SQL statements, statement blocks, user-defined functions, and stored procedures.

Table shows some of the commonly used control-of-flow statements in Transact-SQL.

Comments

Slides 25 and 26





Using slides 25 and 26, explain the comments in SQL server.

Comments are descriptive text strings, also known as remarks, in program code that will be ignored by the compiler. Comments can be included inside the source code of a single statement, a batch, or a stored procedure. Comments explain the purpose of the program, special execution conditions, and provide revision history information.

Microsoft SQL Server supports two types of commenting styles:

- **- (double hyphens)**: A complete line of code or a part of a code can be marked as a comment, if two hyphens (- -) are placed at the beginning. The remainder of the line becomes a comment. Explain Code Snippet displays the use of this style of comment.
- **/* … */ (forward slash-asterisk character pairs)**: These comment characters can be used on the same line as code to be executed, on lines by themselves, or even within executable code. Everything in the line beginning from the open comment pair (/*) to the close comment pair (*/) is considered part of the comment. For a multiple-line comment, the open-comment character pair (/*) must begin the comment, and the close-comment character pair (*/) must end the comment. Explain Code Snippet which displays the use of this style of comment.

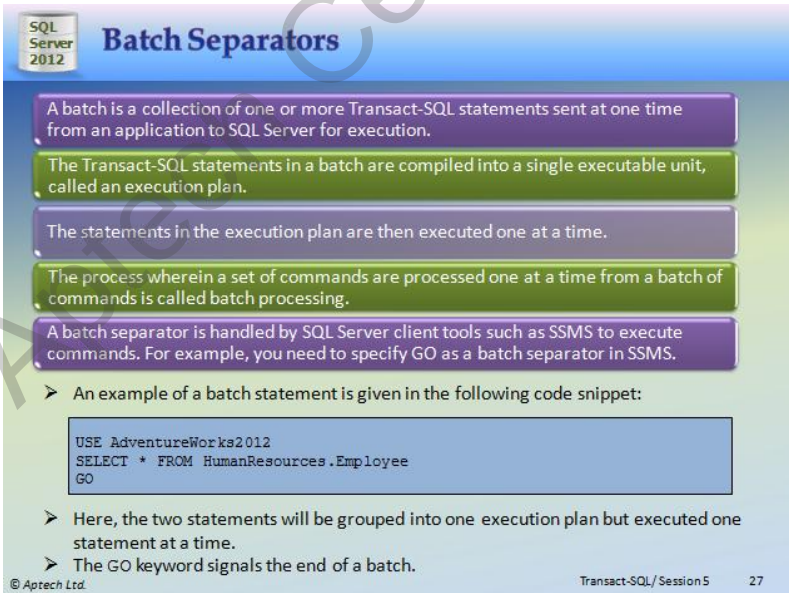
In-Class Question:

What symbol is used to comment a single line of code in SQL server?

Answer:
A complete line of code or a part of a code can be marked as a comment, if two hyphens (- -) are placed at the beginning.

Batch Separators

Slide 27



Using slide 27, explain the batch separators.

A batch is a collection of one or more Transact-SQL statements sent at one time from an application to SQL Server for execution. The Transact-SQL statements in a batch are compiled into a single executable unit, called an execution plan. The statements in the

execution plan are then executed one at a time. The process wherein a set of commands are processed one at a time from a batch of commands is called batch processing.

A batch separator is handled by SQL Server client tools such as SSMS to execute commands. For example, you need to specify GO as a batch separator in SSMS.

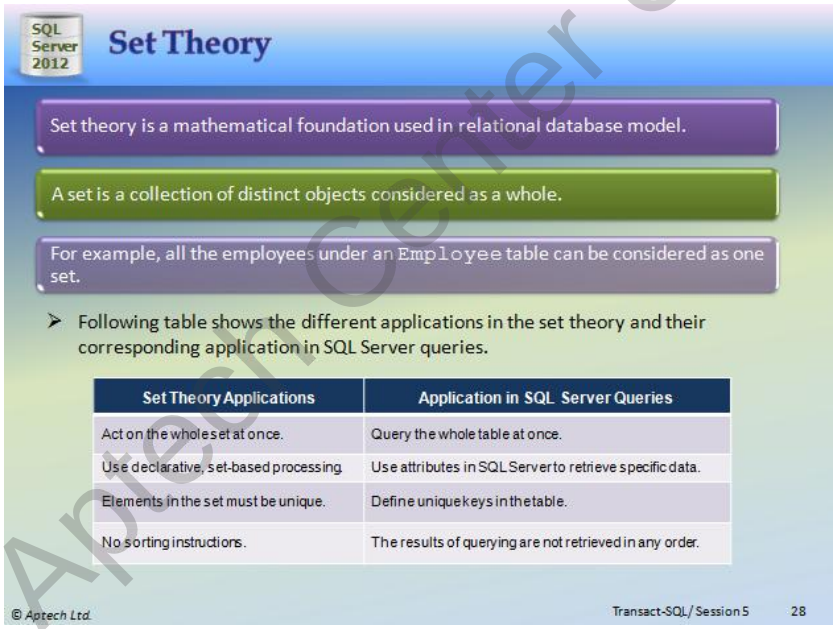
Explain the example of a batch statement is given in Code Snippet.

In Code Snippet, the two statements will be grouped into one execution plan, but executed one statement at a time. The GO keyword signals the end of a batch.
GO [count]

Here, the COUNT is a positive integer. The batch preceding GO will execute the specified number of times.

Set Theory

Slide 28



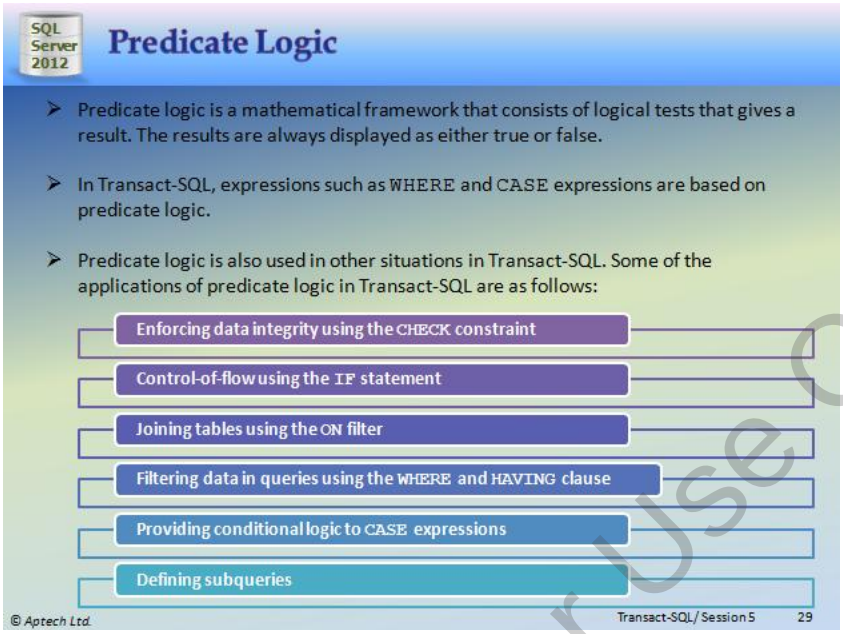
Using slide 28, explain the set theory.

Sets and Predicate Logic are the two mathematical fundamentals that are used in SQL Server 2012. Both these theories are used in querying of data in SQL Server 2012.
Set theory is a mathematical foundation used in relational database model. A set is a collection of distinct objects considered as a whole. For example, all the employees under an Employee table can be considered as one set. The employees are the different objects that form a part of the set in the Employee table.

Explain that joins are used for the operations on tables as shown in the set theory. Example of full outer join in SQL is like Union operation in set theory.

Predicate Logic

Slide 29



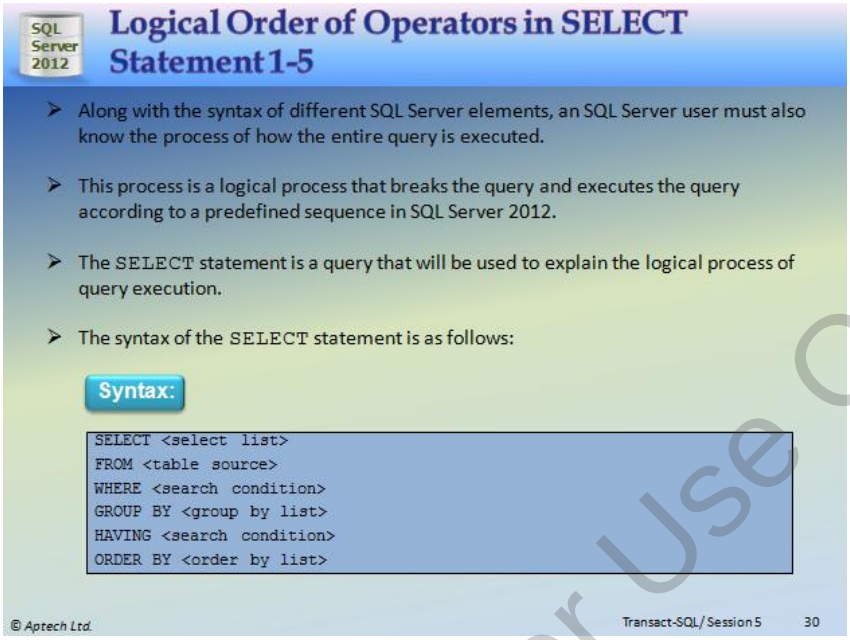
Using slide 29, explain the predicate logic.

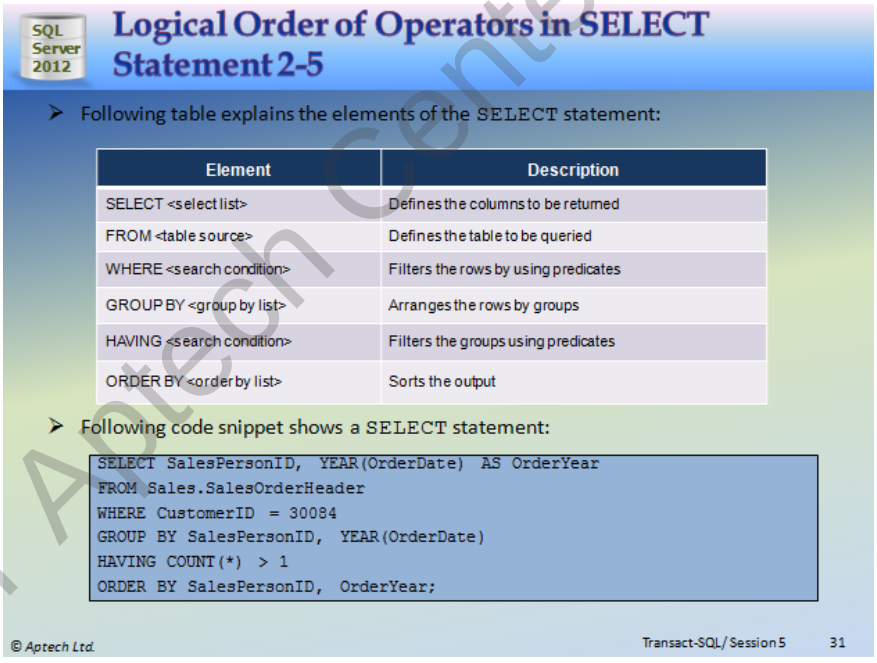
Predicate logic is a mathematical framework that consists of logical tests that gives a result. The results are always displayed as either true or false. In Transact-SQL, expressions such as WHERE and CASE expressions are based on predicate logic. Predicate logic is also used in other situations in Transact-SQL. Some of the applications of predicate logic in Transact-SQL are as follows:

- Enforcing data integrity using the CHECK constraint
- Control-of-flow using the IF statement
- Joining tables using the ON filter
- Filtering data in queries using the WHERE and HAVING clause
- Providing conditional logic to CASE expressions
- Defining subqueries

Logical Order of Operators in SELECT Statement

Slides 30 to 34



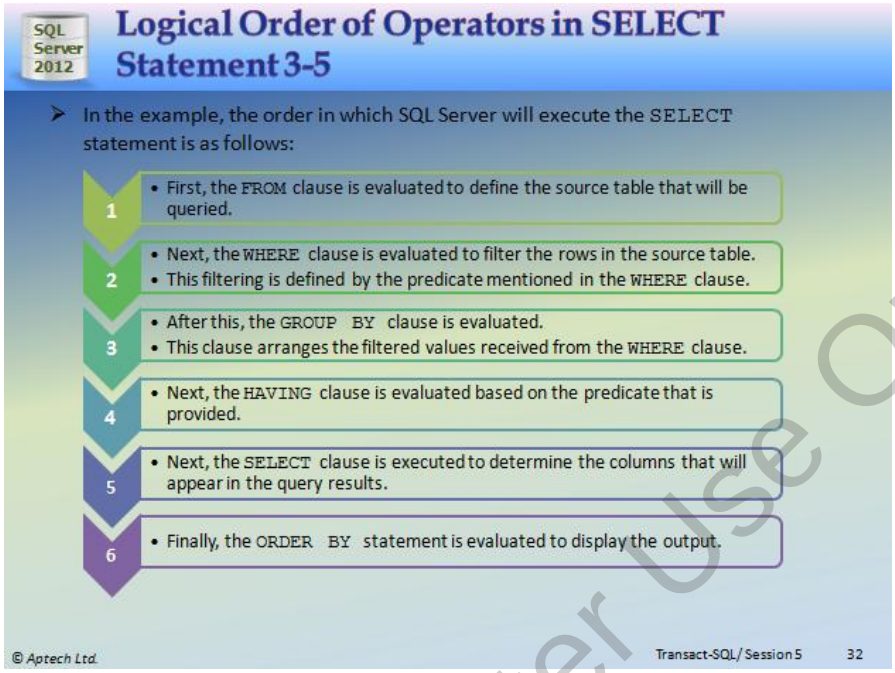


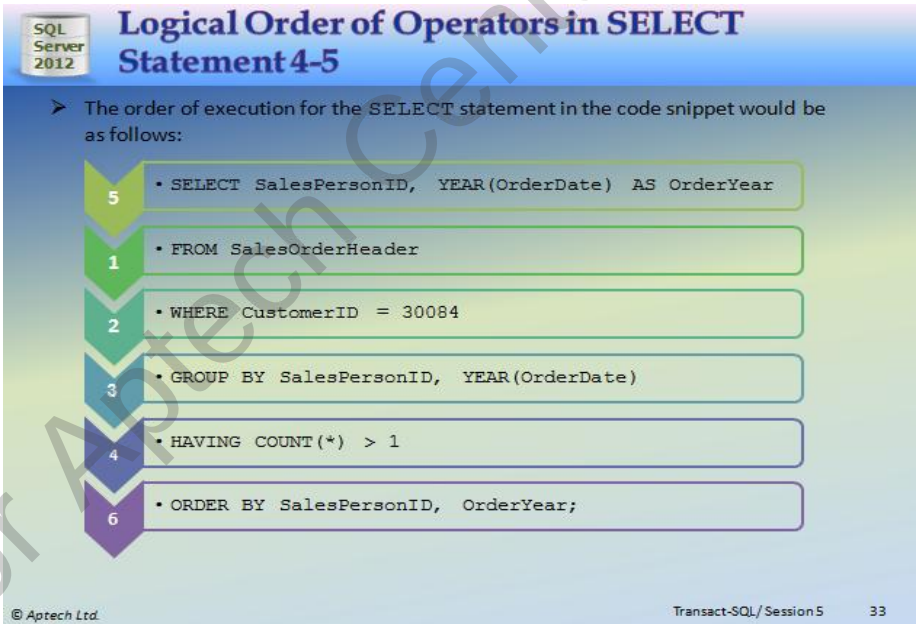
Using slides 30 and 31, explain the logical order of operators in select statement.

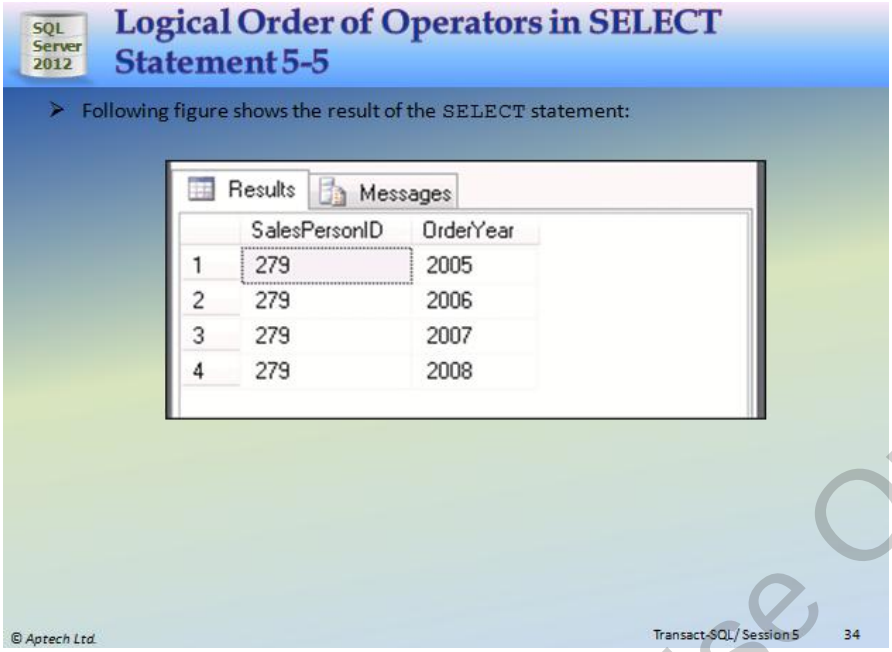
Along with the syntax of different SQL Server elements, an SQL Server user must also know the process of how the entire query is executed. This process is a logical process that breaks the query and executes the query according to a predefined sequence in SQL Server 2012. The SELECT statement is a query that will be used to explain the logical process of query execution.

Table explains the elements of the SELECT statement.

Explain Code Snippet which shows a SELECT statement in slide 31.







Using slides 32 to 34, explain the order in which SQL Server will execute the SELECT statement.

In the example, the order in which SQL Server will execute the SELECT statement is as follows:

- First, the FROM clause is evaluated to define the source table that will be queried.
- Next, the WHERE clause is evaluated to filter the rows in the source table. This filtering is defined by the predicate mentioned in the WHERE clause.
- After this, the GROUP BY clause is evaluated. This clause arranges the filtered values received from the WHERE clause.
- Next, the HAVING clause is evaluated based on the predicate that is provided. Next, the SELECT clause is executed to determine the columns that will appear in the query results.
- Finally, the ORDER BY statement is evaluated to display the output.

The order of execution for the SELECT statement in Code Snippet would be as follows:

1. SELECT SalesPersonID, YEAR(OrderDate) AS OrderYear
2. FROM SalesOrderHeader
3. WHERE CustomerID = 30084
4. GROUP BY SalesPersonID, YEAR(OrderDate)
5. HAVING COUNT(*) > 1
6. ORDER BY SalesPersonID, OrderYear;

Figure shows the result of the SELECT statement.

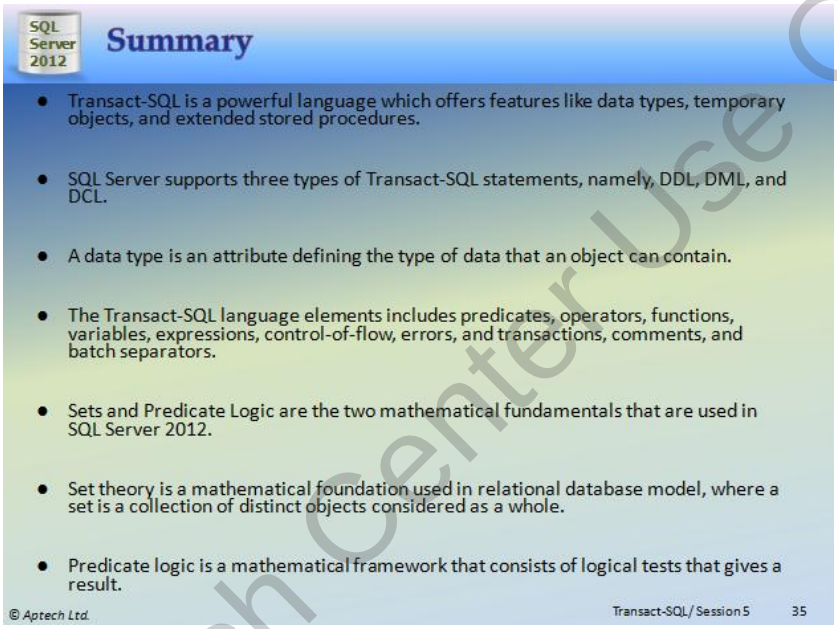
The query may have some other operations and thus it will be processed in the following order:

1. FROM
2. ON
3. OUTER

4. WHERE
5. GROUP BY
6. CUBE | ROLLUP
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY
11. TOP

Summarize Session

Slide 35



Using slide 35, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- Transact-SQL is a powerful language which offers features such as data types, temporary objects, and extended stored procedures.
- SQL Server supports three types of Transact-SQL statements, namely, DDL, DML, and DCL.
- A data type is an attribute defining the type of data that an object can contain.
- The Transact-SQL language elements includes predicates, operators, functions, variables, expressions, control-of-flow, errors, and transactions, comments, and batch separators.
- Set and Predicate Logic are the two mathematical fundamentals that are used in SQL Server 2012.

- Set theory is a mathematical foundation used in relational database model, where a set is a collection of distinct objects considered as a whole.
- Predicate logic is a mathematical framework that consists of logical tests that gives a result.

5.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Creating and Managing Databases topic that is offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 6 – Creating and Managing Databases

6.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

6.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe system and user-defined databases
- List the key features of the AdventureWorks2012 sample database
- Describe adding of filegroups and transaction logs
- Describe the procedure to create a database
- List and describe types of database modifications
- Describe the procedure to drop a database
- Describe database snapshots

6.1.2 Teaching Skills

To teach this session, you should be well-versed with system and user defined databases. The key features of the AdventureWorks2012 database should also be known. The session also covers types of database modification.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slide and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

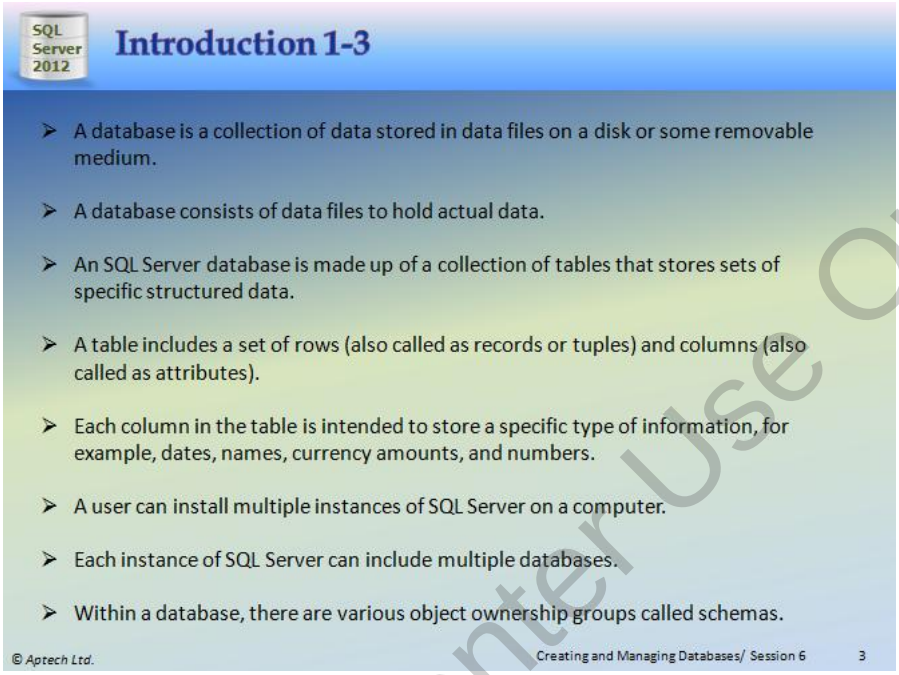
In-Class Activities:
Follow the order given here during In-Class activities.

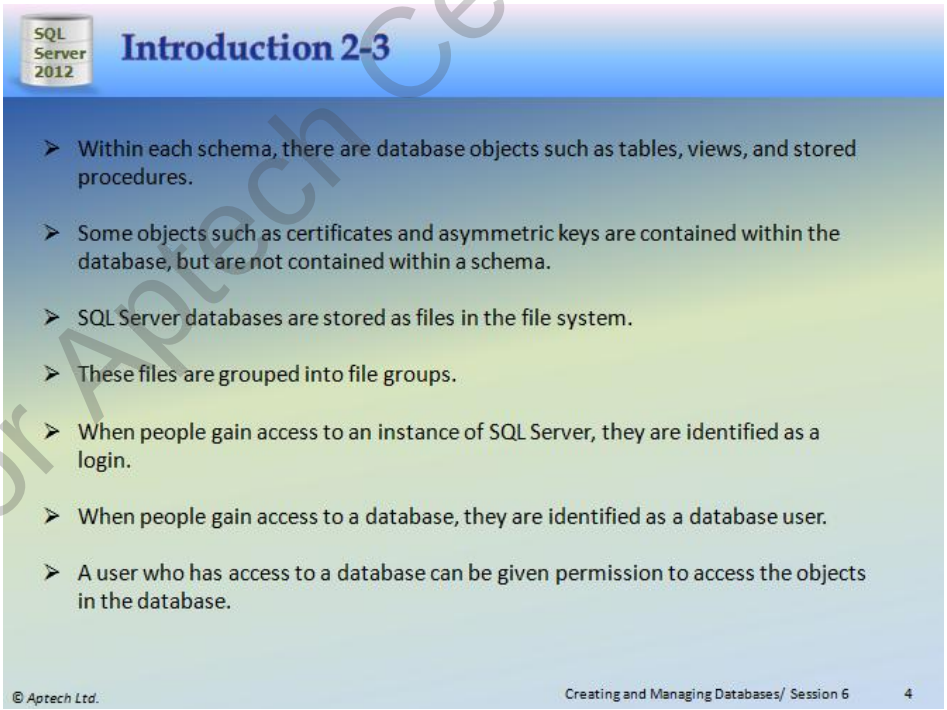
Overview of the Session:
Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces system and user defined databases. They will learn about the key features of the AdventureWorks2012 database. They will also know about types of database modification.

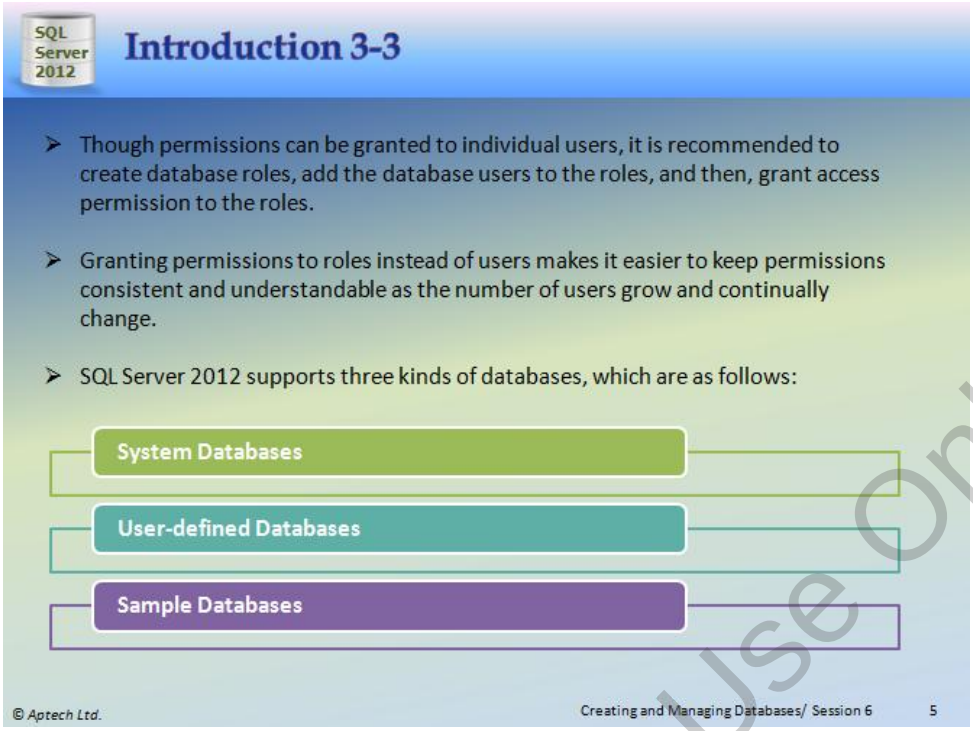
6.2 In-Class Explanations

Introduction

Slides 3 to 5







Using slides 3 to 5, explain the concept of database.

A database is a collection of data stored in data files on a disk or some removable medium. A database consists of data files to hold actual data.

An SQL Server database is made up of a collection of tables that stores sets of specific structured data. A table includes a set of rows (also called as records or tuples) and columns (also called as attributes). Each column in the table is intended to store a specific type of information, for example, dates, names, currency amounts, and numbers.

A user can install multiple instances of SQL Server on a computer. Each instance of SQL Server can include multiple databases. Within a database, there are various object ownership groups called schemas. Within each schema, there are database objects such as tables, views, and stored procedures. Some objects such as certificates and asymmetric keys are contained within the database, but are not contained within a schema.

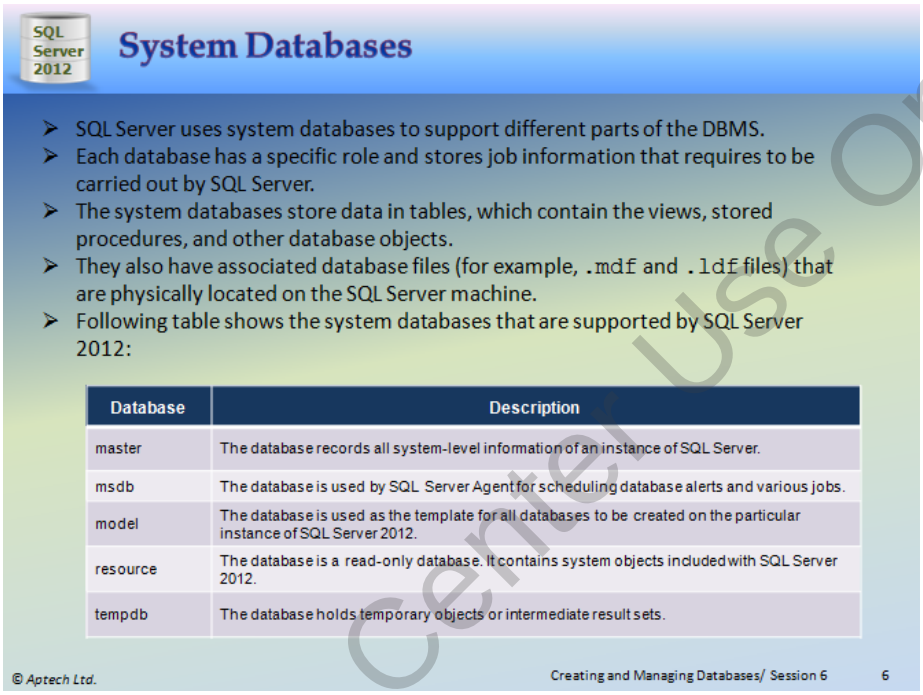
SQL Server databases are stored as files in the file system. These files are grouped into file groups. When people gain access to an instance of SQL Server, they are identified as a login. When people gain access to a database, they are identified as a database user.
A user who has access to a database can be given permission to access the objects in the database. Though permissions can be granted to individual users, it is recommended to create database roles, add the database users to the roles, and then, grant access permission to the roles. Granting permissions to roles instead of users makes it easier to keep permissions consistent and understandable as the number of users grow and continually change.

SQL Server 2012 supports three kinds of databases, which are as follows:

- System Databases
- User-defined Databases
- Sample Databases

System Databases

Slide 6



Using slide 6, explain the system databases.

SQL Server uses system databases to support different parts of the DBMS. Each database has a specific role and stores job information that requires to be carried out by SQL Server. The system databases store data in tables, which contain the views, stored procedures, and other database objects. They also have associated database files (for example, .mdf and .ldf files) that are physically located on the SQL Server machine.

Explain table which shows the system databases that are supported by SQL Server 2012.

In-Class Question:

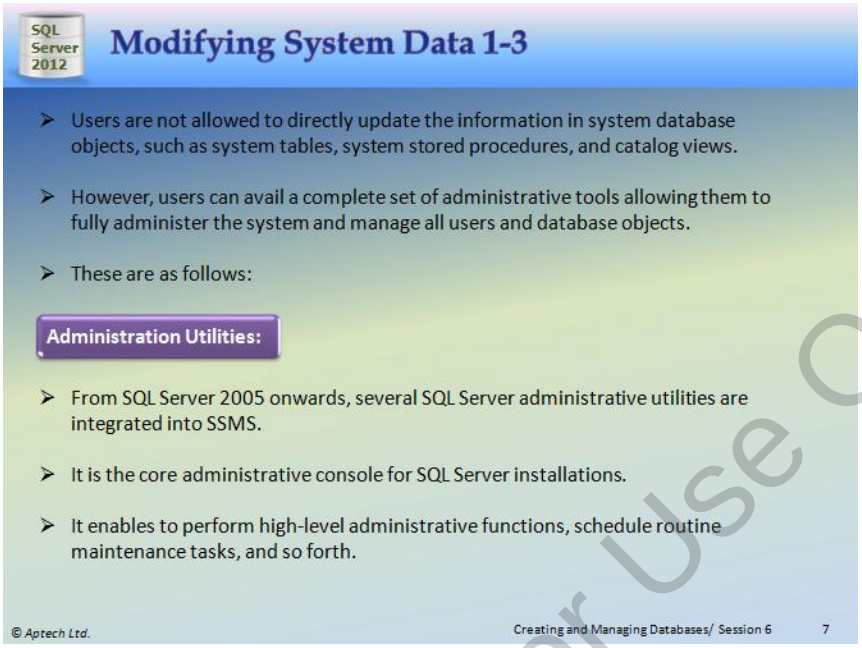
What is the use of system databases?

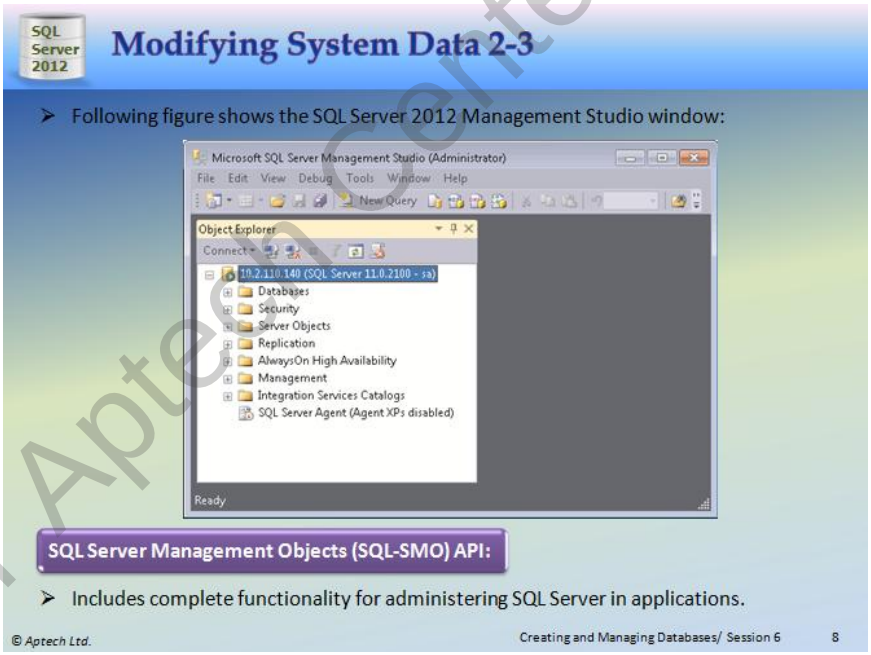
Answer:

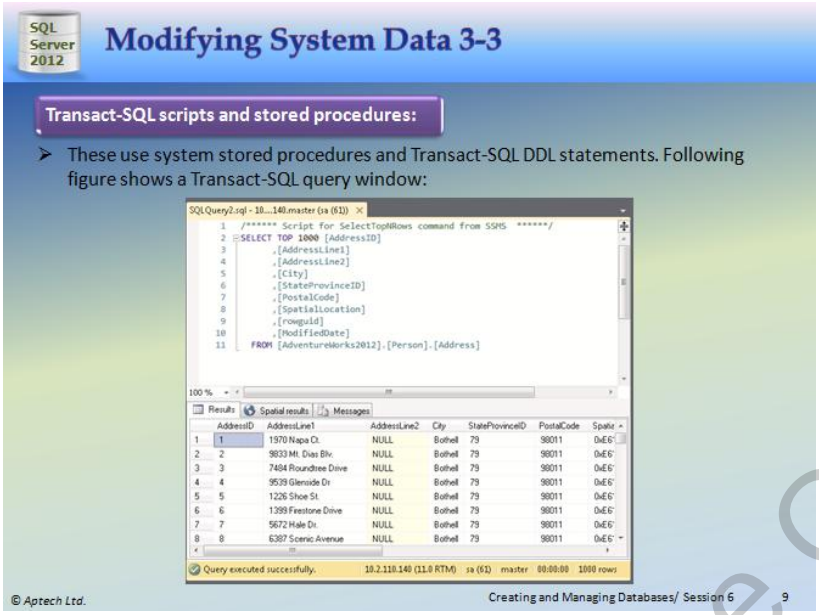
SQL Server uses system databases to support different parts of the DBMS. Each database has a specific role and stores job information that requires to be carried out by SQL Server.

Modifying System Data

Slides 7 to 9







Using slides 7 to 9, explain how to modify system data.

Users are not allowed to directly update the information in system database objects, such as system tables, system stored procedures, and catalog views. However, users can avail a complete set of administrative tools allowing them to fully administer the system and manage all users and database objects. These are as follows:

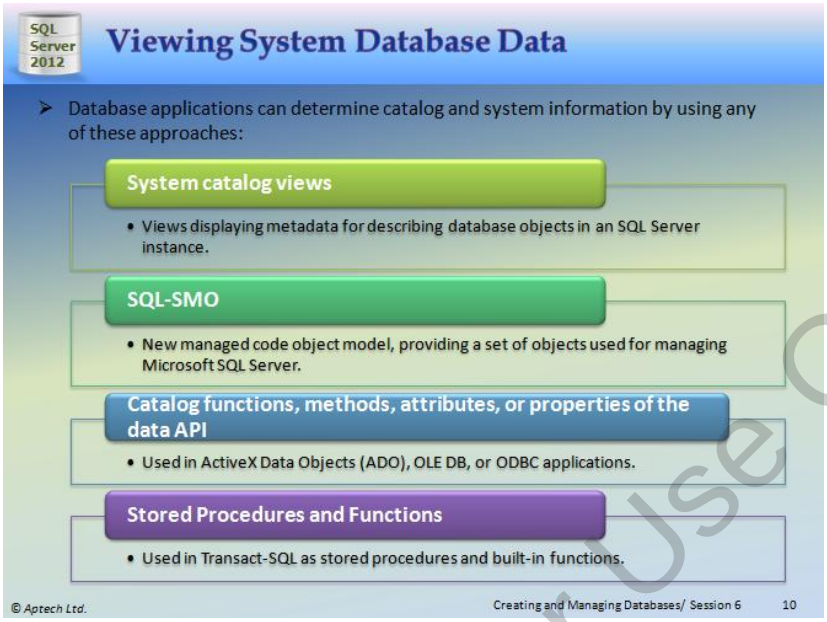
- **Administration utilities**: From SQL Server 2005 onwards, several SQL Server administrative utilities are integrated into SSMS. It is the core administrative console for SQL Server installations. It enables to perform high-level administrative functions, schedule routine maintenance tasks, and so forth. Figure shows the SQL Server 2012 Management Studio window.
- **SQL Server Management Objects (SQL-SMO) API**: Includes complete functionality for administering SQL Server in applications.
- **Transact-SQL scripts and stored procedures:** These use system stored procedures and Transact- SQL DDL statements. Figure shows a Transact-SQL query window.

These tools also guard applications from making changes in the system objects.

Tips:
SQL Server does not support triggers defined on the system tables, because they might modify the operation of the system.

Viewing System Database Data

Slide 10



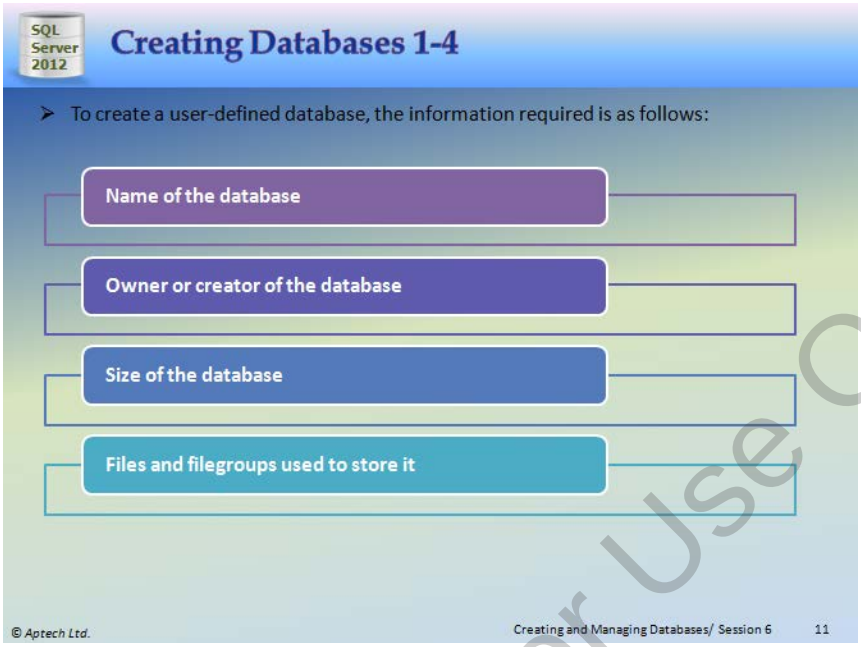
Using slide 10, explain methods to view system database data.

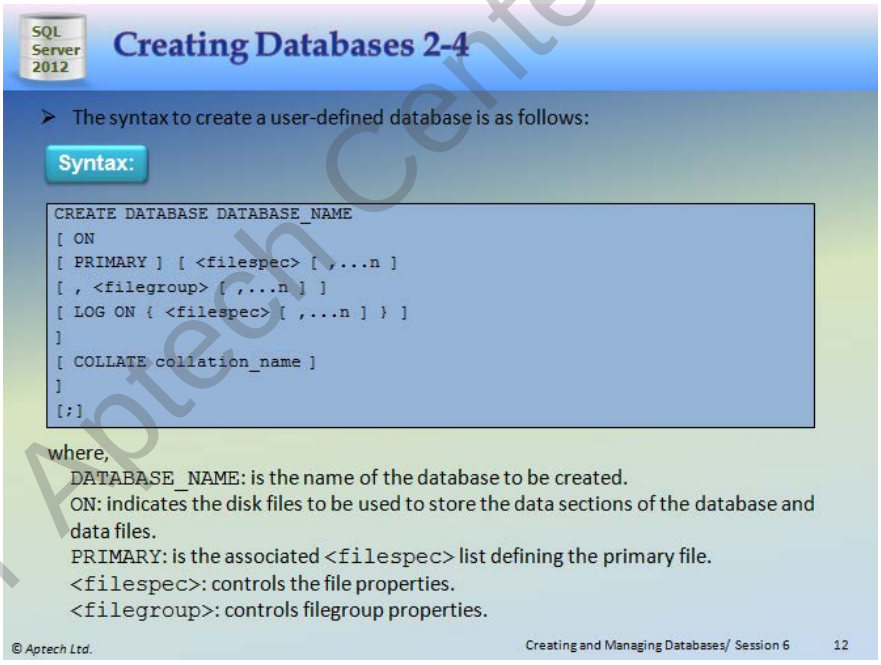
Database applications can determine catalog and system information by using any of these approaches:

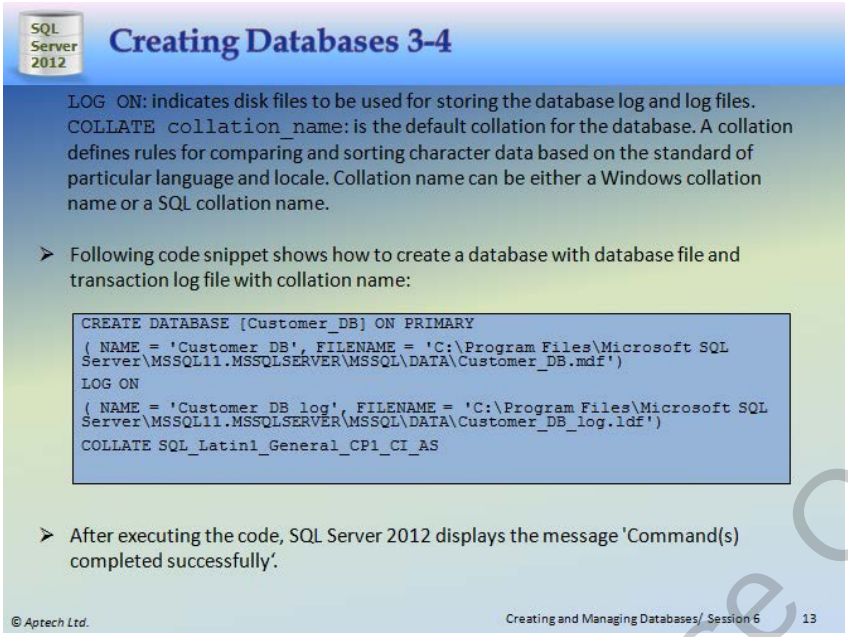
- **System catalog views**: Views displaying metadata for describing database objects in an SQL Server instance.
- **SQL-SMO**: New managed code object model, providing a set of objects used for managing Microsoft SQL Server.
- **Catalog functions, methods, attributes, or properties of the data API**: Used in ActiveX Data Objects (ADO), OLE DB, or ODBC applications.
- **Stored Procedures and Functions**: Used in Transact-SQL as stored procedures and built-in functions.

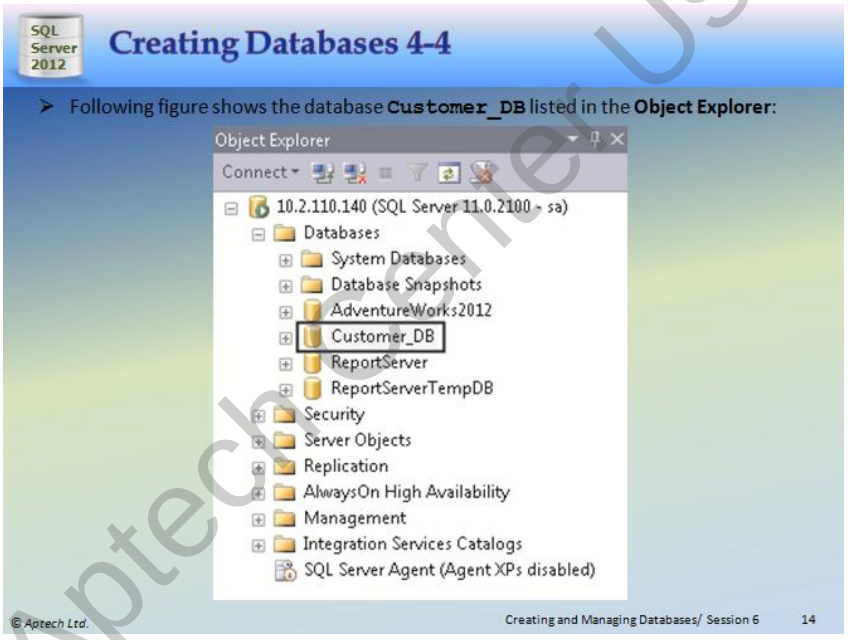
Creating Databases

Slides 11 to 14









Using slides 11 to 14, explain how to create database.

Using SQL Server 2012, users can create their own databases, also called user-defined databases, and work with them. The purpose of these databases is to store user data.

To create a user-defined database, the information required is as follows:

- Name of the database
- Owner or creator of the database
- Size of the database
- Files and filegroups used to store it

Explain the syntax to create a user-defined database using slide 12.

Explain Code Snippet which shows how to create a database with database file and transaction log file with collation name.

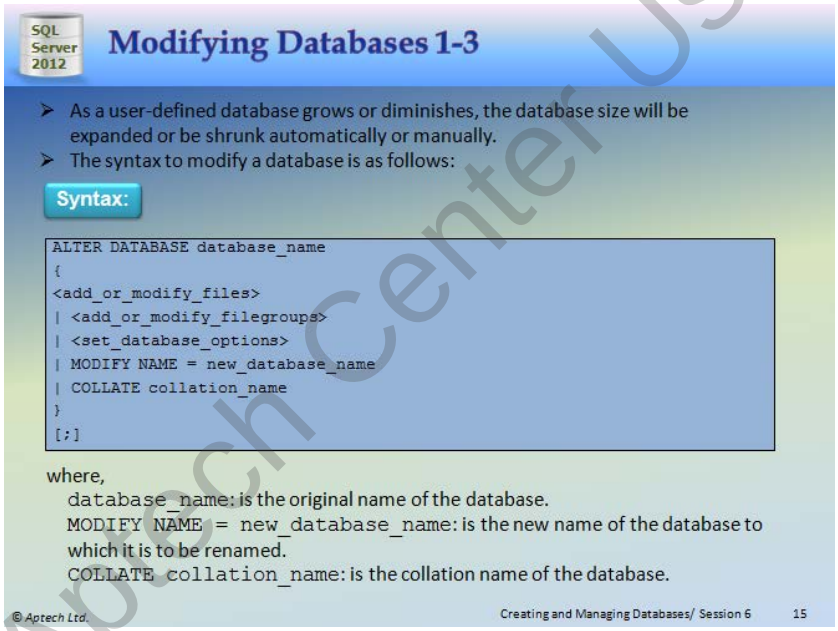
After executing the code in Code Snippet, SQL Server 2012 displays the message 'Command(s) completed successfully'.

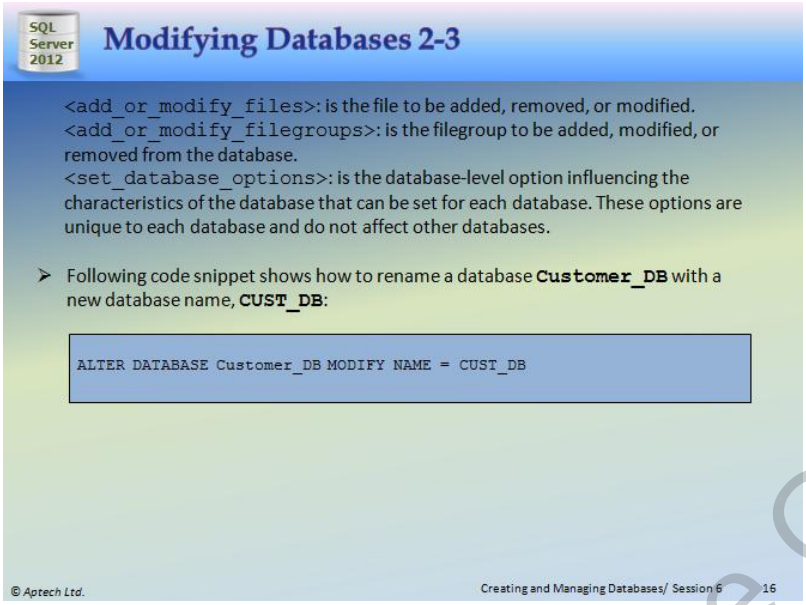
The figure in slide 14 shows the database Customer_DB listed in the Object Explorer.

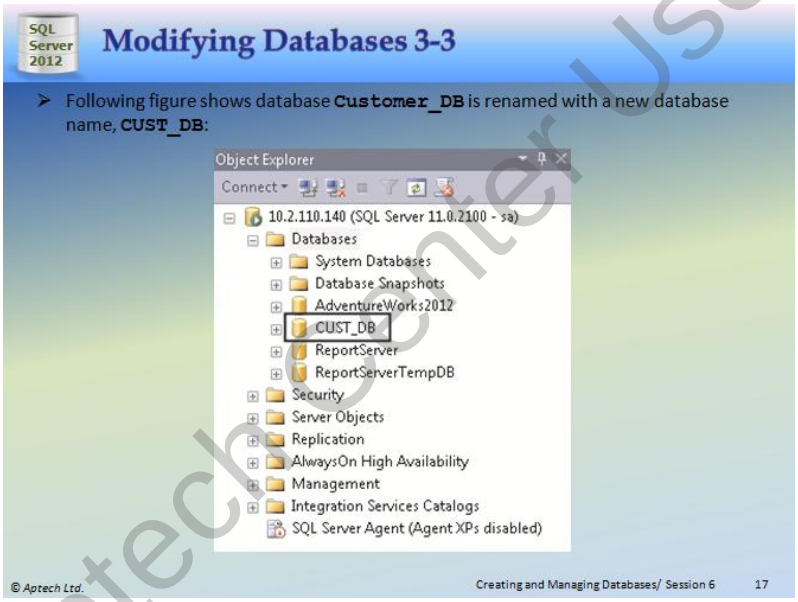
The database can also be created by right-clicking the databases option in object explorer and selecting new database option. Further click OK to create a database with default value or there would be process to define many other options like changing the owner of the database and so on.

Modifying Databases

Slides 15 to 17







Using slides 15 to 17, explain how to modify the database.

As a user-defined database grows or diminishes, the database size will be expanded or be shrunk automatically or manually. Based on changing requirements from time to time, it may be found necessary to modify a database. Explain the syntax to modify a database.

Explain Code Snippet in slide 17 which shows how to rename a database Customer_DB with a new database name, CUST_DB.

Figure shows database Customer_DB is renamed with a new database name, CUST_DB.

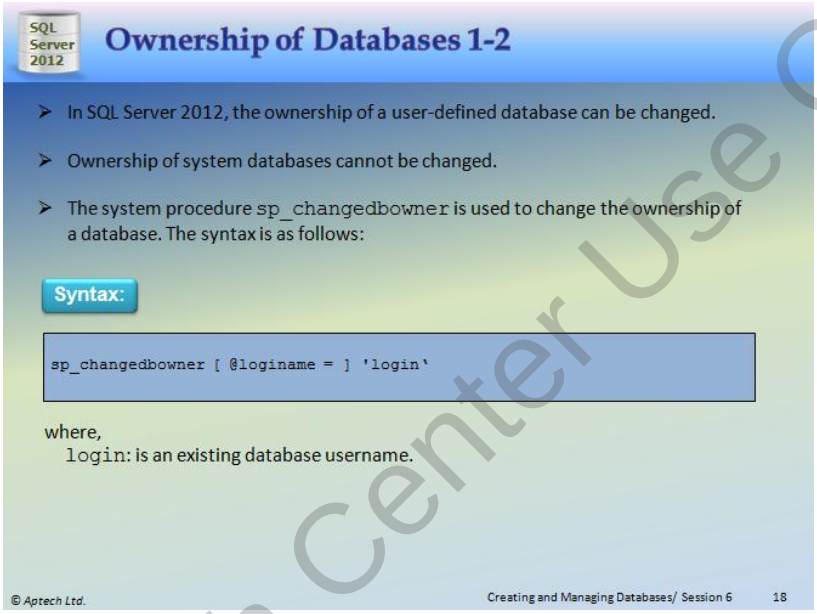
In-Class Question:

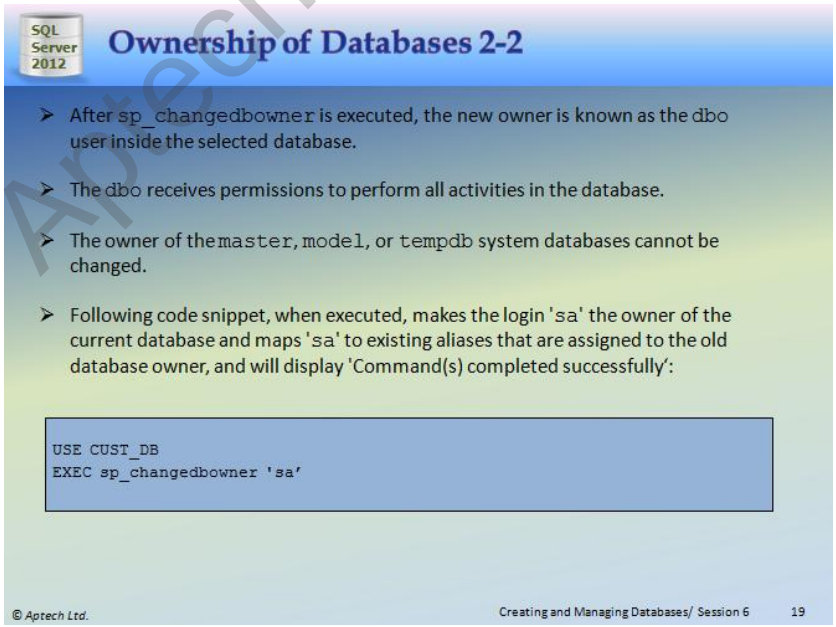
What keyword is used to modify a database?

Answer:
Alter keyword is used to modify a database.

Ownership of Databases

Slides 18 and 19





Using slides 18 and 19, explain the ownership of database.

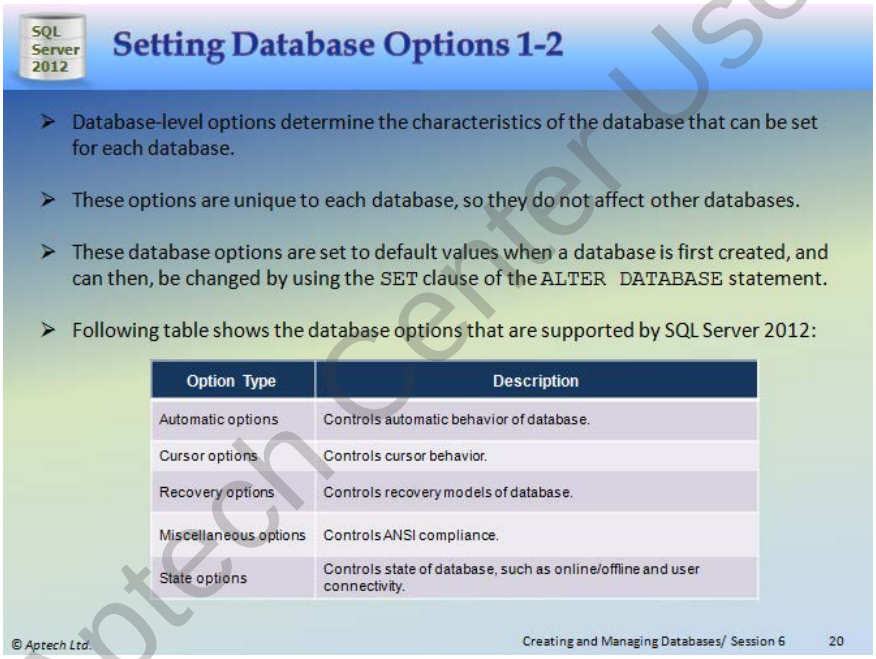
In SQL Server 2012, the ownership of a user-defined database can be changed. Ownership of system databases cannot be changed. The system procedure sp_changedbowner is used to change the ownership of a database. Explain the syntax.

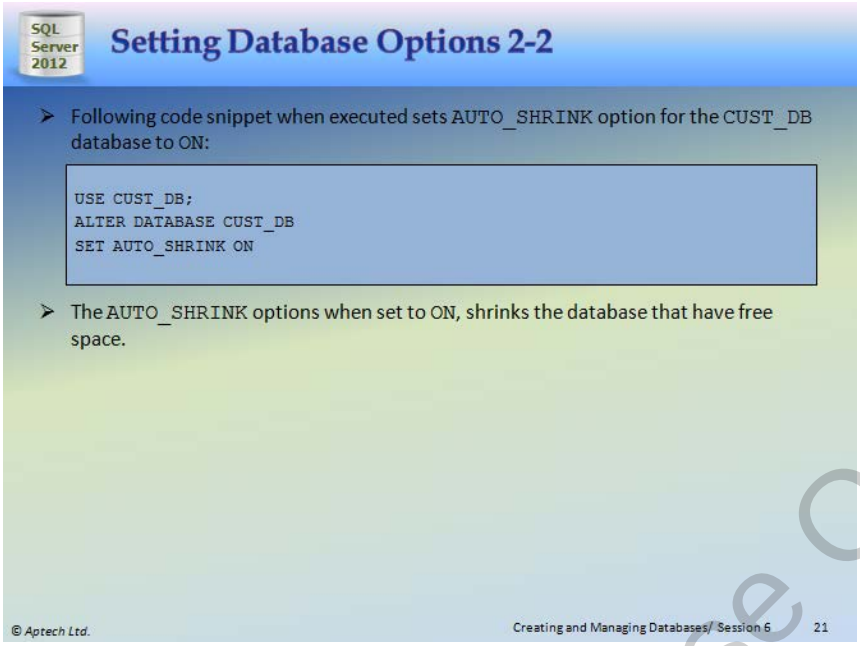
After sp_changedbowner is executed, the new owner is known as the dbo user inside the selected database. The dbo receives permissions to perform all activities in the database. The owner of the master, model, or tempdb system databases cannot be changed.

Code Snippet, when executed, makes the login 'sa' the owner of the current database and maps 'sa' to existing aliases that are assigned to the old database owner, and will display 'Command(s) completed successfully'.

Setting Database Options

Slides 20 and 21





Using slides 20 and 21, explain how to set database options.

Database-level options determine the characteristics of the database that can be set for each database. These options are unique to each database, so they do not affect other databases. These database options are set to default values when a database is first created, and can then, be changed by using the SET clause of the ALTER DATABASE statement.

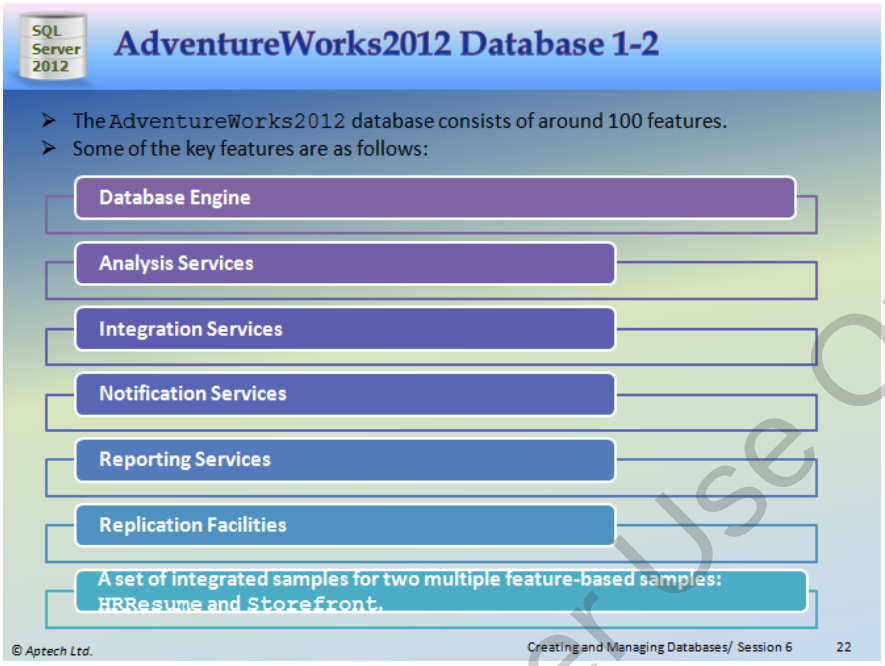
Explain the database options as shown in table that are supported by SQL Server 2012.

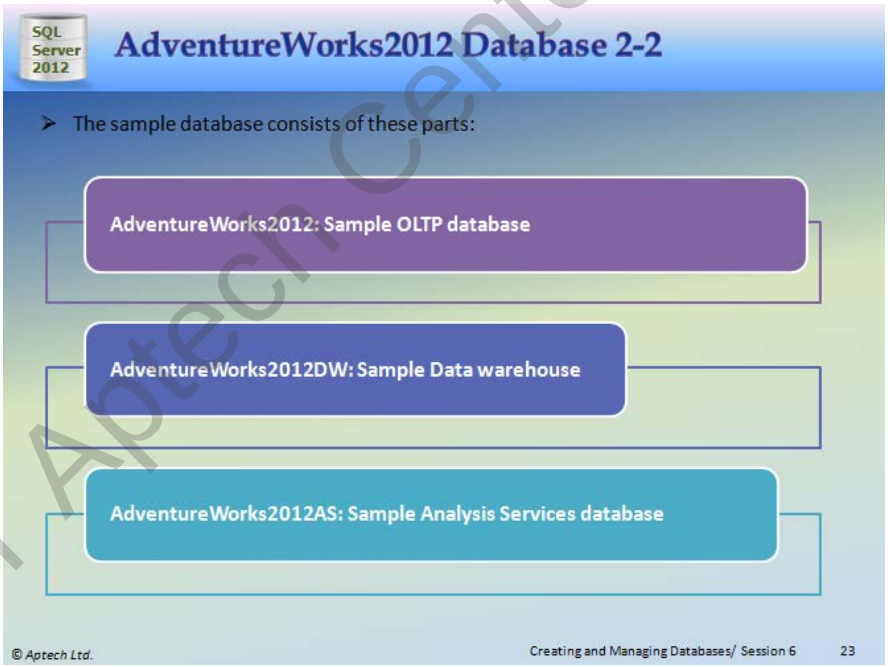
Code Snippet when executed sets AUTO_SHRINK option for the CUST_DB database to ON. The AUTO_SHRINK options when set to ON, shrinks the database that have free space.

Tips:
The AUTO_SHRINK option is not available in a Contained Database.

AdventureWorks2012 Database

Slides 22 and 23





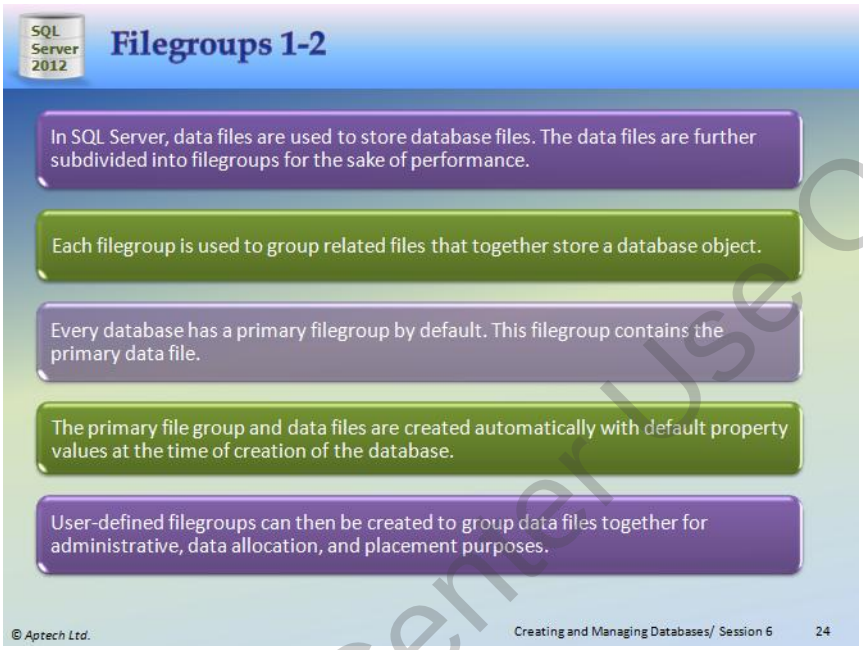
Using slides 22 and 23, explain the AdventureWorks2012 database.

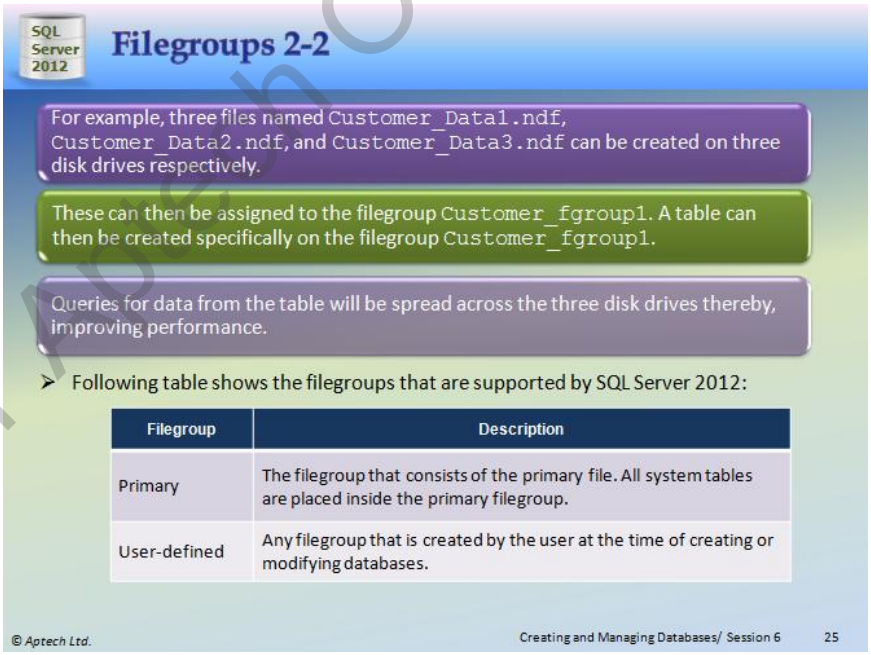
The sample database, AdventureWorks, was introduced from SQL Server 2005 onwards. This database demonstrates the use of new features introduced in SQL Server. A fictitious company called Adventure Works Cycles is created as a scenario in the database. Adventure Works Cycles is a large, multinational manufacturing company. The company manufactures and sells metal and composite bicycles to North American, European, and Asian commercial

markets. In SQL Server 2012, a new version of the sample database AdventureWorks2012 is used. The AdventureWorks2012 database consists of around 100 features. Explain some features of it.

Filegroups

Slides 24 and 25





Using slides 24 and 25, explain the filegroups.

In SQL Server, data files are used to store database files. The data files are further subdivided into filegroups for the sake of performance. Each filegroup is used to group

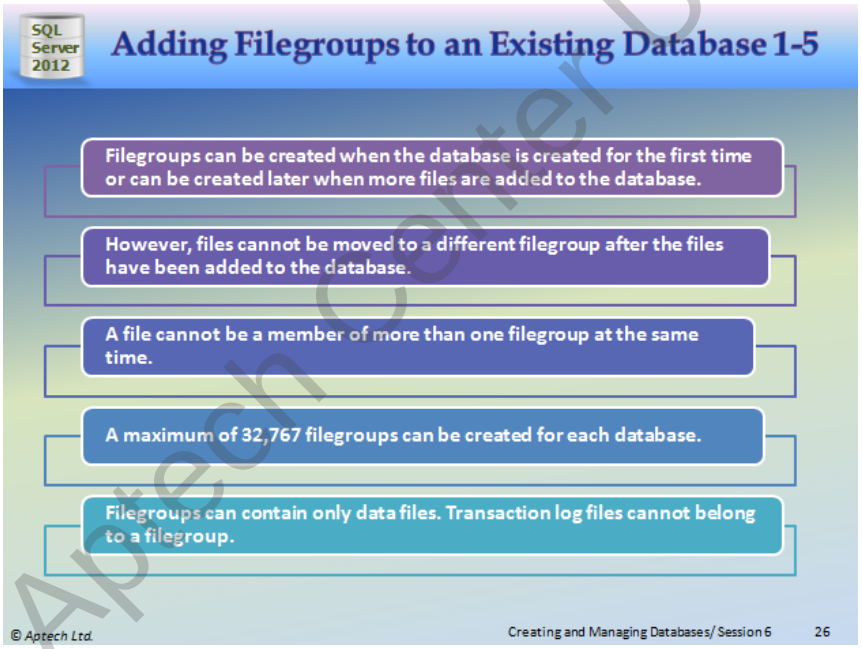
related files that together store a database object. Every database has a primary filegroup by default. This filegroup contains the primary data file. The primary file group and data files are created automatically with default property values at the time of creation of the database. User-defined filegroups can then be created to group data files together for administrative, data allocation, and placement purposes.

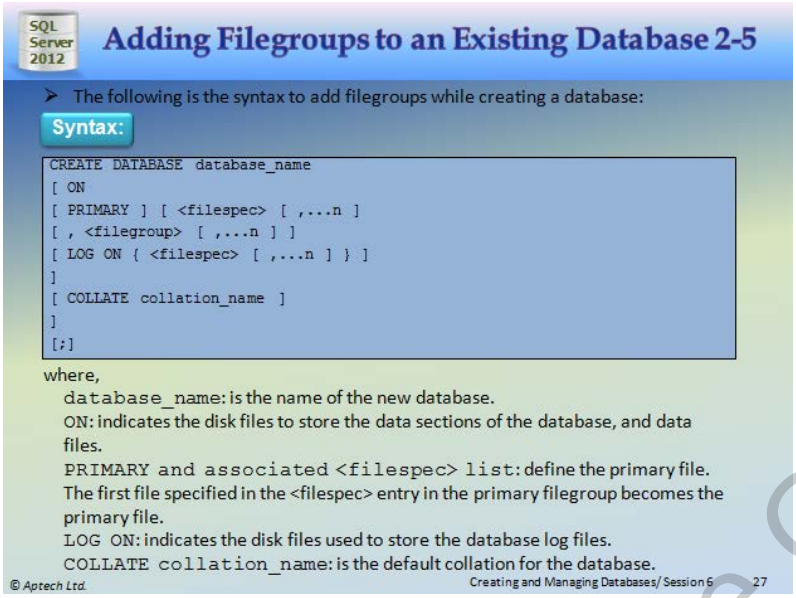
For example, three files named Customer_Data1.ndf, Customer_Data2.ndf, and Customer_Data3.ndf, can be created on three disk drives respectively. These can then be assigned to the filegroup Customer_fgroup1. A table can then be created specifically on the filegroup Customer_fgroup1. Queries for data from the table will be spread across the three disk drives thereby, improving performance.

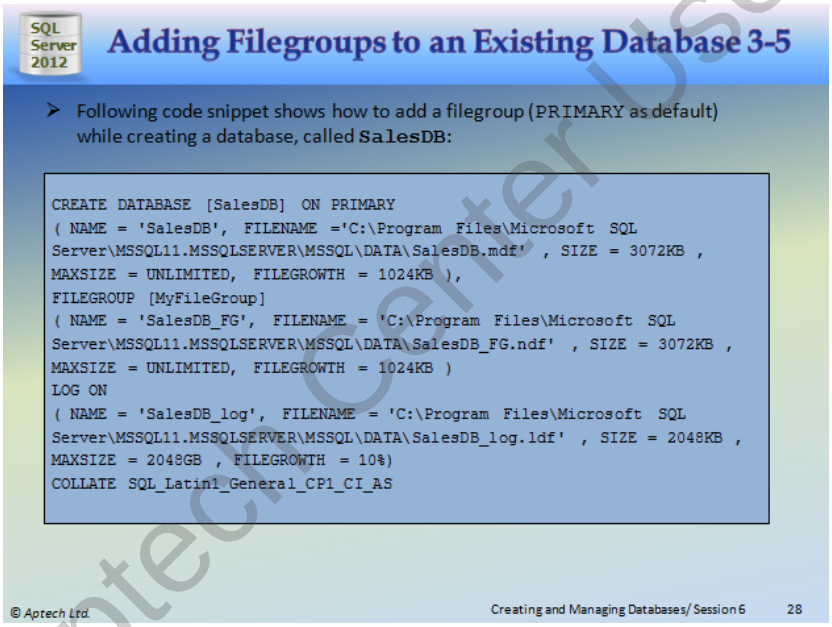
Table shows the filegroups that are supported by SQL Server 2012.

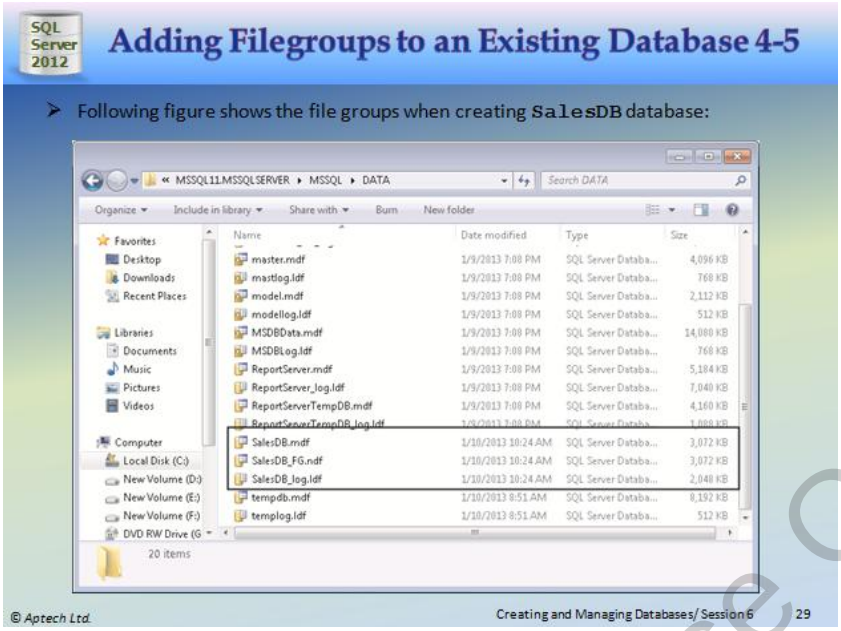
Adding Filegroups to an Existing Database

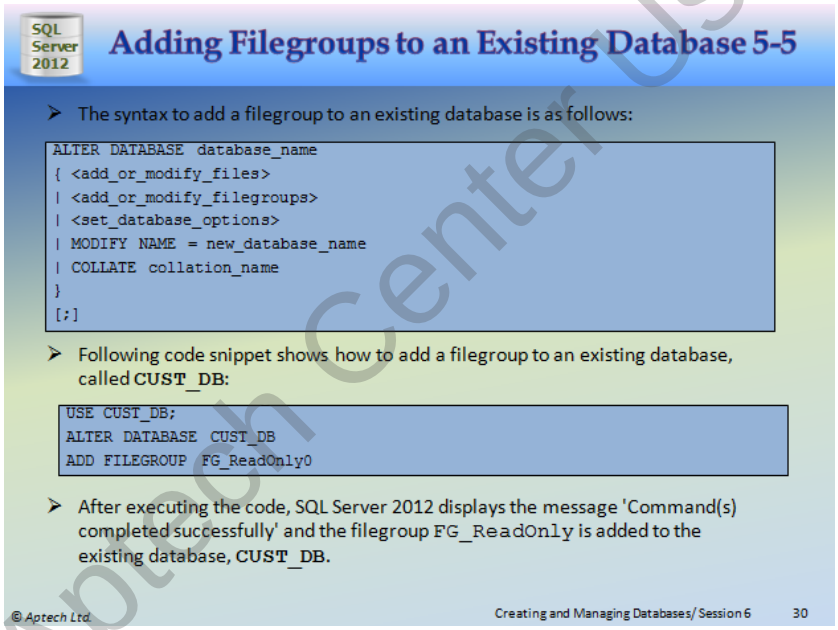
Slides 26 to 30











Using slides 26 to 30, explain how to add filegroups to an existing database.

Filegroups can be created when the database is created for the first time or can be created later when more files are added to the database. However, files cannot be moved to a different filegroup after the files have been added to the database.

A file cannot be a member of more than one filegroup at the same time. A maximum of 32,767 filegroups can be created for each database. Filegroups can contain only data files. Transaction log files cannot belong to a filegroup.

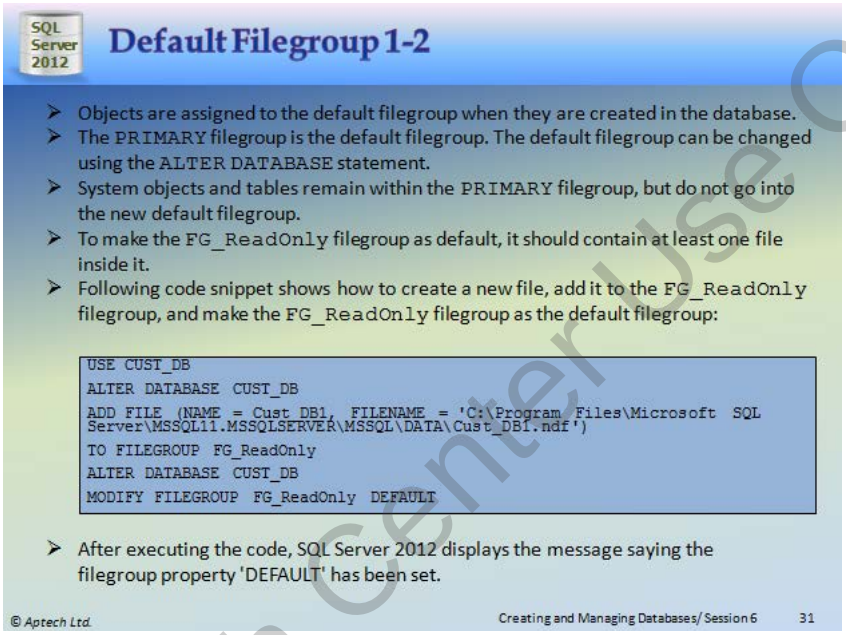
Explain the syntax to add filegroups while creating a database using slide 27. Explain Code Snippet using slide 28 which shows how to add a filegroup (PRIMARY as default) while creating a database, called SalesDB.

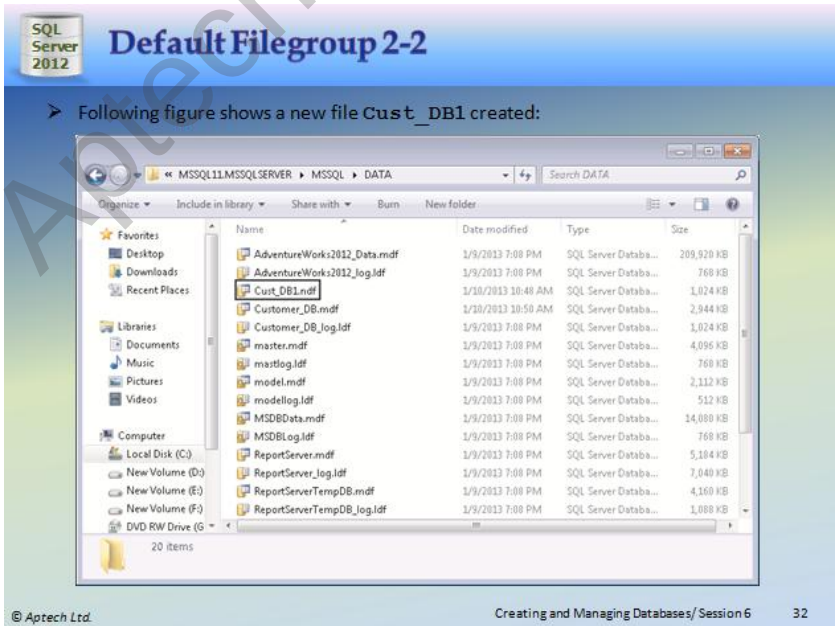
Figure on slide 29 shows the file groups when creating SalesDB database. Explain Code Snippet on slide 30 which shows how to add a filegroup to an existing database, called CUST_DB.

After executing the code, SQL Server 2012 displays the message 'Command(s) completed successfully' and the filegroup FG_ReadOnly is added to the existing database, CUST_DB.

Default Filegroup

Slides 31 and 32





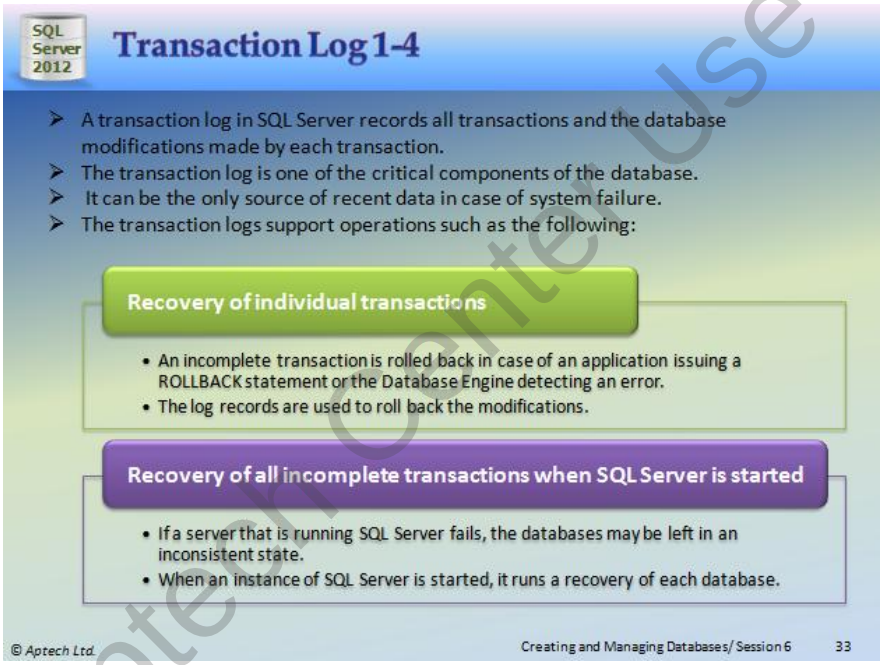
Using slides 31 and 32, explain the default filegroup.

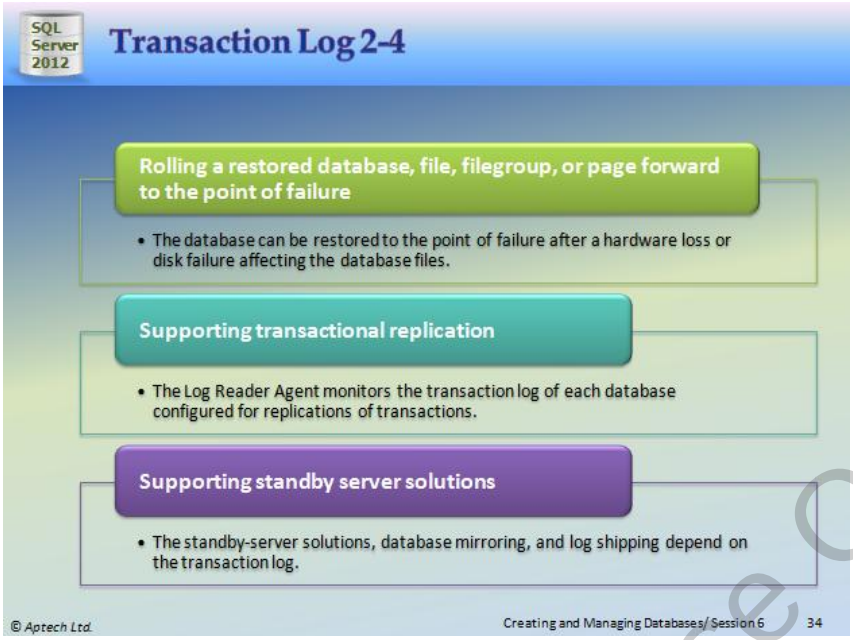
Objects are assigned to the default filegroup when they are created in the database. The PRIMARY filegroup is the default filegroup. The default filegroup can be changed using the ALTER DATABASE statement. System objects and tables remain within the PRIMARY filegroup, but do not go into the new default filegroup.

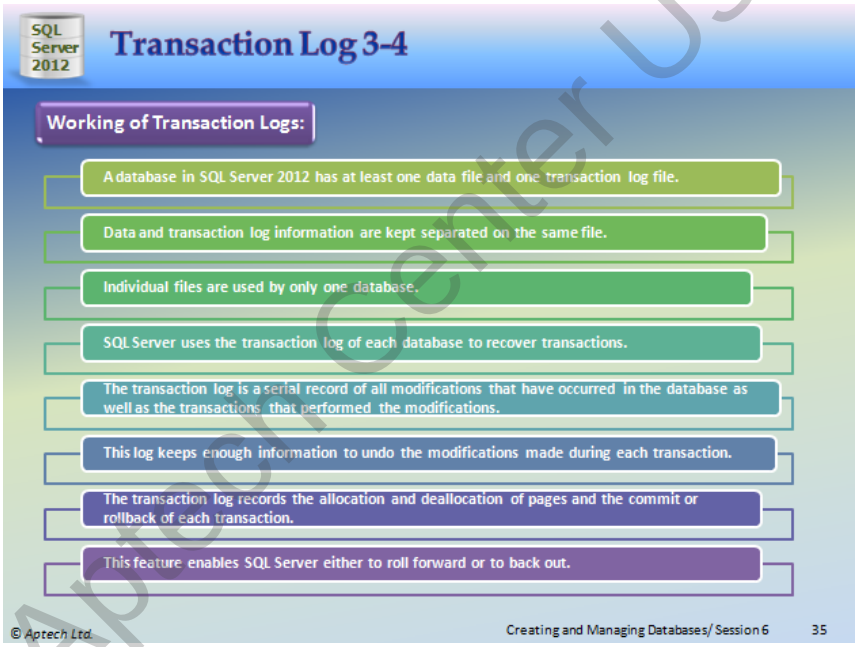
To make the FG_ReadOnly filegroup as default, it should contain at least one file inside it. Code Snippet shows how to create a new file, add it to the FG_ReadOnly filegroup and make the FG_ReadOnly filegroup that was created in Code Snippet as the default filegroup. After executing the code in Code Snippet, SQL Server 2012 displays the message saying the filegroup property 'DEFAULT' has been set. Figure shows a new file Cust_DB1 created.

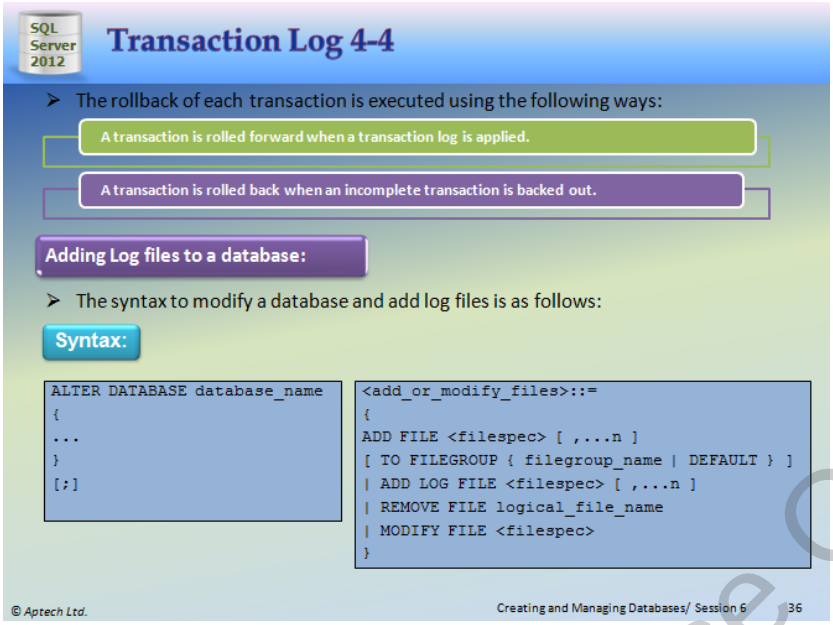
Transaction Log

Slides 33 to 36









Using slides 33 to 36, explain the transaction log.

A transaction log in SQL Server records all transactions and the database modifications made by each transaction. The transaction log is one of the critical components of the database. It can be the only source of recent data in case of system failure. The transaction logs support operations such as the following:

- Recovery of individual transactions: An incomplete transaction is rolled back in case of an application issuing a ROLLBACK statement or the Database Engine detecting an error. The log records are used to roll back the modifications.
- Recovery of all incomplete transactions when SQL Server is started: If a server that is running SQL Server fails, the databases may be left in an inconsistent state. When an instance of SQL Server is started, it runs a recovery of each database.
- Rolling a restored database, file, filegroup, or page forward to the point of failure: The database can be restored to the point of failure after a hardware loss or disk failure affecting the database files.
- Supporting transactional replication: The Log Reader Agent monitors the transaction log of each database configured for replications of transactions.
- Supporting standby server solutions: The standby-server solutions, database mirroring, and log shipping depend on the transaction log.

The next section explains the working of transaction logs and adding log files to a database.

Working of Transaction Logs: A database in SQL Server 2012 has at least one data file and one transaction log file. Data and transaction log information are kept separated on the same file. Individual files are used by only one database.

SQL Server uses the transaction log of each database to recover transactions. The transaction log is a serial record of all modifications that have occurred in the database as well as the transactions that performed the modifications. This log keeps enough information to undo the modifications made during each transaction. The transaction log

records the allocation and deallocation of pages and the commit or rollback of each transaction. This feature enables SQL Server either to roll forward or to back out. The rollback of each transaction is executed using the following ways:

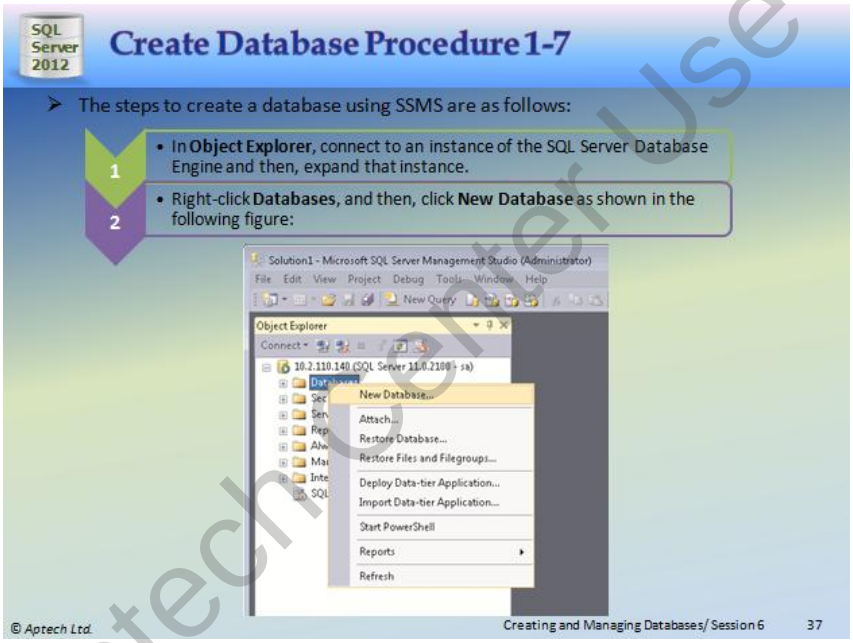
- A transaction is rolled forward when a transaction log is applied.
- A transaction is rolled back when an incomplete transaction is backed out.

Adding Log files to a database: Explain the syntax to modify a database and add log files.

By default, the data and transaction logs are put on the same drive and path to accommodate single-disk systems, but may not be optimal for production environments.

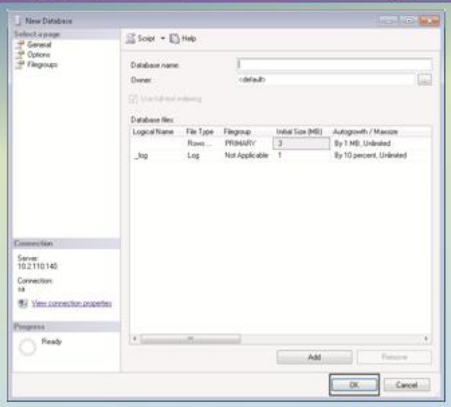
Procedure to Create a Database

Slides 37 to 43



Create Database Procedure 2-7

3. In **New Database**, enter a database name.
4. To create the database by accepting all default values, click **OK**, as shown in the following figure; otherwise, continue with the following optional steps:

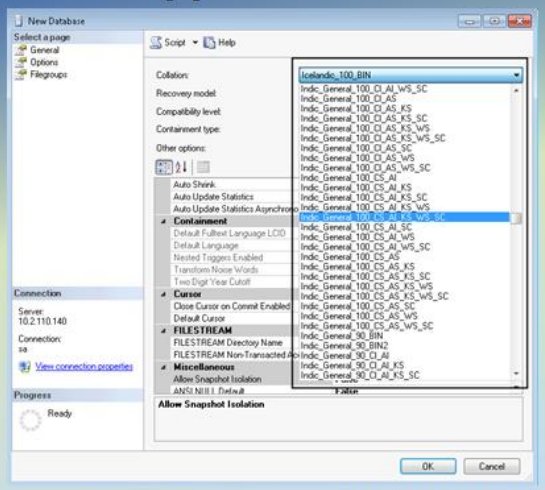


Create Database Procedure 3-7

5. To change the owner name, click (...) to select another owner.
6. To change the default values of the primary data and transaction log files, in the Database files grid, click the appropriate cell and enter the new value.
7. To change the collation of the database, select the **Options** page, and then, select a collation from the list.

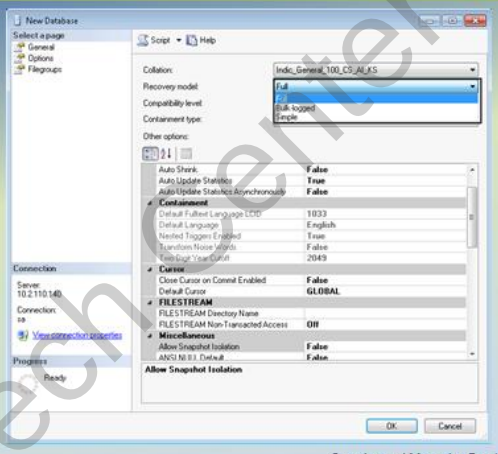
Create Database Procedure 4-7

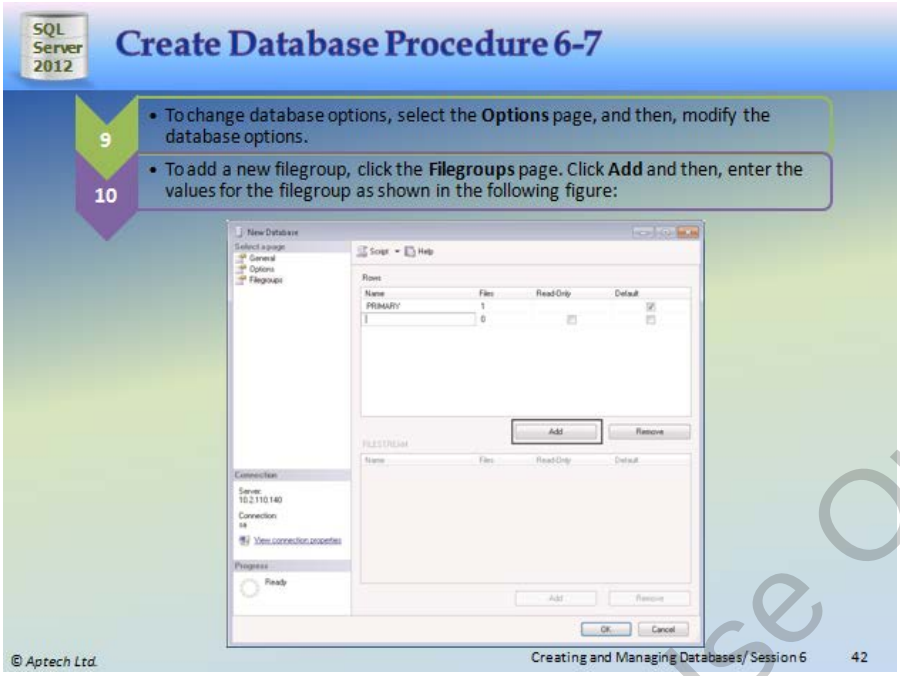
- This is shown in the following figure:

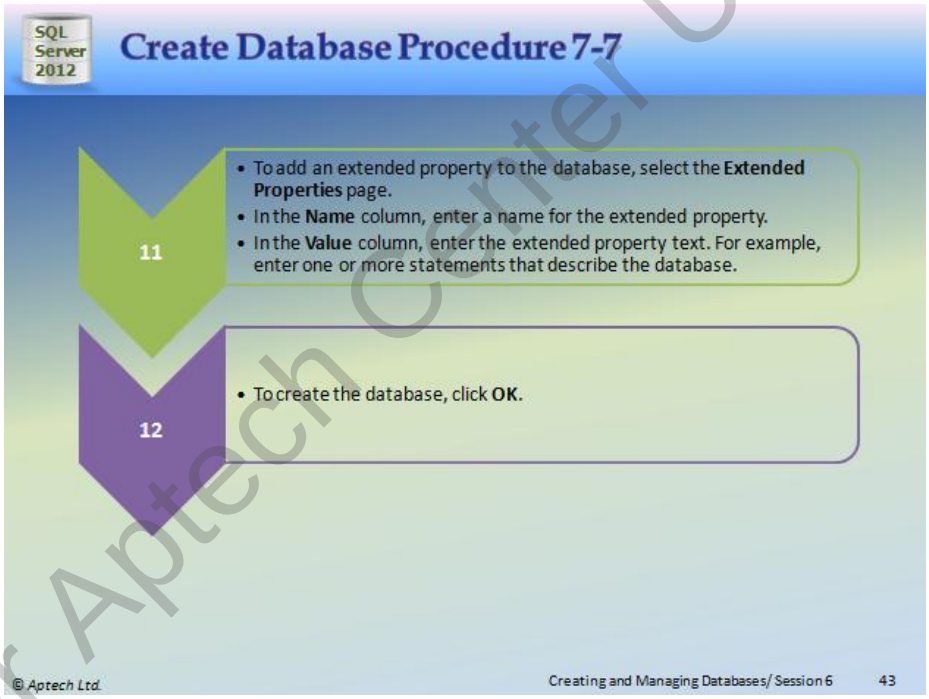


Create Database Procedure 5-7

8 • To change the recovery model, select the **Options** page and then, select a recovery model from the list as shown in the following figure:







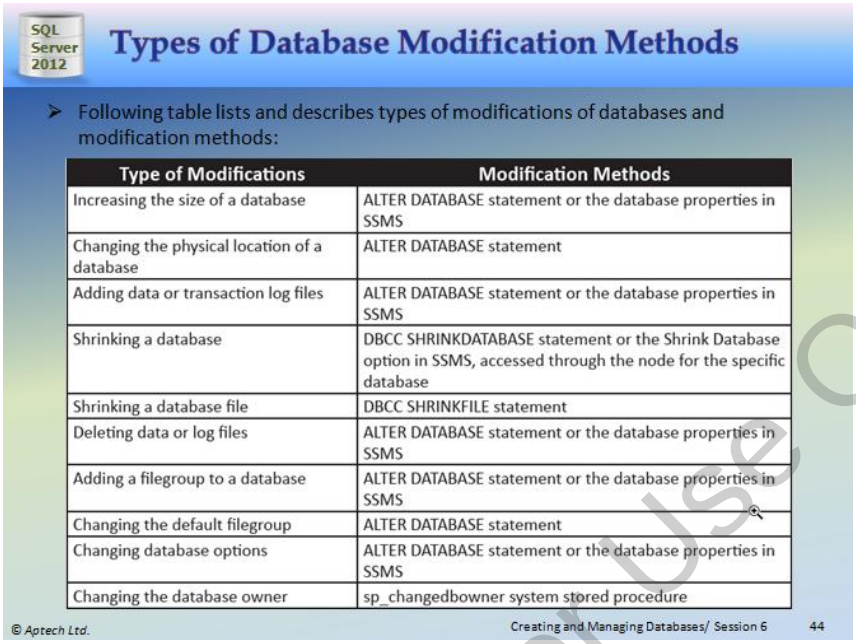
Using slides 37 to 43, explain the procedure to create a database.

When a database created, the data files needs to be as large as possible based on the maximum amount of data, which is expected in the database.

Explain the steps to create a database using SSMS using the figures and steps as shown in the slides.

Types of Database Modification Methods

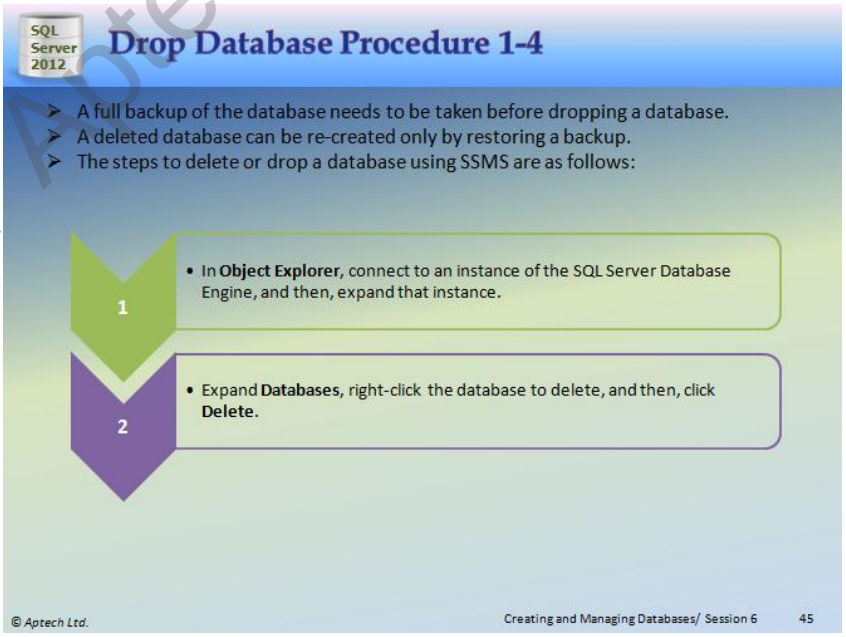
Slide 44



Using slide 44, explain the types of modifications of databases and modification methods. The user-defined database can be deleted when it is no longer required. The files and the data associated with the database are automatically deleted from the disk when the database is deleted.

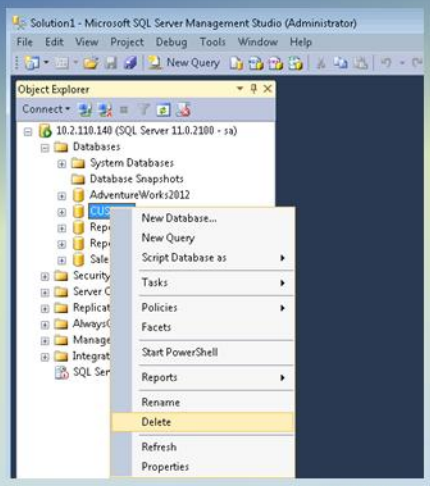
Procedure to Drop a Database

Slides 45 to 48



Drop Database Procedure 2-4

- This is shown in the following figure:



Drop Database Procedure 3-4

3. Confirm that the correct database is selected, and then, click **OK**.

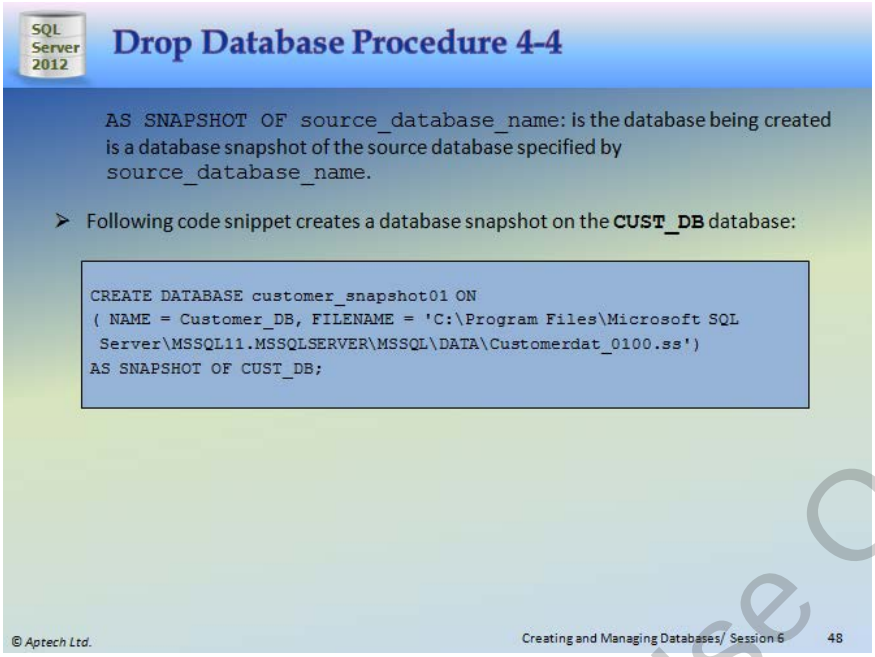
- The syntax to delete or drop a database using Transact-SQL is as follows:

```
CREATE DATABASE database_snapshot_name
ON
(
NAME = logical_file_name,
FILENAME = 'os_file_name'
) [ ,...n ]
AS SNAPSHOT OF source_database_name
[;]
```

where,

`database_snapshot_name`: is the name of the new database snapshot.

`ON ( NAME = logical_file_name, FILENAME = 'os_file_name' ) [ ,... n ]`: is the list of files in the source database. For the snapshot to work, all the data files must be specified individually.



Using slides 45 to 48, explain the procedure to drop a database.

A full backup of the database needs to be taken before dropping a database. A deleted database can be re-created only by restoring a backup. The steps to delete or drop a database using SSMS are as follows:

1. In **Object Explorer**, connect to an instance of the SQL Server Database Engine, and then, expand that instance.

2. Expand **Databases**, right-click the database to delete, and then, click **Delete**, as shown in figure.

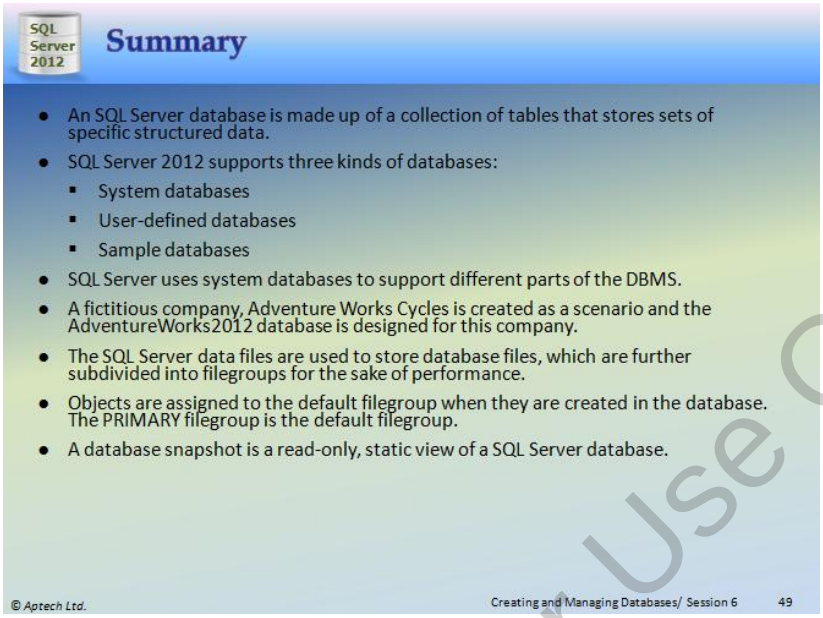
3. Confirm that the correct database is selected, and then, click **OK**.

Explain the syntax to delete or drop a database using Transact-SQL.

Code Snippet creates a database snapshot on the CUST_DB database.

Summarize Session

Slide 49



Using slide 49, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- An SQL Server database is made up of a collection of tables that stores sets of specific structured data.
- SQL Server 2012 supports three kinds of databases:
    - System databases
    - User-defined databases
    - Sample databases
- SQL Server uses system databases to support different parts of the DBMS.
- A fictitious company, Adventure Works Cycles is created as a scenario and the AdventureWorks2012 database is designed for this company.
- The SQL Server data files are used to store database files, which are further subdivided into filegroups for the sake of performance.
- Objects are assigned to the default filegroup when they are created in the database. The PRIMARY filegroup is the default filegroup.
- A database snapshot is a read-only, static view of a SQL Server database.

6.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Creating Tables topic offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 7 – Creating Tables

7.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

7.1.1 Objectives

By the end of this session, the learners will be able to:

- List SQL Server 2012 data types
- Describe the procedure to create, modify, and drop tables in an SQL Server database
- Describe the procedure to add, modify, and drop columns in a table

7.1.2 Teaching Skills

To teach this session, you should be well-versed with system and user defined databases. The key features of the AdventureWorks2012 database should also be known. You should also have in-depth knowledge about the types of database modification.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slide and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

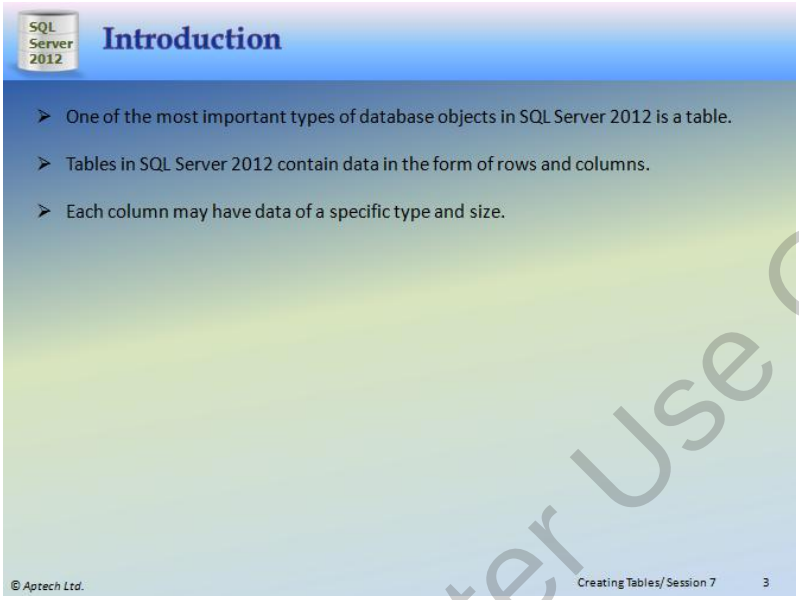
Overview of the Session:

Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces system and user defined databases. They will learn about the key features of the AdventureWorks2012 database. They will also know about types of database modification.

7.2 In-Class Explanations

Introduction

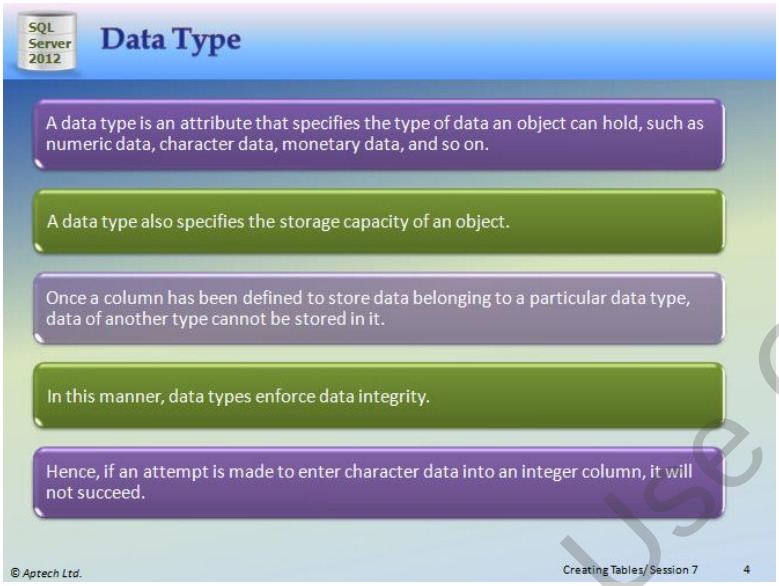
Slide 3



Using slide 3, explain the database.
One of the most important types of database objects in SQL Server 2012 is a table. Tables in SQL Server 2012 contain data in the form of rows and columns. Each column may have data of a specific type and size.

Data Type

Slide 4



Using slide 4, explain data type.

A data type is an attribute that specifies the type of data an object can hold, such as numeric data, character data, monetary data, and so on. A data type also specifies the storage capacity of an object. Once a column has been defined to store data belonging to a particular data type, data of another type cannot be stored in it. In this manner, data types enforce data integrity.

Consider a column named, BasicSalary in a table, Employee. To ensure that only numeric data is entered, this column is defined to belong to an integer data type. Hence, if an attempt is made to enter character data into that BasicSalary column, it will not succeed.

In-Class Question:

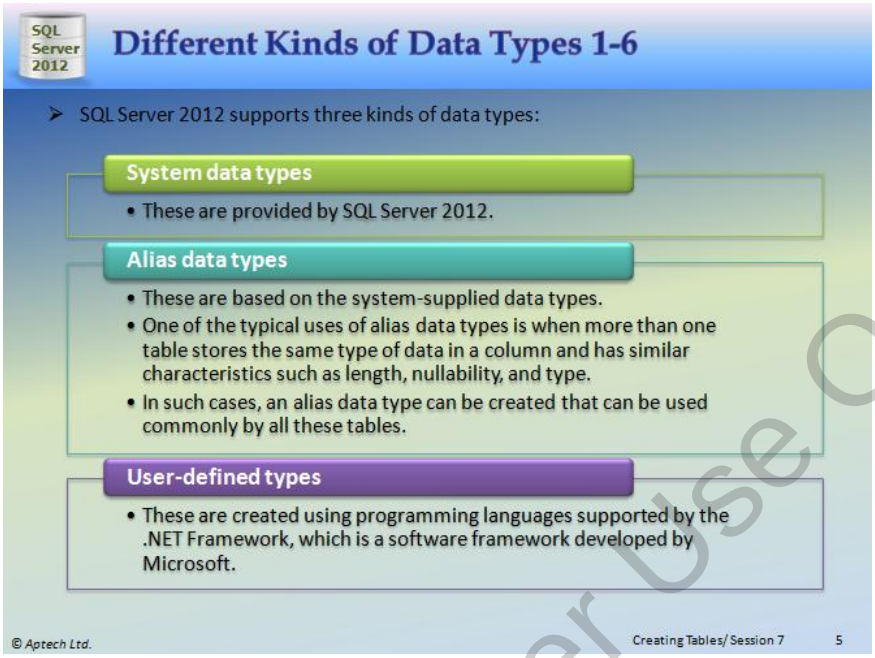
After you finish explaining the data type, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

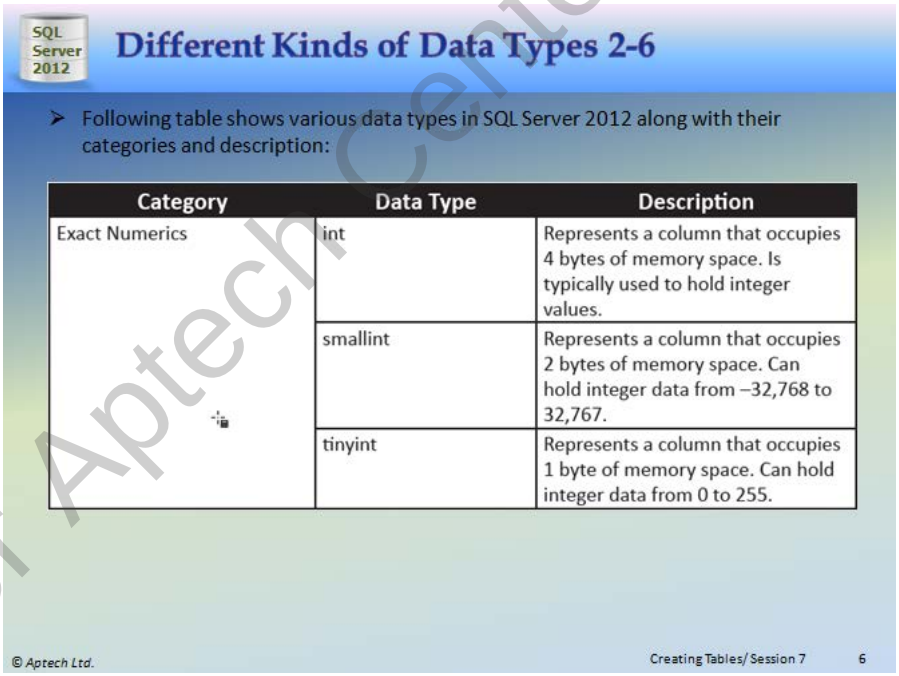
What is data type?

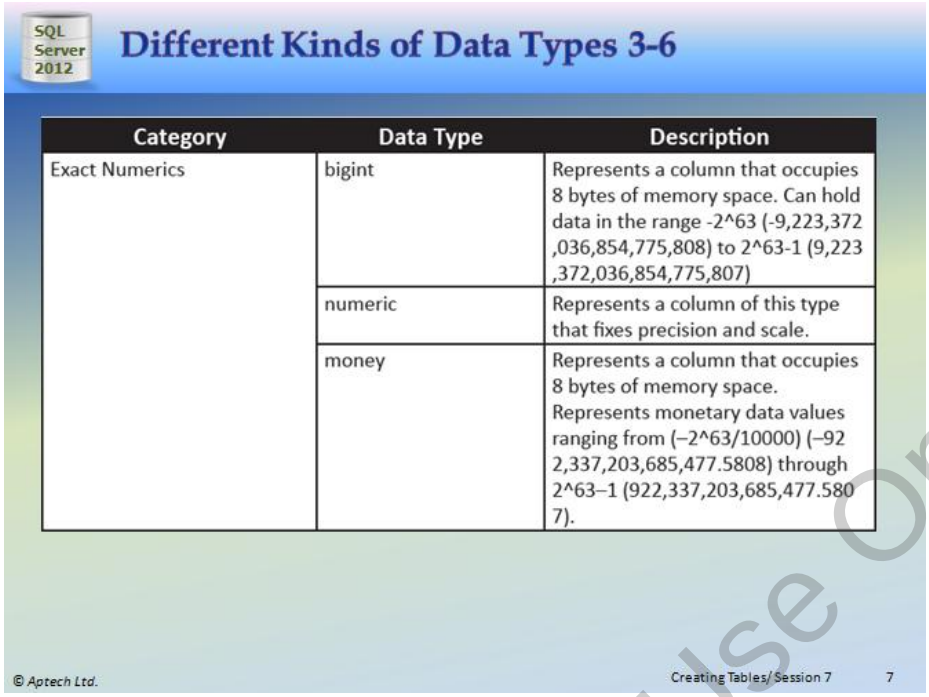
Answer:
A data type is an attribute that specifies the type of data an object can hold, such as numeric data, character data, monetary data, and so on.

Different Kinds of Data Types

Slides 5 to 10

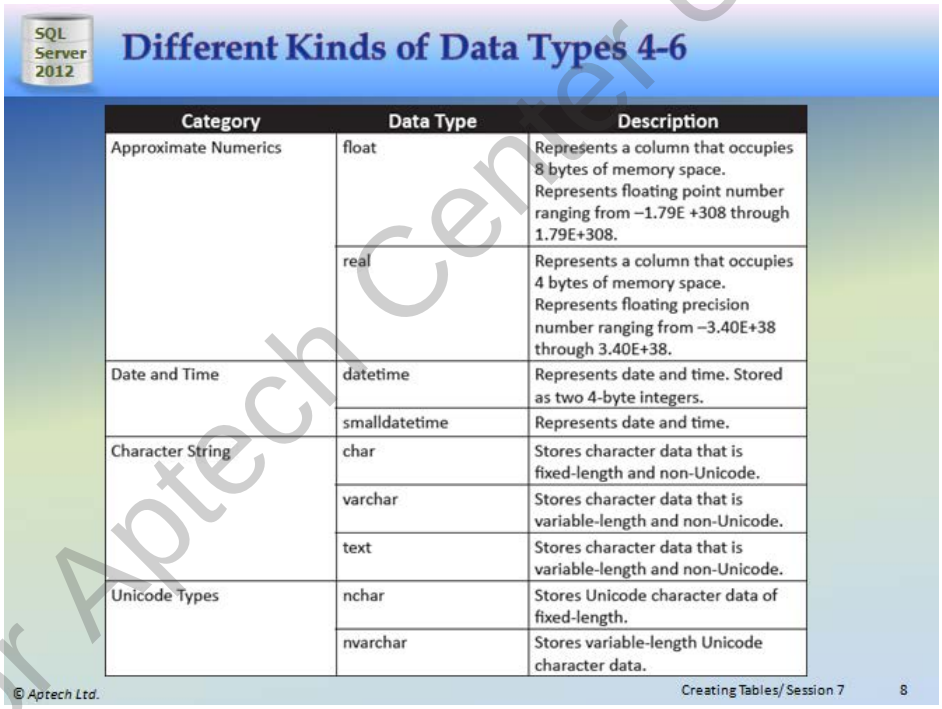






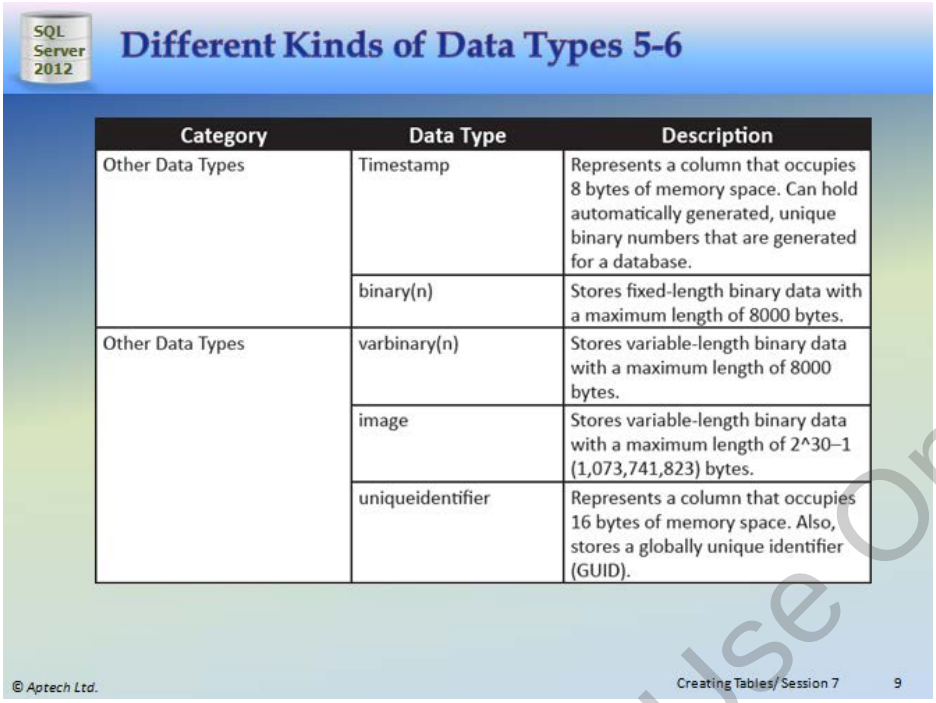
Different Kinds of Data Types 3-6

Category	Data Type	Description
Exact Numerics	bigint	Represents a column that occupies 8 bytes of memory space. Can hold data in the range -2^63 (-9,223,372 ,036,854,775,808) to 2^63-1 (9,223 ,372,036,854,775,807)
	numeric	Represents a column of this type that fixes precision and scale.
	money	Represents a column that occupies 8 bytes of memory space. Represents monetary data values ranging from (–2^63/10000) (–92 2,337,203,685,477.5808) through 2^63–1 (922,337,203,685,477.580 7).



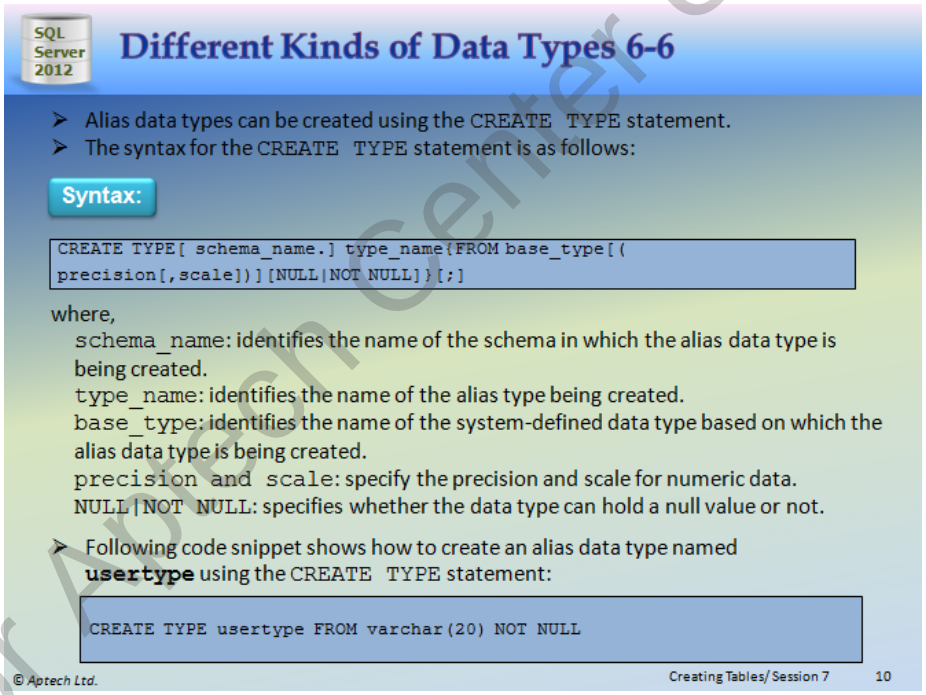
Different Kinds of Data Types 4-6

Category	Data Type	Description
Approximate Numerics	float	Represents a column that occupies 8 bytes of memory space. Represents floating point number ranging from –1.79E +308 through 1.79E+308.
	real	Represents a column that occupies 4 bytes of memory space. Represents floating precision number ranging from –3.40E+38 through 3.40E+38.
Date and Time	datetime	Represents date and time. Stored as two 4-byte integers.
	smalldatetime	Represents date and time.
Character String	char	Stores character data that is fixed-length and non-Unicode.
	varchar	Stores character data that is variable-length and non-Unicode.
	text	Stores character data that is variable-length and non-Unicode.
Unicode Types	nchar	Stores Unicode character data of fixed-length.
	nvarchar	Stores variable-length Unicode character data.



Different Kinds of Data Types 5-6

Category	Data Type	Description
Other Data Types	Timestamp	Represents a column that occupies 8 bytes of memory space. Can hold automatically generated, unique binary numbers that are generated for a database.
	binary(n)	Stores fixed-length binary data with a maximum length of 8000 bytes.
Other Data Types	varbinary(n)	Stores variable-length binary data with a maximum length of 8000 bytes.
	image	Stores variable-length binary data with a maximum length of 2^30–1 (1,073,741,823) bytes.
	uniqueidentifier	Represents a column that occupies 16 bytes of memory space. Also, stores a globally unique identifier (GUID).



Different Kinds of Data Types 6-6

- Alias data types can be created using the CREATE TYPE statement.
- The syntax for the CREATE TYPE statement is as follows:

Syntax:

```
CREATE TYPE[ schema_name.] type_name{FROM base_type[(
precision[,scale])][NULL|NOT NULL]}[;]
```

where,

schema_name: identifies the name of the schema in which the alias data type is being created.
type_name: identifies the name of the alias type being created.
base_type: identifies the name of the system-defined data type based on which the alias data type is being created.
precision and scale: specify the precision and scale for numeric data.
NULL|NOT NULL: specifies whether the data type can hold a null value or not.

- Following code snippet shows how to create an alias data type named **usertype** using the CREATE TYPE statement:

```
CREATE TYPE usertype FROM varchar(20) NOT NULL
```


Using slides 5 to 10, explain the different kinds of data types.

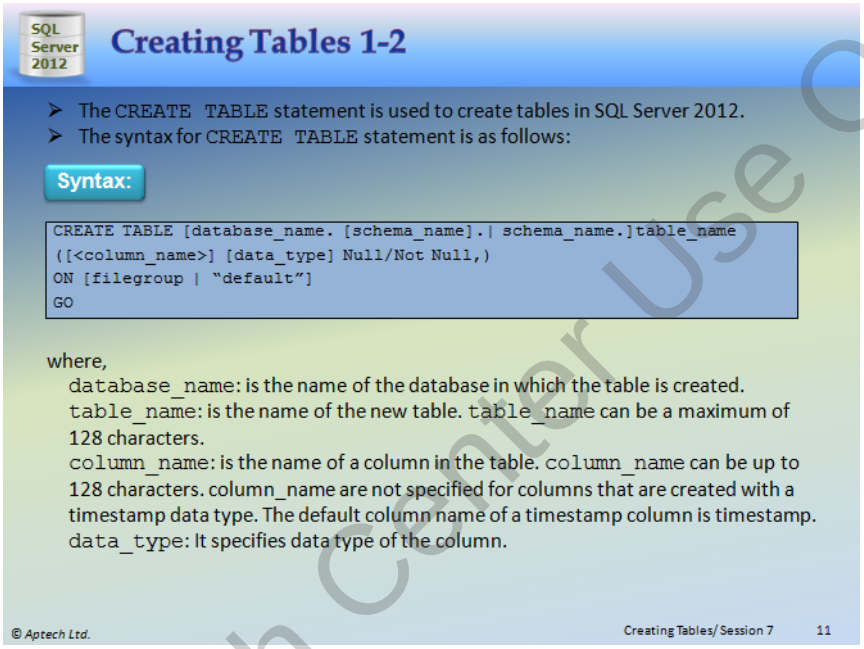
SQL Server 2012 supports three kinds of data types:

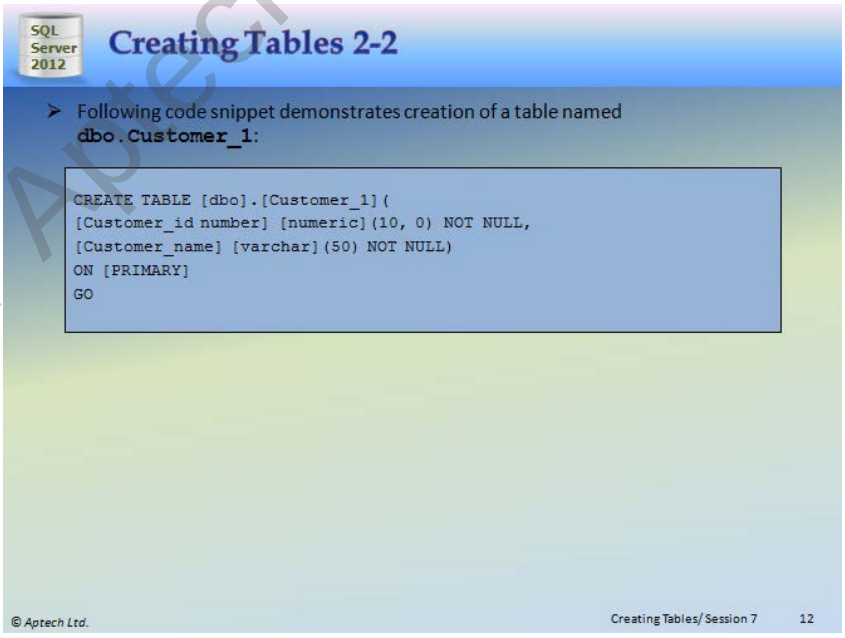
- **System data types** - These are provided by SQL Server 2012.
- **Alias data types** - These are based on the system-supplied data types. One of the typical uses of alias data types is when more than one table stores the same type of data in a column and has similar characteristics such as length, nullability, and type. In such cases, an alias data type can be created that can be used commonly by all these tables.
- **User-defined types** - These are created using programming languages supported by the .NET Framework, which is a software framework developed by Microsoft.

Using table on slides 6 to 9, explain various data types in SQL Server 2012 along with their categories and description.
Alias data types can be created using the CREATE TYPE statement.
Explain the syntax for the CREATE TYPE statement on slide 10.
Explain code snippet which shows how to create an alias data type named, usertype using the CREATE TYPE statement.

Creating Tables

Slides 11 and 12





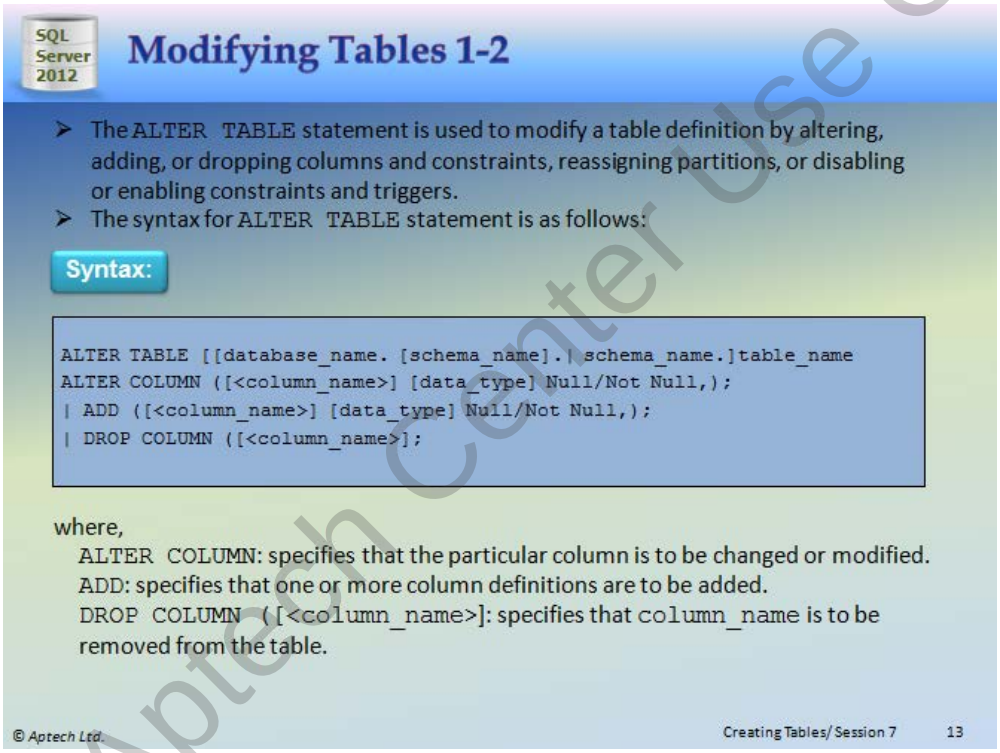
Using slides 11 and 12, explain how to create table.
Most tables have a primary key, made up of one or more columns of the table. A primary key is always unique. The Database Engine will enforce the restriction that any primary key

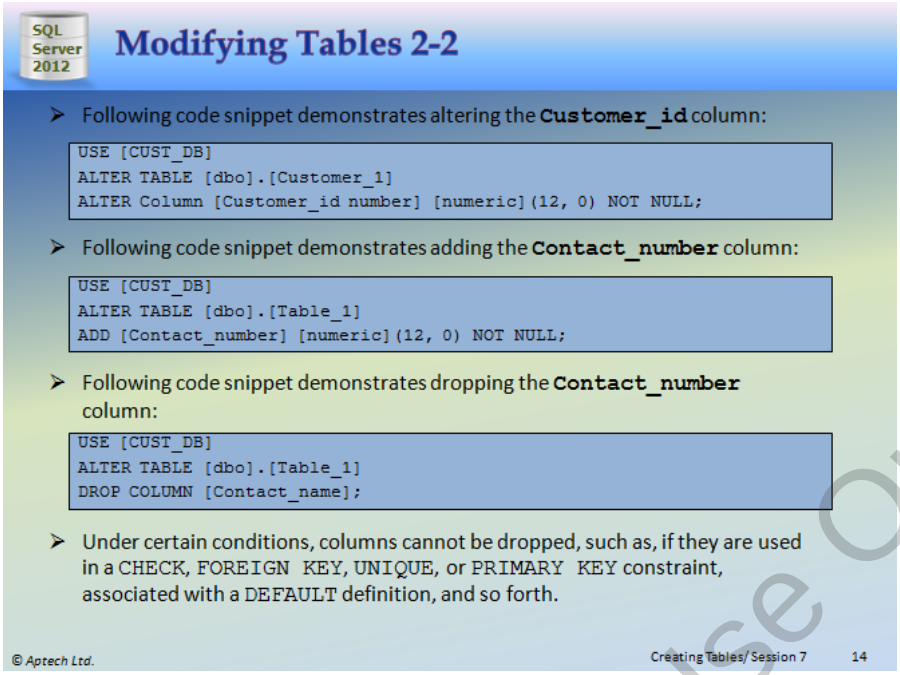
value cannot be repeated in the table. Thus, the primary key can be used to identify each record uniquely.
The CREATE TABLE statement is used to create tables in SQL Server 2012. Explain the syntax for CREATE TABLE statement.
Explain the code snippet using slide 12 which demonstrates creation of a table named, dbo.Customer_1.

Tips:
The empty table can be filled with data with the INSERT INTO statement.

Modifying Tables

Slides 13 and 14





Using slides 13 and 14, explain how to modify a table.
The ALTER TABLE statement is used to modify a table definition by altering, adding, or dropping columns and constraints, reassigning partitions, or disabling or enabling constraints and triggers.

Explain the syntax for ALTER TABLE statement.

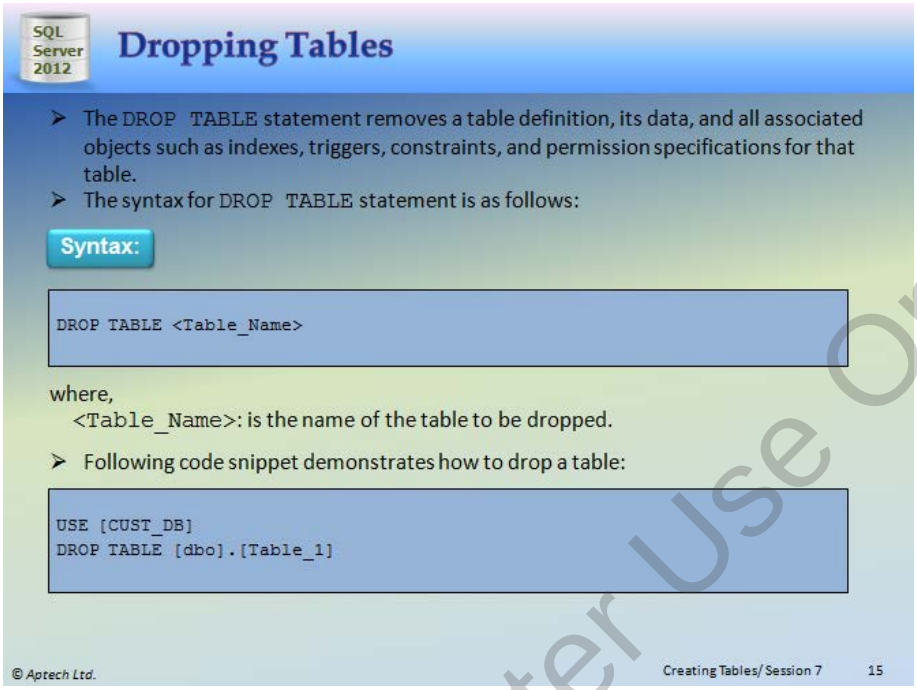
First code snippet on slide 14 demonstrates altering the Customer_id column.

Second and third code snippets on slide 14 demonstrate adding the Contact_number column and dropping the Contact_number column.

Before attempting to drop columns, however, it is important to ensure that the columns can be dropped. Under certain conditions, columns cannot be dropped, such as if they are used in a CHECK, FOREIGN KEY, UNIQUE, or PRIMARY KEY constraint, associated with a DEFAULT definition, and so forth.

Dropping Tables

Slide 15



Using slide 15, explain how to drop a table.

The DROP TABLE statement removes a table definition, its data, and all associated objects such as indexes, triggers, constraints, and permission specifications for that table.

Explain the syntax for DROP TABLE statement.

Explain code snippet which demonstrates how to drop a table.

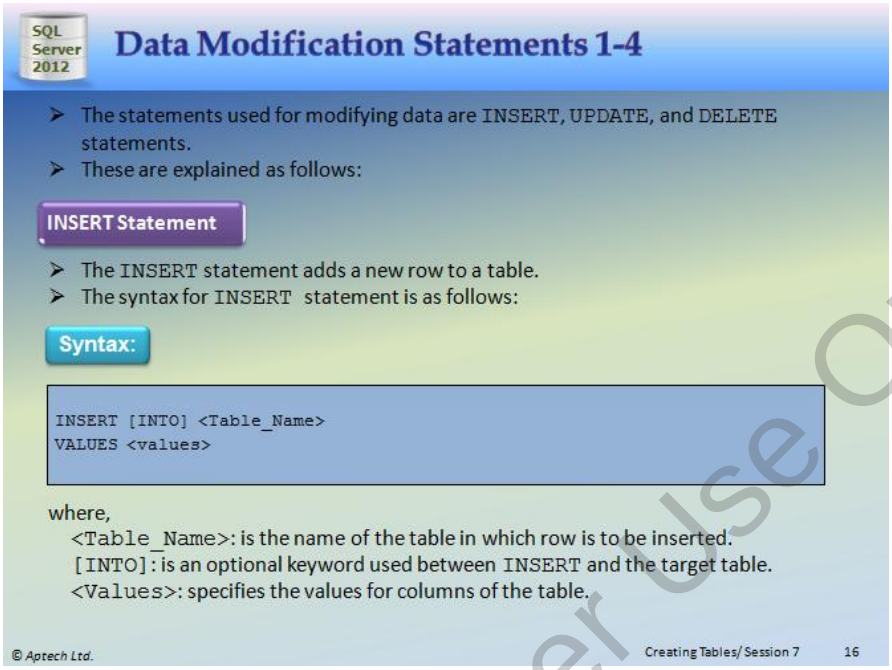
In-Class Question:

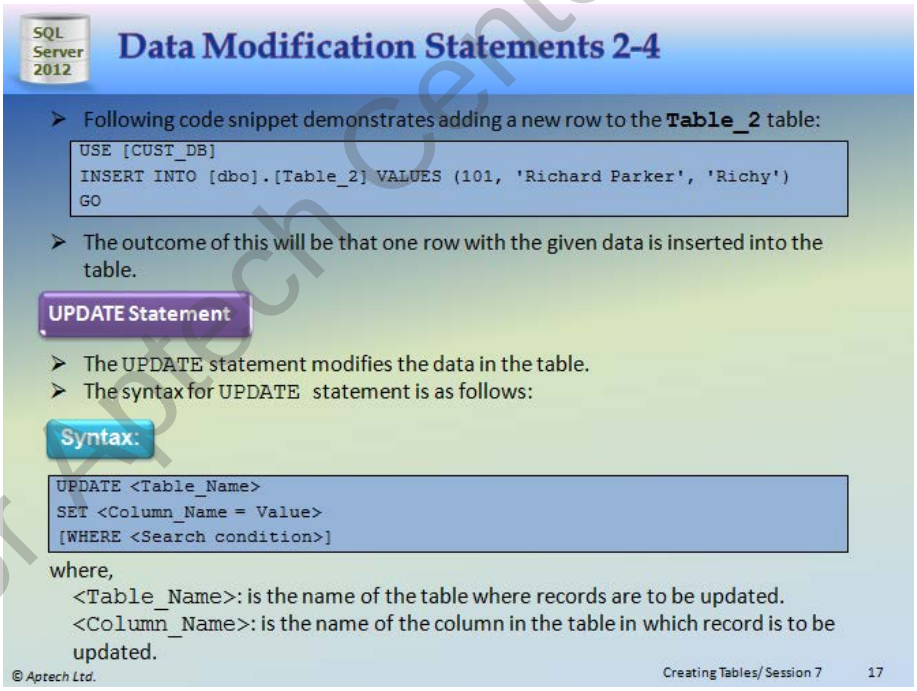
What is syntax to delete a table?

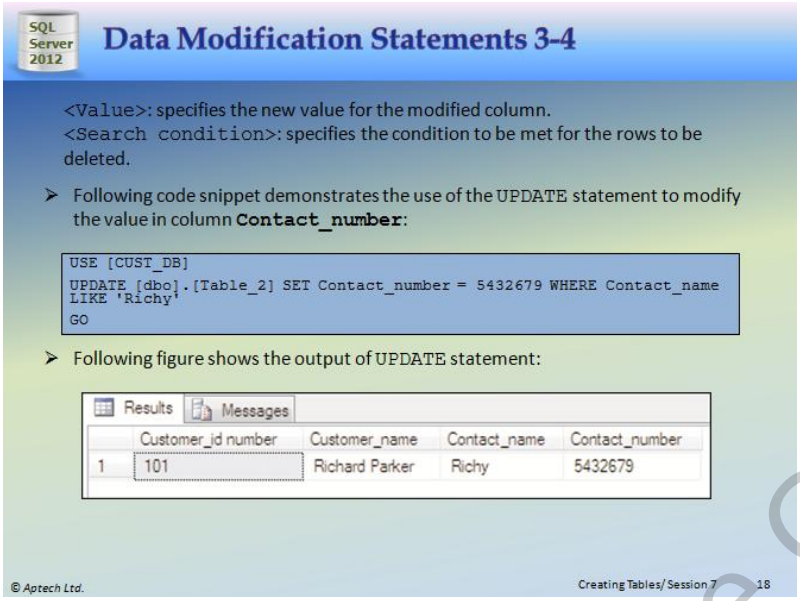
Answer:
Drop table <table_name>

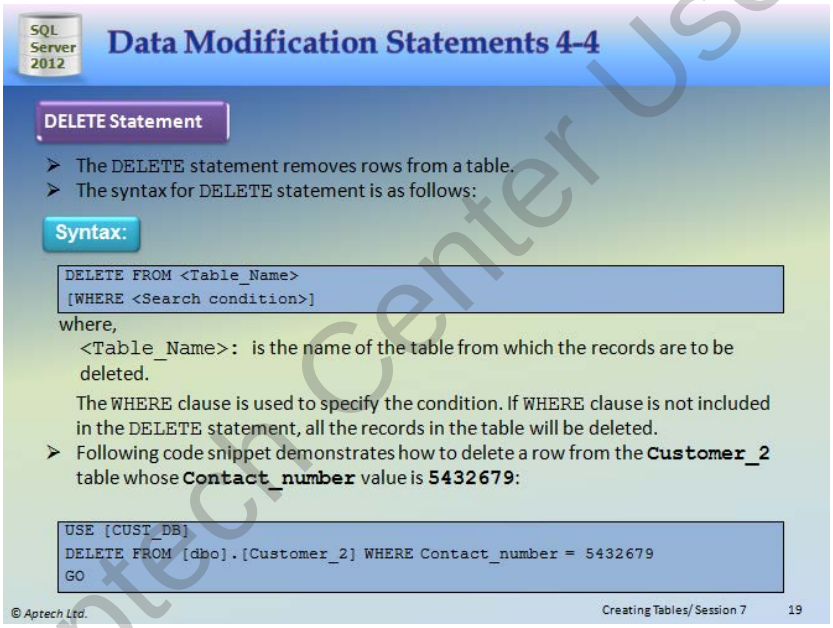
Data Modification Statements

Slides 16 to 19









Using slides 16 to 19, explain the statements used for data modification.

The statements used for modifying data are INSERT, UPDATE, and DELETE statements. These are explained as follows:

- **INSERT Statement** - The INSERT statement adds a new row to a table. Explain the syntax.

Explain code snippet which demonstrates adding a new row to the Table_2.
The outcome of this will be that one row with the given data is inserted into the table.

- **UPDATE Statement** - The UPDATE statement modifies the data in the table. Explain the syntax.

Explain code snippet which demonstrates the use of the UPDATE statement to modify the value in column Contact_number.
Figure on slide 18 shows the output of UPDATE statement.

- **DELETE Statement** - The DELETE statement removes rows from a table. Explain the syntax.

Explain code snippet which demonstrates how to delete a row from the Customer_2 table whose Contact_number value is 5432679.

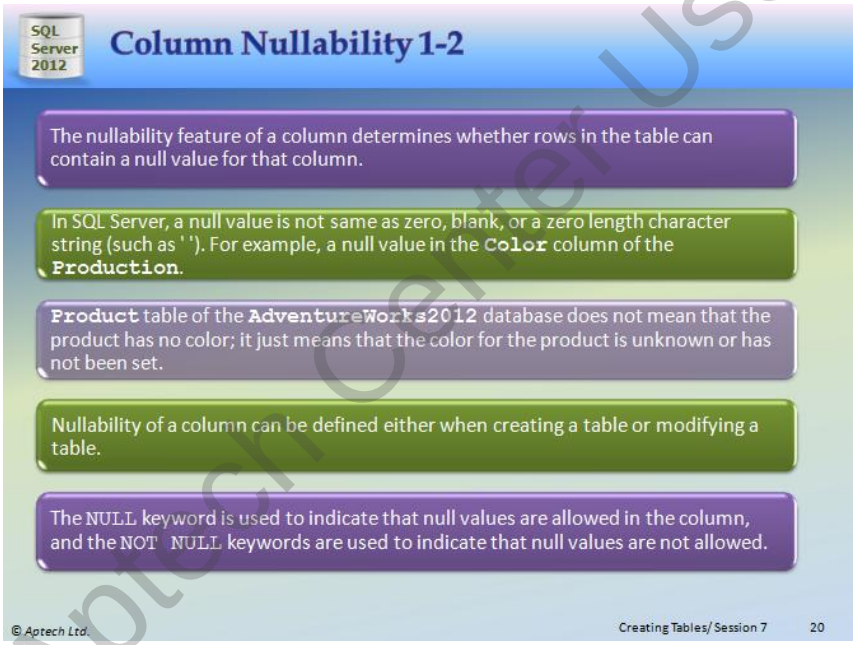
In-Class Question:

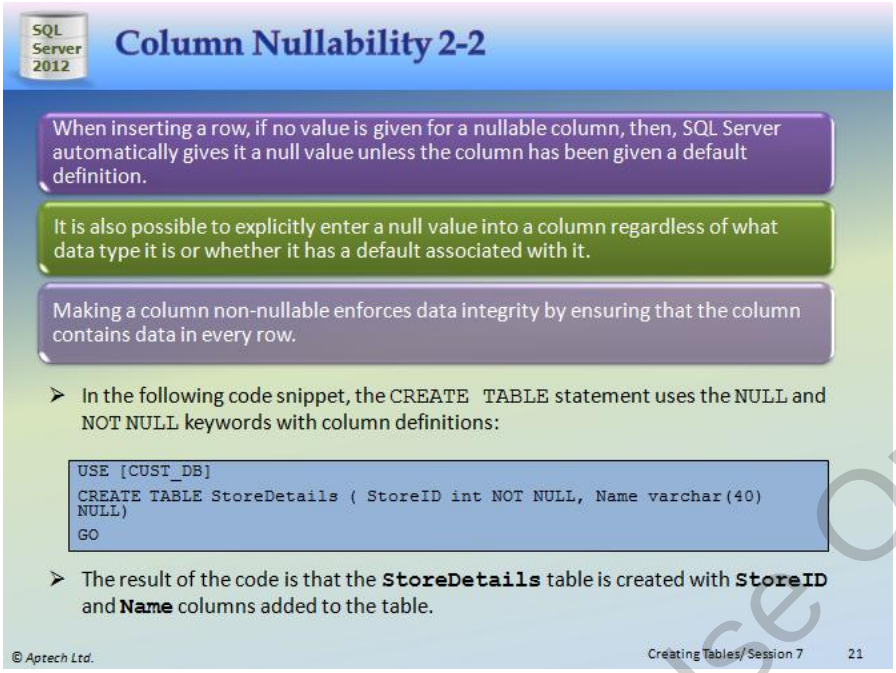
What statement is used to add new row in a table?

Answer:
Insert statement is used to add new row in the table.

Column Nullability

Slides 20 and 21





Using slides 20 and 21, explain the column nullability concept.

The nullability feature of a column determines whether rows in the table can contain a null value for that column. In SQL Server, a null value is not same as zero, blank, or a zero length character string (such as ' '). For example, a null value in the Color column of the Production.Product table of the AdventureWorks2012 database does not mean that the product has no color; it just means that the color for the product is unknown or has not been set.

Nullability of a column can be defined either when creating a table or modifying a table.

The NULL keyword is used to indicate that null values are allowed in the column, and the NOT NULL keywords are used to indicate that null values are not allowed.

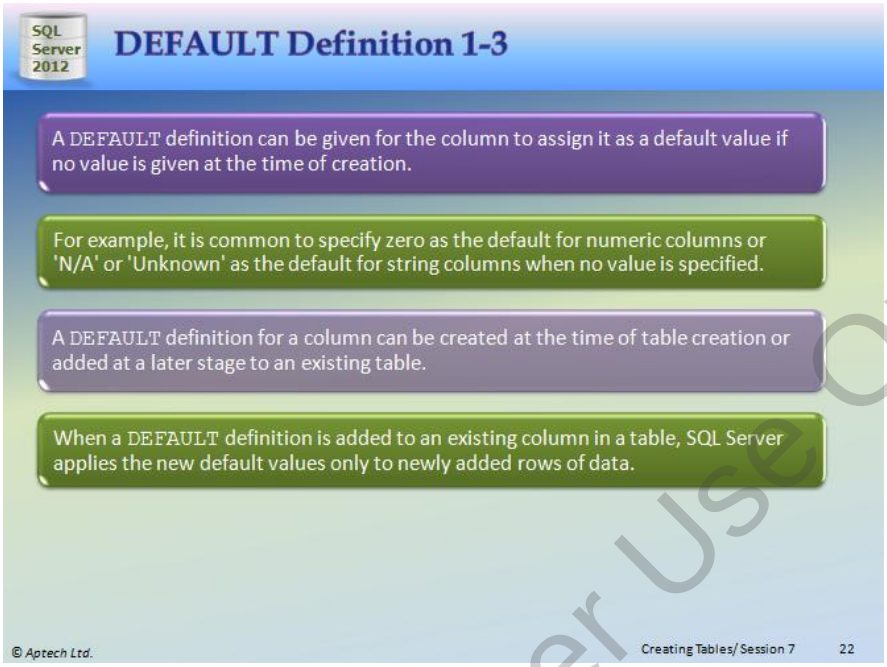
When inserting a row, if no value is given for a nullable column (that is, it allows null values), then, SQL Server automatically gives it a null value unless the column has been given a default definition. It is also possible to explicitly enter a null value into a column regardless of what data type it is or whether it has a default associated with it. Making a column non-nullable (that is, not permitting null values) enforces data integrity by ensuring that the column contains data in every row.

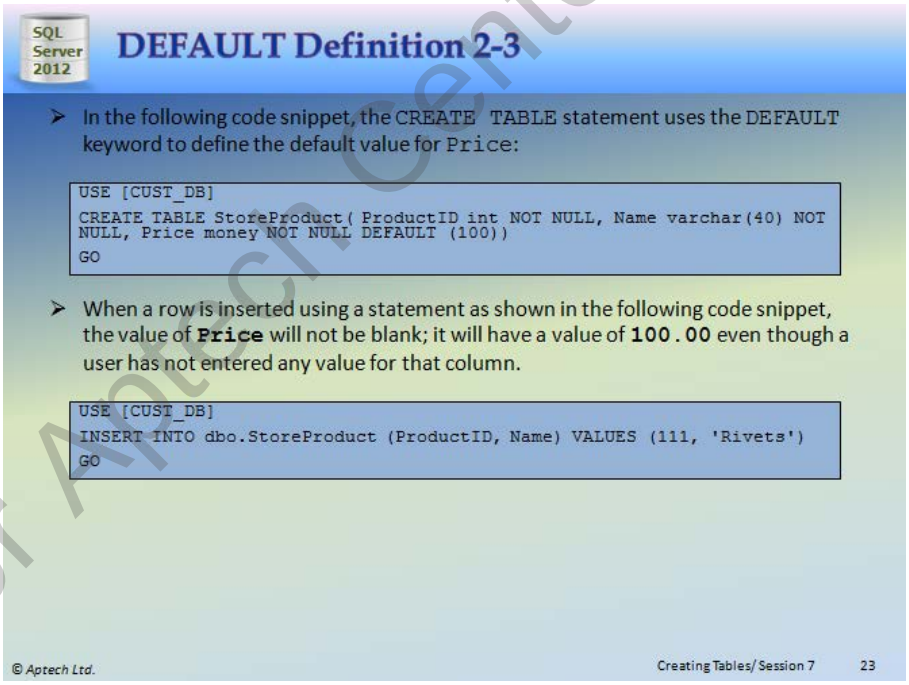
In the code snippet, the CREATE TABLE statement uses the NULL and NOT NULL keywords with column definitions.

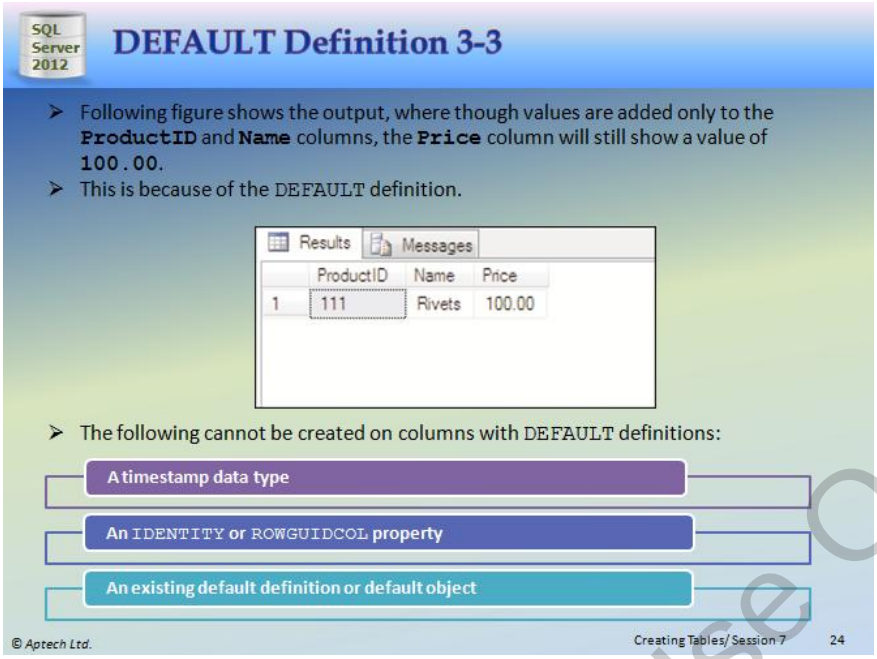
The result of the code is that the StoreDetails table is created with StoreID and Name columns added to the table.

DEFAULT Definition

Slides 22 to 24







Using slides 22 to 24, explain the default definition.

All values for the product details may not be known even at the time of data insertion. However, as per data consistency and integrity rules, the columns in a record should typically contain a value. Storing null values into such columns where the exact value of data is not known may not be desirable or practical.

In such situations, a DEFAULT definition can be given for the column to assign it as a default value if no value is given at the time of creation. For example, it is common to specify zero as the default for numeric columns or 'N/A' or 'Unknown' as the default for string columns when no value is specified.

A DEFAULT definition for a column can be created at the time of table creation or added at a later stage to an existing table. When a DEFAULT definition is added to an existing column in a table, SQL Server applies the new default values only to those rows of data, which have been newly added to the table.

In code snippet on slide 24, the CREATE TABLE statement uses the DEFAULT keyword to define the default value for Price.

When a row is inserted using a statement as shown in code snippet, the value of Price will not be blank; it will have a value of 100.00 even though a user has not entered any value for that column.

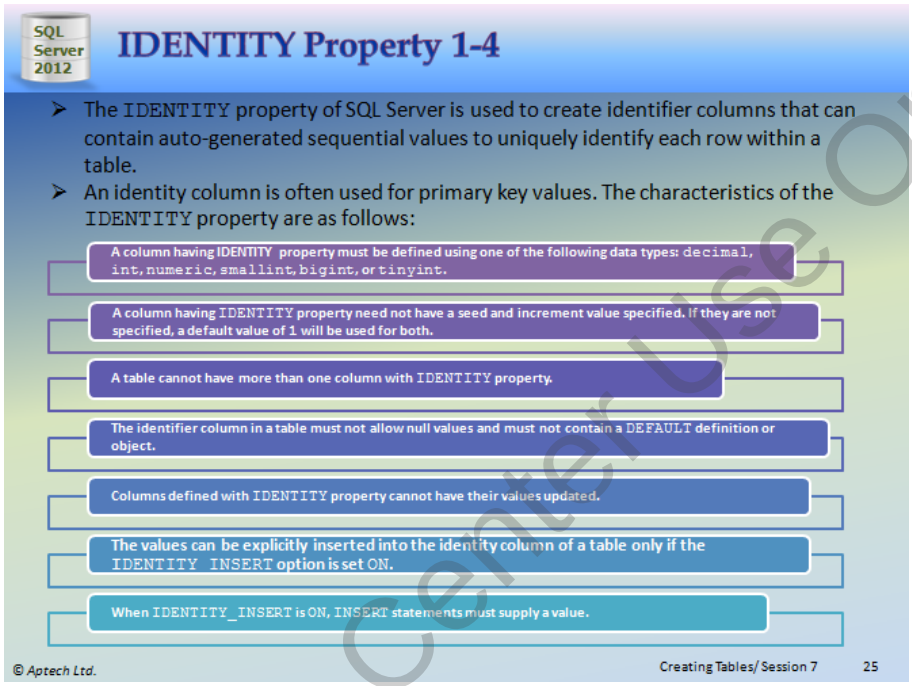
Figure shows the output of code snippet, where though values are added only to the ProductID and Name columns, the Price column will still show a value of 100.00. This is because of the DEFAULT definition.

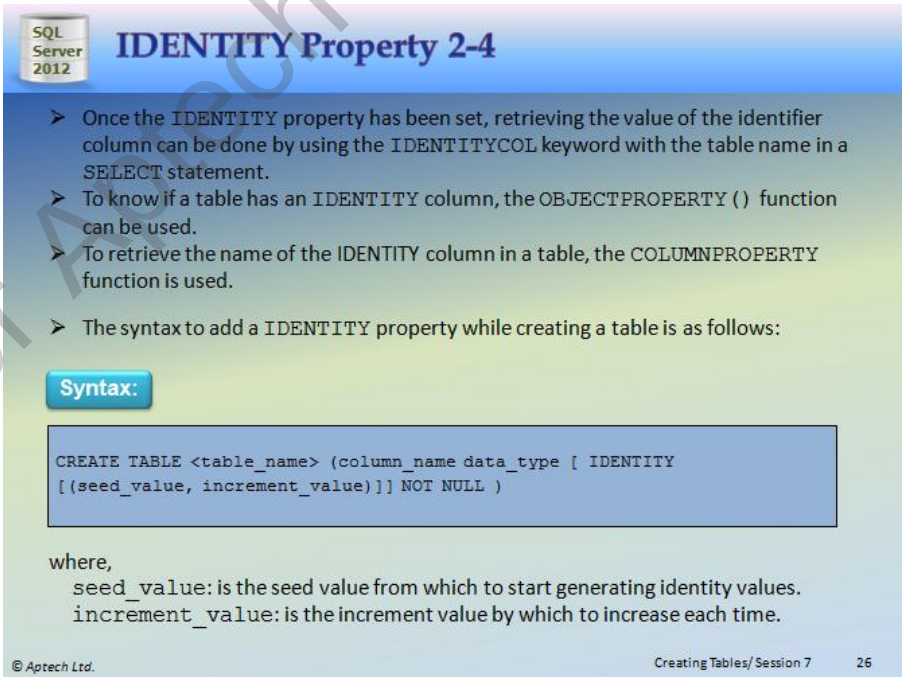
Mention that the following cannot be created on columns with DEFAULT definitions:

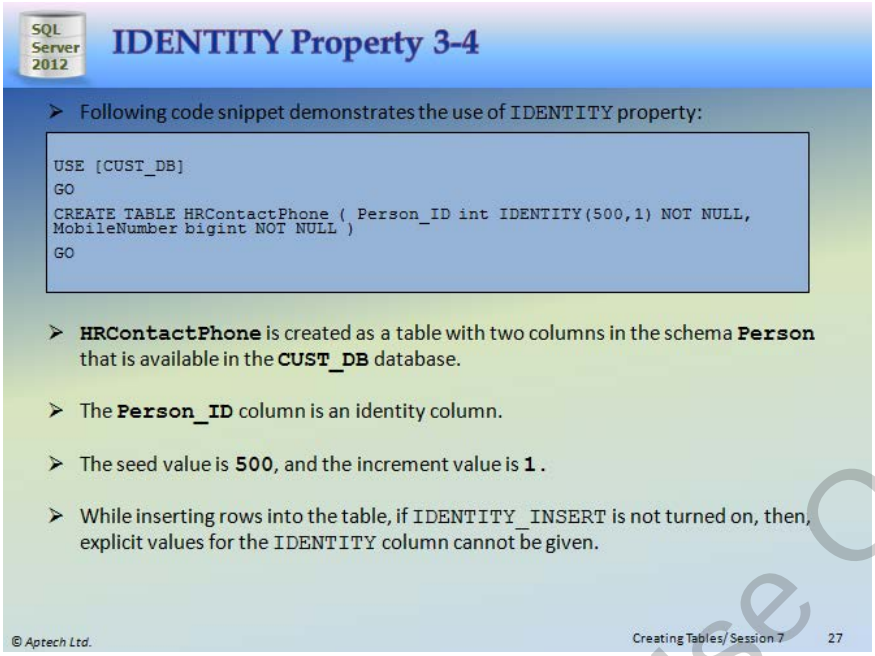
- A timestamp data type
- An IDENTITY or ROWGUIDCOL property
- An existing default definition or default object

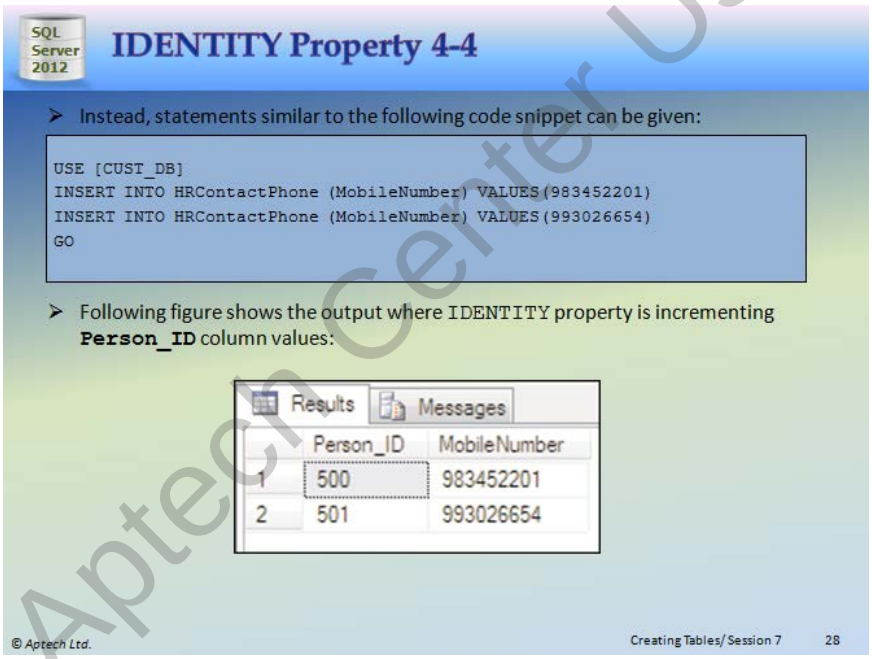
IDENTITY Property

Slides 25 to 28









Using slides 25 to 28, explain the Relational operators.

The IDENTITY property of SQL Server is used to create identifier columns that can contain auto-generated sequential values to uniquely identify each row within a table. For example, an identifier column could be created to generate unique student registration numbers automatically whenever new rows are inserted into the Students table. The identity number for the first row inserted into the table is called seed value and the increment, also called Identity Increment property, is added to the seed in order to generate further identity numbers in sequence. When a new row is inserted into a table with an identifier column, the next identity value is automatically generated by SQL Server by adding the increment to the seed. An identity column is often used for primary key values.

The characteristics of the IDENTITY property are as follows:

- A column having IDENTITY property must be defined using one of the following data types: decimal, int, numeric, smallint, bigint, or tinyint.
- A column having IDENTITY property need not have a seed and increment value specified. If they are not specified, a default value of 1 will be used for both.
- A table cannot have more than one column with IDENTITY property. The identifier column in a table must not allow null values and must not contain a DEFAULT definition or object.
- Columns defined with IDENTITY property cannot have their values updated.
- The values can be explicitly inserted into the identity column of a table only if the IDENTITY_INSERT option is set ON. When IDENTITY_INSERT is ON, INSERT statements must supply a value.

The advantage of identifier columns is that SQL Server can automatically provide key values, thus reducing costs and improving performance. Using identifier columns simplifies programming and keeps primary key values short.

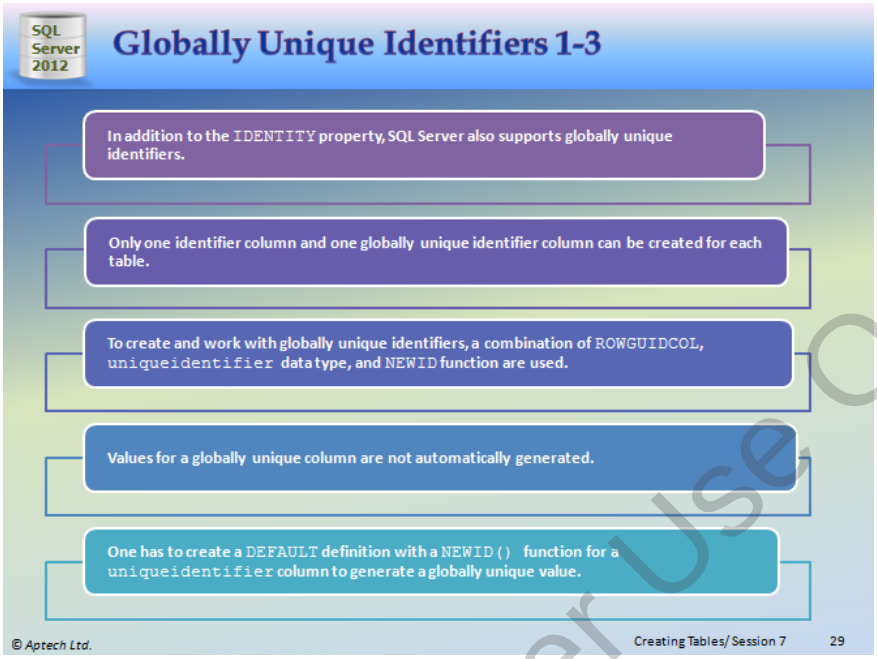
Once the IDENTITY property has been set, retrieving the value of the identifier column can be done by using the IDENTITYCOL keyword with the table name in a SELECT statement. To know if a table has an IDENTITY column, the OBJECTPROPERTY() function can be used. To retrieve the name of the IDENTITY column in a table, the COLUMNPROPERTY function is used.

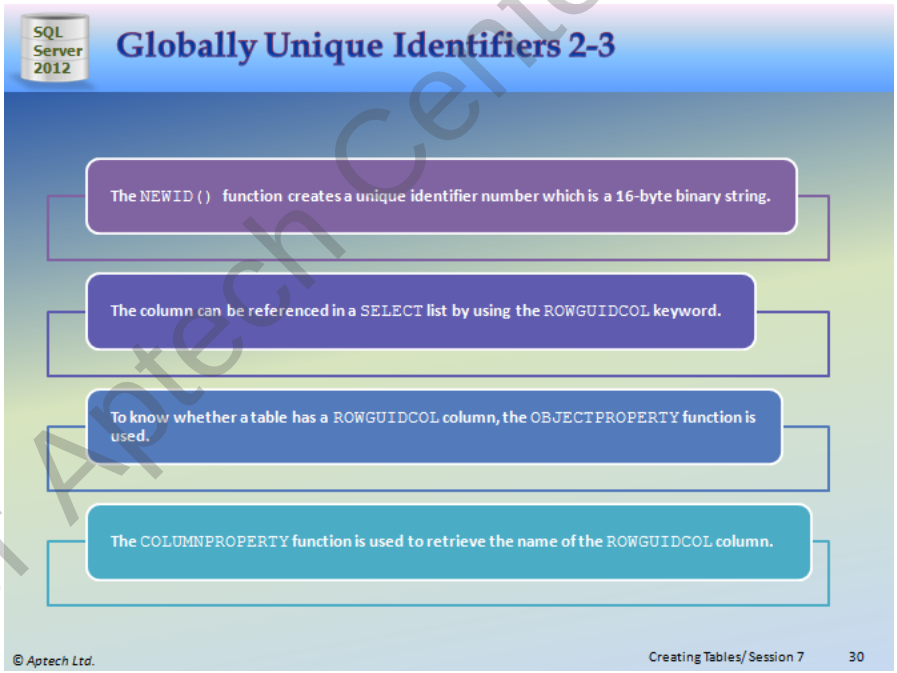
Explain the syntax for IDENTITY property using slide 26.
Code snippet on slide 27 demonstrates the use of IDENTITY property. HRContactPhone is created as a table with two columns in the schema Person that is available in the CUST_DB database. The Person_ID column is an identity column. The seed value is 500, and the increment value is 1.

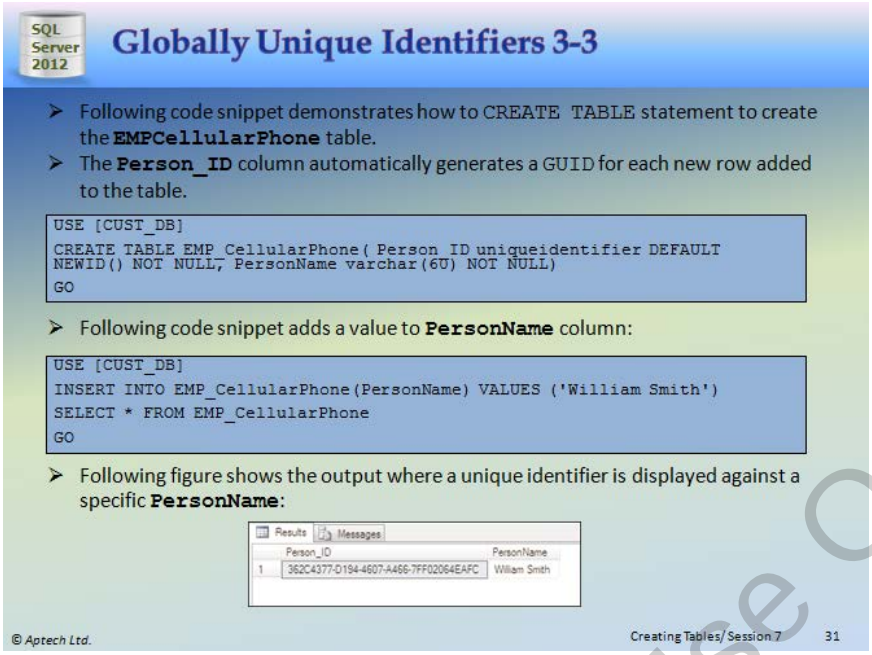
While inserting rows into the table, if IDENTITY_INSERT is not turned on, then explicit values for the IDENTITY column cannot be given is shown on slide 28. Figure shows the output where IDENTITY property is incrementing Person_ID column values.

Globally Unique Identifiers

Slides 29 to 31







Using slides 29 to 31, explain the globally unique identifier.

In addition to the IDENTITY property, SQL Server also supports globally unique identifiers. Often, in a networked environment, many tables may need to have a column consisting of a common globally unique value. Consider a scenario where data from multiple database systems such as banking databases must be consolidated at a single location. When the data from around the world is collated at the central site for consolidation and reporting, using globally unique values prevents customers in different countries from having the same bank account number or customer ID. To satisfy this need, SQL Server provides globally unique identifier columns. These can be created for each table containing values that are unique across all the computers in a network. Only one identifier column and one globally unique identifier column can be created for each table. To create and work with globally unique identifiers, a combination of ROWGUIDCOL, uniqueidentifier data type, and NEWID function are used.

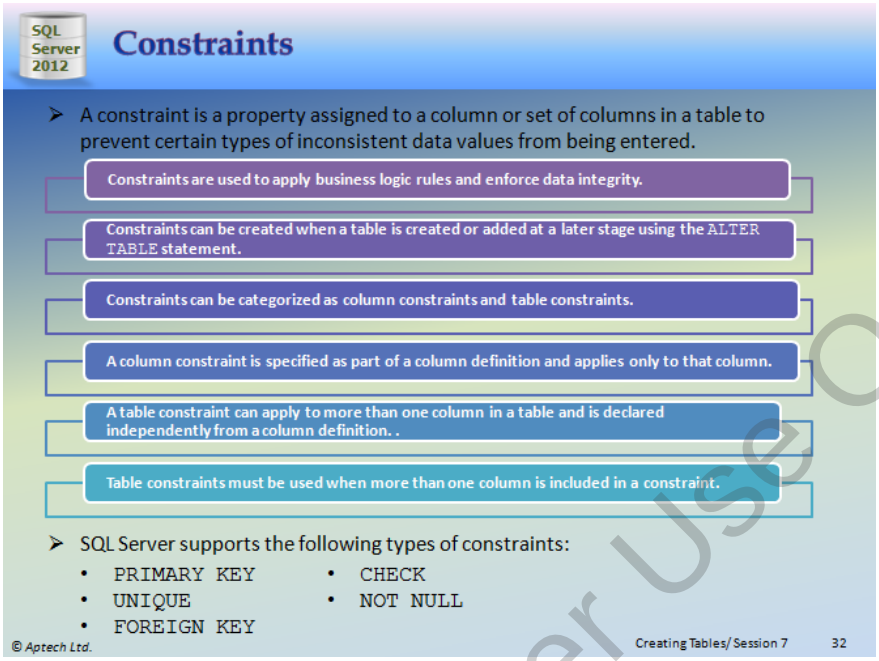
Values for a globally unique column are not automatically generated. One has to create a DEFAULT definition with a NEWID() function for a uniqueidentifier column to generate a globally unique value. The NEWID() function creates a unique identifier number which is a 16-byte binary string. The column can be referenced in a SELECT list by using the ROWGUIDCOL keyword.

To know whether a table has a ROWGUIDCOL column, the OBJECTPROPERTY function is used. The COLUMNPROPERTY function is used to retrieve the name of the ROWGUIDCOL column. Code snippet on slide 31 demonstrates how to CREATE TABLE statement to create the EMPCellularPhone table.

The Person_ID column automatically generates a GUID for each new row added to the table. Code snippet on slide 31 inserts a value to PersonName column. Figure shows the output where a unique identifier is displayed against a specific PersonName.

Constraints

Slide 32



Using slide 32, explain the constraints.
One of the important functions of SQL Server is to maintain and enforce data integrity. There are a number of means to achieve this but one of the commonly used and preferred methods is to use constraints. A constraint is a property assigned to a column or set of columns in a table to prevent certain types of inconsistent data values from being entered. Constraints are used to apply business logic rules and enforce data integrity. Constraints can be created when a table is created, as part of the table definition by using the CREATE TABLE statement or can be added at a later stage using the ALTER TABLE statement.

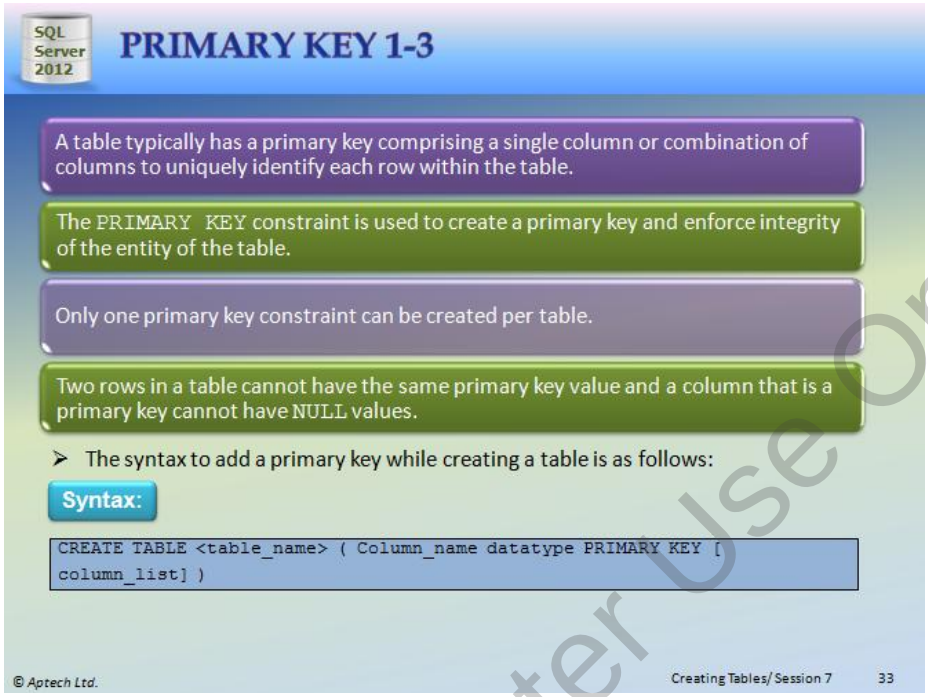
Constraints can be categorized as column constraints and table constraints. A column constraint is specified as part of a column definition and applies only to that column. A table constraint can apply to more than one column in a table and is declared independently from a column definition. Table constraints must be used when more than one column is included in a constraint.

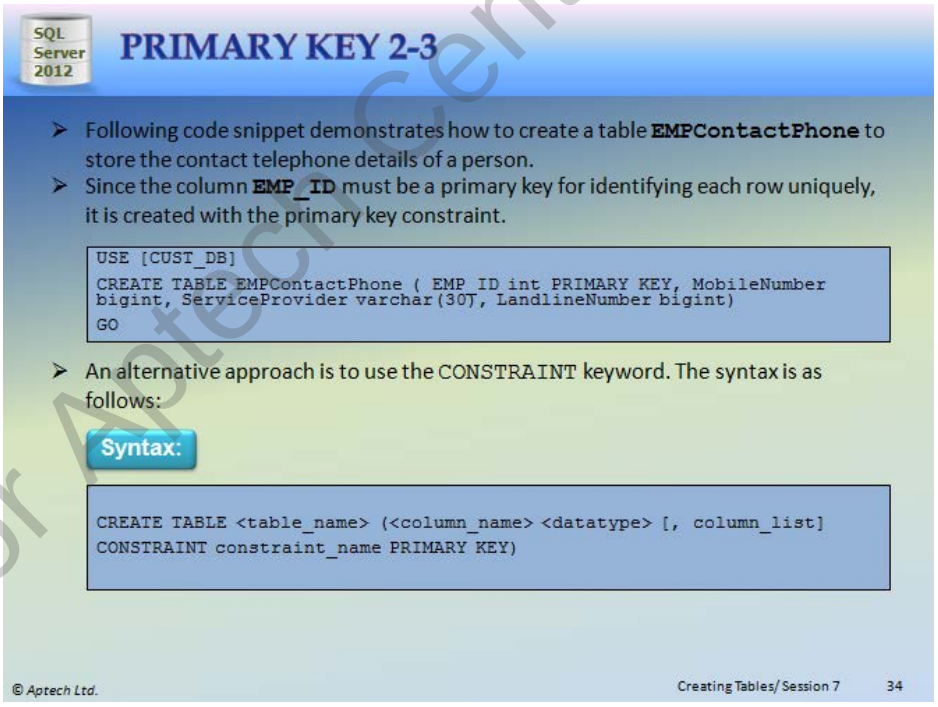
SQL Server supports the following types of constraints:

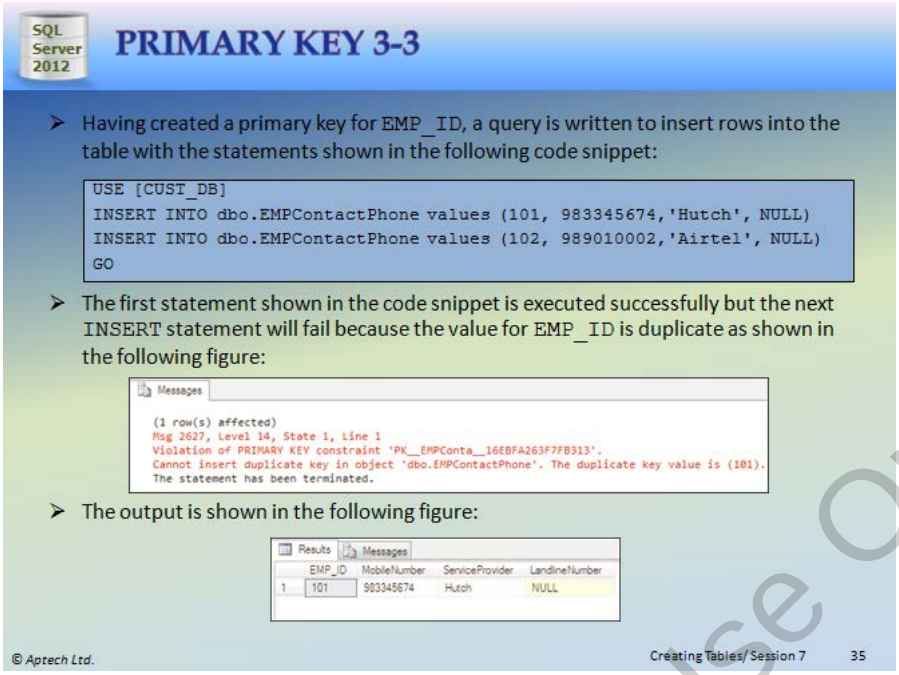
- **NOT NULL** - Indicates that a column cannot store NULL value
- **CHECK** - Ensures that the value in a column meets a specific condition
- **UNIQUE** - Ensures that each row for a column must have a unique value
- **FOREIGN KEY** - Ensures the referential integrity of the data in one table to match values in another table
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly
- **DEFAULT** - Specifies a default value when specified none for this column

Primary Key

Slides 33 to 35







Using slides 33 to 35, explain the primary key.

A table typically has a primary key comprising a single column or combination of columns to uniquely identify each row within the table. The PRIMARY KEY constraint is used to create a primary key and enforce integrity of the entity of the table. Only one primary key constraint can be created per table. Two rows in a table cannot have the same primary key value and a column that is a primary key cannot have NULL values. Hence, when a primary key constraint is added to existing columns of a table, SQL Server 2012 checks to see if the rules for primary key are complied with. If the existing data in the columns do not comply with the rules for primary key, then the constraint will not be added. Explain the syntax to add a primary key while creating a table.

Explain the code snippet that demonstrates how to create a table EMPContactPhone to store the contact telephone details of a person. Since the column EMP_ID must be a primary key for identifying each row uniquely, it is created with the primary key constraint.
An alternative approach is to use the CONSTRAINT keyword. Explain the syntax.
Having created a primary key for EMP_ID, a query is written to insert rows into the table with the statements shown in the code snippet.

The first statement shown in the code snippet is executed successfully but the next INSERT statement will fail because the value for EMP_ID is duplicated, as shown in figure.
The output of code snippet is shown in figure.

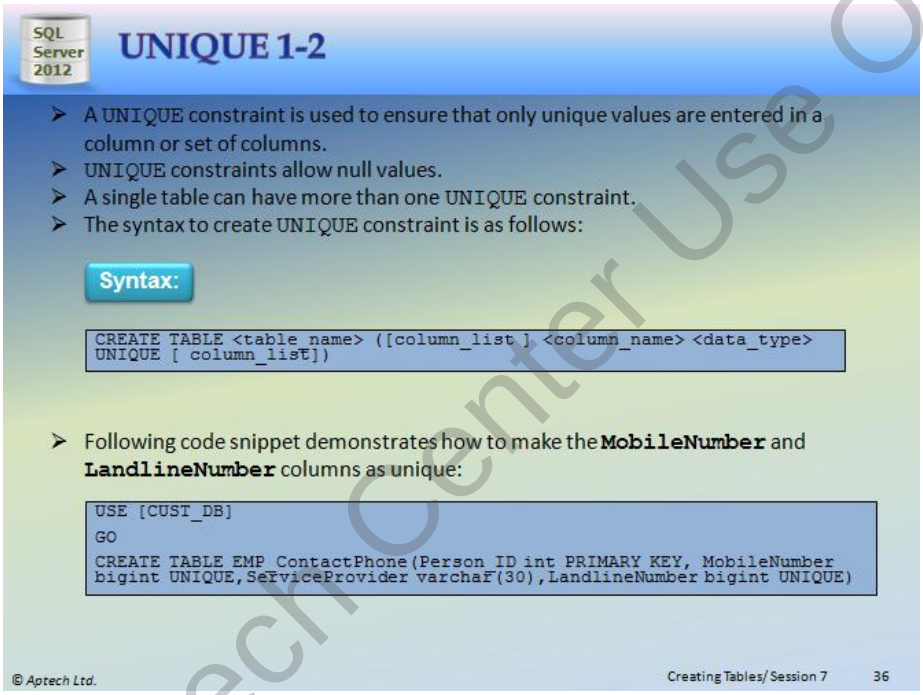
In-Class Question:

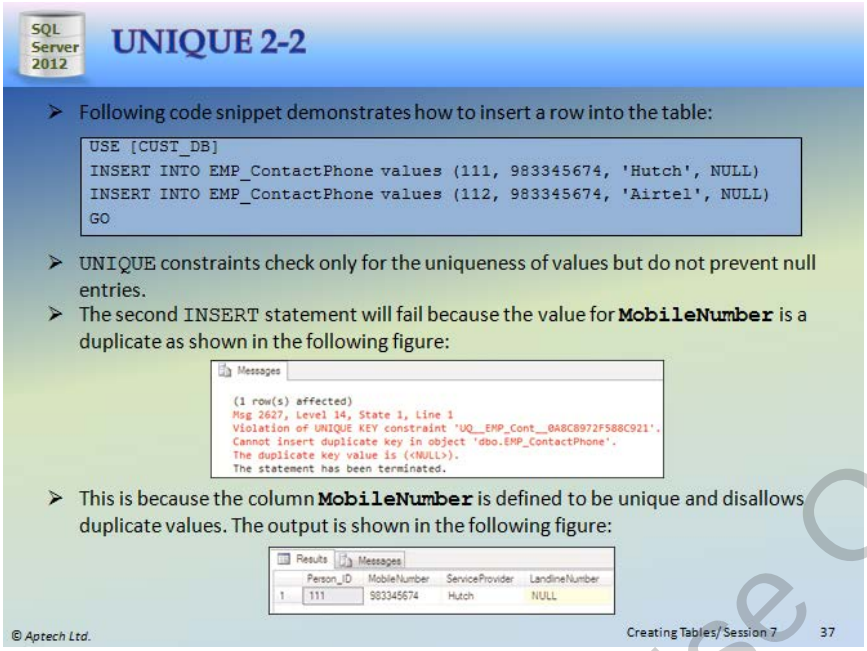
What is the use of primary key?

Answer:
A table typically has a primary key comprising a single column or combination of columns to uniquely identify each row within the table.

UNIQUE Constraint

Slides 36 and 37





Using slides 36 and 37, explain the unique constraints.

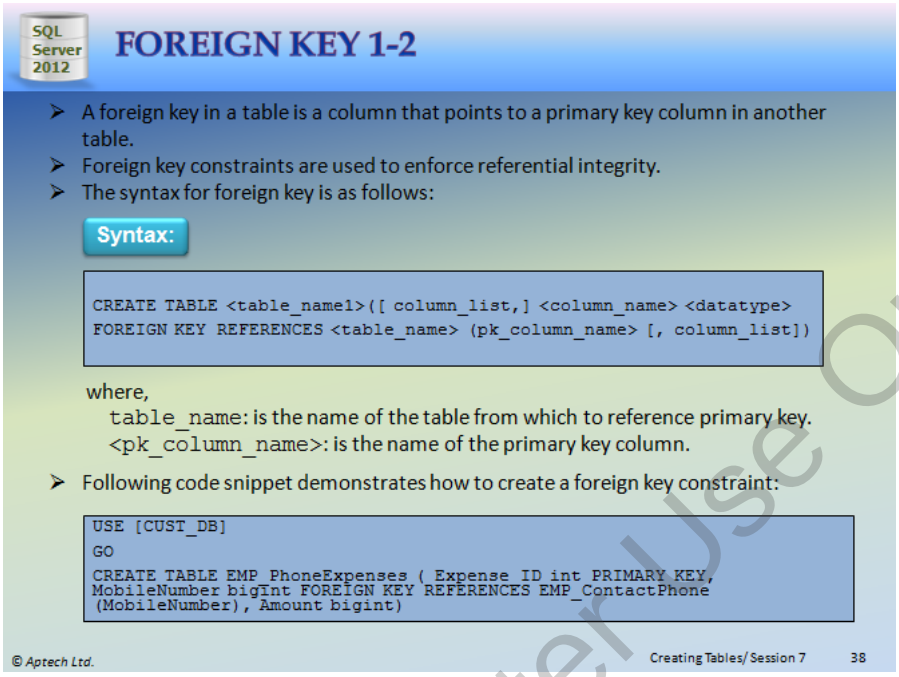
A UNIQUE constraint is used to ensure that only unique values are entered in a column or set of columns. It allows developers to make sure that no duplicate values are entered. Primary keys are implicitly unique. Unique key constraints enforce entity integrity because once the constraints are applied; no two rows in the table can have the same value for the columns. UNIQUE constraints allow null values. A single table can have more than one UNIQUE constraint. Explain the syntax to create UNIQUE constraint.

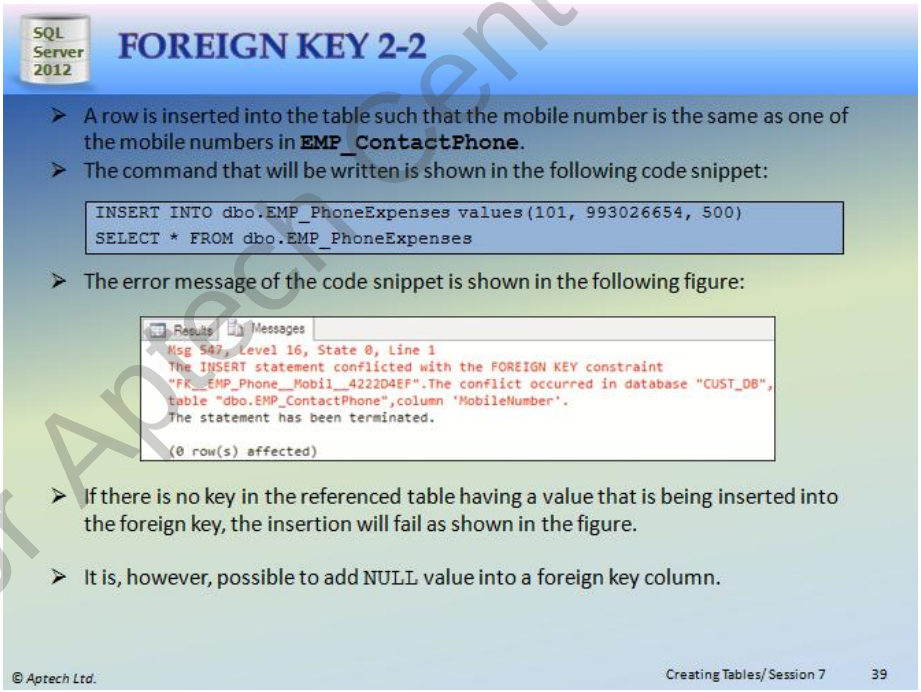
Explain the code snippet that demonstrates how to make the MobileNumber and LandlineNumber columns as unique. Explain the code snippet that demonstrates how to insert a row into the table.

Though a value of NULL has been given for the LandlineNumber columns, which are defined as UNIQUE, the command will execute successfully because UNIQUE constraints check only for the uniqueness of values but do not prevent null entries. The first statement shown in code snippet is executed successfully but the next INSERT statement will fail even though the primary key value is different because the value for MobileNumber is a duplicate as shown in figure. This is because the column MobileNumber is defined to be unique and disallows duplicate values. The output of code snippet is shown in figure.

Foreign Key

Slides 38 and 39





Using slides 38 and 39, explain the foreign key.

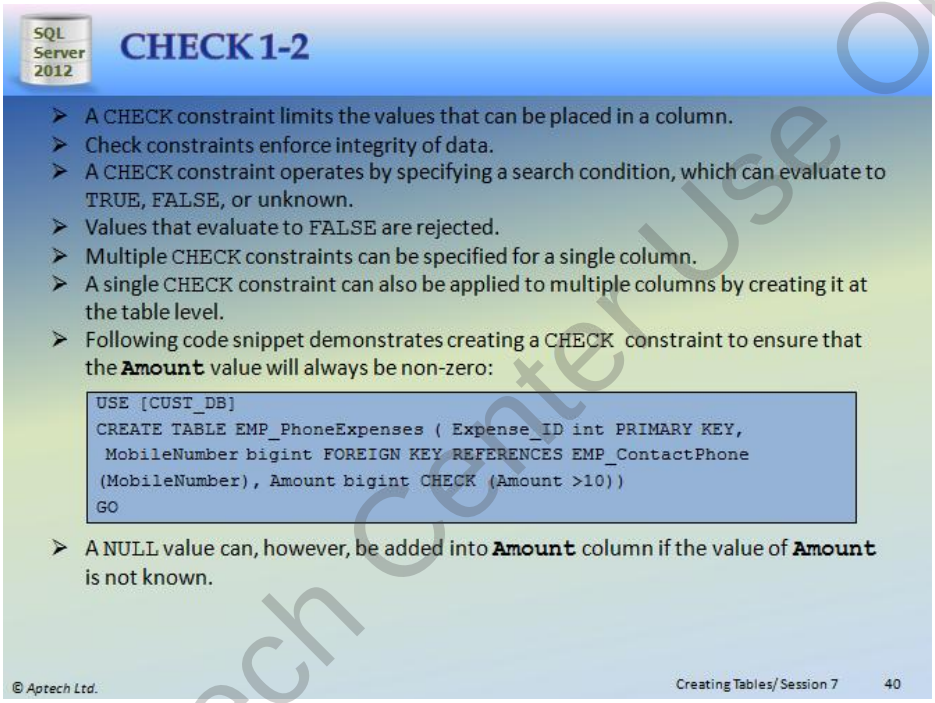
A foreign key in a table is a column that points to a primary key column in another table. Foreign key constraints are used to enforce referential integrity. Explain the syntax for foreign key.

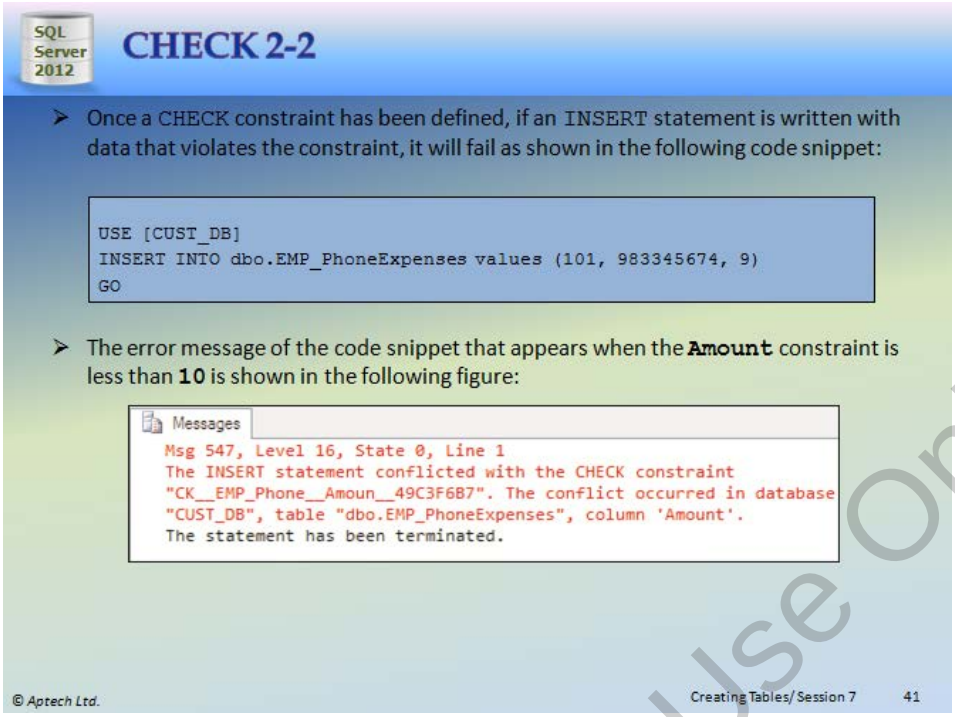
A row is inserted into the table such that the mobile number is the same as one of the mobile numbers in EMP_ContactPhone. The command that will be written is shown in code snippet. The error message of code snippet is shown in figure.

If there is no key in the referenced table having a value that is being inserted into the foreign key, the insertion will fail as shown in figure. It is, however, possible to add NULL value into a foreign key column.

CHECK Constraint

Slides 40 and 41





Using slides 40 and 41, explain the CHECK constraint.
A CHECK constraint limits the values that can be placed in a column. Check constraints enforce integrity of data. For example, a CHECK constraint can be given to check if the value being entered into VoterAge is greater than or equal to 18. If the data being entered for the column does not satisfy the condition, then insertion will fail.

A CHECK constraint operates by specifying a search condition, which can evaluate to TRUE, FALSE, or unknown. Values that evaluate to FALSE are rejected. Multiple CHECK constraints can be specified for a single column. A single CHECK constraint can also be applied to multiple columns by creating it at the table level.

Code snippet demonstrates creating a CHECK constraint to ensure that the Amount value will always be non-zero. A NULL value can, however, be added into Amount column if the value of Amount is not known.

Once a CHECK constraint has been defined, if an INSERT statement is written with data that violates the constraint, it will fail as shown in code snippet.
The error message of code snippet that appears when the Amount constraint is less than 10 is shown in figure.

NOT NULL

Slide 42

Using slide 42, explain NOT NULL constraint.

A NOT NULL constraint enforces that the column will not accept null values. The NOT NULL constraints are used to enforce domain integrity, similar to CHECK constraints.

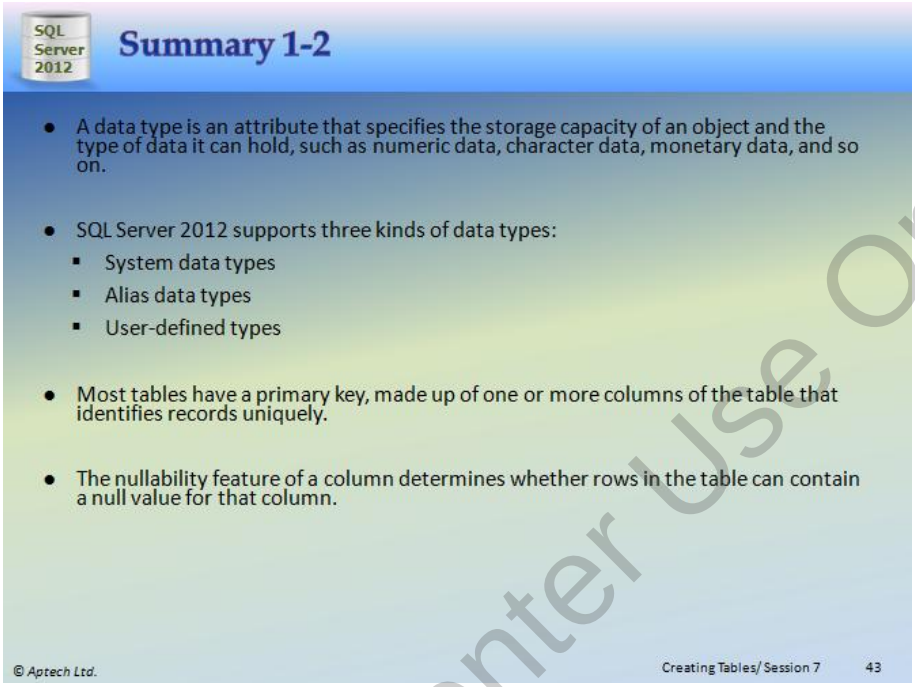
The following code snippet is an example of creating not null constraint on columns:

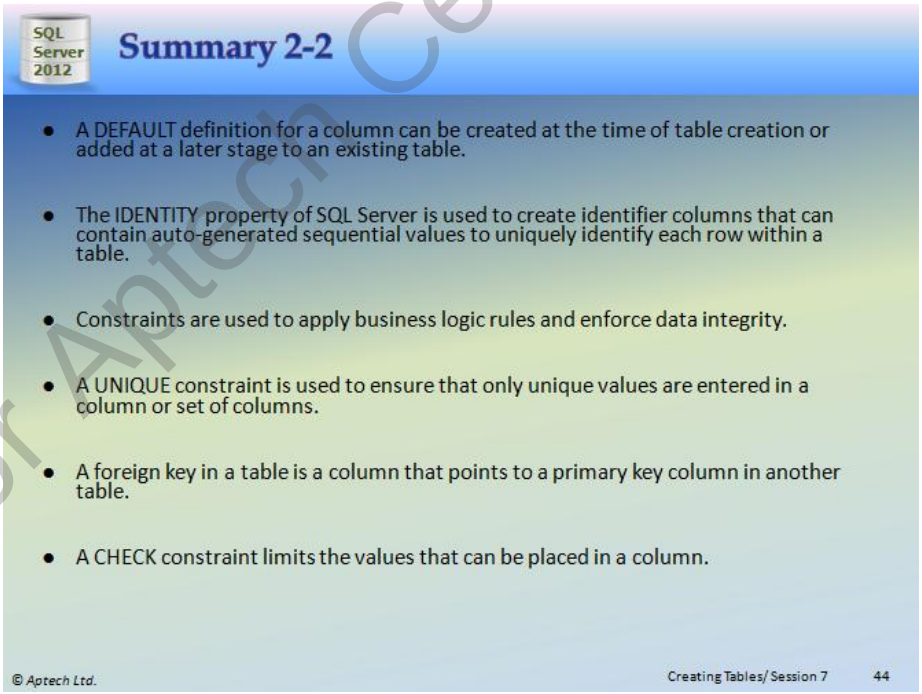
```
CREATE TABLE [dbo].[Message](
  [Id] [int] IDENTITY(1,1) NOT NULL,
  [Message] [int] NOT NULL
)
```

The message value in the table cannot be null.

Summarize Session

Slides 43 and 44





Using slides 43 and 44, summarize the session. End the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session.

Explain each of the following points in brief. Tell them that:

- A data type is an attribute that specifies the storage capacity of an object and the type of data it can hold, such as numeric data, character data, monetary data, and so on.
- SQL Server 2012 supports three kinds of data types:
  - System data types
  - Alias data types
  - User-defined types
- Most tables have a primary key, made up of one or more columns of the table that identifies records uniquely.
- The nullability feature of a column determines whether rows in the table can contain a null value for that column.
- A DEFAULT definition for a column can be created at the time of table creation or added at a later stage to an existing table.
- The IDENTITY property of SQL Server is used to create identifier columns that can contain auto-generated sequential values to uniquely identify each row within a table.
- Constraints are used to apply business logic rules and enforce data integrity.
- A UNIQUE constraint is used to ensure that only unique values are entered in a column or set of columns.
- A foreign key in a table is a column that points to a primary key column in another table.
- A CHECK constraint limits the values that can be placed in a column.

7.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Accessing Data topic offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 8 – Accessing Data

8.1 Introduction

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

8.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe the SELECT statement, its syntax, and use
- Explain the various clauses used with SELECT
- State the use of ORDER BY clause
- Describe working with typed and untyped XML
- Explain the procedure to create, use, and view XML schemas
- Explain use of XQuery to access XML data

8.1.2 Teaching Skills

To teach this session, you should be well-versed with concept about the SELECT statement, the expressions, and the various clauses used with SELECT statement. Along with this, also prepare yourself with the new xml data type and describes how to work with XML data in SQL Server 2012 tables.

The session also covers the XQuery language, which is used to query XML data, is also discussed in the session.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slide and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:
Follow the order given here during In-Class activities.

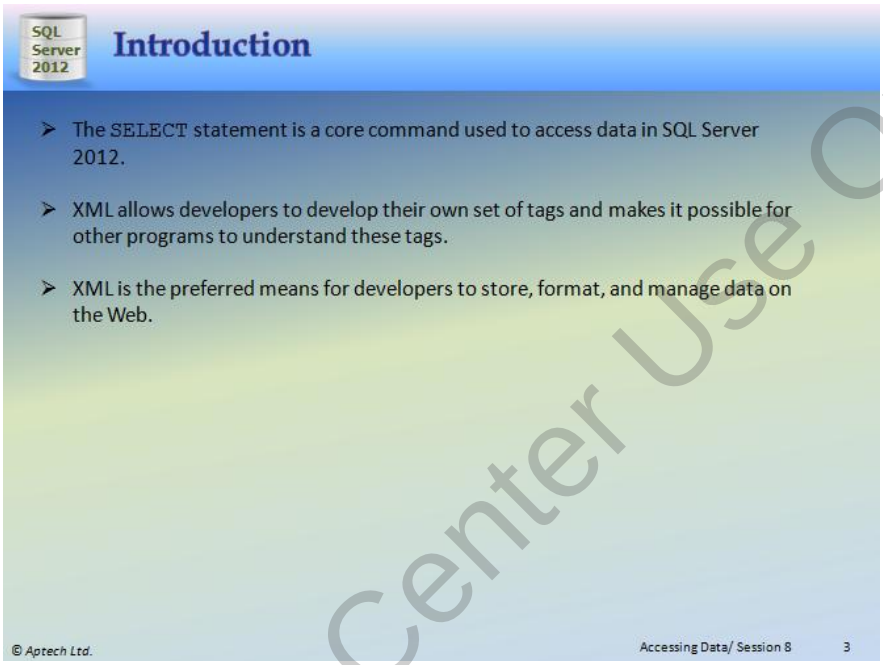
Overview of the Session:
Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces SELECT statement, the expressions, and the various clauses used with SELECT statement. They will learn about the new xml data type and describes how to work with XML data in

SQL Server 2012 tables. They will also know about the new xml data type and describes how to work with XML data in SQL Server 2012 tables.

8.2 In-Class Explanations

Introduction

Slide 3

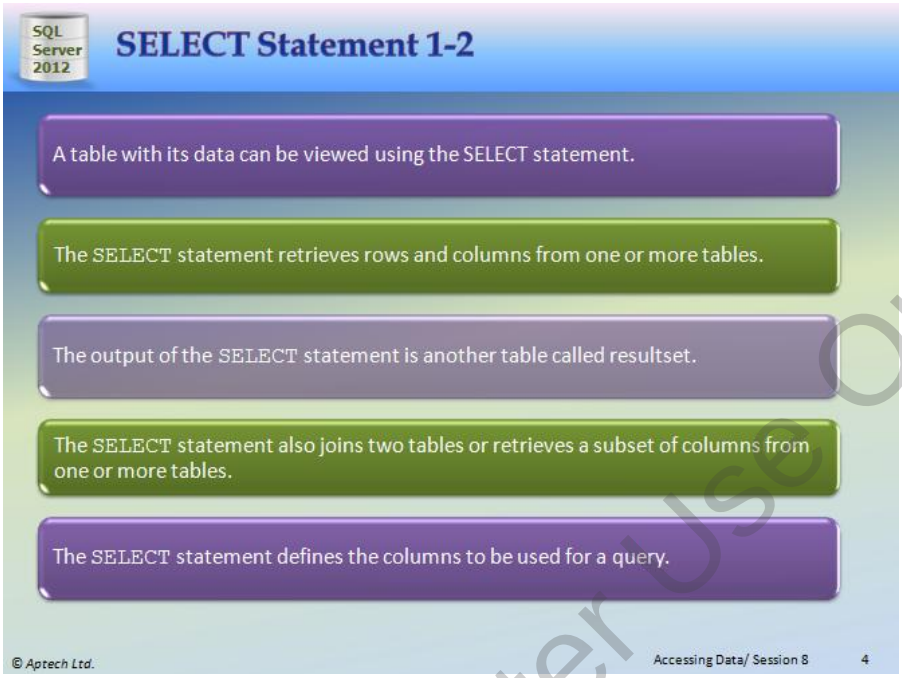


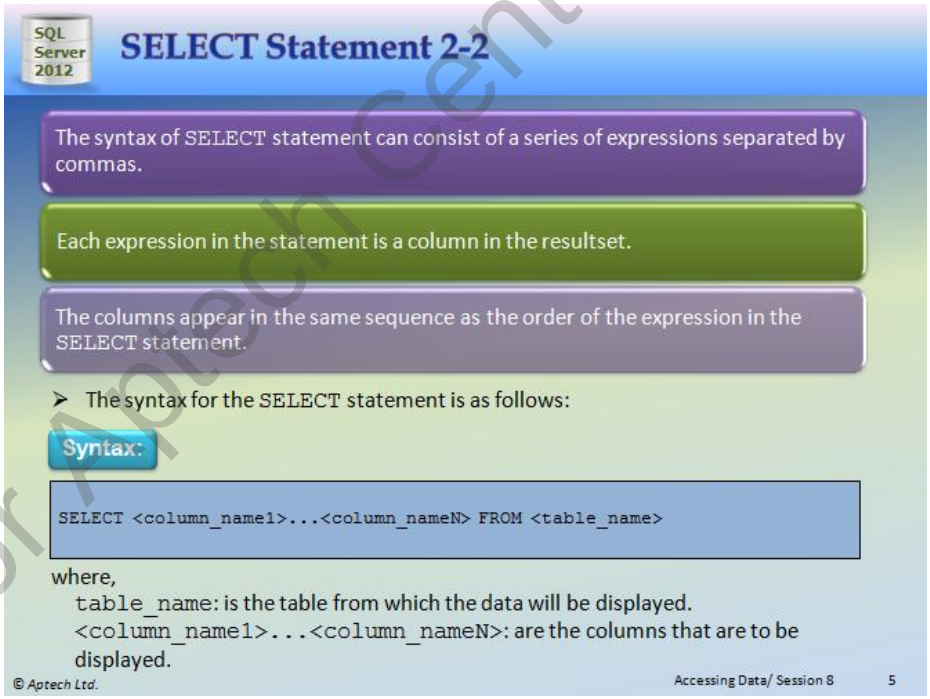
Using slide 3, explain SELECT statement.

The SELECT statement is a core command used to access data in SQL Server 2012. XML allows developers to develop their own set of tags and makes it possible for other programs to understand these tags. XML is the preferred means for developers to store, format, and manage data on the Web.

SELECT Statement

Slides 4 and 5





Using slides 4 and 5, explain the SELECT statement in details.

A table with its data can be viewed using the SELECT statement. The SELECT statement in a query will display the required information in a table.

The SELECT statement retrieves rows and columns from one or more tables. The output of the SELECT statement is another table called resultset. The SELECT statement also joins two

tables or retrieves a subset of columns from one or more tables. The SELECT statement defines the columns to be used for a query. The syntax of SELECT statement can consist of a series of expressions separated by commas. Each expression in the statement is a column in the resultset. The columns appear in the same sequence as the order of the expression in the SELECT statement.

The SELECT statement retrieves rows from the database and enables the selection of one or many rows or columns from a table. Explain the syntax for the SELECT statement.

All commands in SQL Server 2012 do not end with a semicolon.

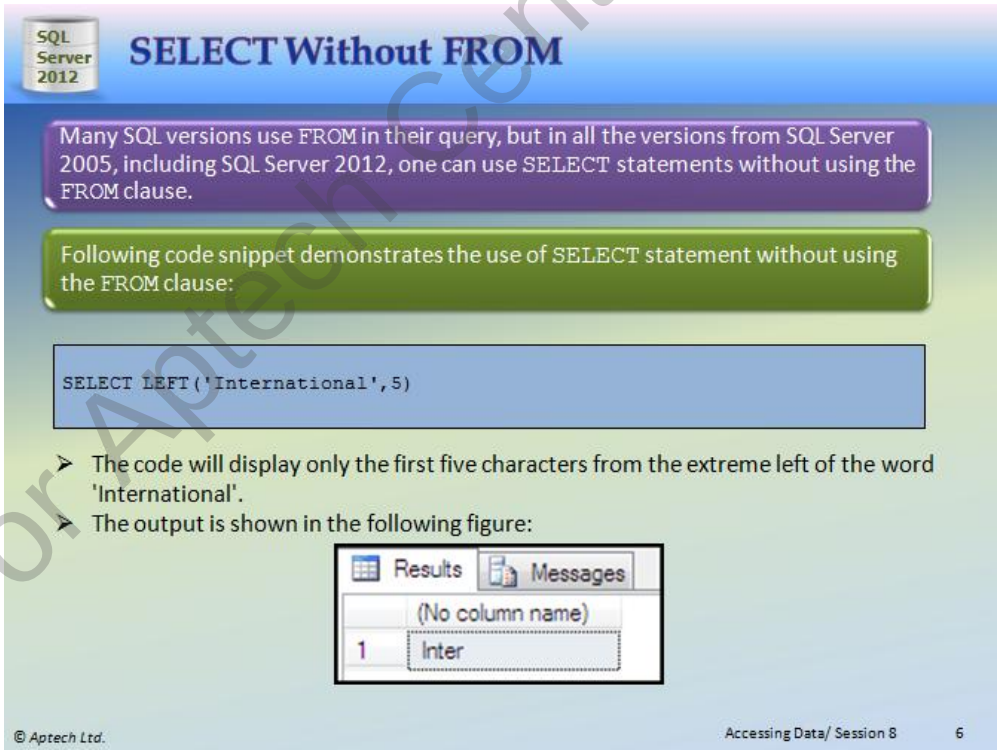
In-Class Question:

What is use of SELECT statement?

Answer:
The SELECT statement retrieves rows and columns from one or more tables.

SELECT Without FROM

Slide 6

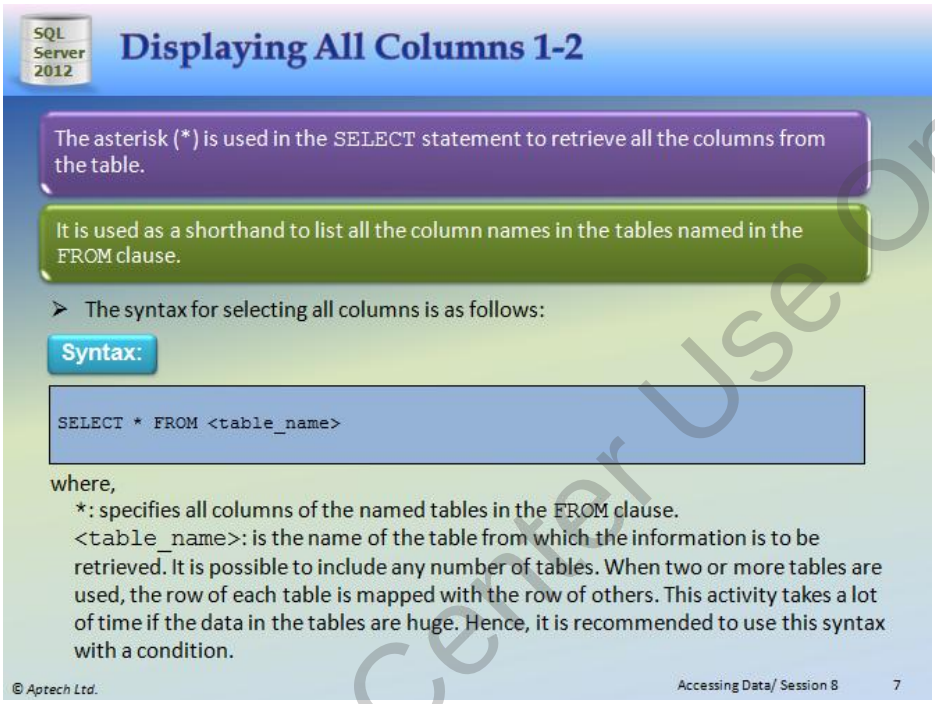


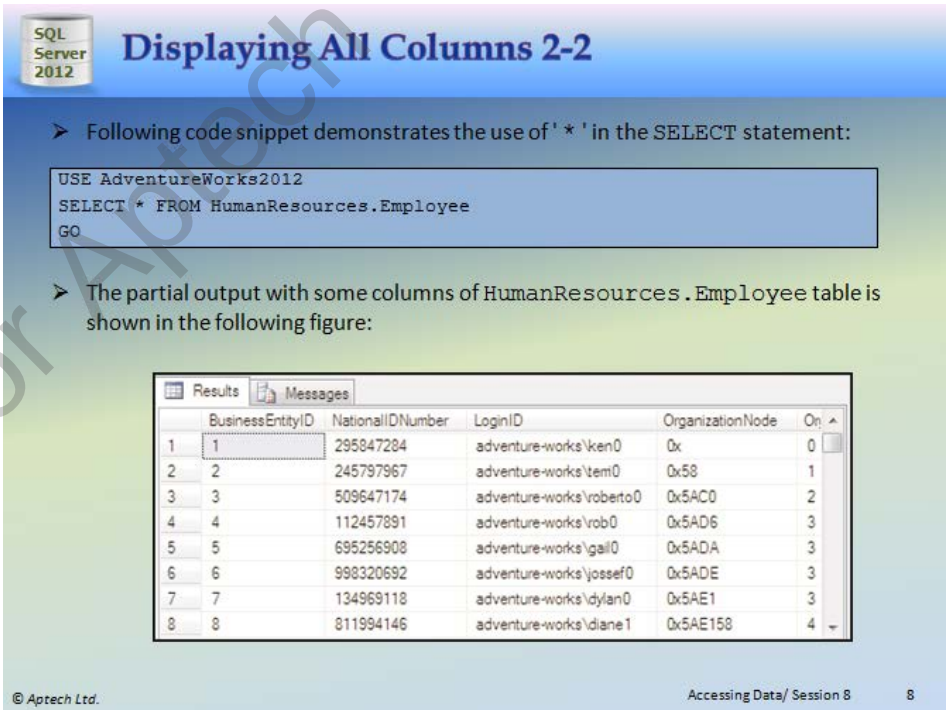
Using slide 6, explain the SELECT statement without FROM.
Many SQL versions use FROM in their query, but in all the versions from SQL Server 2005, including SQL Server 2012, one can use SELECT statements without using the FROM clause.

Explain code snippet which demonstrates the use of SELECT statement without using the FROM clause. The code will display only the first five characters from the extreme left of the word 'International'. The output is shown in figure.

Displaying All Columns

Slides 7 and 8





Using slides 7 and 8, explain how to display all columns.

The asterisk (*) is used in the SELECT statement to retrieve all the columns from the table. It is used as a shorthand to list all the column names in the tables named in the FROM clause. Explain the syntax for selecting all columns.

Explain the code snippet which demonstrates the use of ' * ' in the SELECT statement.
The partial output of code snippet with some columns of HumanResources.Employee table is shown in figure.

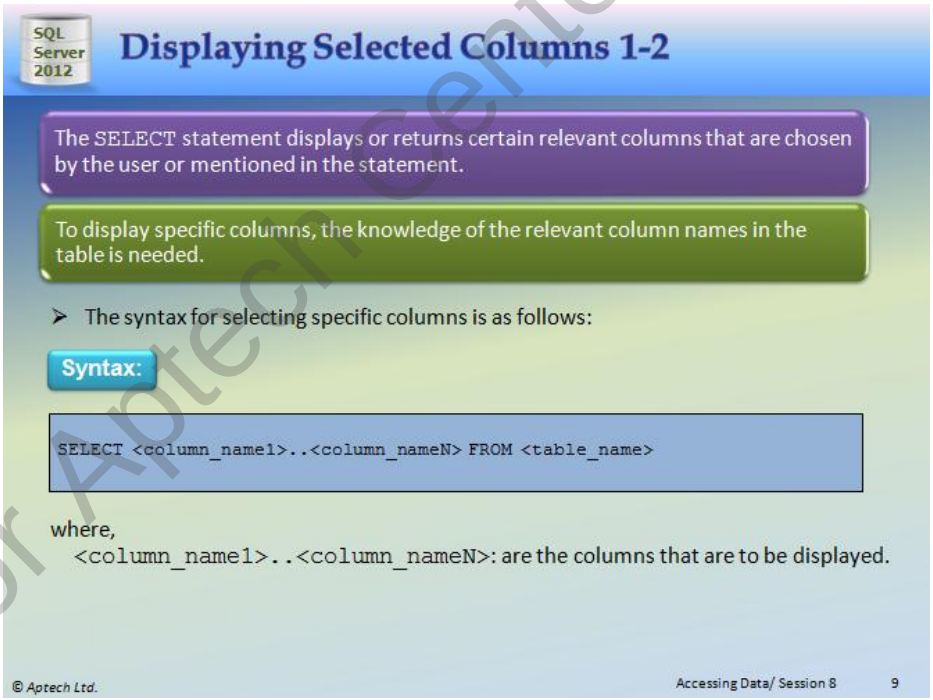
In-Class Question:
After you finish explaining how to display all columns, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

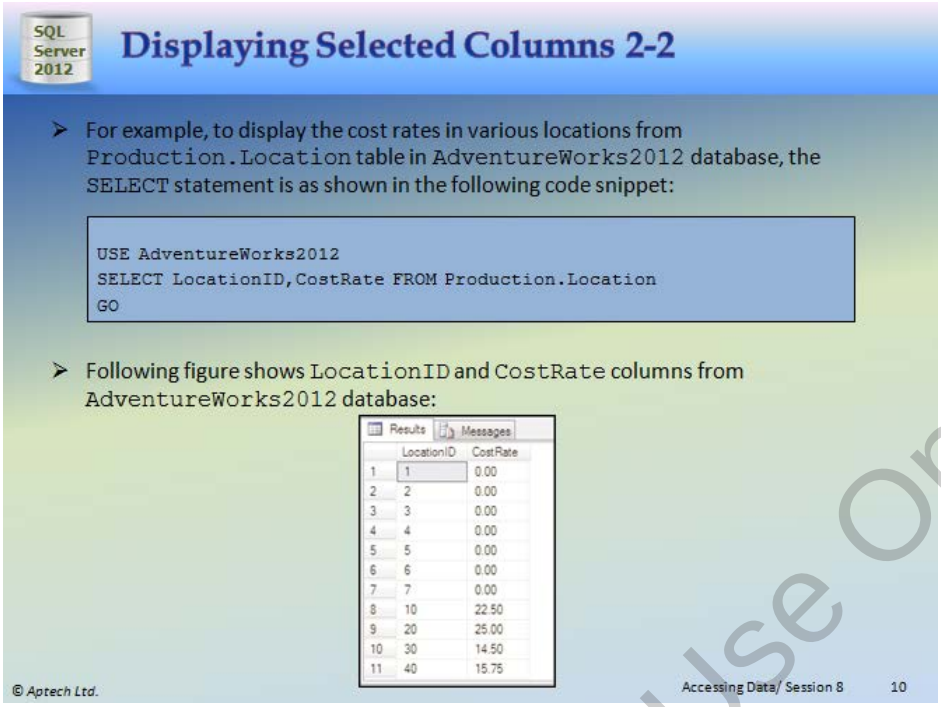
 What is syntax to display all columns?

Answer:
SELECT * FROM < tablename>

Displaying Selected Columns

Slides 9 and 10





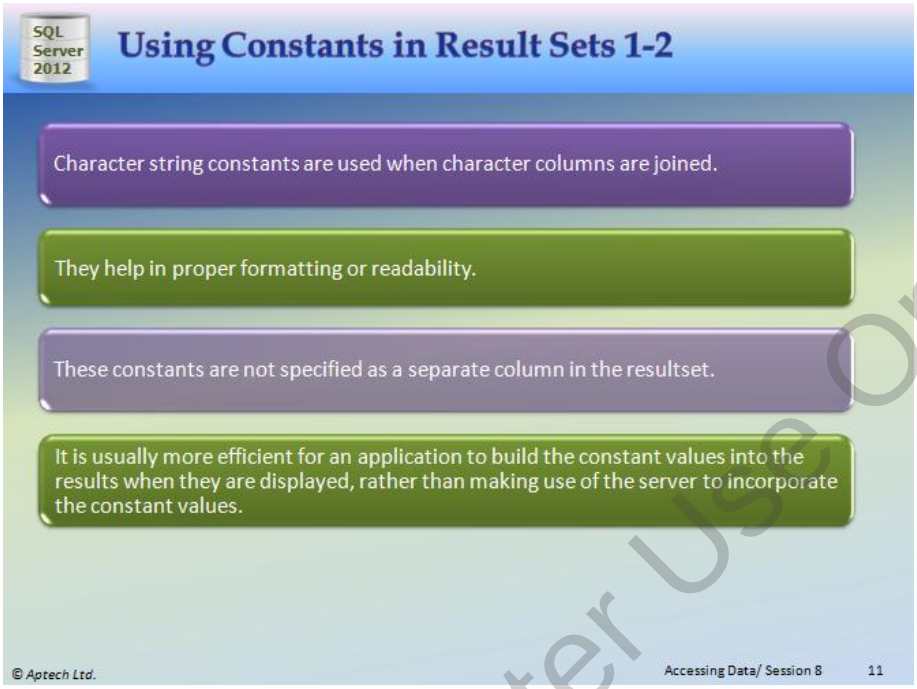
Using slides 9 and 10, explain how to display selected rows.

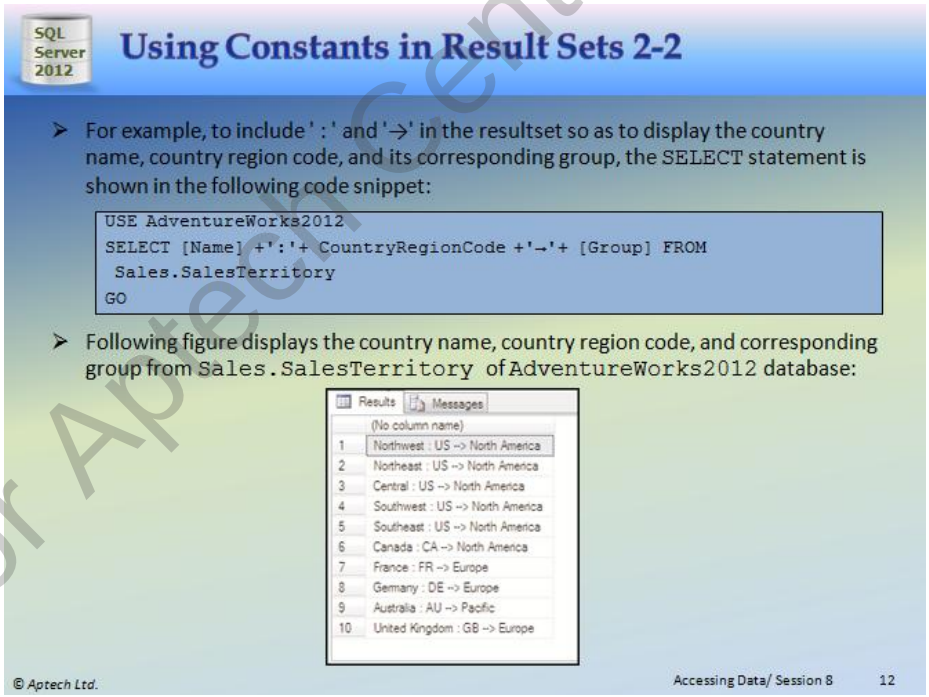
Mention that the SELECT statement displays or returns certain relevant columns that are chosen by the user or mentioned in the statement. To display specific columns, the knowledge of the relevant column names in the table is needed. Explain the syntax for selecting specific columns.

For example, to display the cost rates in various locations from Production.Location table in AdventureWorks2012 database, the SELECT statement is as shown in the code snippet. Figure shows LocationID and CostRate columns from AdventureWorks2012 database.

Constants in Result Sets

Slides 11 and 12





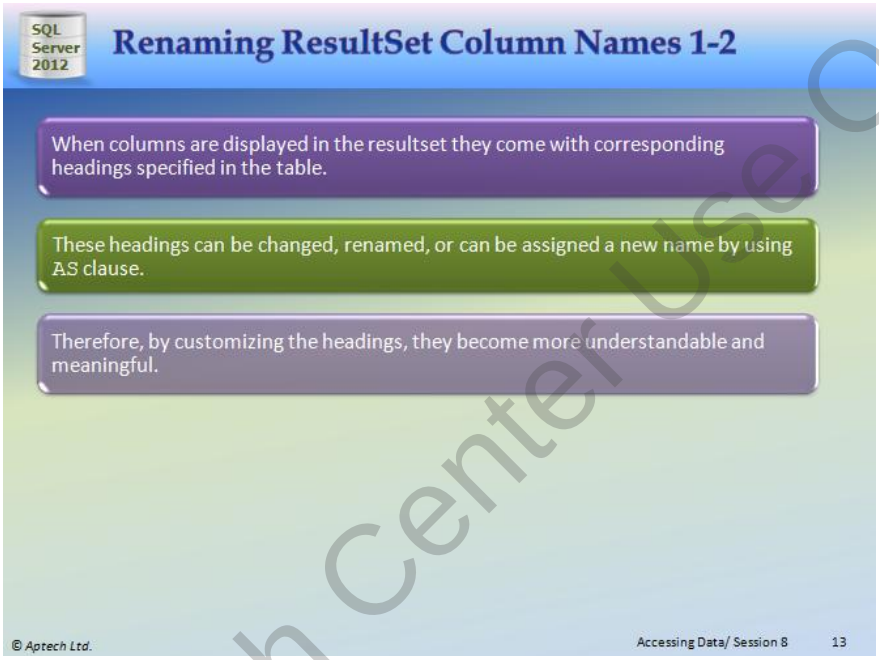
Using slides 11 and 12, explain how to use constants in result sets.
SELECT statement allows the users to specify different expressions in order to view the resultset in an ordered manner. These expressions assign different names to the columns in the resultset, compute values, and eliminate duplicate values.

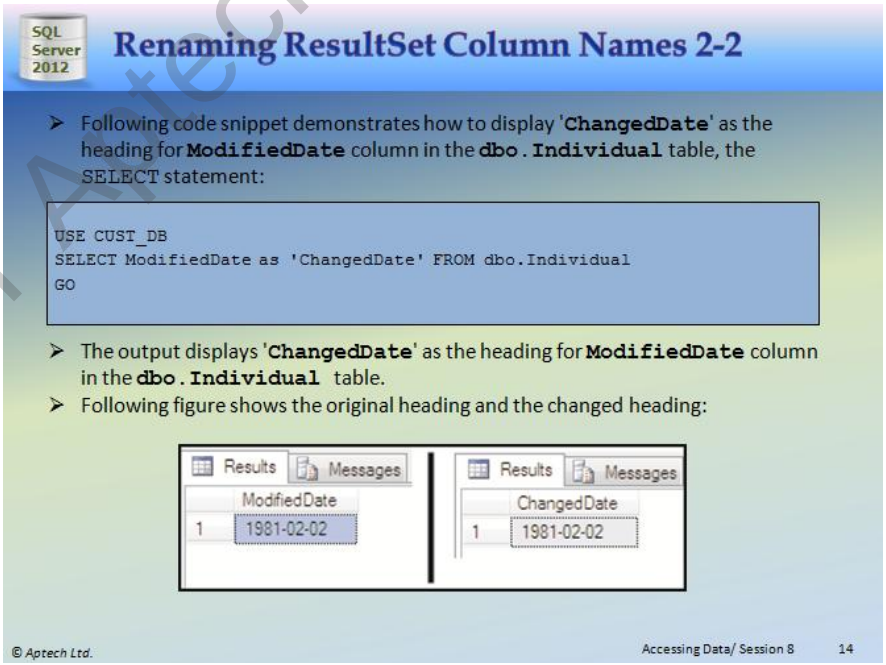
Character string constants are used when character columns are joined. They help in proper formatting or readability. These constants are not specified as a separate column in the

resultset. It is usually more efficient for an application to build the constant values into the results when they are displayed, rather than making use of the server to incorporate the constant values. For example, to include ' : ' and '→' in the resultset so as to display the country name, country region code, and its corresponding group, the SELECT statement is shown in the code snippet. Figure displays the country name, country region code, and corresponding group from Sales.SalesTerritory of AdventureWorks2012 database.

Renaming ResultSet Column Names

Slides 13 and 14





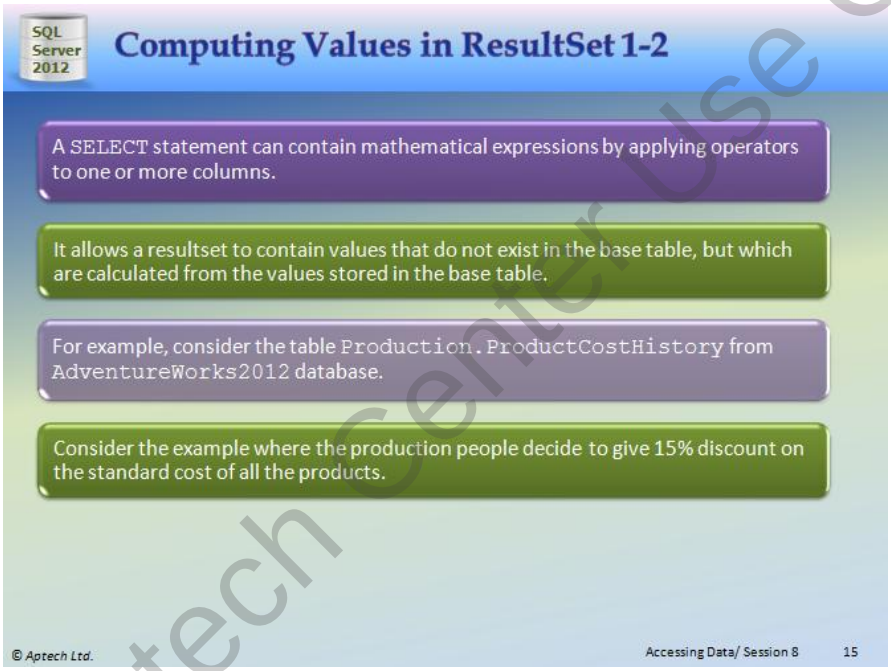
Using slides 13 and 14, explain how to rename column name.

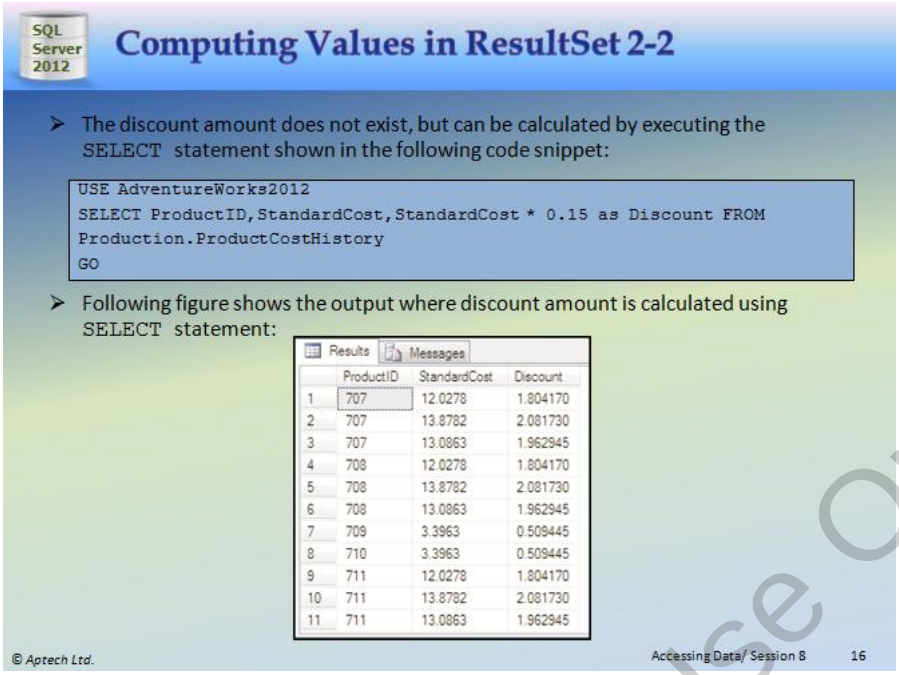
When columns are displayed in the resultset they come with corresponding headings specified in the table. These headings can be changed, renamed, or can be assigned a new name by using AS clause. Therefore, by customizing the headings, they become more understandable and meaningful.

Explain the code snippet which demonstrates how to display 'ChangedDate' as the heading for ModifiedDate column in the dbo.Individual table, the SELECT statement.
The output displays 'ChangedDate' as the heading for ModifiedDate column in the dbo.Individual table. Figure shows the original heading and the changed heading.

Computing Values in ResultSets

Slides 15 and 16





Using slides 15 and 16, explain how to compute values in result sets.

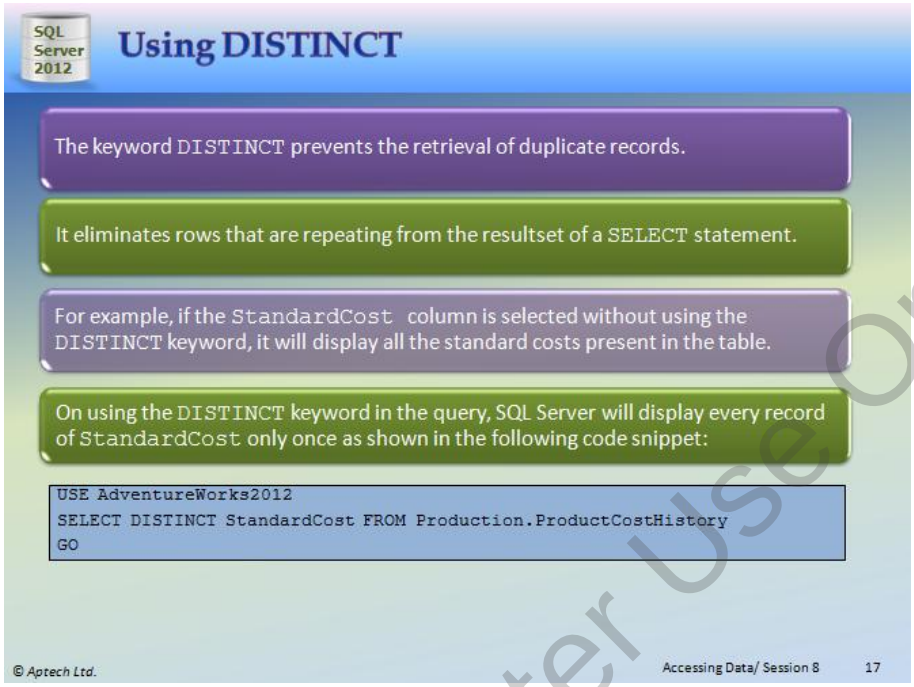
A SELECT statement can contain mathematical expressions by applying operators to one or more columns. It allows a resultset to contain values that do not exist in the base table, but which are calculated from the values stored in the base table.

Mention the table used in the FROM clause of a query is called as a base table.

For example, consider the table, Production.ProductCostHistory from AdventureWorks2012 database. Consider the example where the production people decide to give 15% discount on the standard cost of all the products. The discount amount does not exist but can be calculated by executing the SELECT statement shown in the code snippet. Figure shows the output where discount amount is calculated using SELECT statement.

Using DISTINCT

Slide 17

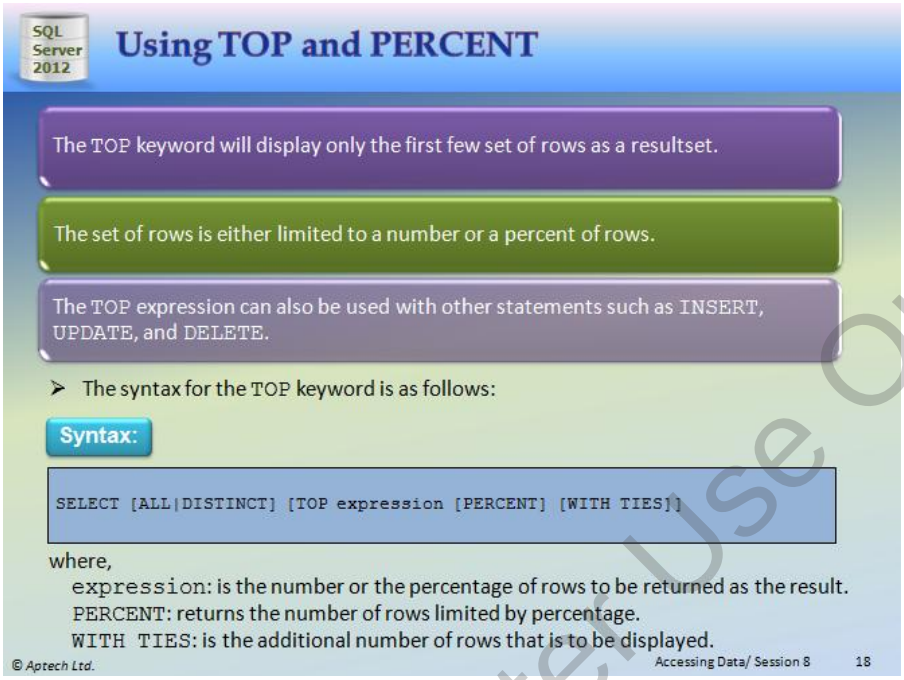


Using slide 17, explain the distinct keyword.

The keyword DISTINCT prevents the retrieval of duplicate records. It eliminates rows that are repeating from the resultset of a SELECT statement. For example, if the StandardCost column is selected without using the DISTINCT keyword, it will display all the standard costs present in the table. Using the DISTINCT keyword in the query, SQL Server will display every record of StandardCost only once as shown in the code snippet.

TOP and PERCENT

Slide 18



Using slide 18, explain the TOP and PERCENT keywords.

The TOP keyword will display only the first few set of rows as a resultset. The set of rows is either limited to a number or a percent of rows. The TOP expression can also be used with other statements such as INSERT, UPDATE, and DELETE. Explain the syntax for the TOP keyword.

The SELECT statement has various clauses associated with it. In this section, each clause is discussed in detail.

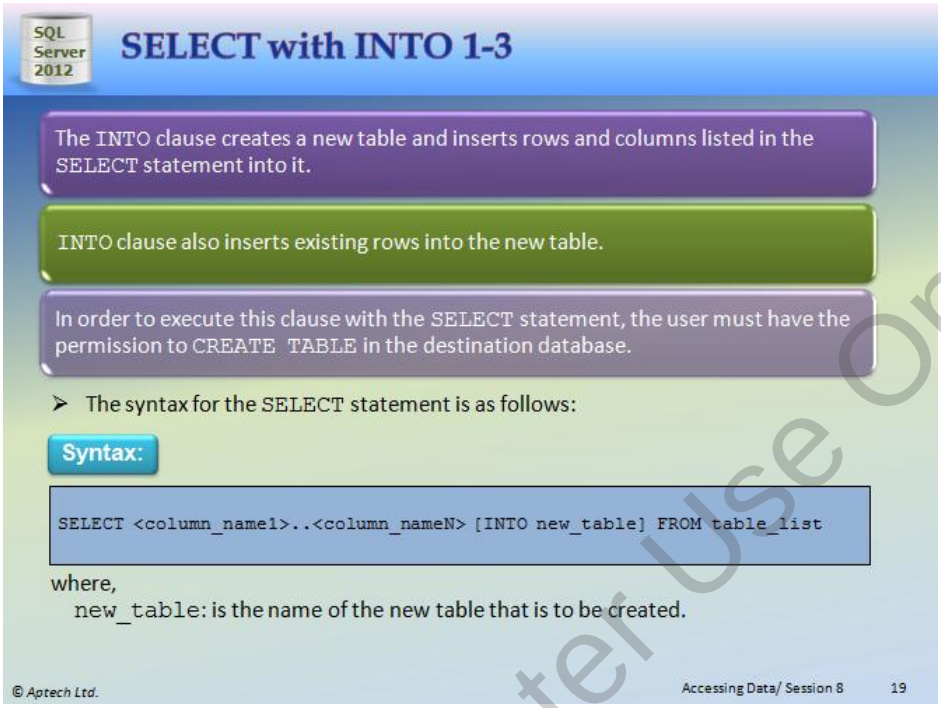
For example,

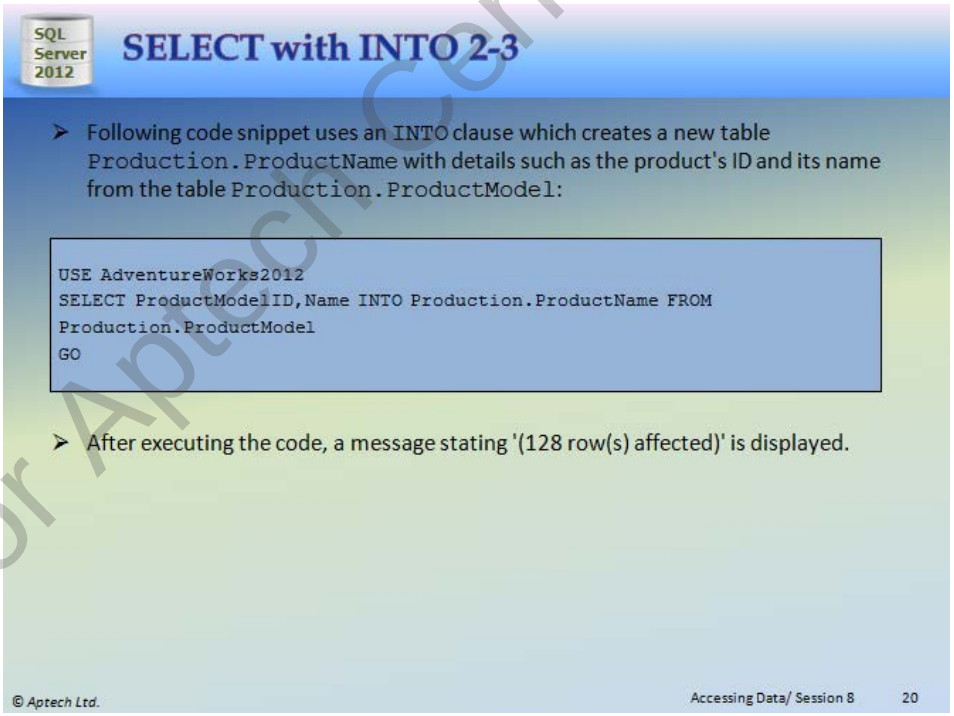
```
Select top 10 * from Production.ProductCostHistory
```

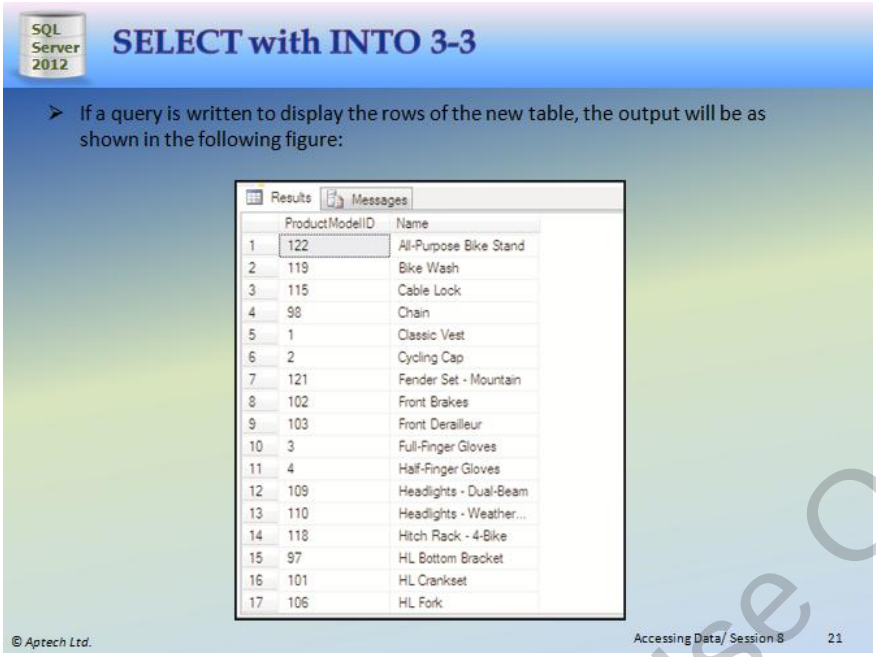
The query will give top 10 rows in the table.

SELECT with INTO

Slides 19 to 21







Using slides 19 to 21, explain the INTO keyword.

The INTO clause creates a new table and inserts rows and columns listed in the SELECT statement into it. INTO clause also inserts existing rows into the new table. In order to execute this clause with the SELECT statement, the user must have the permission to CREATE TABLE in the destination database.

Explain the code snippet which uses an INTO clause which creates a new table, Production.ProductName with details such as the product's ID and its name from the table Production.ProductModel.

After executing the code, a message stating '(128 row(s) affected)' is displayed. If a query is written to display the rows of the new table, the output will be as shown in figure.

Tips:
The SELECT INTO statement can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data.

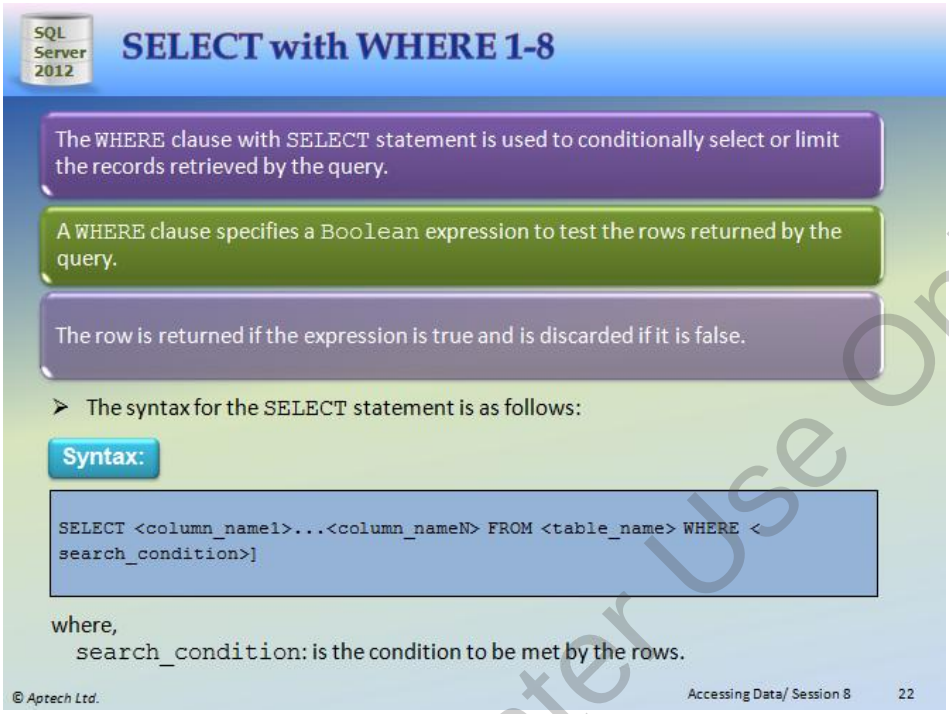
In-Class Question:

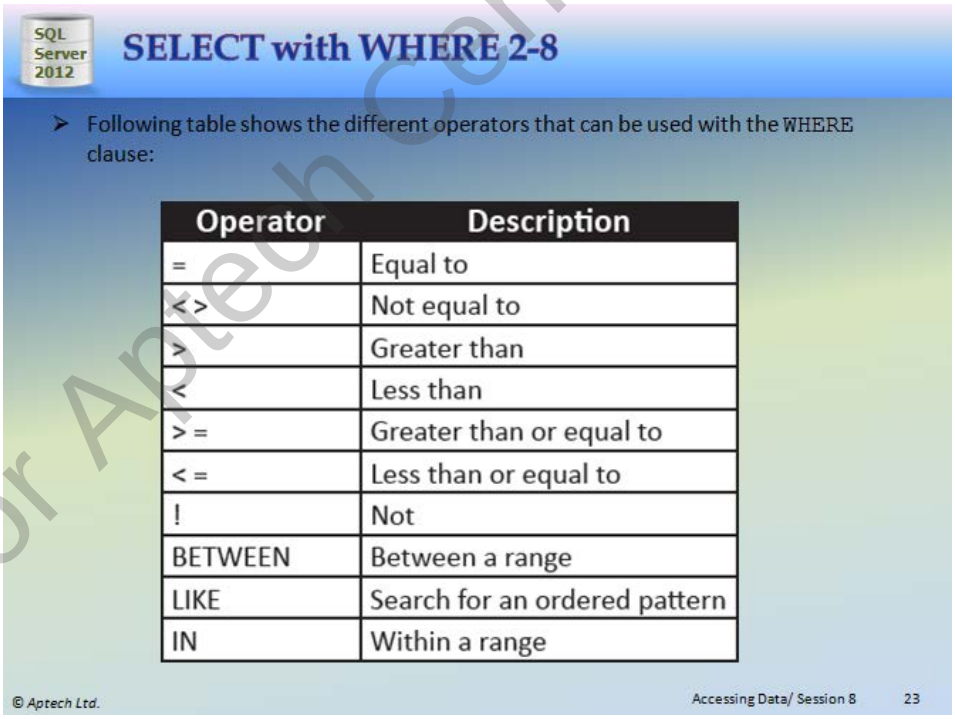
What is the use of INTO clause?

Answer:
The INTO clause creates a new table and inserts rows and columns listed in the SELECT statement into it.

SELECT with WHERE

Slides 22 to 29





SELECT with WHERE 3-8

Following code snippet demonstrates the equal to operator with WHERE clause to display data with EndDate 6/30/2007 12:00:00 AM:

```
USE AdventureWorks2012
SELECT * FROM Production.ProductCostHistory WHERE EndDate = '6/30/2007
12:00:00 AM'
GO
```

- The output SELECT with WHERE clause is shown in the following figure:

Results | Messages

	ProductID	StartDate	EndDate	StandardCost	ModifiedDate
1	707	2006-07-01 00:00:0...	2007-06-30 00:...	13.8782	2007-06-30 00:00:0
2	708	2006-07-01 00:00:0...	2007-06-30 00:...	13.8782	2007-06-30 00:00:0
3	711	2006-07-01 00:00:0...	2007-06-30 00:...	13.8782	2007-06-30 00:00:0
4	712	2006-07-01 00:00:0...	2007-06-30 00:...	5.2297	2007-06-30 00:00:0
5	713	2006-07-01 00:00:0...	2007-06-30 00:...	29.0807	2007-06-30 00:00:0
6	714	2006-07-01 00:00:0...	2007-06-30 00:...	29.0807	2007-06-30 00:00:0
7	715	2006-07-01 00:00:0...	2007-06-30 00:...	29.0807	2007-06-30 00:00:0
8	716	2006-07-01 00:00:0...	2007-06-30 00:...	29.0807	2007-06-30 00:00:0
9	717	2006-07-01 00:00:0...	2007-06-30 00:...	722.2568	2007-06-30 00:00:0

SELECT with WHERE 4-8

- All queries in SQL use single quotes to enclose the text values.
- For example, consider the following query, which retrieves all the records from Person.Address table having Bothell as city.
- Following code snippet demonstrates the equal to operator with WHERE clause to display data with address having Bothell city.

```
USE AdventureWorks2012
SELECT * FROM Person.Address WHERE city LIKE 'Bothell'
GO
```

- The output of the query is shown in the following figure:

Results | Spatial results | Messages

	AddressID	AddressLine1	AddressLine2	City	StateProvinceID	PostalCod
1	5	1226 Shoe St.	NULL	Bothell	79	98011
2	11	1318 Lasalle Street	NULL	Bothell	79	98011
3	6	1399 Firestone Drive	NULL	Bothell	79	98011
4	18	1873 Lion Circle	NULL	Bothell	79	98011
5	40	1902 Santa Cruz	NULL	Bothell	79	98011
6	1	1970 Napa Ct.	NULL	Bothell	79	98011
7	10	250 Race Court	NULL	Bothell	79	98011
8	868	25111 228th St Sw	NULL	Bothell	79	98011
9	19	3148 Rose Street	NULL	Bothell	79	98011

SELECT with WHERE 5-8

- Numeric values are not enclosed within any quotes as shown in the following code snippet:

```
USE AdventureWorks2012
SELECT * FROM HumanResources.Department WHERE DepartmentID < 10
GO
```

- The query displays all those records where the value in `DepartmentID` is less than 10.
- The output of the query is shown in the following figure:

Results | Messages

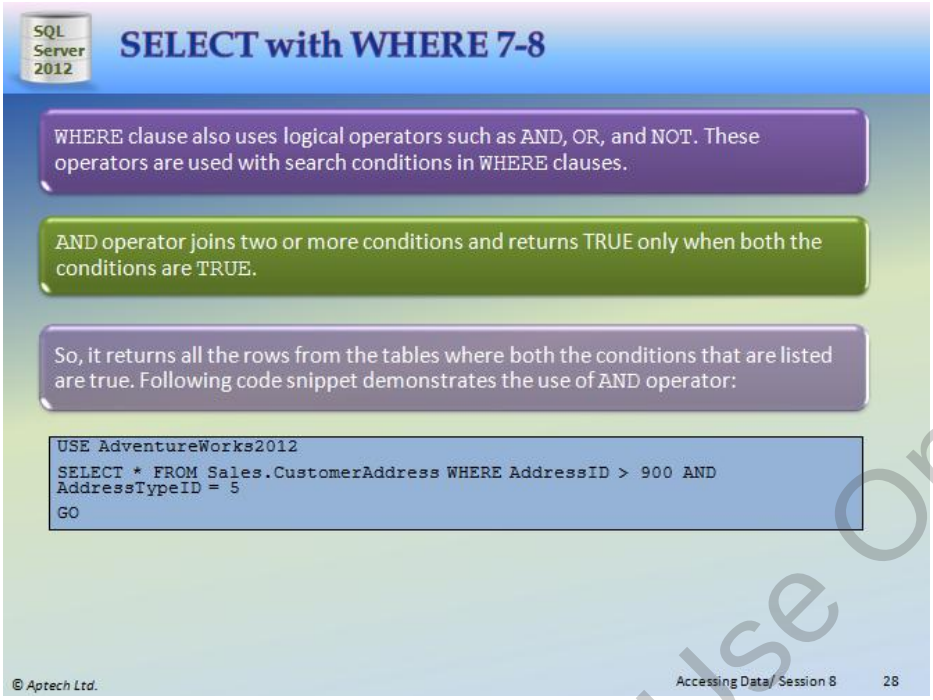
	DepartmentID	Name	GroupName	Modifie
1	1	Engineering	Research and Development	2002-(
2	2	Tool Design	Research and Development	2002-(
3	3	Sales	Sales and Marketing	2002-(
4	4	Marketing	Sales and Marketing	2002-(
5	5	Purchasing	Inventory Management	2002-(
6	6	Research and Development	Research and Development	2002-(
7	7	Production	Manufacturing	2002-(
8	8	Production Control	Manufacturing	2002-(

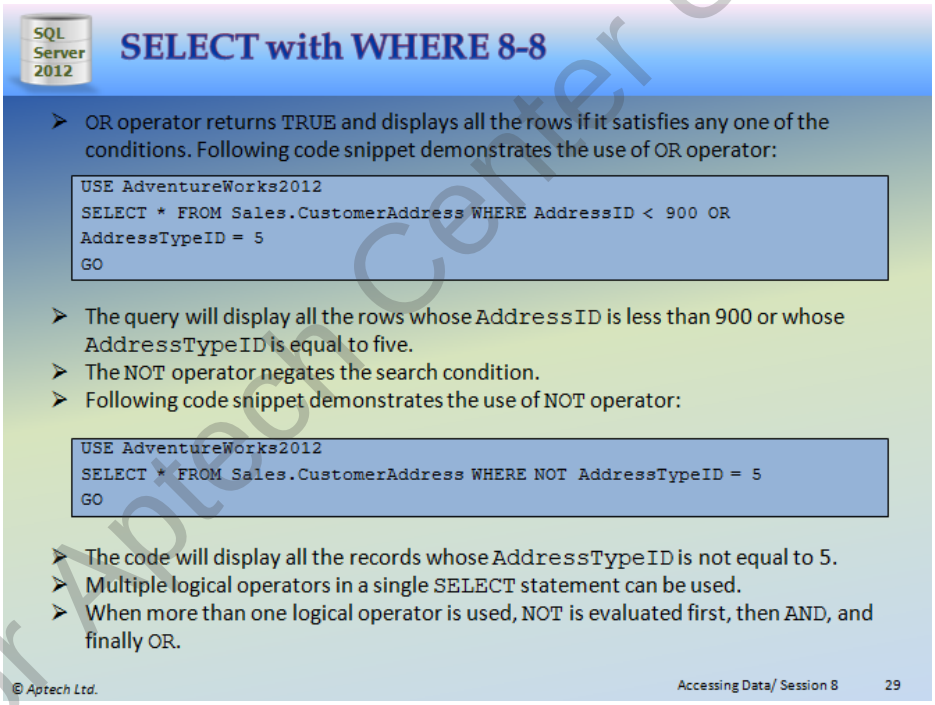
SELECT with WHERE 6-8

- `WHERE` clause can also be used with wildcard characters as shown in the following table:

Wildcard	Description	Example
_	It will display a single character	`SELECT * FROM Person.Contact WHERE Suffix LIKE 'Jr_'`
%	It will display a string of any length	`SELECT * FROM Person.Contact WHERE LastName LIKE 'B%'`
[ ]	It will display a single character within the range enclosed in the brackets	`SELECT * FROM Sales.CurrencyRate WHERE ToCurrencyCode LIKE 'C[AN][DY]'`
[^]	It will display any single character not within the range enclosed in the brackets	`SELECT * FROM Sales.CurrencyRate WHERE ToCurrencyCode LIKE 'A[^R][^S]'`

- All wildcard characters are used along with `LIKE` keyword to make the query accurate and specific.





Using slides 22 to 29, explain the WHERE clause.

Mention WHERE clause with SELECT statement is used to conditionally select or limit the records retrieved by the query. A WHERE clause specifies a Boolean expression to test the rows returned by the query. The row is returned if the expression is true and is discarded if it is false.

Table on slide 23 shows the different operators that can be used with the WHERE clause. Explain the code snippet which demonstrates the Equal to operator with WHERE clause to display data with EndDate 6/30/2007 12:00:00 AM.

Explain the code snippet that returns all records from the table, Production.ProductCostHistory which has the end date as '6/30/2007 12:00:00 AM'. The output of the SELECT query with WHERE clause is shown in figure.

All queries in SQL use single quotes to enclose the text values. For example, consider the query, which retrieves all the records from Person.Address table having 'Bothell' as city.

Explain the code snippet on slide 25 which demonstrates the Equal to operator with WHERE clause to display data with address having 'Bothell' city.
The output of the query is shown in figure.

Numeric values are not enclosed within any quotes as shown in code snippet.
The query in code snippet on slide 26 displays all those records where the value in DepartmentID is less than 10. The output of the query is shown in figure.

The WHERE clause can also be used with wildcard characters as shown in table. All wildcard characters are used along with LIKE keyword to make the query accurate and specific.
For example,

```
Select * from Person.Address where city like 'B%'
```

This code snippet will also display all the rows where the city starts with 'B'.
WHERE clause also uses logical operators such as AND, OR, and NOT. These operators are used with search conditions in WHERE clauses. AND operator joins two or more conditions and returns TRUE only when both the conditions are TRUE. So, it returns all the rows from the tables where both the conditions that are listed, are true.

Explain the code snippet which demonstrates AND operator.
OR operator returns TRUE and displays all the rows if it satisfies any one of the conditions.
Explain the code snippet which demonstrates OR operator.

Explain the query in the code snippet that displays all the rows whose AddressID is less than 900 or whose AddressTypeID is equal to five. The NOT operator negates the search condition. Explain the code snippet demonstrates NOT operator.

Explain the code snippet that displays all the records whose AddressTypeID is not equal to 5. Multiple logical operators in a single SELECT statement can be used. When more than one logical operator is used, NOT is evaluated first, then AND, and finally OR.

Other than the operators mentioned here, the sub-queries can also be used in WHERE clause. Also the CASE statement can be used.

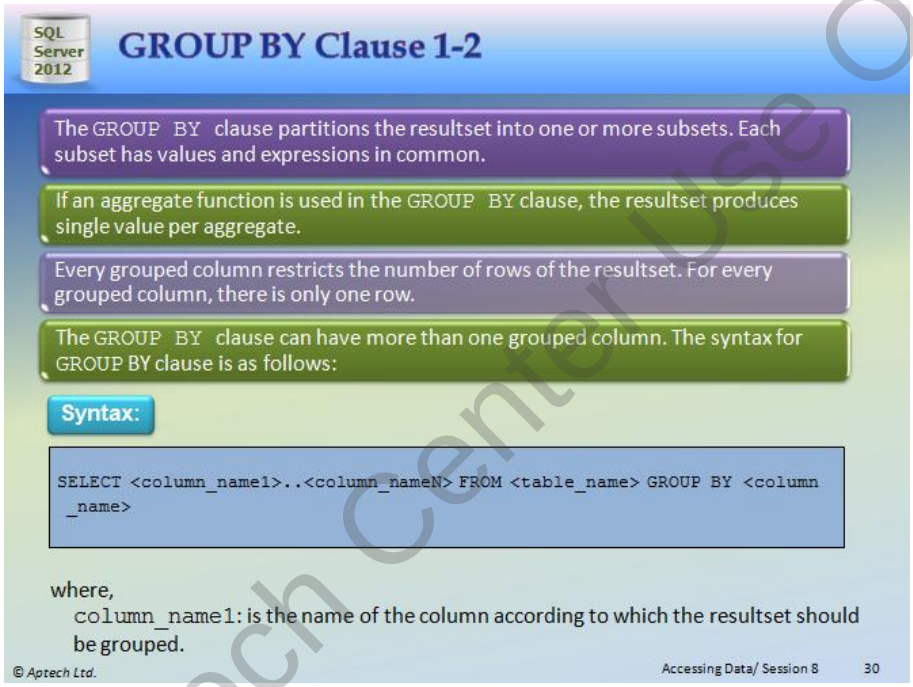
In-Class Question:

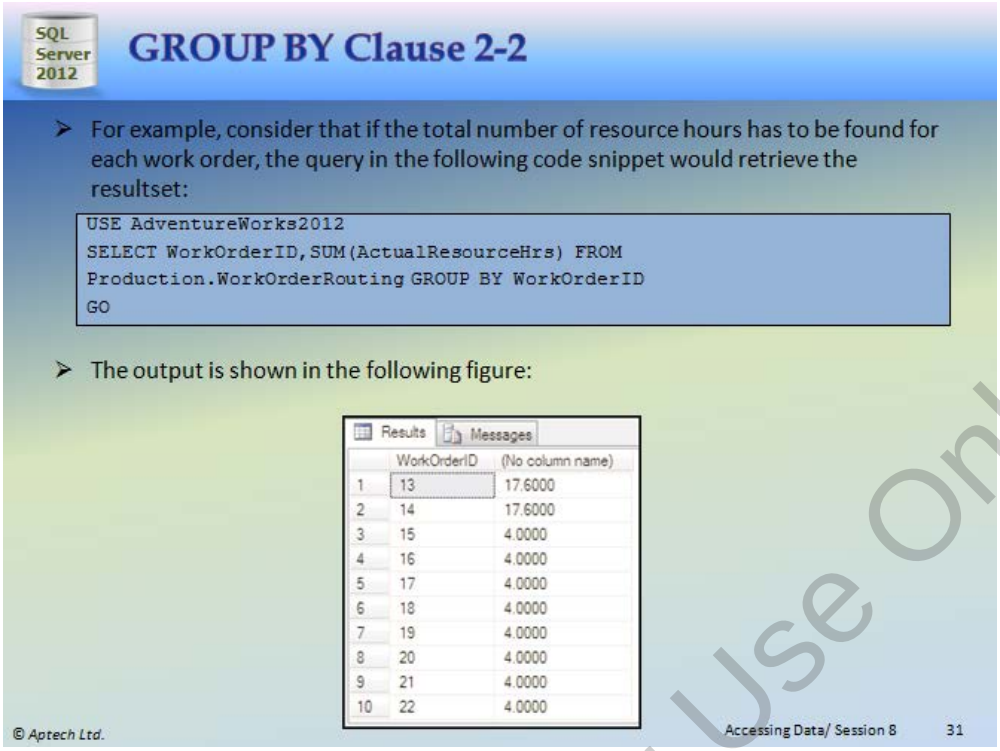
What is the use of WHERE clause?

Answer:
The WHERE clause with SELECT statement is used to conditionally select or limit the records retrieved by the query. It is use for filtering the records in table.

GROUP BY Clause

Slides 30 and 31





Using slides 30 and 31, explain the GROUP BY clause.

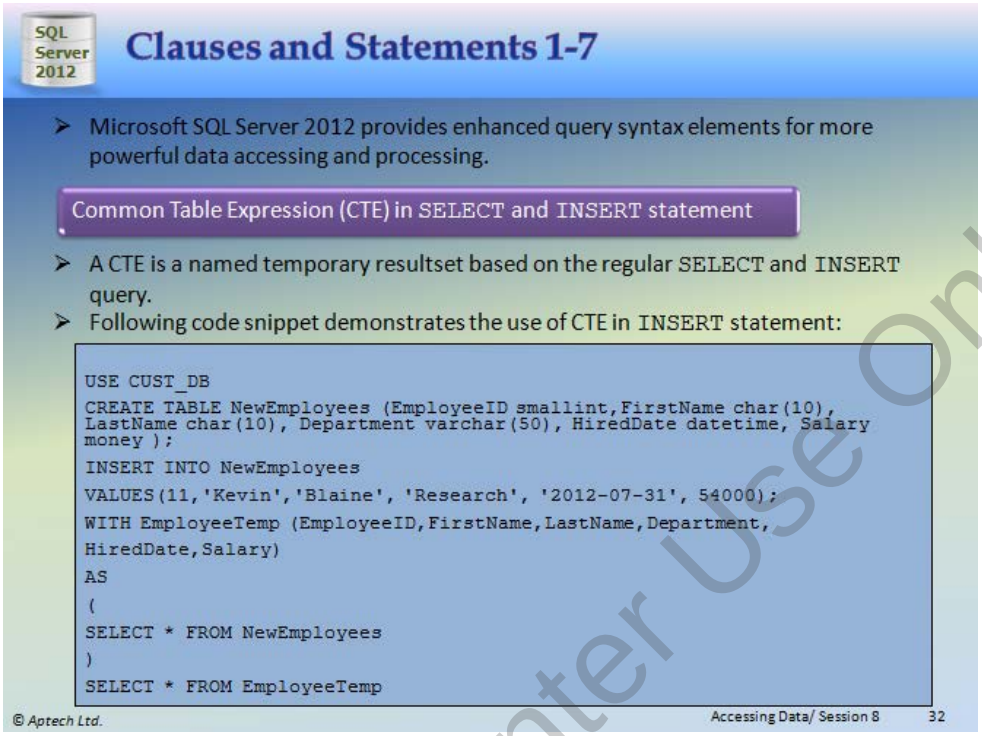
The GROUP BY clause partitions the resultset into one or more subsets. Each subset has values and expressions in common. If an aggregate function is used in the GROUP BY clause, the resultset produces single value per aggregate.

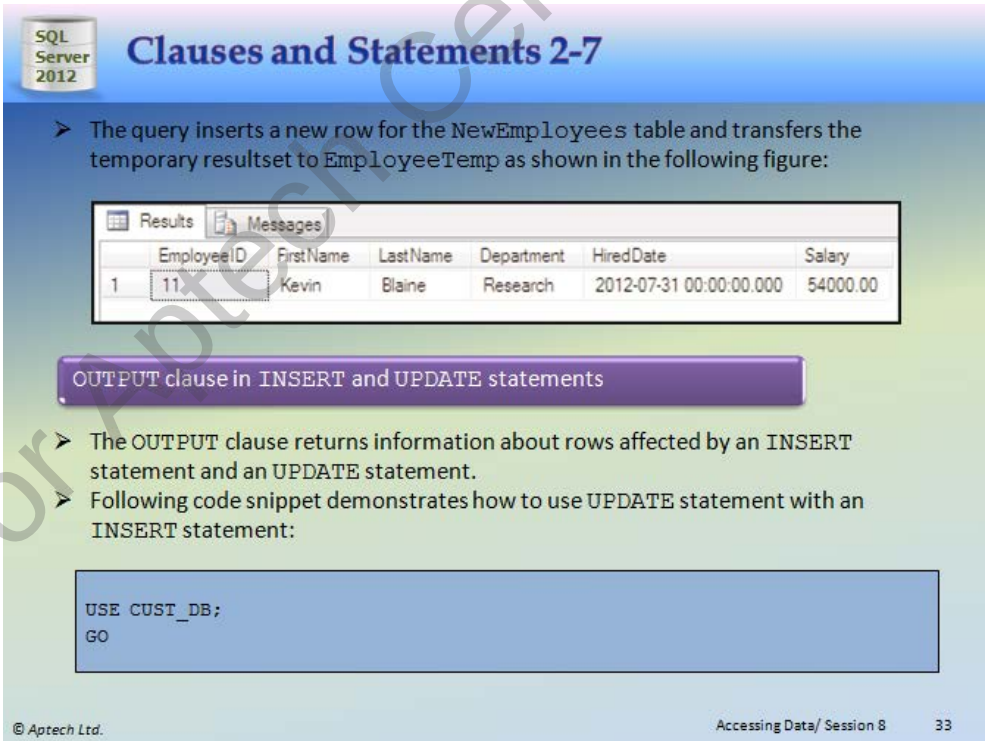
The GROUP BY keyword is followed by a list of columns, known as grouped columns. Every grouped column restricts the number of rows of the resultset. For every grouped column, there is only one row. The GROUP BY clause can have more than one grouped column. Explain the syntax for GROUP BY clause.

For example, consider that if the total number of resource hours has to be found for each work order, the query in code snippet would retrieve the resultset. The output is shown in figure.

Clauses and Statements

Slides 32 to 38





Clauses and Statements 3-7

```
CREATE TABLE dbo.table_3
(
id INT,
employee VARCHAR(32)
)
go
INSERT INTO dbo.table_3 VALUES
(1, 'Matt')
,(2, 'Joseph')
,(3, 'Renny')
,(4, 'Daisy');
GO
DECLARE @updatedTable TABLE
(
id INT, olddata_employee VARCHAR(32), newdata_employee VARCHAR(32)
);
UPDATE dbo.table_3
Set employee= UPPER(employee)
```

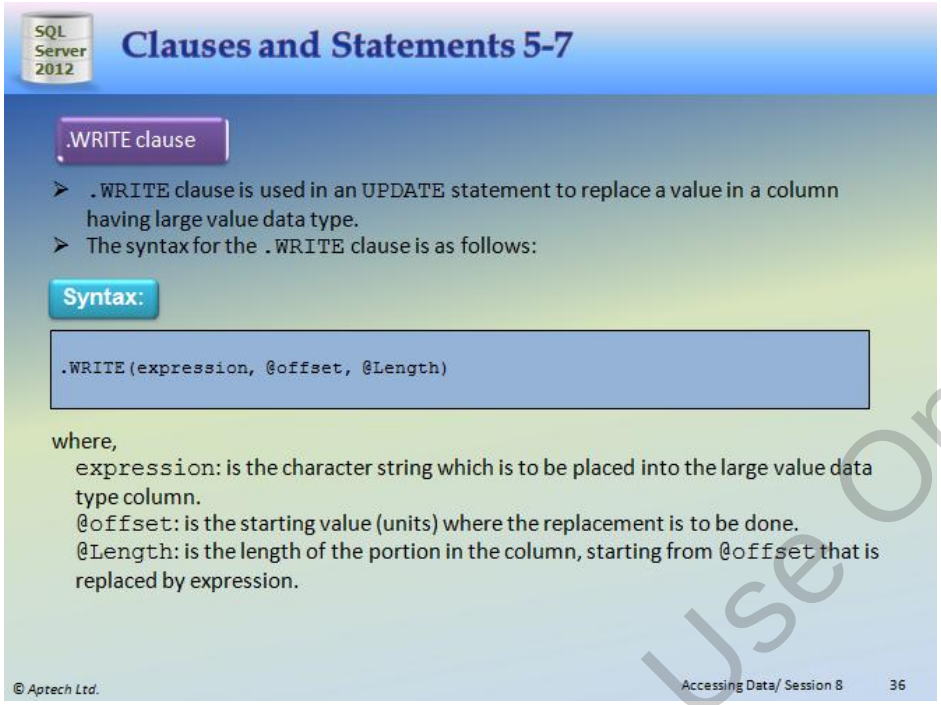

Clauses and Statements 4-7

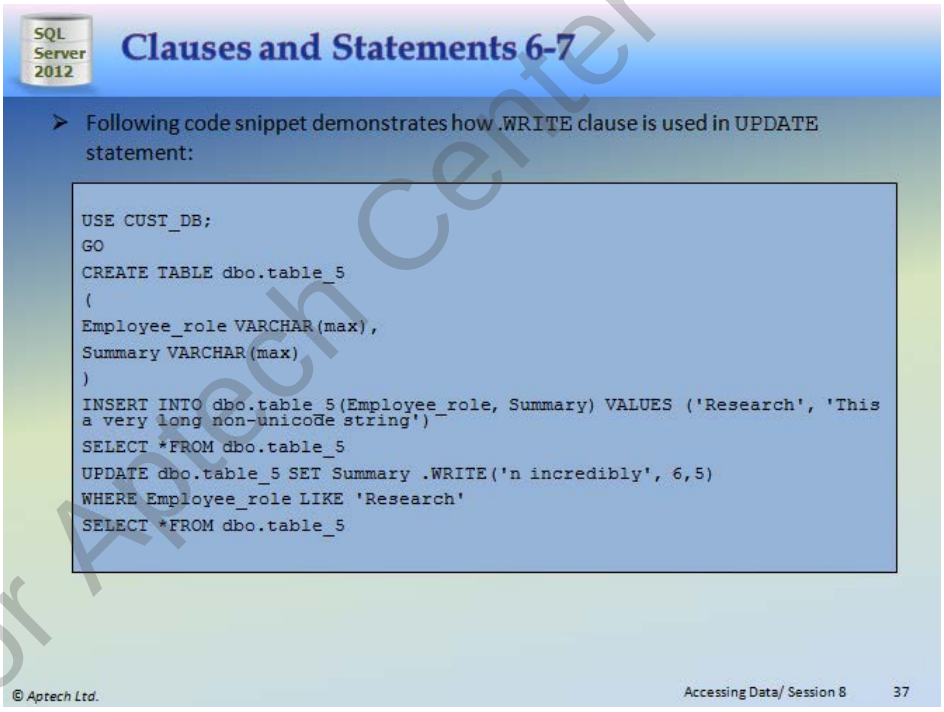
```
OUTPUT
inserted.id,
deleted.employee,
inserted.employee
INTO @updatedTable
SELECT * FROM @updatedTable
```

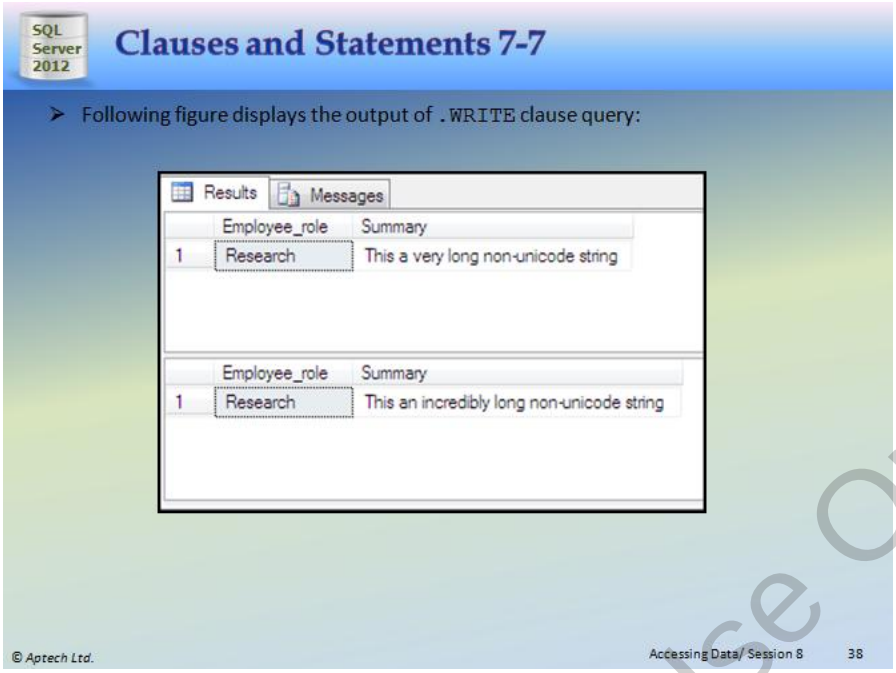
- The output where rows are affected by an `INSERT` statement and an `UPDATE` statement is shown in the following figure:

Results | Messages

	id	olddata_employee	newdata_employee
1	1	Matt	MATT
2	2	Joseph	JOSEPH
3	3	Renny	RENNY
4	4	Daisy	DAISY







Using slides 32 to 38, explain the clauses and statements.

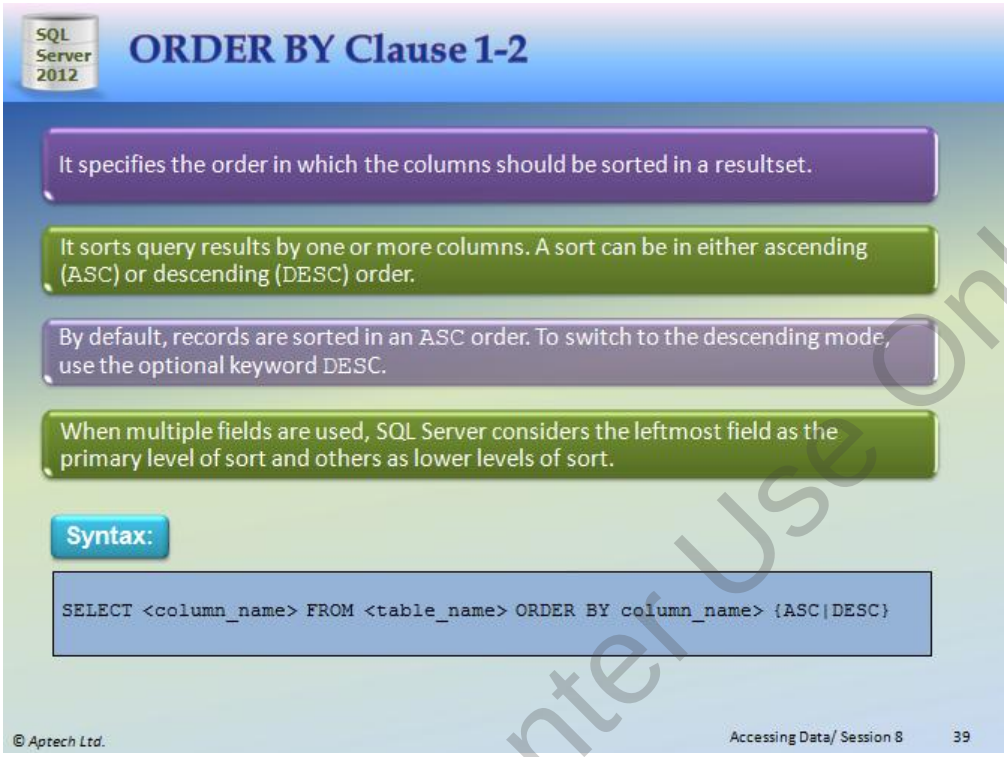
Microsoft SQL Server 2012 provides enhanced query syntax elements for more powerful data accessing and processing.

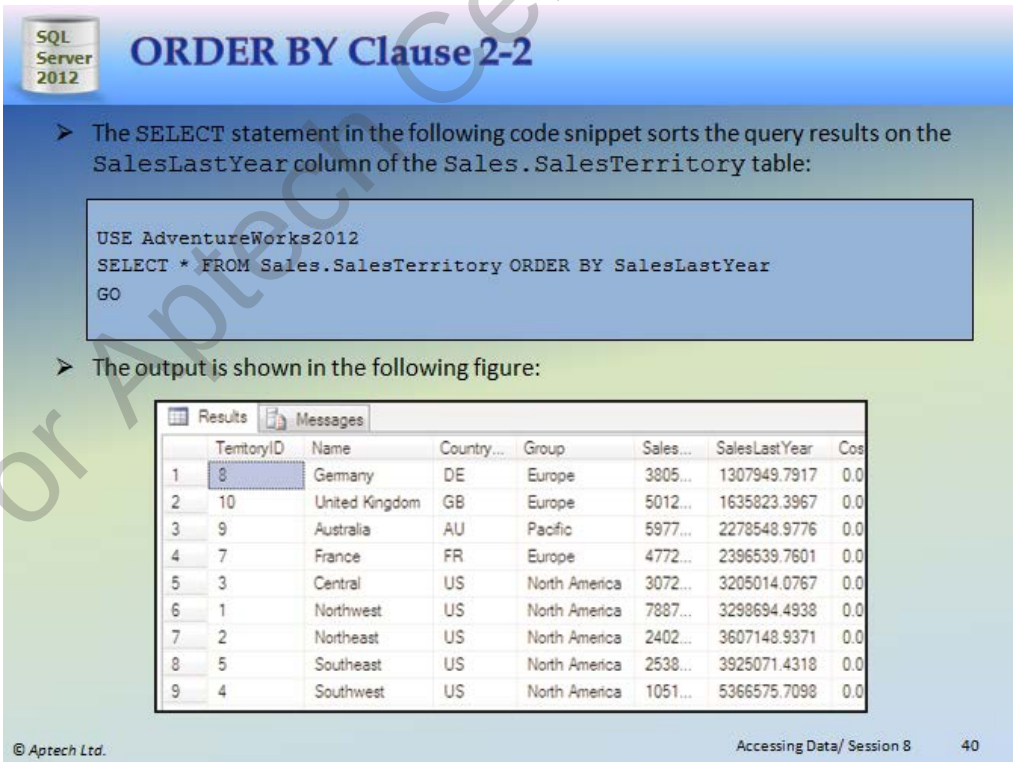
- **Common Table Expression (CTE) in SELECT and INSERT statement:**
  A CTE is a named temporary resultset based on the regular SELECT and INSERT query. Explain the code snippet which demonstrates the use of CTE in INSERT statement.
- **OUTPUT clause in INSERT and UPDATE statements:**
  The OUTPUT clause returns information about rows affected by an INSERT statement and an UPDATE statement. Explain the code snippet that demonstrates how to use UPDATE statement with an OUTPUT clause.
  After executing the code snippet, the output where rows are affected by an INSERT statement and an UPDATE statement is shown in figure.
- **.WRITE clause:**
  .WRITE clause is used in an UPDATE statement to replace a value in a column having large value data type. Explain the syntax for the .WRITE clause.

Explain the code snippet which demonstrates how .WRITE clause is used in UPDATE statement. Figure displays the output of .WRITE clause query.

ORDER BY Clause

Slides 39 and 40





Using slides 39 and 40, explain the ORDER BY clause.

It specifies the order in which the columns should be sorted in a resultset. It sorts query results by one or more columns. A sort can be in either ascending (ASC) or descending (DESC) order. By default, records are sorted in an ASC order. To switch to the descending

mode, use the optional keyword, DESC. When multiple fields are used, SQL Server considers the leftmost field as the primary level of sort and others as lower levels of sort.
The SELECT statement in code snippet sorts the query results on the SalesLastYear column of the Sales.SalesTerritory table. The output is shown in figure.

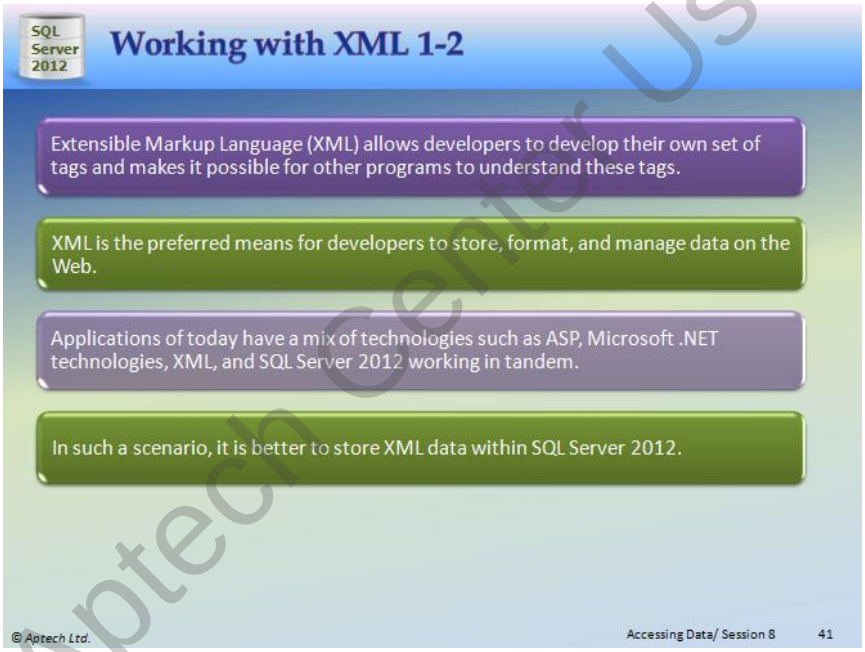
In-Class Question:

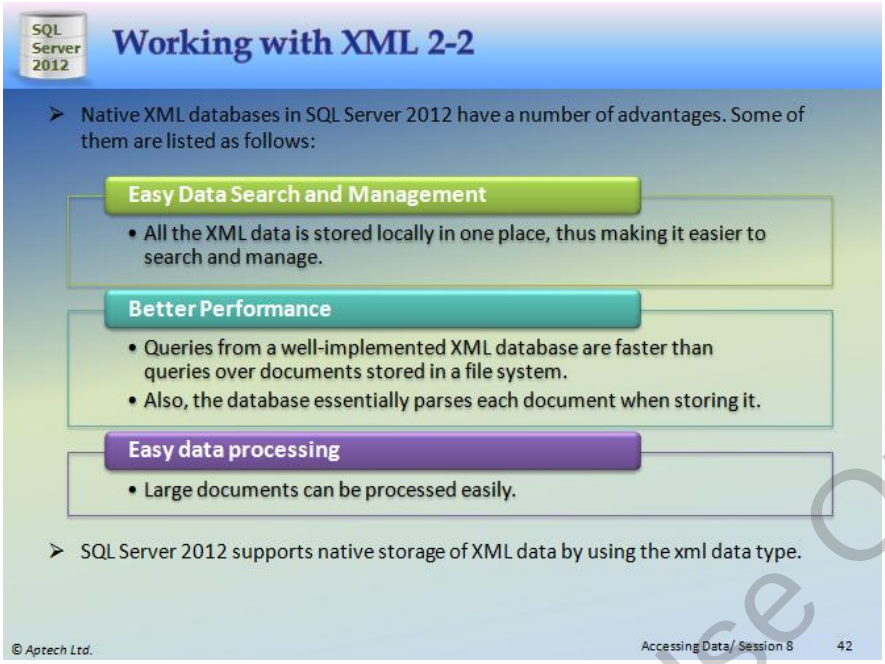
Which clause is used to view the result set in sorted order?

Answer:
The ORDER BY clause is used to view the result set in sorted order.

Working with XML

Slides 41 and 42





Using slides 41 and 42, explain the working with XML.

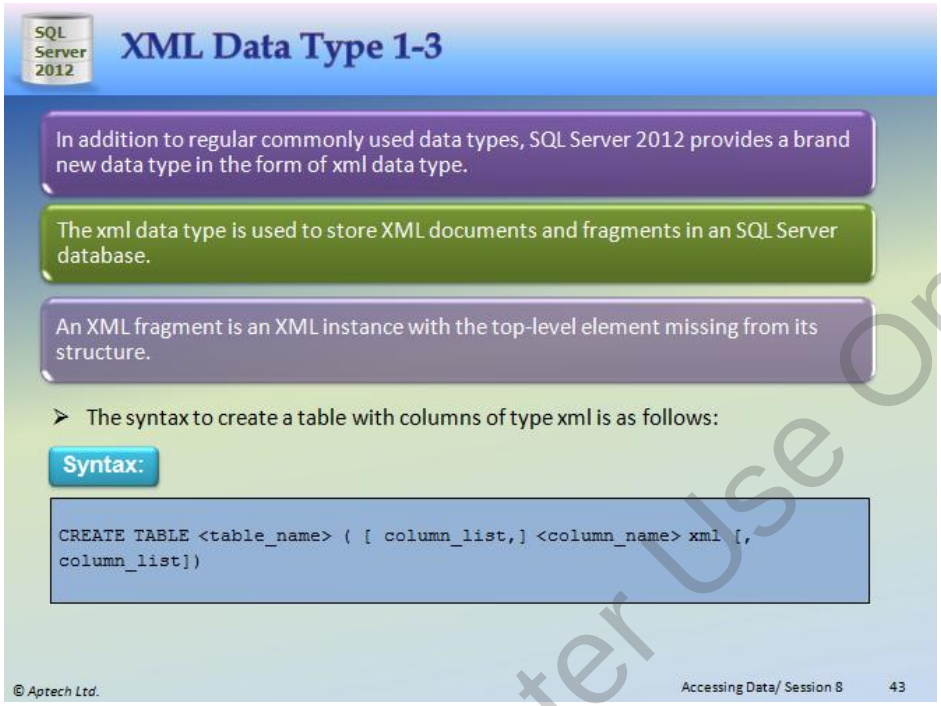
Extensible Markup Language (XML) allows developers to develop their own set of tags and makes it possible for other programs to understand these tags. XML is the preferred means for developers to store, format, and manage data on the Web. Applications of today have a mix of technologies such as ASP, Microsoft .NET technologies, XML, and SQL Server 2012 working in tandem. In such a scenario, it is better to store XML data within SQL Server 2012. Native XML databases in SQL Server 2012 have a number of advantages. Some of them are listed as follows:

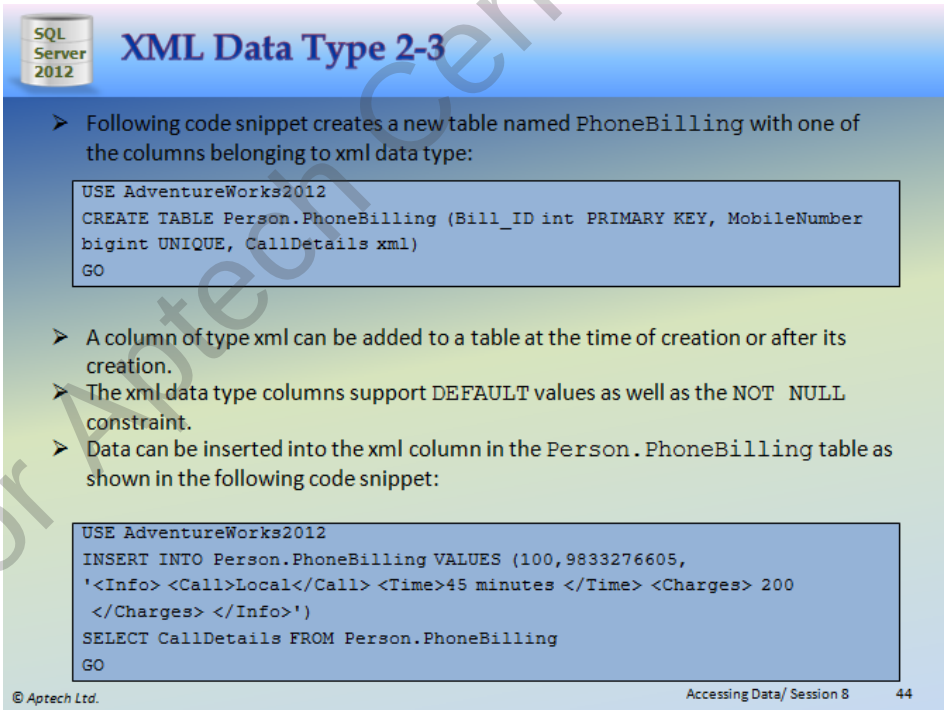
- **Easy Data Search and Management** - All the XML data is stored locally in one place, thus making it easier to search and manage.
- **Better Performance** - Queries from a well-implemented XML database are faster than queries over documents stored in a file system. Also, the database essentially parses each document when storing it.
- **Easy data processing** - Large documents can be processed easily.

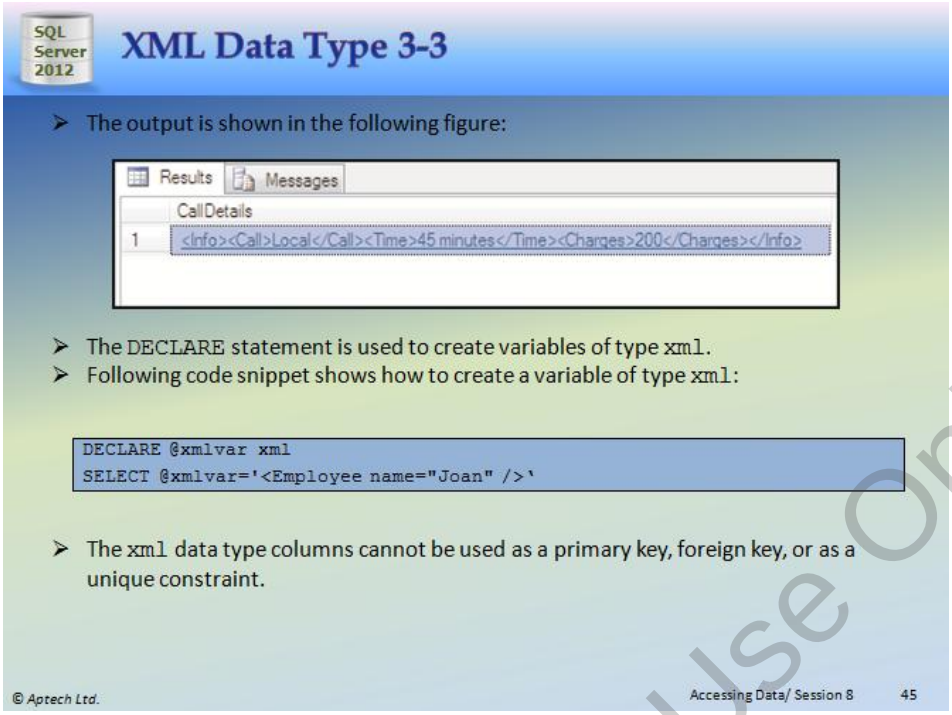
SQL Server 2012 supports native storage of XML data by using the xml data type. The following sections explore the xml data type, working with typed and untyped XML, storing them in SQL Server 2012, and using XQuery to retrieve data from columns of xml data type.

XML Data Type

Slides 43 to 45







Using slides 43 to 45, explain the xml data type.

In addition to regular commonly used data types, SQL Server 2012 provides a brand new data type in the form of xml data type.

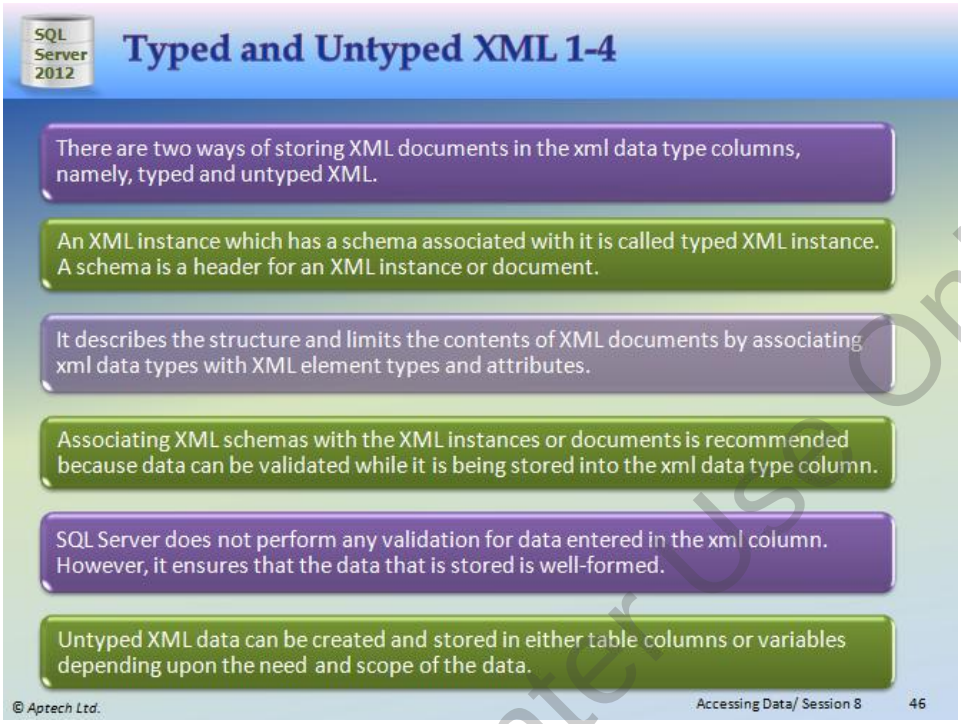
The xml data type is used to store XML documents and fragments in an SQL Server database. An XML fragment is an XML instance with the top-level element missing from its structure.

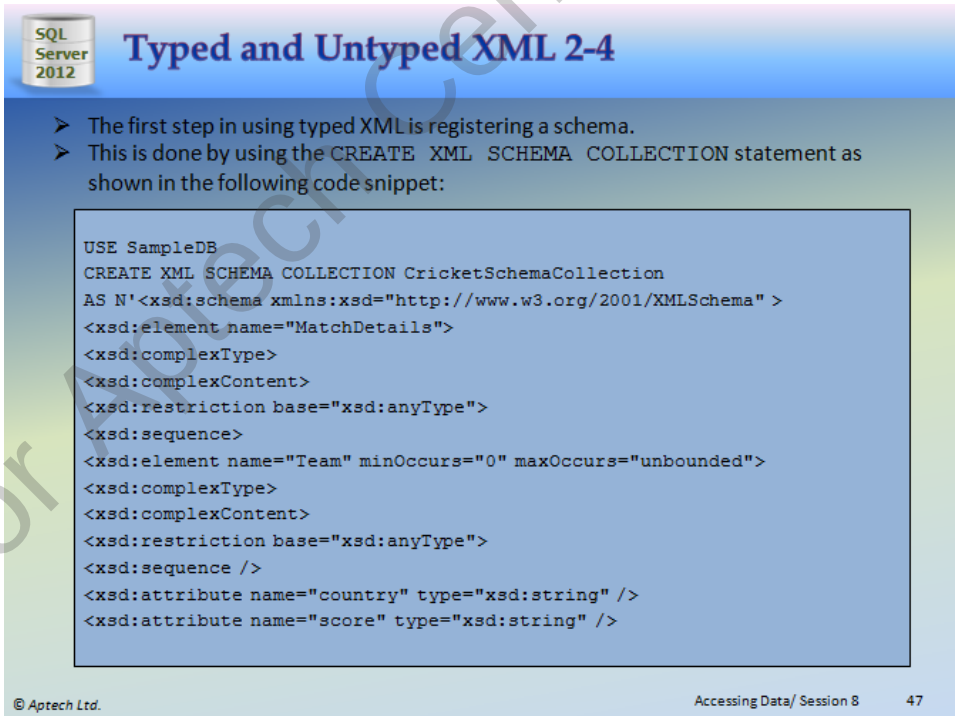
Explain the syntax to create a table with columns of type xml.
Explain the code snippet that creates a new table named, PhoneBilling with one of the columns belonging to xml data type.

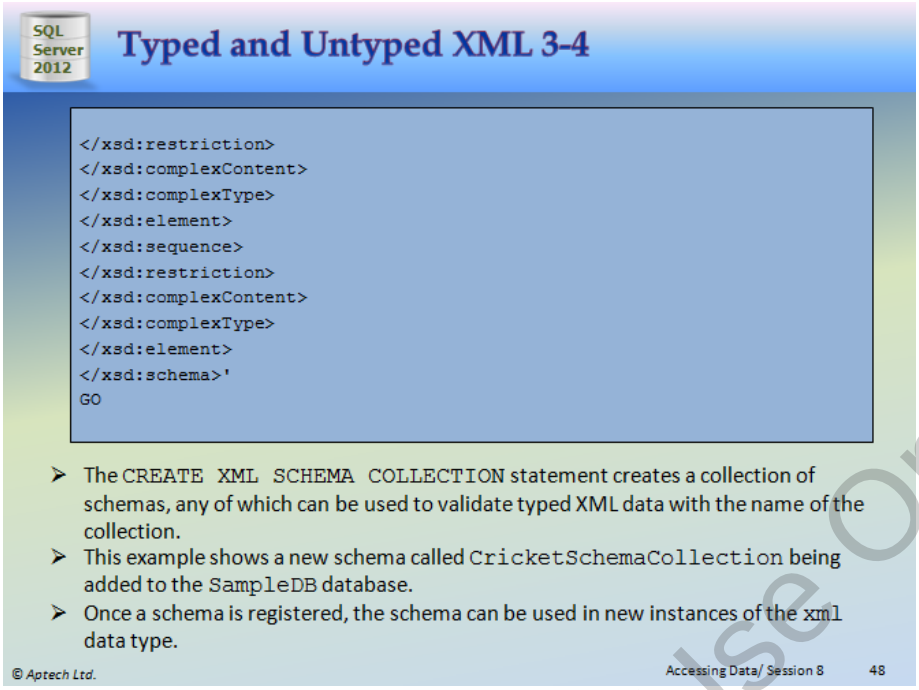
A column of type xml can be added to a table at the time of creation or after its creation.
The xml data type columns support DEFAULT values as well as the NOT NULL constraint.
Data can be inserted into the xml column in the Person.PhoneBilling table as shown in code snippet. The output is shown in figure.
The DECLARE statement is used to create variables of type xml. Explain the code snippet that shows how to create a variable of type xml.

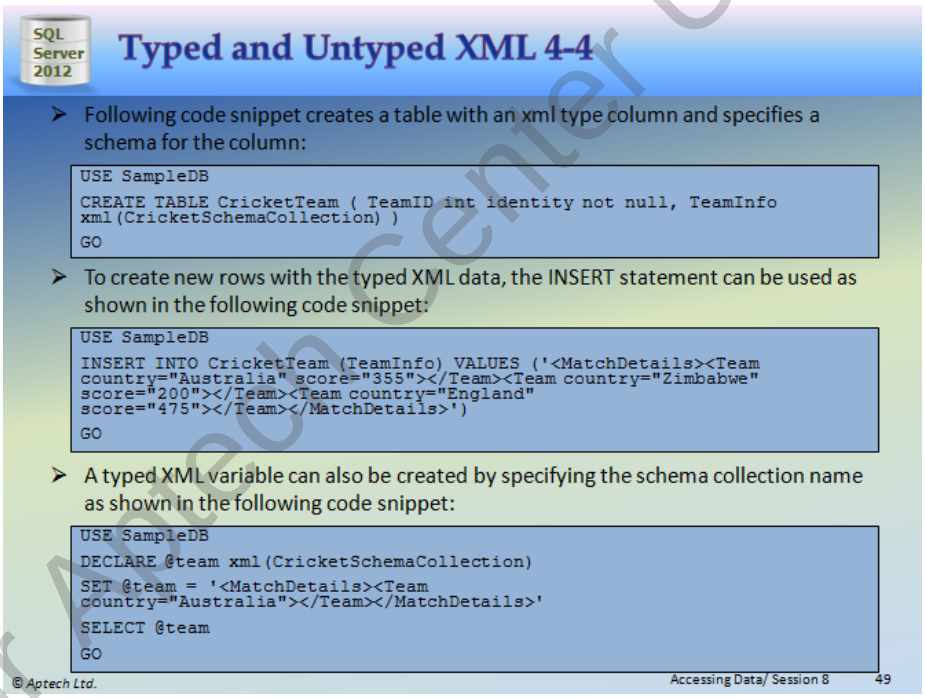
Typed and Untyped XML

Slides 46 to 49









Using slides 46 to 49, explain the typed and untyped XML.

There are two ways of storing XML documents in the xml data type columns namely, typed and untyped XML. An XML instance which has a schema associated with it is called typed XML instance. A schema is a header for an XML instance or document. It describes the structure and limits the contents of XML documents by associating xml data types with XML element types and attributes. Associating XML schemas with the XML instances or documents is recommended because data can be validated while it is being stored into the xml data type column.

SQL Server does not perform any validation for data entered in the xml column. However, it ensures that the data that is stored is well-formed. Untyped XML data can be created and stored in either table columns or variables depending upon the need and scope of the data.

The first step in using typed XML is registering a schema. This is done by using the CREATE XML SCHEMA COLLECTION statement as shown in code snippet.

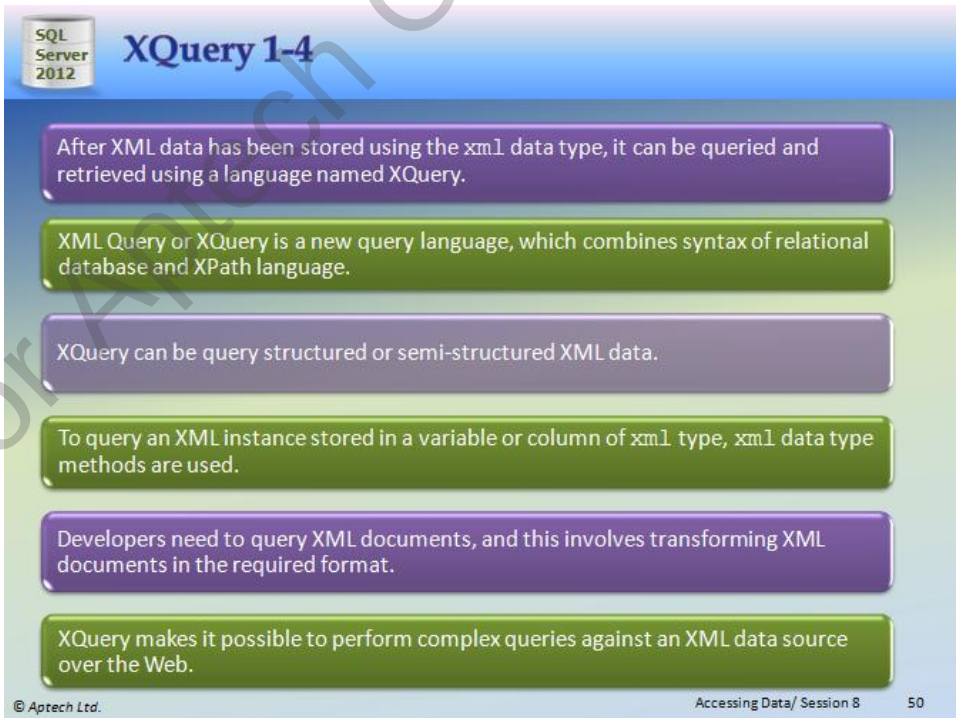
The CREATE XML SCHEMA COLLECTION statement creates a collection of schemas, any of which can be used to validate typed XML data with the name of the collection. This example shows a new schema called CricketSchemaCollection being added to the SampleDB database. Once a schema is registered, the schema can be used in new instances of the xml data type. Code snippet creates a table with an xml type column and specifies a schema for the column.

To create new rows with the typed XML data, the INSERT statement can be used as shown in code snippet using slide 49.

A typed XML variable can also be created by specifying the schema collection name. For instance, in code snippet, a variable team is declared as a typed XML variable with schema name as CricketSchemaCollection. The SET statement is used to assign the variable as an XML fragment.

XQuery

Slides 50 to 53



XQuery 2-4

- Some of the xml data type methods used with XQuery are described as follows:

exist()

- This method is used to determine if one or more specified nodes are present in the XML document.
- It returns 1 if the XQuery expression returned at least one node, 0 if the Xquery expression evaluated to an empty result, and NULL if the xml data type instance against which the query was executed is NULL.
- Following code snippet demonstrates the use of `exist()` method:

```
USE SampleDB
SELECT TeamID FROM CricketTeam WHERE
TeamInfo.exist('(/MatchDetails/Team)') = 1
GO
```

- This will return only those `TeamID` values where the `Team` element has been specified in the `TeamInfo`. The output is shown in the following figure:

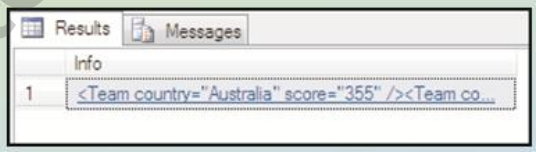
XQuery 3-4

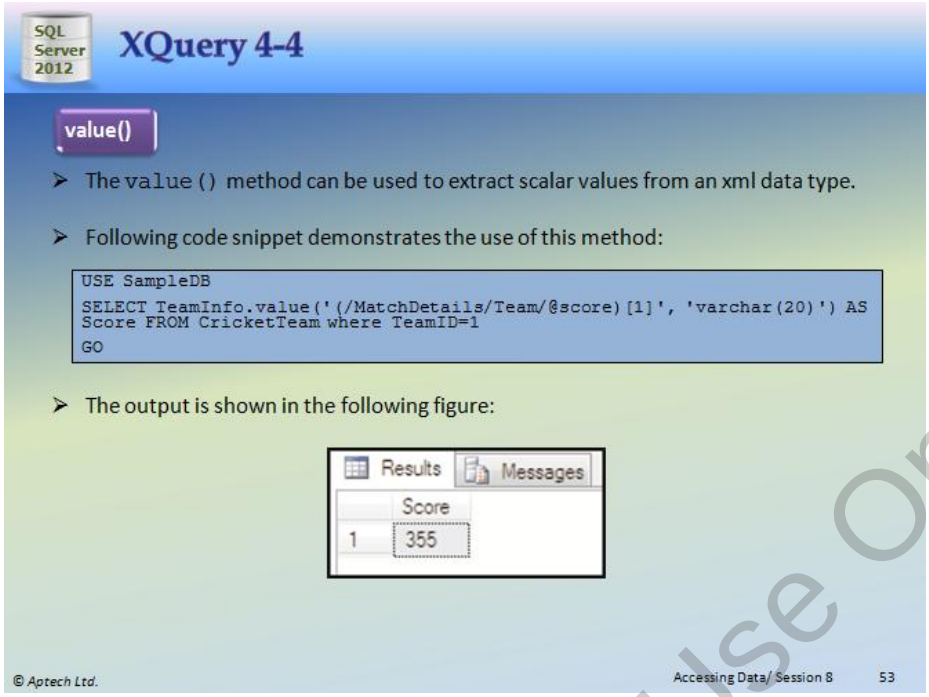
query()

- The `query()` method can be used to retrieve either the entire contents of an XML document or a selected section of the XML document.
- Following code snippet shows the use of `query()` method:

```
USE SampleDB
SELECT TeamInfo.query('/MatchDetails/Team') AS Info FROM CricketTeam
GO
```

- The output is shown in the following figure:





Using slides 50 to 53, explain the XQuery.

After XML data has been stored using the xml data type, it can be queried and retrieved using a language named, XQuery. XML Query or XQuery is a new query language, which combines syntax that is familiar to developers who work with the relational database, and XPath language, that is used to select individual sections or collections of elements from an XML document. XQuery can be query structured or semi-structured XML data. To query an XML instance stored in a variable or column of xml type, xml data type methods are used. For example, a variable of xml type is declared and queried by using methods of the xml data type. Developers need to query XML documents, and this involves transforming XML documents in the required format. XQuery makes it possible to perform complex queries against an XML data source over the Web.

Some of the xml data type methods used with XQuery are described as follows:

- **exist()**: This method is used to determine if one or more specified nodes are present in the XML document. It returns 1 if the XQuery expression returned at least one node, 0 if the Xquery expression evaluated to an empty result, and NULL if the xml data type instance against which the query was executed is NULL.
  Explain the code snippet demonstrates the use of exist() method. It is assumed that many records have been inserted into the table.
  This will return only those TeamID values where the Team element has been specified in the TeamInfo. The output is shown in figure on slide 51.
- **query()**: The query() method can be used to retrieve either the entire contents of an XML document or a selected section of the XML document. Code snippet shows the use of query() method using slide 52. The output of the command is shown in figure on slide 52.
- **value():** The value() method can be used to extract scalar values from an xml data type. Code snippet demonstrates this method.
  The output of the command is shown in figure on slide 53.

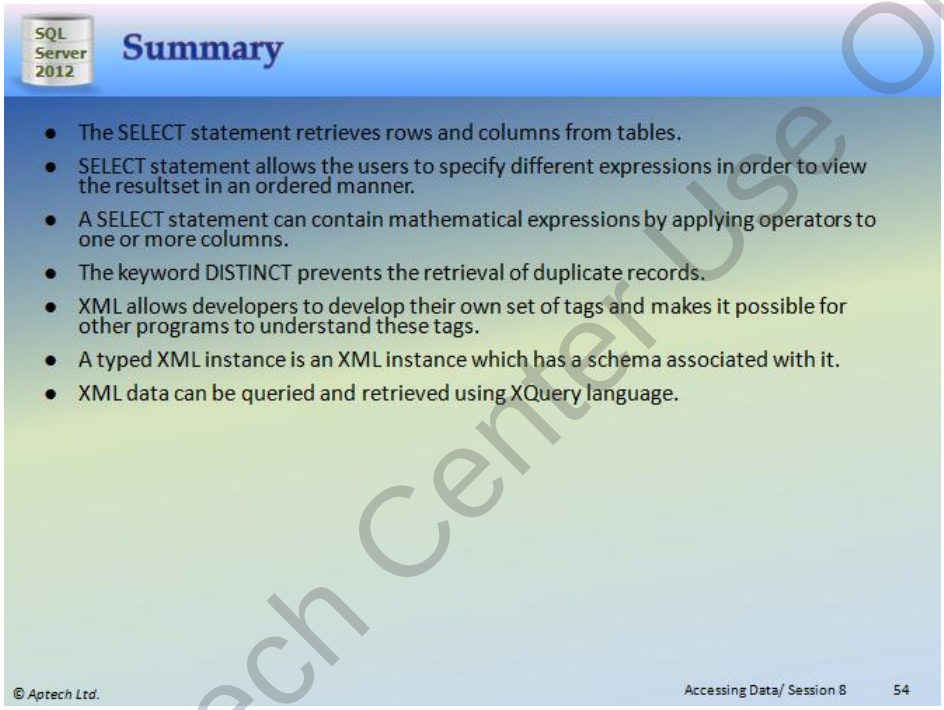
Also there are modify() and nodes() methods.

The modify() is used to modify the contents of an XML document. The modify() method of the xml data type can only be used in the SET clause of an UPDATE statement.

When you want to shared an xml data type instance into relational data, the nodes() method is useful. It allows you to identify nodes that will be mapped into a new row.

Summarize Session

Slide 54



Using slide 54, summarize the session. End the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- The SELECT statement retrieves rows and columns from tables.
- SELECT statement allows the users to specify different expressions in order to view the resultset in an ordered manner.
- A SELECT statement can contain mathematical expressions by applying operators to one or more columns.
- The keyword DISTINCT prevents the retrieval of duplicate records.
- XML allows developers to develop their own set of tags and makes it possible for other programs to understand these tags.
- A typed XML instance is an XML instance which has a schema associated with it.
- XML data can be queried and retrieved using XQuery language.

8.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Advanced Queries and Joins topic offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 9 – Advanced Queries and Joins

9.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

9.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain grouping and aggregating data
- Describe subqueries
- Describe table expressions
- Explain joins
- Describe various types of joins
- Explain the use of various set operators to combine data
- Describe pivoting and grouping set operations

9.1.2 Teaching Skills

To teach this session, you should be well-versed with the various techniques to group and aggregate data and describes the concept of subqueries, table expressions, joins, and explores various set operators.

The session also covers pivoting and grouping set operations.
You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slide and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

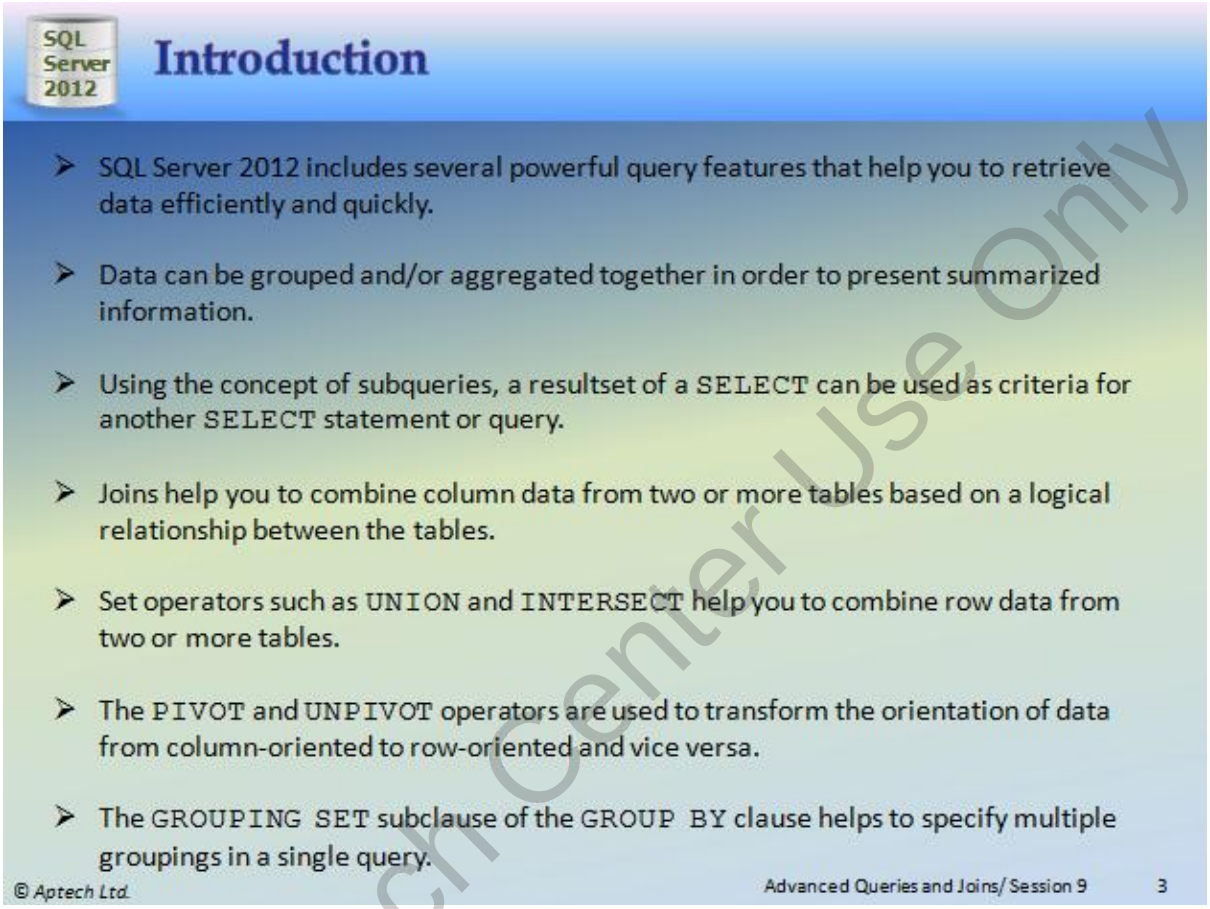
In-Class Activities:
Follow the order given here during In-Class activities.

Overview of the Session:
Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces the various techniques to group and aggregate data and describes the concept of subqueries, table expressions, joins, and explores various set operators. They will also know about pivoting and grouping set operations.

9.2 In-Class Explanations

Introduction

Slide 3

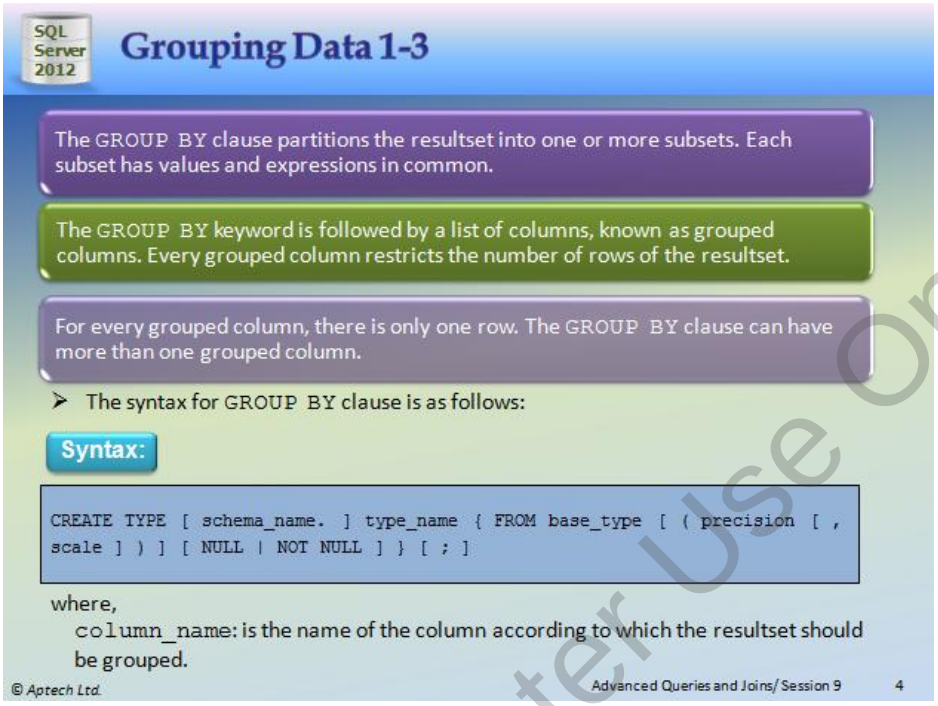


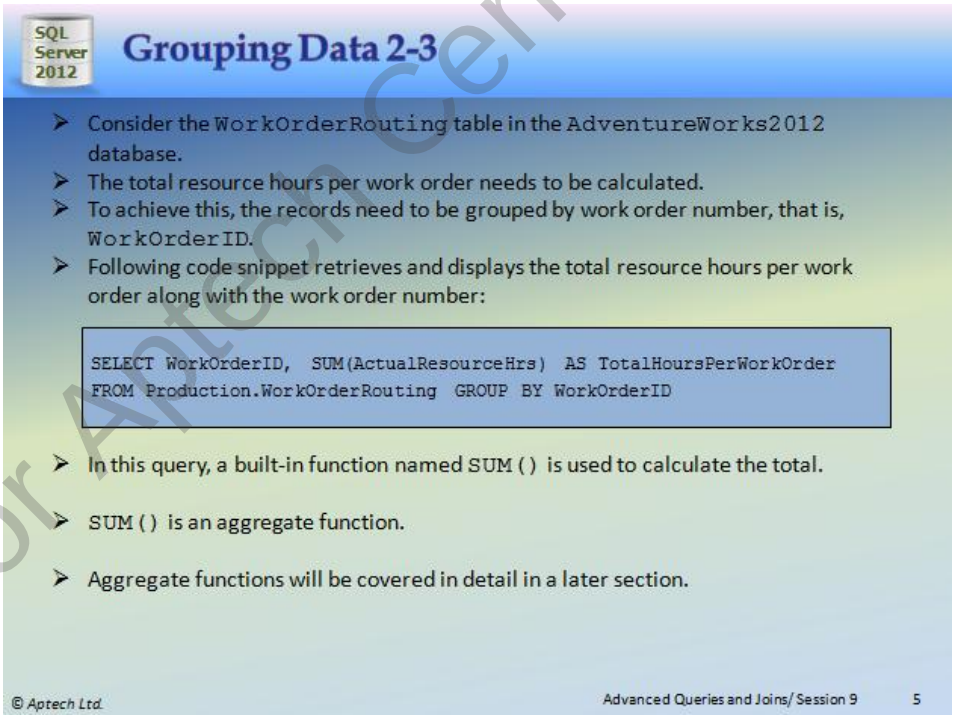
Using slide 3, explain the introduction.

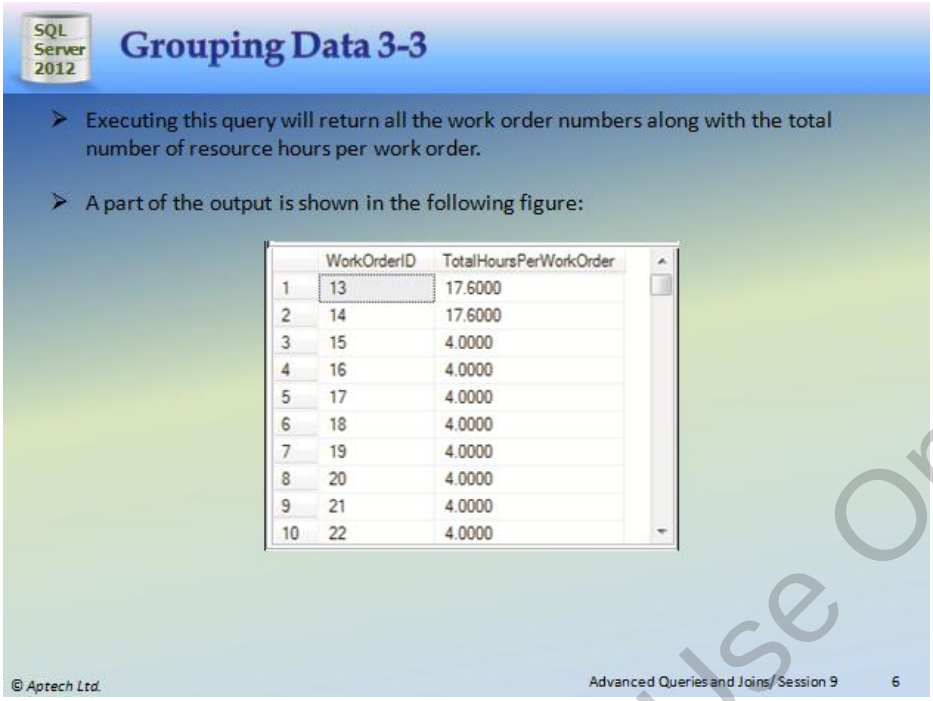
SQL Server 2012 includes several powerful query features that help you to retrieve data efficiently and quickly. Data can be grouped and/or aggregated together in order to present summarized information. Using the concept of subqueries, a resultset of a SELECT can be used as criteria for another SELECT statement or query. Joins help you to combine column data from two or more tables based on a logical relationship between the tables. On the other hand, set operators such as UNION and INTERSECT help you to combine row data from two or more tables. The PIVOT and UNPIVOT operators are used to transform the orientation of data from column-oriented to row-oriented and vice versa. The GROUPING SET subclause of the GROUP BY clause helps to specify multiple groupings in a single query.

Grouping Data

Slides 4 to 6







Using slides 4 to 6, explain the grouping data.

The GROUP BY clause partitions the resultset into one or more subsets. Each subset has values and expressions in common. The GROUP BY keyword is followed by a list of columns, known as grouped columns. Every grouped column restricts the number of rows of the resultset. For every grouped column, there is only one row. The GROUP BY clause can have more than one grouped column.

Explain the syntax of the GROUP BY clause.

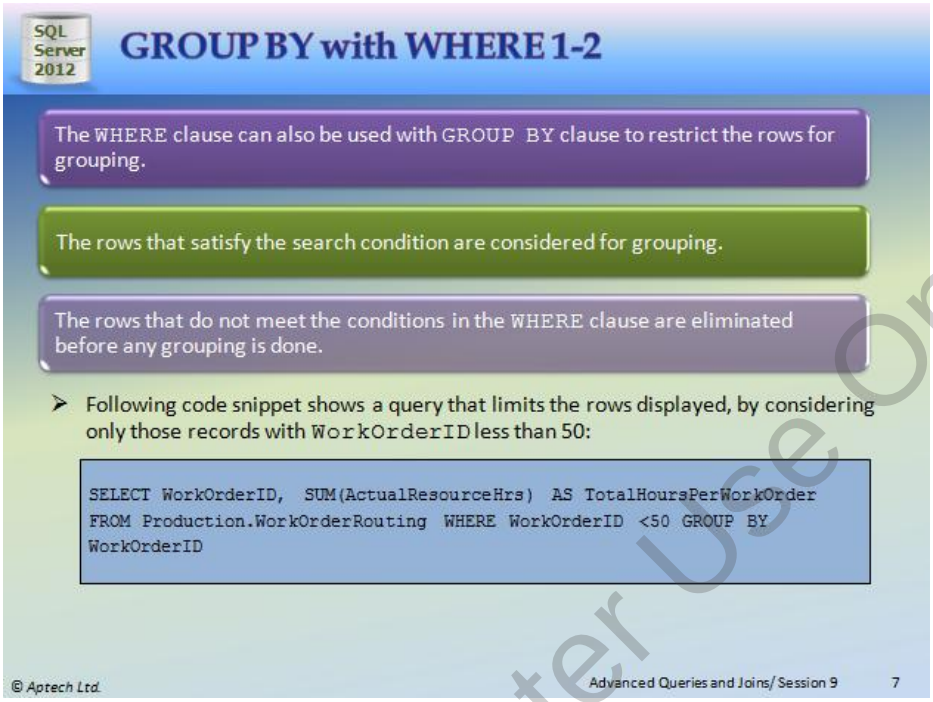
Consider the WorkOrderRouting table in the AdventureWorks2012 database. The total resource hours per work order needs to be calculated. To achieve this, the records need to be grouped by work order number, that is, WorkOrderID.

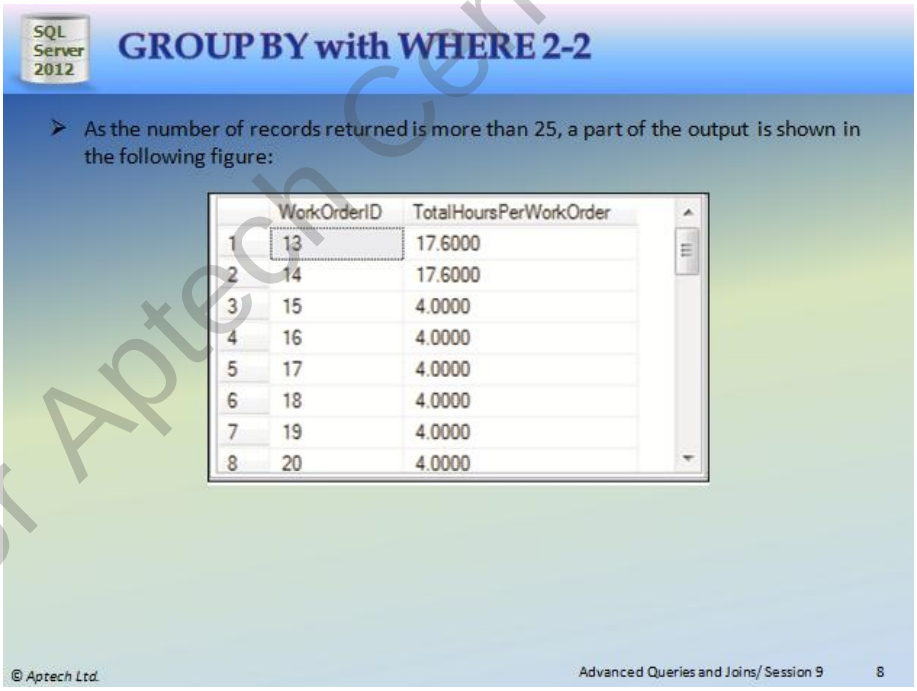
Explain the code snippet which retrieves and displays the total resource hours per work order along with the work order number. In this query, a built-in function named SUM() is used to calculate the total. SUM() is an aggregate function. Aggregate functions will be covered in detail in a later section.

Executing this query will return all the work order numbers along with the total number of resource hours per work order.

GROUP BY with WHERE

Slides 7 and 8





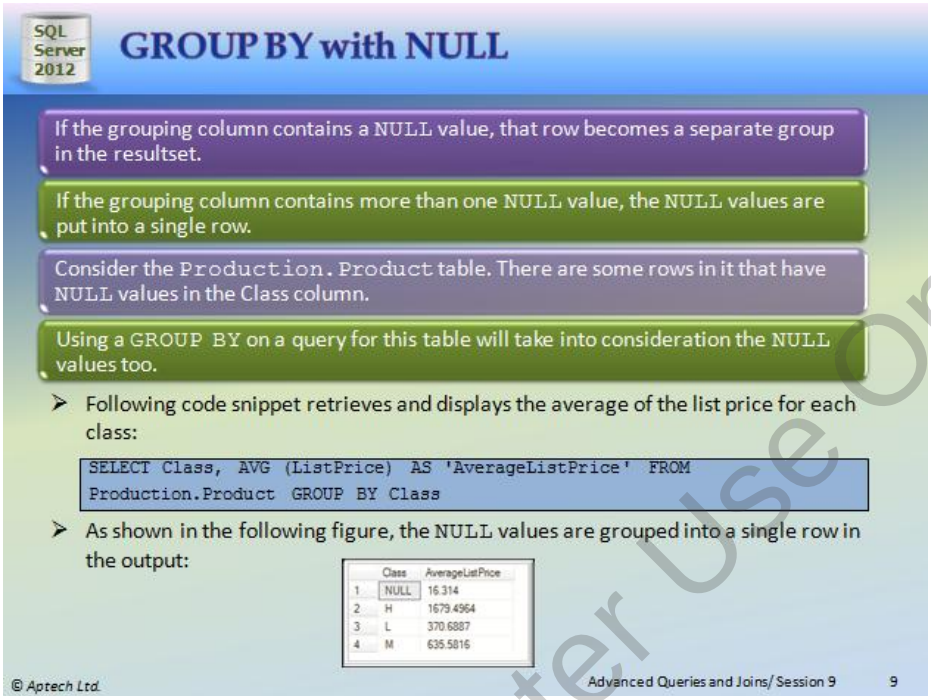
Using slides 7 and 8, explain the GROUP BY with WHERE.

The WHERE clause can also be used with GROUP BY clause to restrict the rows for grouping. The rows that satisfy the search condition are considered for grouping. The rows that do not meet the conditions in the WHERE clause are eliminated before any grouping is done.

Explain the code snippet that shows a query limits the rows displayed, by considering only those records with WorkOrderID less than 50.

GROUP BY with NULL

Slide 9



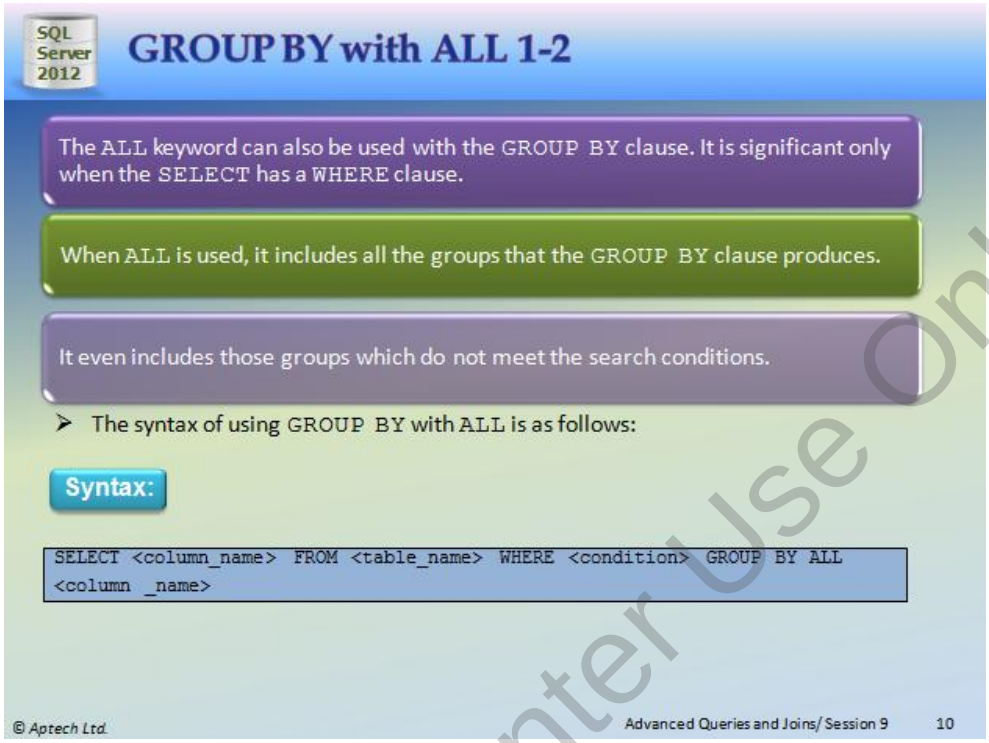
Using slide 9, explain the GROUP BY with NULL.

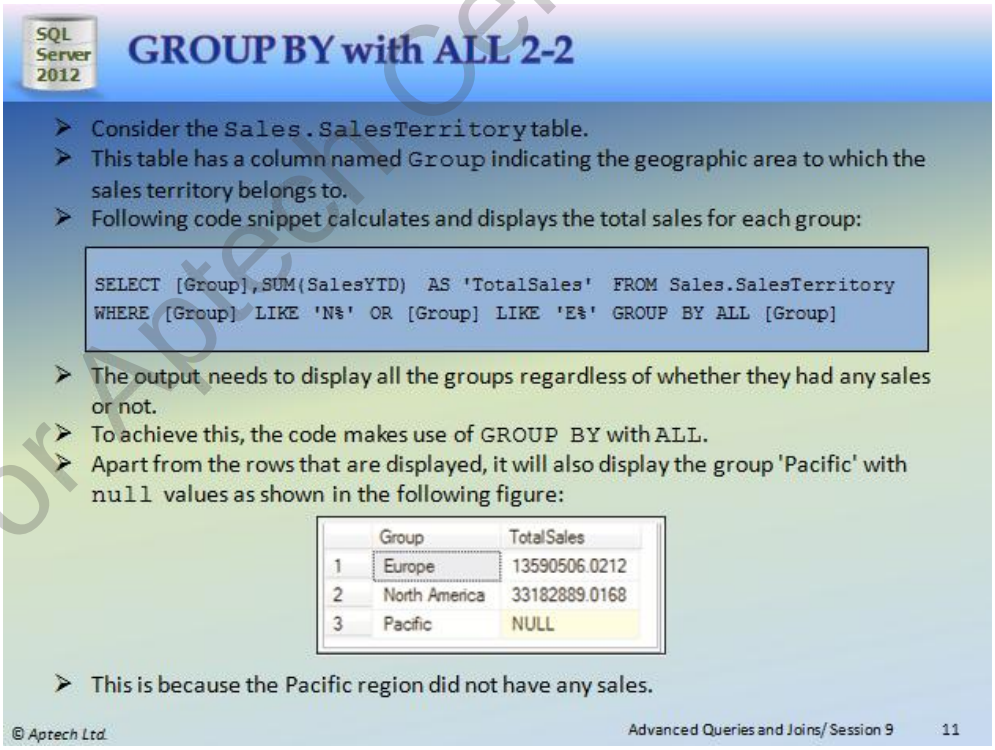
If the grouping column contains a NULL value, that row becomes a separate group in the resultset. If the grouping column contains more than one NULL value, the NULL values are put into a single row. Consider the Production.Product table. There are some rows in it that have NULL values in the Class column.

Using a GROUP BY on a query for this table will take into consideration the NULL values too. For example, code snippet retrieves and displays the average of the list price for each Class. As shown in figure, the NULL values are grouped into a single row in the output.

GROUP BY with ALL

Slides 10 and 11





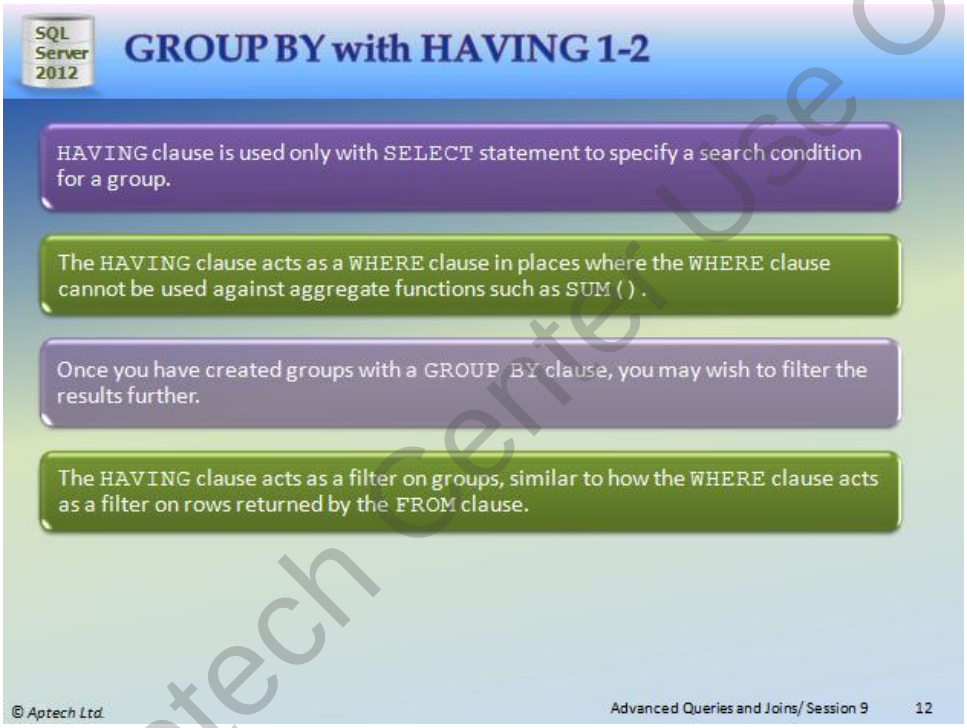
Using slides 10 and 11, explain the GROUP BY with ALL.

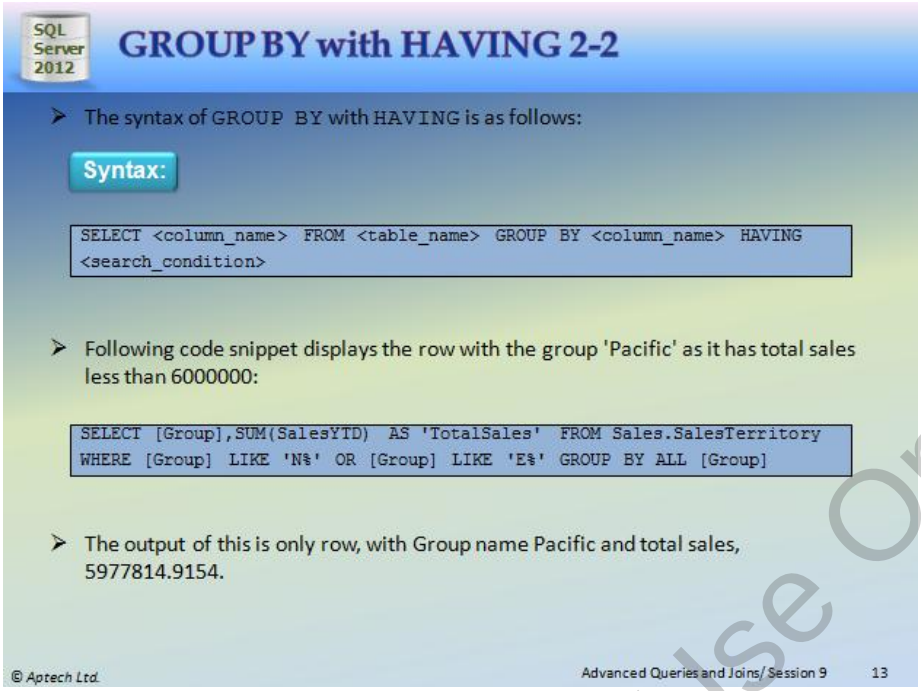
The ALL keyword can also be used with the GROUP BY clause. It is significant only when the SELECT has a WHERE clause. When ALL is used, it includes all the groups that the GROUP BY clause produces. It even includes those groups which do not meet the search conditions.

Explain the syntax of using GROUP BY with ALL.
Consider the Sales.SalesTerritory table. This table has a column named Group indicating the geographic area to which the sales territory belongs to. Code snippet calculates and displays the total sales for each group. The output needs to display all the groups regardless of whether they had any sales or not. To achieve this, the code makes use of GROUP BY with ALL.
Apart from the rows that are displayed in code snippet, it will also display the group 'Pacific' with null values as shown in figure. This is because the Pacific region did not have any sales.

GROUP BY with HAVING

Slides 12 and 13





Using slides 12 and 13, explain the GROUP BY with HAVING.

HAVING clause is used only with SELECT statement to specify a search condition for a group. The HAVING clause acts as a WHERE clause in places where the WHERE clause cannot be used against aggregate functions such as SUM(). Once you have created groups with a GROUP BY clause, you may wish to filter the results further. The HAVING clause acts as a filter on groups, similar to how the WHERE clause acts as a filter on rows returned by the FROM clause. Explain the syntax of GROUP BY with HAVING.

Explain the code snippet which displays the row with the group 'Pacific' as it has total sales less than 6000000.

The output of this is only row, with Group name Pacific and total sales, 5977814.9154.

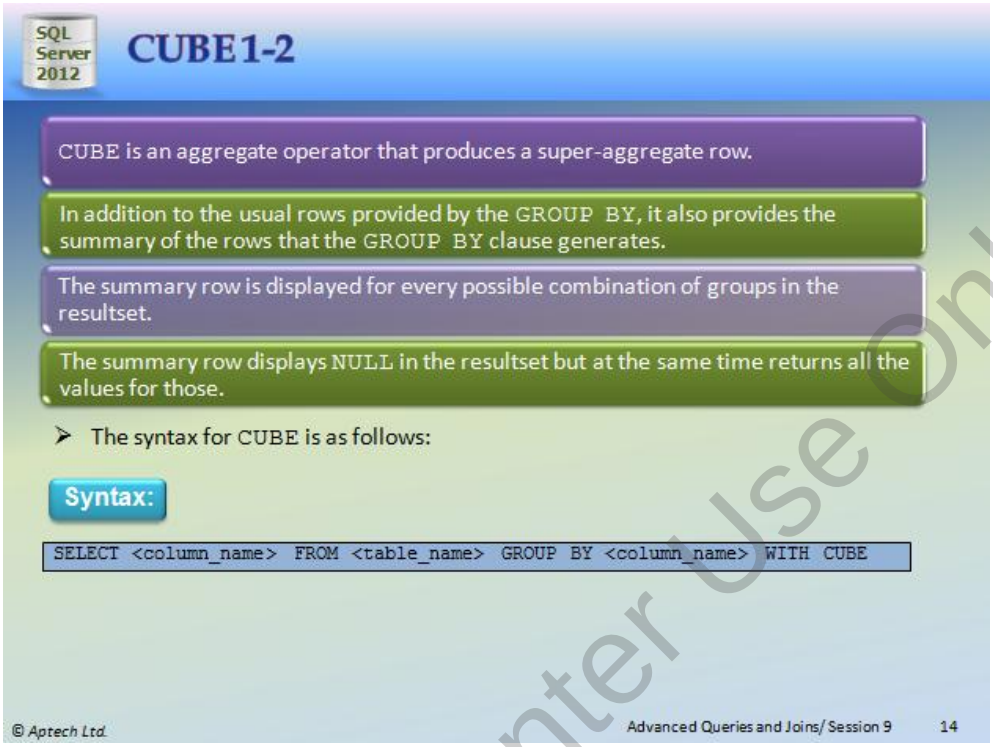
In-Class Question:

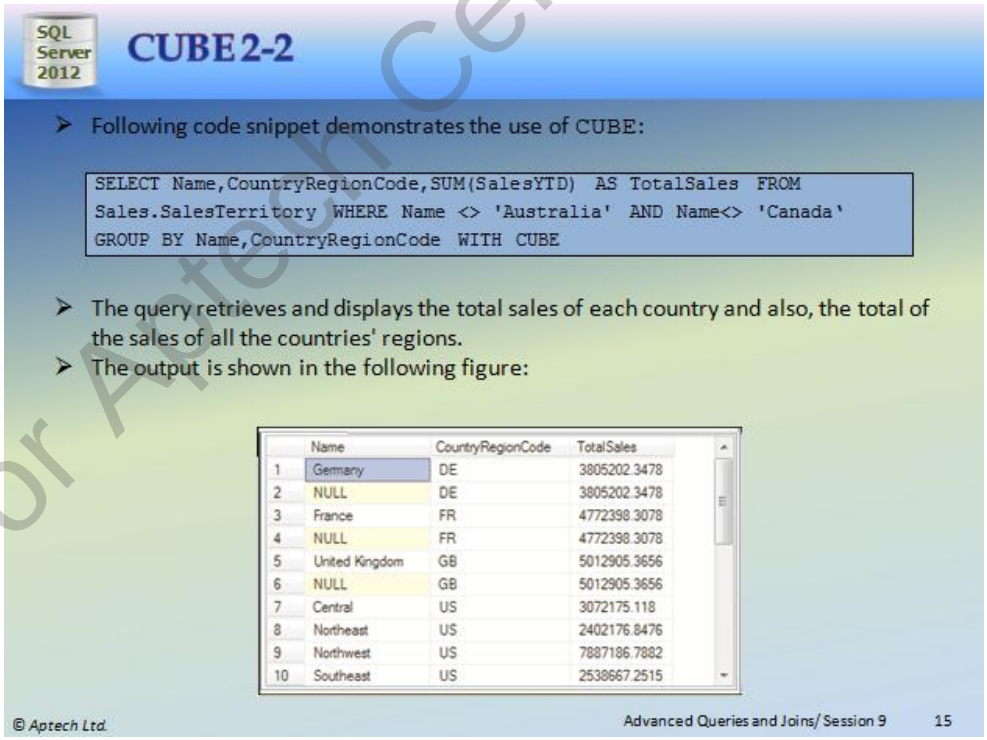

What is HAVING clause used for?

Answer:
The HAVING clause acts as a WHERE clause in places where the WHERE clause cannot be used against aggregate functions such as SUM().

CUBE

Slides 14 to 15





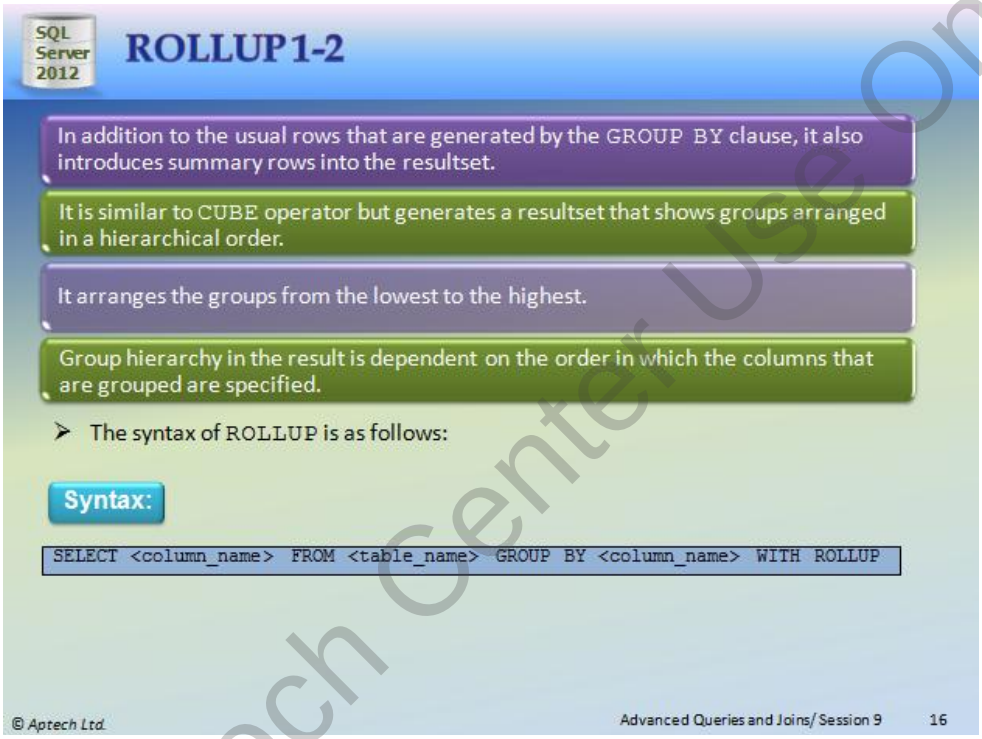
Using slides 14 and 15, explain the CUBE operator.

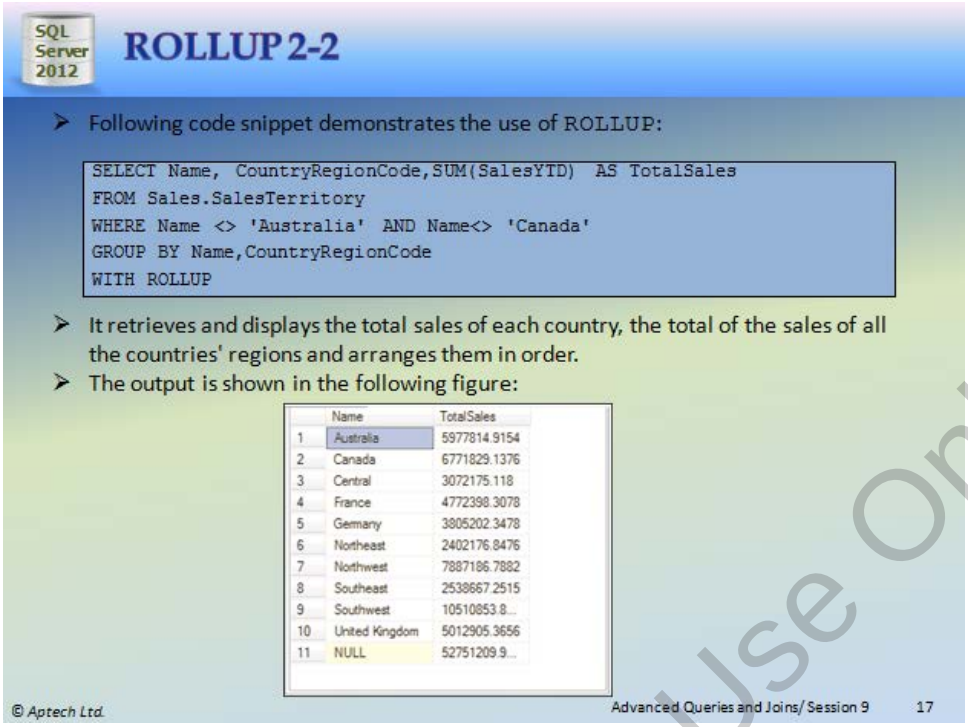
CUBE is an aggregate operator that produces a super-aggregate row. In addition to the usual rows provided by the GROUP BY, it also provides the summary of the rows that the GROUP BY clause generates. The summary row is displayed for every possible combination of

groups in the resultset. The summary row displays NULL in the resultset but at the same time returns all the values for those.
Explain the syntax of CUBE.
Explain the code snippet which demonstrates the use of CUBE.
Code snippet retrieves and displays the total sales of each country and also, the total of the sales of all the countries' regions. The output is shown in figure.

ROLLUP

Slides 16 and 17





Using slides 16 and 17, explain the ROLLUP operator.
In addition to the usual rows that are generated by the GROUP BY, it also introduces summary rows into the resultset. It is similar to CUBE operator but generates a resultset that shows groups arranged in a hierarchical order. It arranges the groups from the lowest to the highest. Group hierarchy in the result is dependent on the order in which the columns that are grouped are specified.

Explain the syntax of ROLLUP.

Explain the code snippet which demonstrates the use of ROLLUP. It retrieves and displays the total sales of each country, the total of the sales of all the countries' regions and arranges them in order.

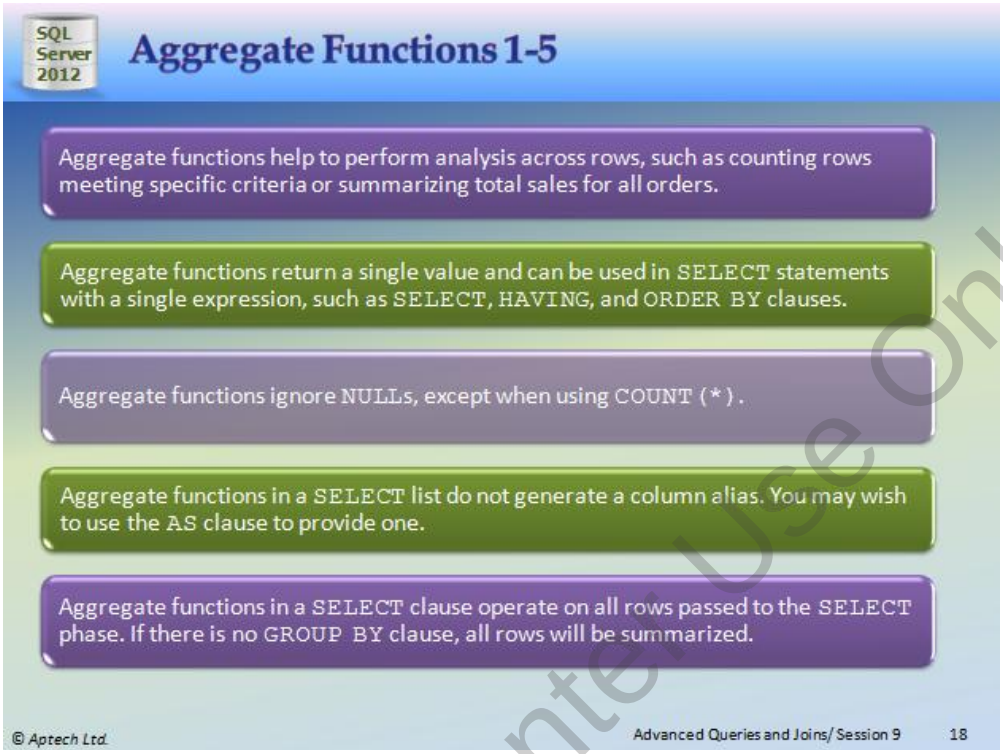
In-Class Question:

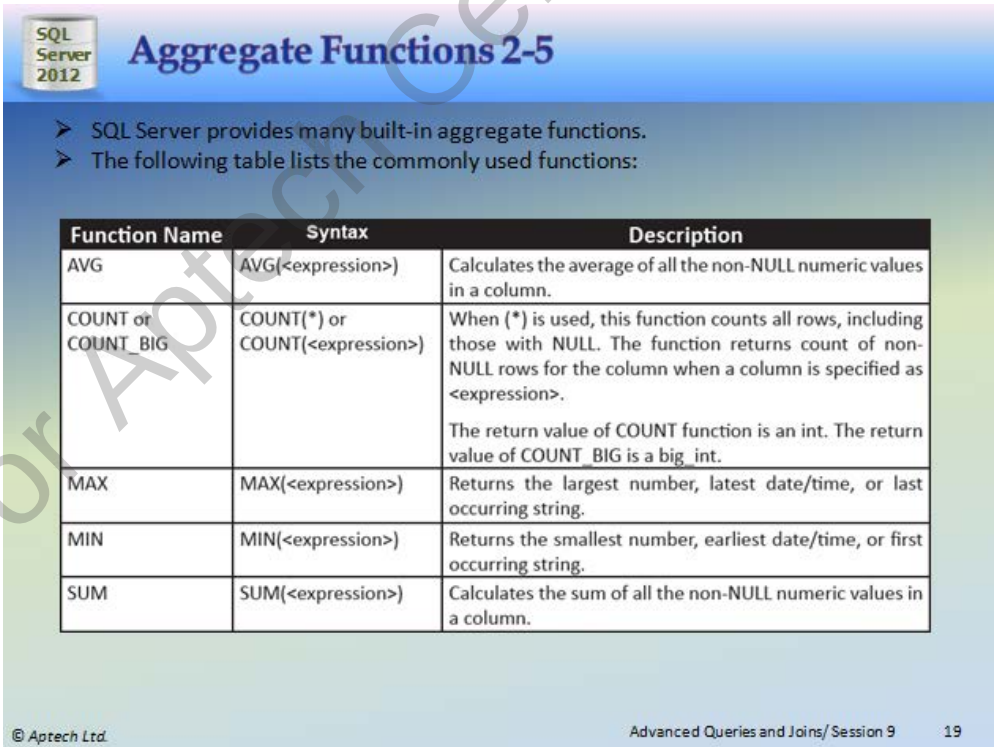
What is used to add summary rows in the sorted result set?

Answer:
ROLLUP operator is used to add summary rows in the sorted result set.

Aggregate Functions

Slides 18 to 22





Function Name	Syntax	Description
AVG	AVG(<expression>)	Calculates the average of all the non-NULL numeric values in a column.
COUNT or COUNT_BIG	COUNT(*) or COUNT(<expression>)	When (*) is used, this function counts all rows, including those with NULL. The function returns count of non-NULL rows for the column when a column is specified as <expression>. The return value of COUNT function is an int. The return value of COUNT_BIG is a big_int.
MAX	MAX(<expression>)	Returns the largest number, latest date/time, or last occurring string.
MIN	MIN(<expression>)	Returns the smallest number, earliest date/time, or first occurring string.
SUM	SUM(<expression>)	Calculates the sum of all the non-NULL numeric values in a column.

Aggregate Functions 3-5

- The following code snippet demonstrates the use of built-in aggregate in a SELECT clause:

```
SELECT AVG([UnitPrice]) AS AvgUnitPrice,
MIN([OrderQty])AS MinQty,
MAX([UnitPriceDiscount]) AS MaxDiscount
FROM Sales.SalesOrderDetail;
```

- Since the query does not use a GROUP BY clause, all rows in the table will be summarized by the aggregate formulas in the SELECT clause.
- The output is shown in the following figure:

	AvgUnitPrice	MinQty	MaxDiscount
1	465.0934	1	0.40

- When using aggregates in a SELECT clause, all columns referenced in the SELECT list must be used as inputs for an aggregate function or must be referenced in a GROUP BY clause.
- Failing this, there will be an error.

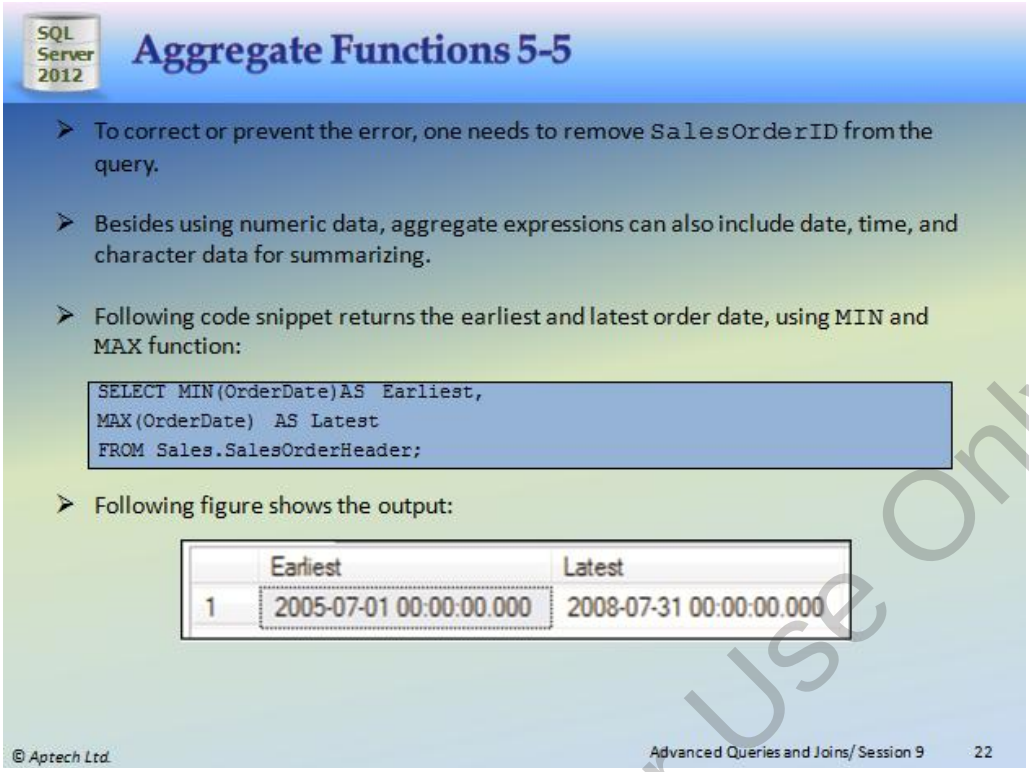
Aggregate Functions 4-5

- The following code snippet will return an error:

```
SELECT SalesOrderID, AVG(UnitPrice) AS AvgPrice
FROM Sales.SalesOrderDetail;
```

This returns an error stating that the column Sales.SalesOrderDetail.SalesOrderID is invalid in the SELECT list because it is not contained in either an aggregate function or the GROUP BY clause.

As the query is not using a GROUP BY clause, all rows will be treated as a single group. All columns, therefore, must be used as inputs to aggregate functions.



Using slides 18 to 22, explain the aggregate functions.
Occasionally, developers may also need to perform analysis across rows, such as counting rows meeting specific criteria or summarizing total sales for all orders. Aggregate functions enable to accomplish this.

Since aggregate functions return a single value, they can be used in SELECT statements where a single expression is used, such as SELECT, HAVING, and ORDER BY clauses.
Aggregate functions ignore NULLs, except when using COUNT(*).

Aggregate functions in a SELECT list do not generate a column alias. You may wish to use the AS clause to provide one.
Aggregate functions in a SELECT clause operate on all rows passed to the SELECT phase. If there is no GROUP BY clause, all rows will be summarized.
SQL Server provides many built-in aggregate functions. Commonly used functions are included in table shown on slide 19.
To use a built-in aggregate in a SELECT clause, consider the query shown in code snippet on slide 20.
Since the query does not use a GROUP BY clause, all rows in the table will be summarized by the aggregate formulas in the SELECT clause. The output is shown in figure.

When using aggregates in a SELECT clause, all columns referenced in the SELECT list must be used as inputs for an aggregate function or must be referenced in a GROUP BY clause. Failing this, there will be an error.

For example, the query in code snippet will return an error.

This returns an error stating that the column Sales.SalesOrderDetail.SalesOrderID is invalid in the SELECT list because it is not contained in either an aggregate function or the GROUP BY clause. As the query is not using a GROUP BY clause, all rows will be treated as a single group. All columns, therefore, must be used as inputs to aggregate functions. To correct or prevent the error, one needs to remove SalesOrderID from the query.
Besides using numeric data, aggregate expressions can also include date, time, and character data for summarizing.
Code snippet returns the earliest and latest order date on slide 22, using MIN and MAX.
Figure shows the output.
Other functions may also be used in combination with aggregate functions.

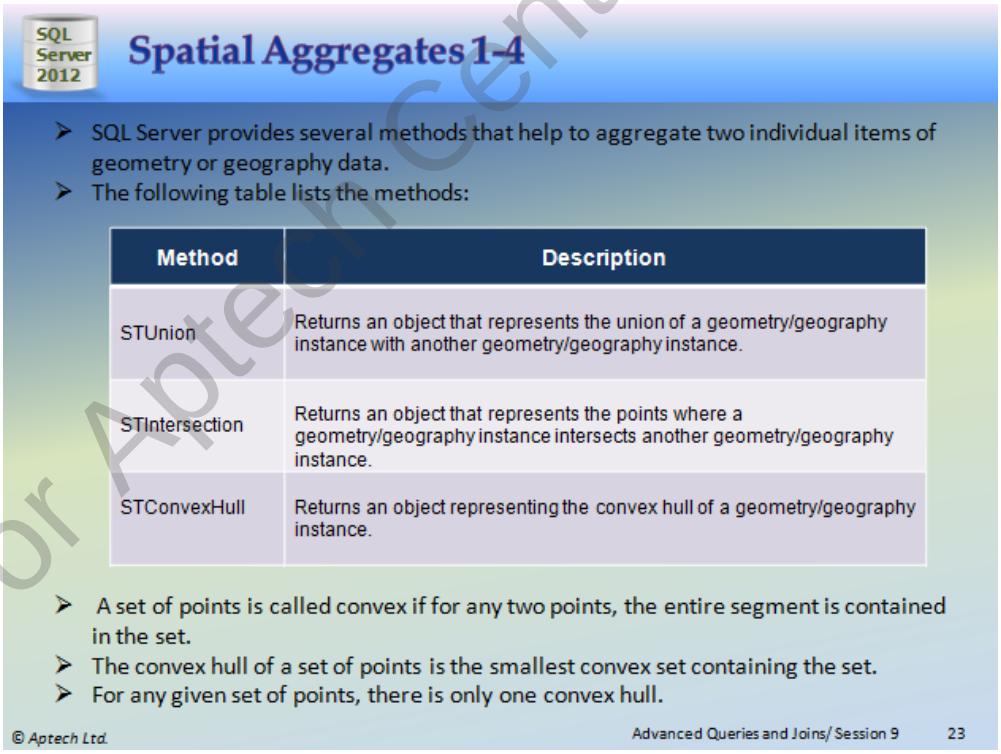
In-Class Question:

Which function is used to count the number of rows in a table?

Answer:
count(*) function is used to count the number of rows in a table.

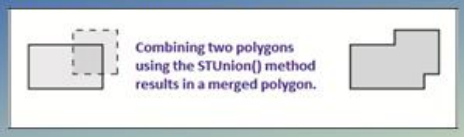
Spatial Aggregates

Slides 23 to 26

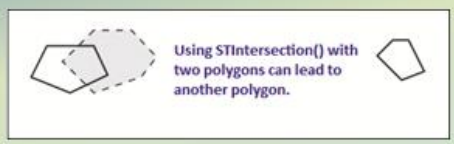


Spatial Aggregates 2-4

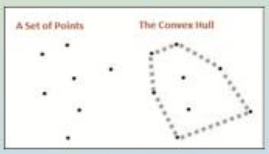
- Following figure depicts an example of `STUnion()`:



- Following figure depicts an example of `STIntersection()`:



- Following figure depicts an example of `STConvexHull()`:

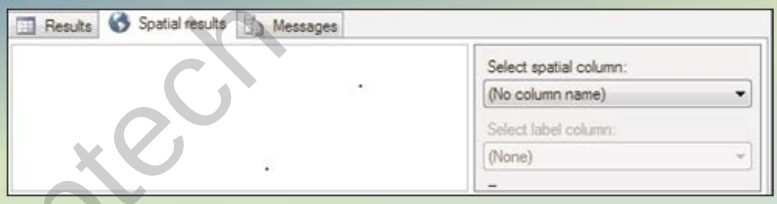


Spatial Aggregates 3-4

- The following code snippet demonstrates the use of `STUnion()`:

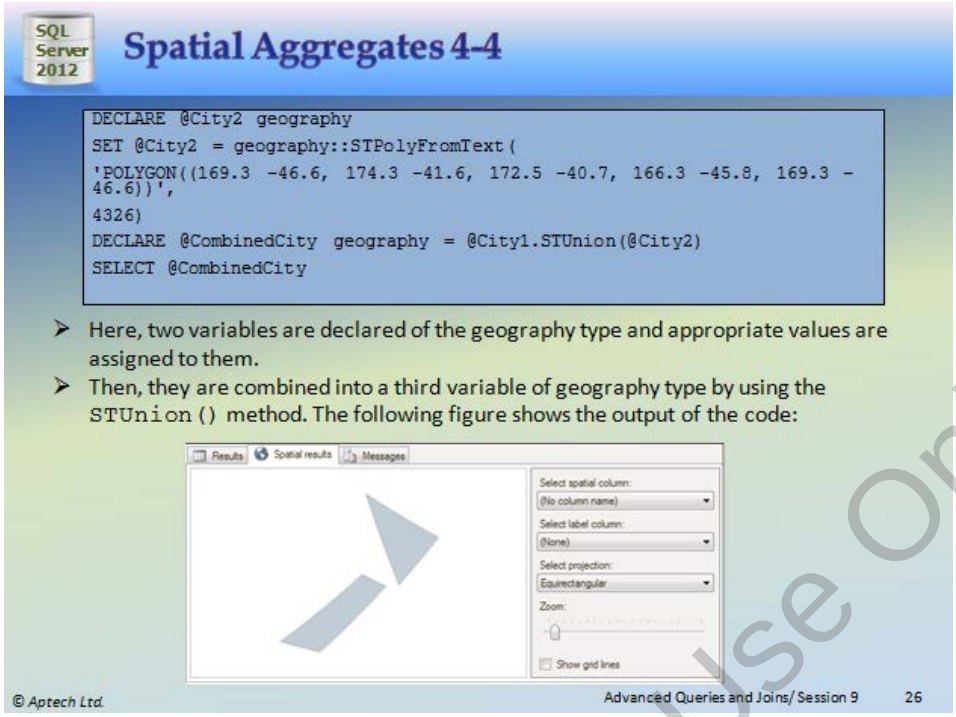
```
SELECT geometry::Point(251, 1,
4326).STUnion(geometry::Point(252,2,4326));
```

- The following figure displays the output as two points:



- The following code snippet displays another example:

```
DECLARE @City1 geography
SET @City1 = geography::STPolyFromText(
'POLYGON((175.3 -41.5, 178.3 -37.9, 172.8 -34.6, 175.3 -41.5))',
4326)
```



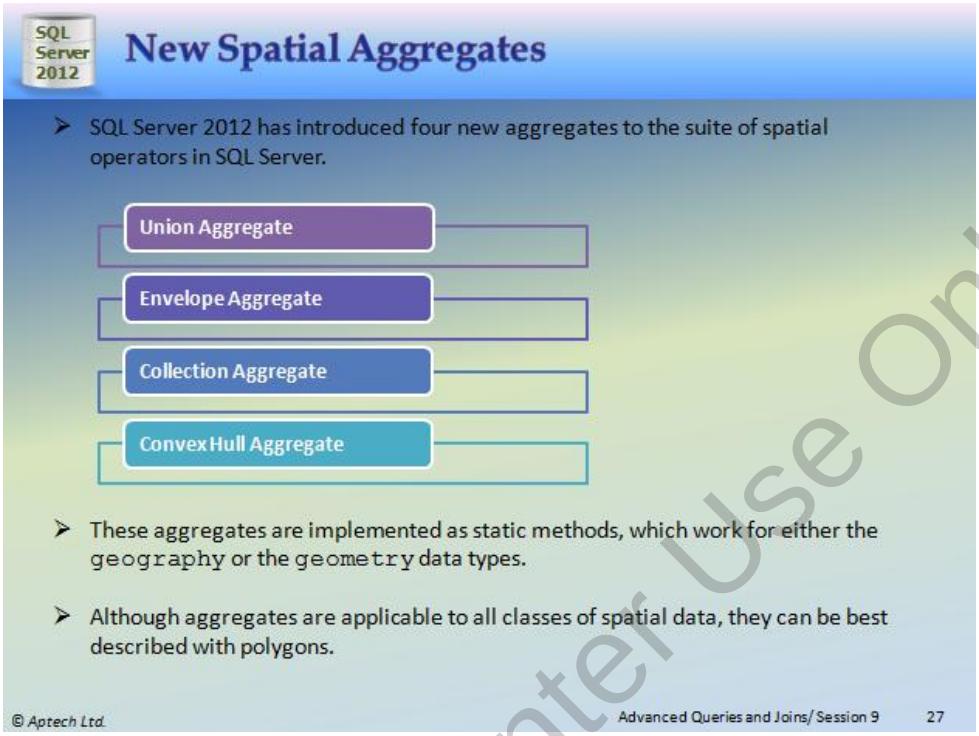
Using slides 23 to 26, explain the spatial aggregates.

SQL Server provides several methods that help to aggregate two individual items of geometry or geography data. These methods are listed in table.
Figures visually depict an example of these methods are shown in slide 24.
Explain the code snippet which demonstrates the use of STUnion().
The output is shown in figure on slide. It shows two points.
Another example is given in code snippet on slide 25.

Here, two variables are declared of the geography type and appropriate values are assigned to them. Then, they are combined into a third variable of geography type by using the STUnion() method. The output of the code is shown in figure on slide 26.

New Spatial Aggregates

Slide 27



Using slide 27, explain the new spatial aggregates.

It is easy to compute unions of regular numeric data by using basic operators with queries such as SELECT x + y or by using the UNION operator. You can compute the union of two individual geometries or two geographies using the STUnion() operator. What if there was a need to compute the union of a set of geometry/geography objects or all the elements in a spatial column? What if there was a need to find out the average of a set of Point elements? It would not be possible to use the AVG() function here. In such a case, you will make use of the new spatial aggregate functions in SQL Server 2012.

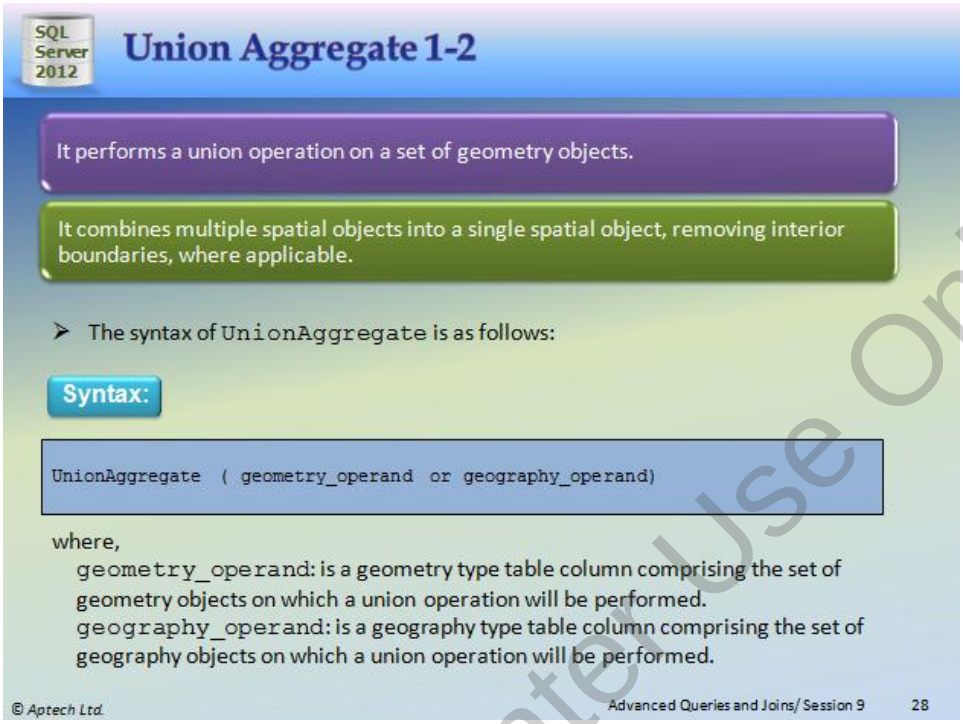
SQL Server 2012 has introduced four new aggregates to the suite of spatial operators in SQL Server:

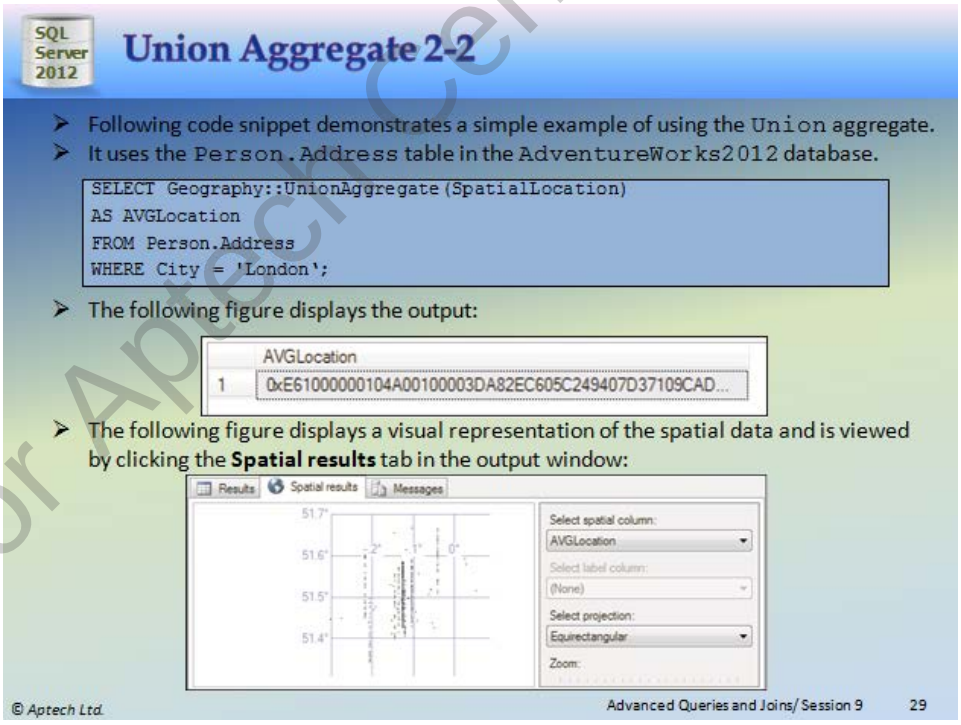
- Union Aggregate
- Envelope Aggregate
- Collection Aggregate
- Convex Hull Aggregate

These aggregates are implemented as static methods, which work for either the geography or the geometry data types. Although aggregates are applicable to all classes of spatial data, they can be best described with polygons.

Union Aggregate

Slides 28 and 29





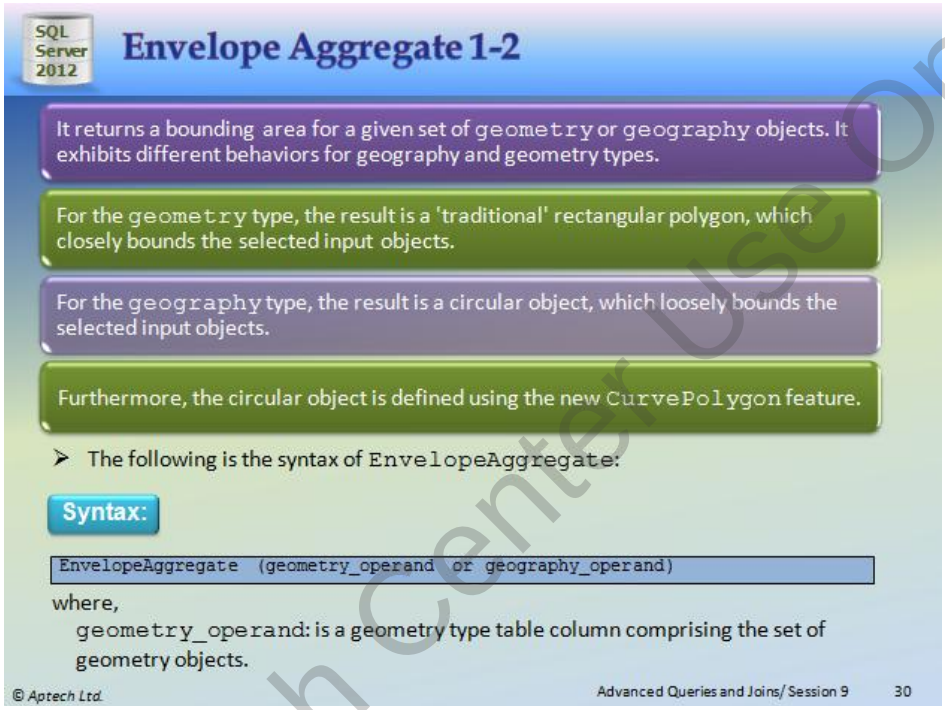
Using slides 28 and 29, explain the Union Aggregate.

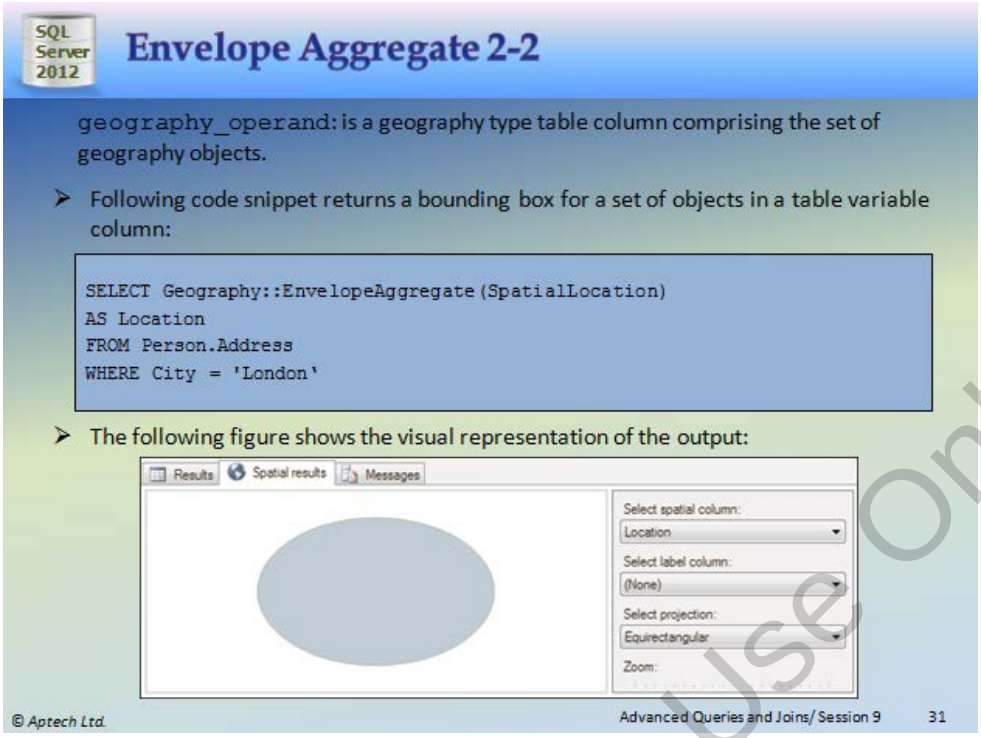
It performs a union operation on a set of geometry objects. It combines multiple spatial objects into a single spatial object, removing interior boundaries, where applicable. Explain the syntax of Union Aggregate.

Explain the code snippet which demonstrates a simple example of using the Union aggregate. It uses the Person.Address table in the AdventureWorks2012 database.
The output of this will be as shown in figure.
To view a visual representation of the spatial data, you can click the **Spatial results** tab in the output window. This will display the output as shown in figure.

Envelope Aggregate

Slides 30 and 31





Using slides 30 and 31, explain the Envelope Aggregate.

It returns a bounding area for a given set of geometry or geography objects.
The Envelope Aggregate exhibits different behaviors for geography and geometry types. Based on the type of object it is applied to, it returns different results. For the geometry type, the result is a 'traditional' rectangular polygon, which closely bounds the selected input objects. For the geography type, the result is a circular object, which loosely bounds the selected input objects. Furthermore, the circular object is defined using the new CurvePolygon feature.

Explain the syntax of Envelope Aggregate.
Code snippet returns a bounding box for a set of objects in a table variable column.
The visual representation of the output is shown in figure.

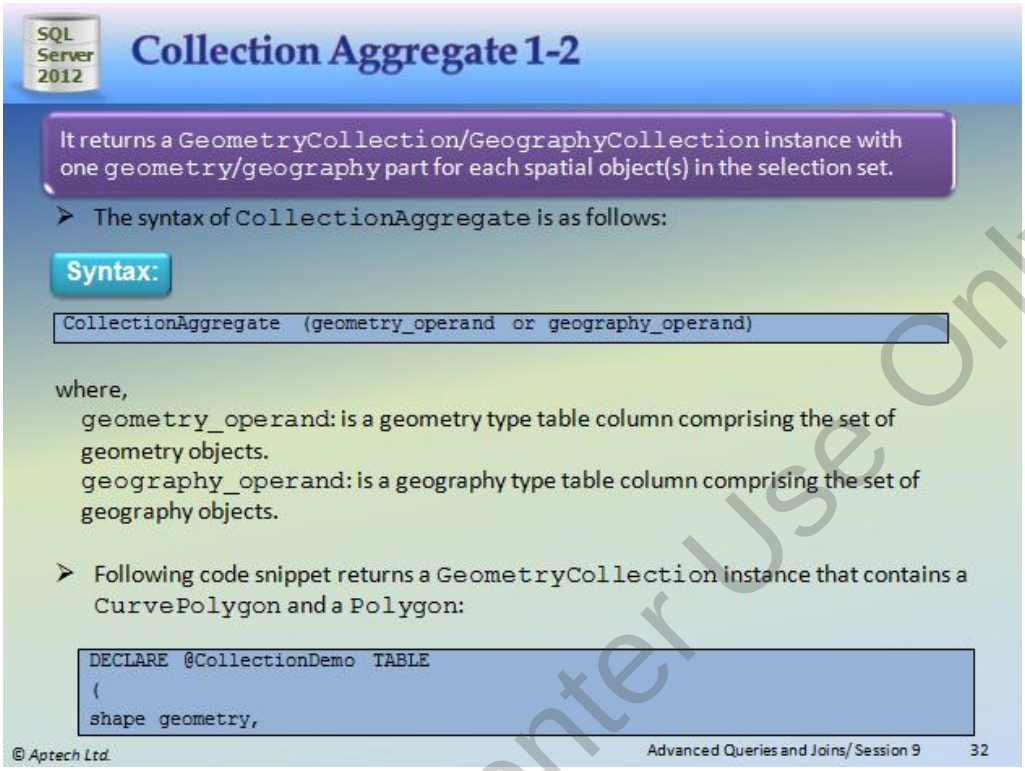
In-Class Question:

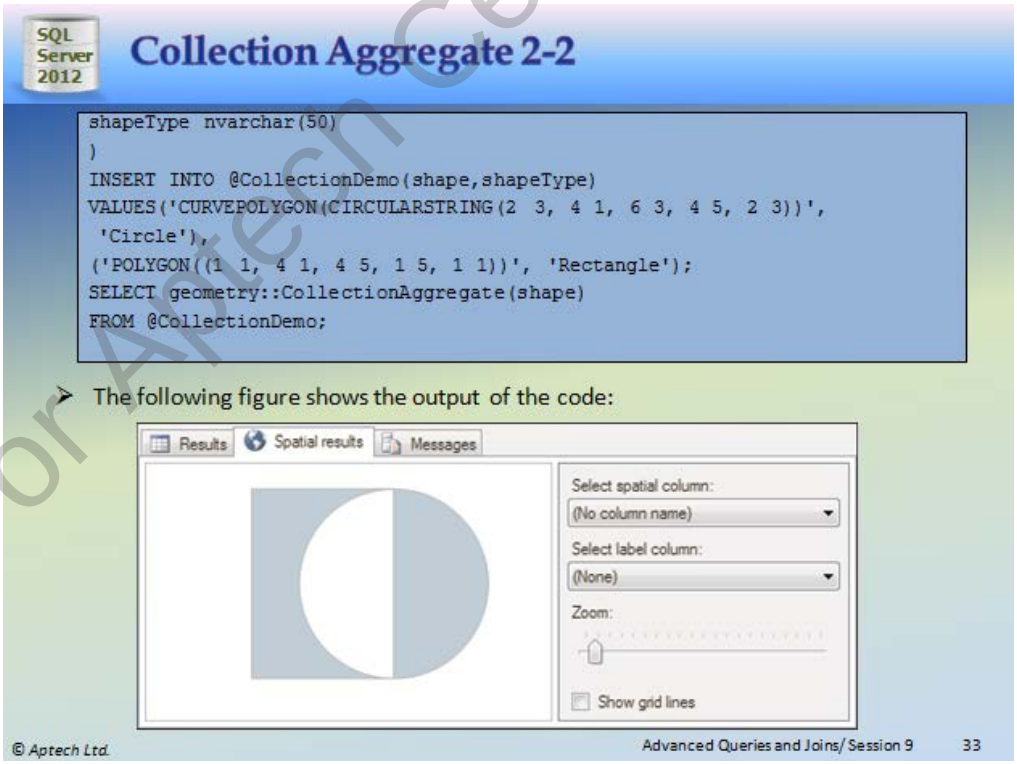
What is result for the geography type object?

Answer:
Circular object, which loosely bounds the selected input objects is the result for the geography type object.

Collection Aggregate

Slides 32 and 33



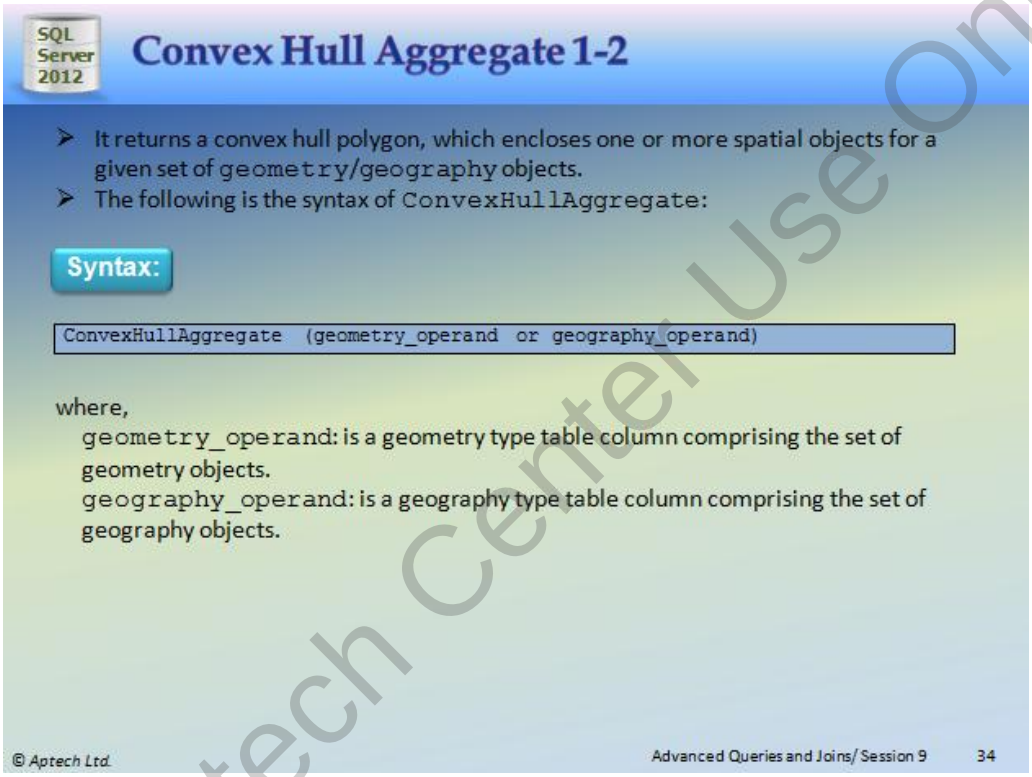


Using slides 32 and 33, explain the Collection Aggregate.

It returns a GeometryCollection/GeographyCollection instance with one geometry/ geography part for each spatial object(s) in the selection set. Explain the syntax of Collection Aggregate.
Code snippet returns a GeometryCollection instance that contains a CurvePolygon and a Polygon.
The output of the code is shown in figure on slide 33.

Convex Hull Aggregate

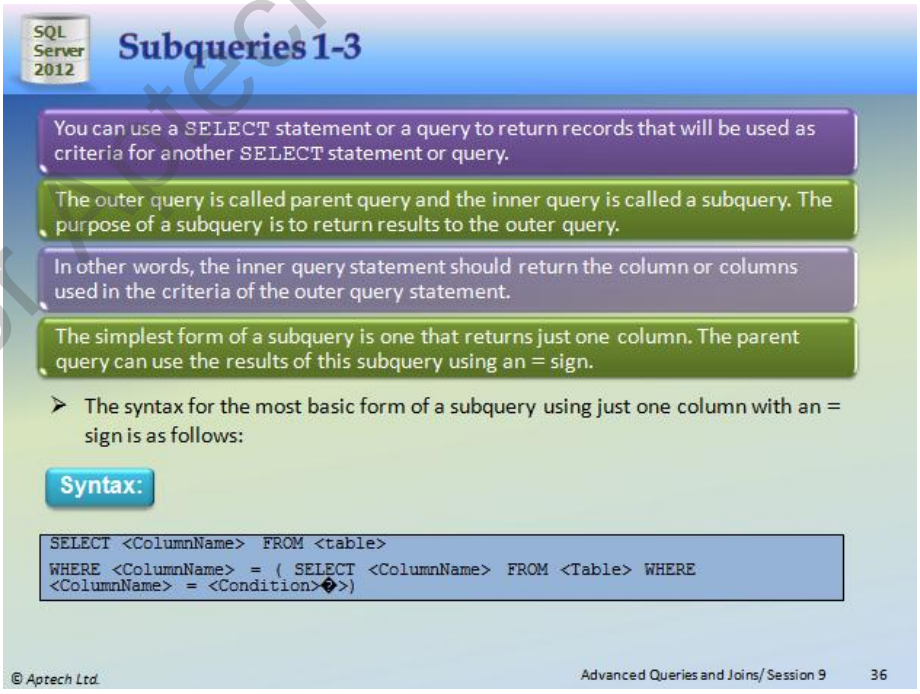
Slides 34 and 35

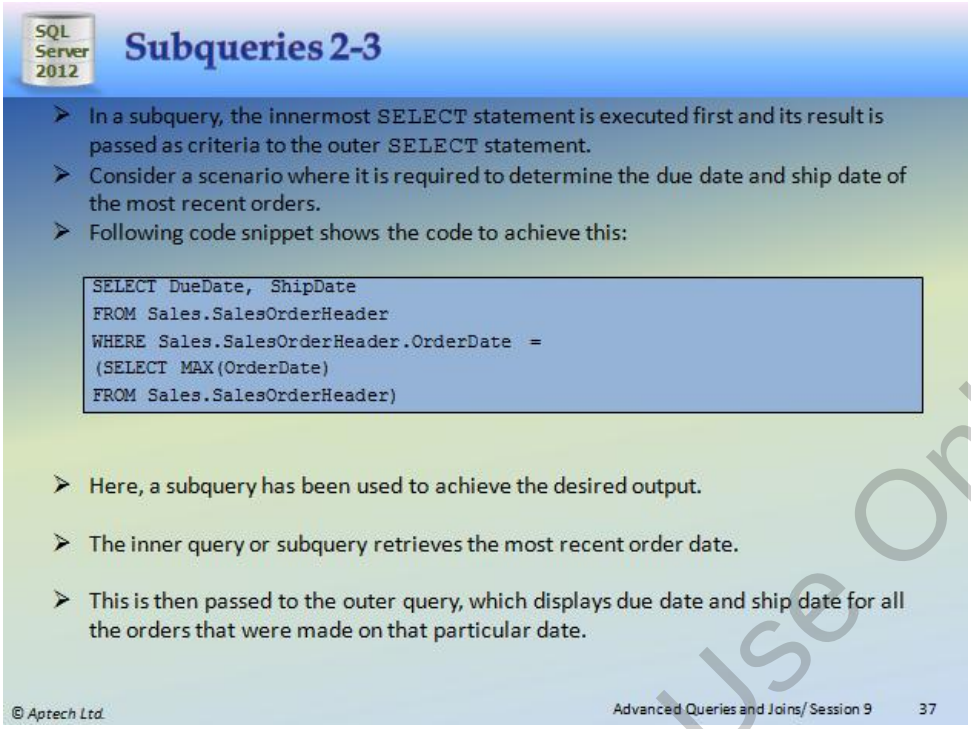


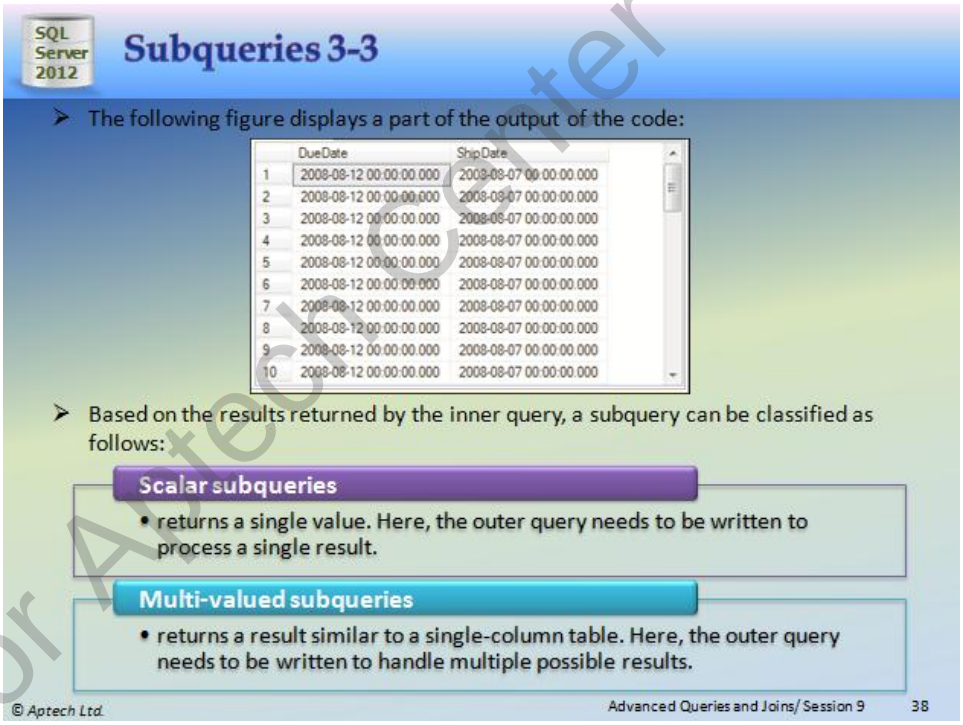
Using slides 34 and 35, explain the Convex Hull Aggregate.
Mention it returns a convex hull polygon, which encloses one or more spatial objects for a given set of geometry/geography objects.
Explain the syntax of ConvexHullAggregate. Explain the code snippet which demonstrates the use of ConvexHullAggregate.
The output is shown in figure on slide 35.

Subqueries

Slides 36 to 38







Using slides 36 to 38, explain the subqueries.
Mention SELECT statement or a query to return records that will be used as criteria for another SELECT statement or query. The outer query is called parent query and the inner query is called a subquery. The purpose of a subquery is to return results to the outer query. In other words, the inner query statement should return the column or columns used in the criteria of the outer query statement.

The simplest form of a subquery is one that returns just one column. The parent query can use the results of this subquery using an = sign.

The syntax for the most basic form of a subquery using just one column with an = sign is shown.

In a subquery, the innermost SELECT statement is executed first and its result is passed as criteria to the outer SELECT statement.
Consider a scenario where it is required to determine the due date and ship date of the most recent orders. Explain the code snippet that shows the code to achieve this.

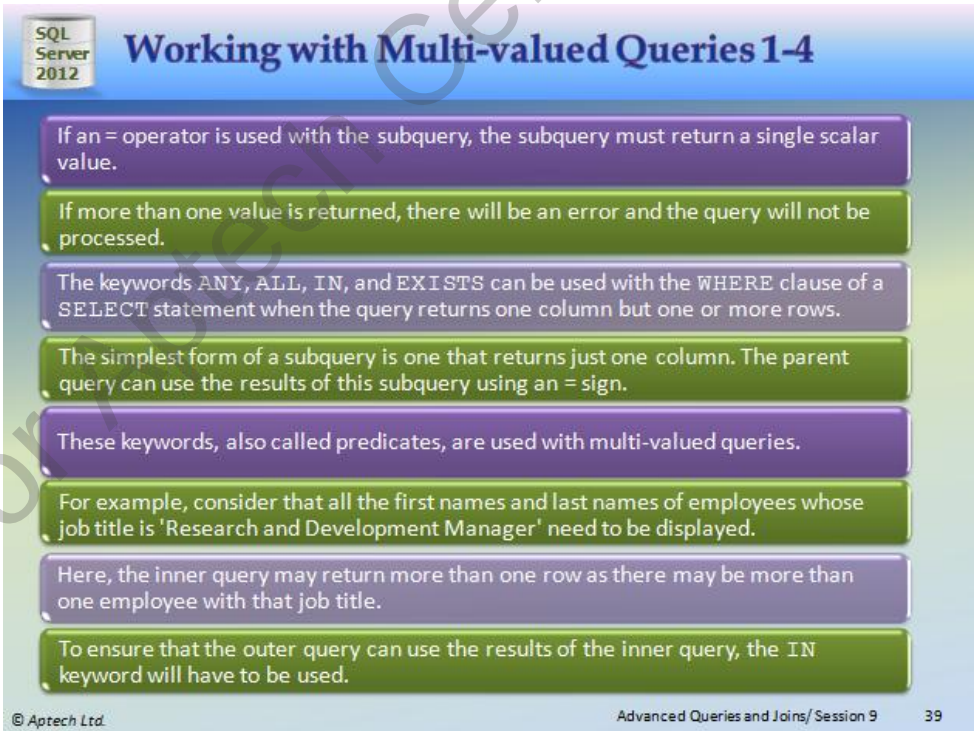
Here, a subquery has been used to achieve the desired output. The inner query or subquery retrieves the most recent order date. This is then passed to the outer query, which displays due date and ship date for all the orders that were made on that particular date. A part of the output of the code is shown in figure.

Based on the results returned by the inner query, a subquery can be classified as a scalar subquery or a multi-valued subquery. These are described as follows:

- Scalar subqueries return a single value. Here, the outer query needs to be written to process a single result.
- Multi-valued subqueries return a result similar to a single-column table. Here, the outer query needs to be written to handle multiple possible results.

Multi-valued Queries

Slides 39 to 42



Working with Multi-valued Queries 2-4

- Following code snippet demonstrates this:

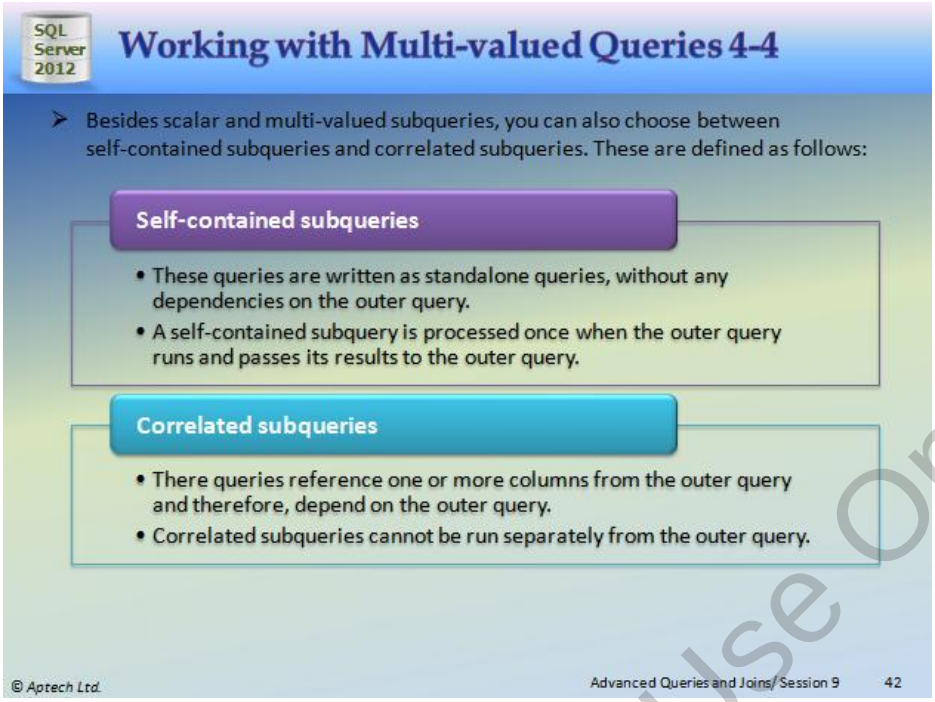
```
SELECT FirstName, LastName FROM Person.Person
WHERE Person.Person.BusinessEntityID IN (SELECT BusinessEntityID
FROM HumanResources.Employee WHERE JobTitle ='Research and Development
Manager');
```

- Here, the inner query retrieves the `BusinessEntityID` from the `HumanResources.Employee` table for those records having job title 'Research and Development Manager'.
- These results are then passed to the outer query, which matches the `BusinessEntityID` with that in the `Person.Person` table.
- Finally, from the records that are matching, the first and last names are extracted and displayed.
- The following figure displays the output:

	FirstName	LastName
1	Dylan	Miller
2	Michael	Raheem

Working with Multi-valued Queries 3-4

- You should remember the following points when using subqueries:
  - The `ntext`, `text`, and `image` data types cannot be used in the `SELECT` list of subqueries.
  - The `SELECT` list of a subquery introduced with a comparison operator can have only one expression or column name.
  - Subqueries that are introduced by a comparison operator not followed by the keyword `ANY` or `ALL` cannot include `GROUP BY` and `HAVING` clauses.
  - You cannot use `DISTINCT` keyword with subqueries that include `GROUP BY`.
  - You can specify `ORDER BY` only when `TOP` is also specified.



Using slides 39 to 42, explain the working with multi-valued queries.

Mention if an = operator is used with the subquery, the subquery must return a single scalar value. If more than one value is returned, there will be an error and the query will not be processed. In such scenarios, the keywords ANY, ALL, IN, and EXISTS can be used with the WHERE clause of a SELECT statement when the query returns one column but one or more rows.

These keywords, also called predicates, are used with multi-valued queries.
For example, consider that all the first names and last names of employees whose job title is 'Research and Development Manager' need to be displayed. Here, the inner query may return more than one row as there may be more than one employee with that job title. To ensure that the outer query can use the results of the inner query, the IN keyword will have to be used.

Explain the code snippet which demonstrates this on slide 40.
Here, the inner query retrieves the BusinessEntityID from the HumanResources.Employee table for those records having job title 'Research and Development Manager'. These results are then passed to the outer query, which matches the BusinessEntityID with that in the Person.Person table. Finally, from the records that are matching, the first and last names are extracted and displayed. The output is displayed in the figure.

The SOME or ANY keywords evaluate to true if the result is an inner query containing at least one row that satisfies the comparison. They compare a scalar value with a column of values. SOME and ANY are equivalent; both return the same result. They are rarely used.
There are some guidelines to be followed when working with subqueries.

You should remember the following points when using subqueries:

- The ntext, text, and image data types cannot be used in the SELECT list of subqueries.
- The SELECT list of a subquery introduced with a comparison operator can have only one expression or column name.
- Subqueries that are introduced by a comparison operator not followed by the keyword ANY or ALL cannot include GROUP BY and HAVING clauses.
- You cannot use DISTINCT keyword with subqueries that include GROUP BY.
- You can specify ORDER BY only when TOP is also specified.

Besides scalar and multi-valued subqueries, you can also choose between self-contained subqueries and correlated subqueries. These are defined as follows:

- Self-contained subqueries are written as standalone queries, without any dependencies on the outer query. A self-contained subquery is processed once when the outer query runs and passes its results to the outer query.
- Correlated subqueries reference one or more columns from the outer query and therefore, depend on the outer query. Correlated subqueries cannot be run separately from the outer query.

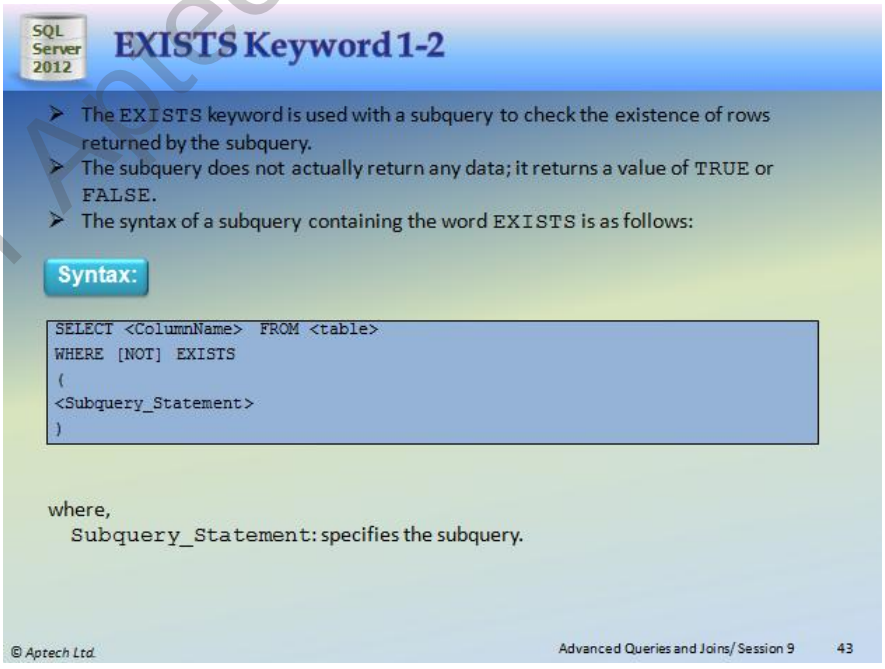
In-Class Question:

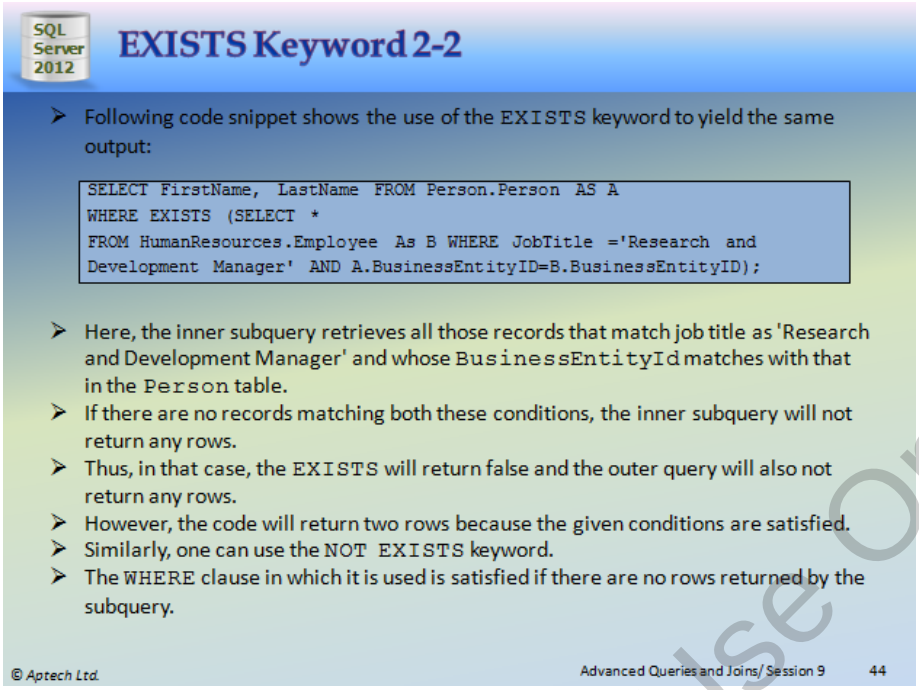
What is a self-contained query?

Answer:
Self-contained subqueries are written as standalone queries, without any dependencies on the outer query.

EXISTS Keyword

Slides 43 and 44





Using slides 43 and 44, explain the EXIST keyword.
The EXISTS keyword is used with a subquery to check the existence of rows returned by the subquery. The subquery does not actually return any data; it returns a value of TRUE or FALSE.

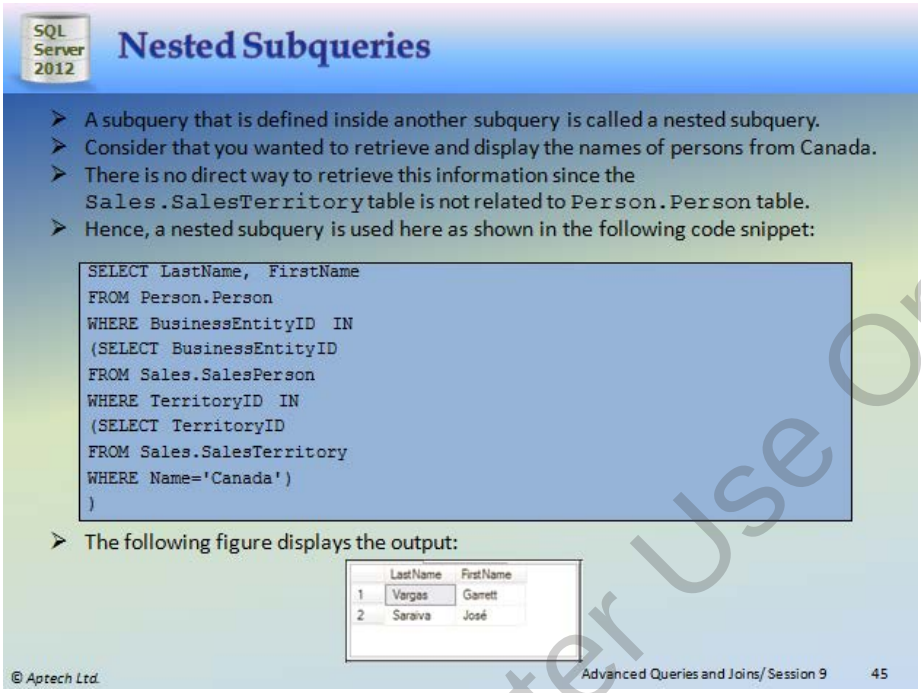
Explain the syntax of a subquery containing the word EXISTS.
The code can be rewritten as shown in code snippet using the EXISTS keyword to yield the same output on slide 44.

Here, the inner subquery retrieves all those records that match job title as 'Research and Development Manager' and whose BusinessEntityId matches with that in the Person table. If there are no records matching both these conditions, the inner subquery will not return any rows. Thus, in that case, the EXISTS will return false and the outer query will also not return any rows. However, the code in code snippet will return two rows because the given conditions are satisfied. The output will be the same as figure on slide 44.

Similarly, one can use the NOT EXISTS keyword. The WHERE clause in which it is used is satisfied if there are no rows returned by the subquery.

Nested Subqueries

Slide 45

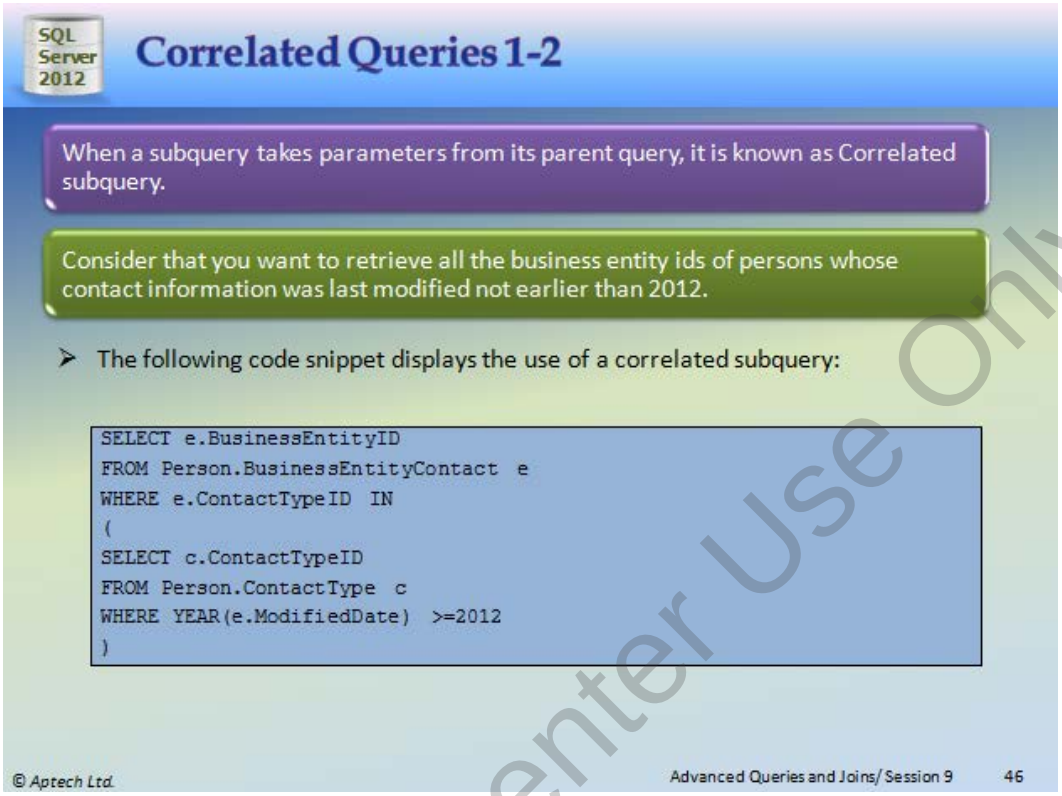


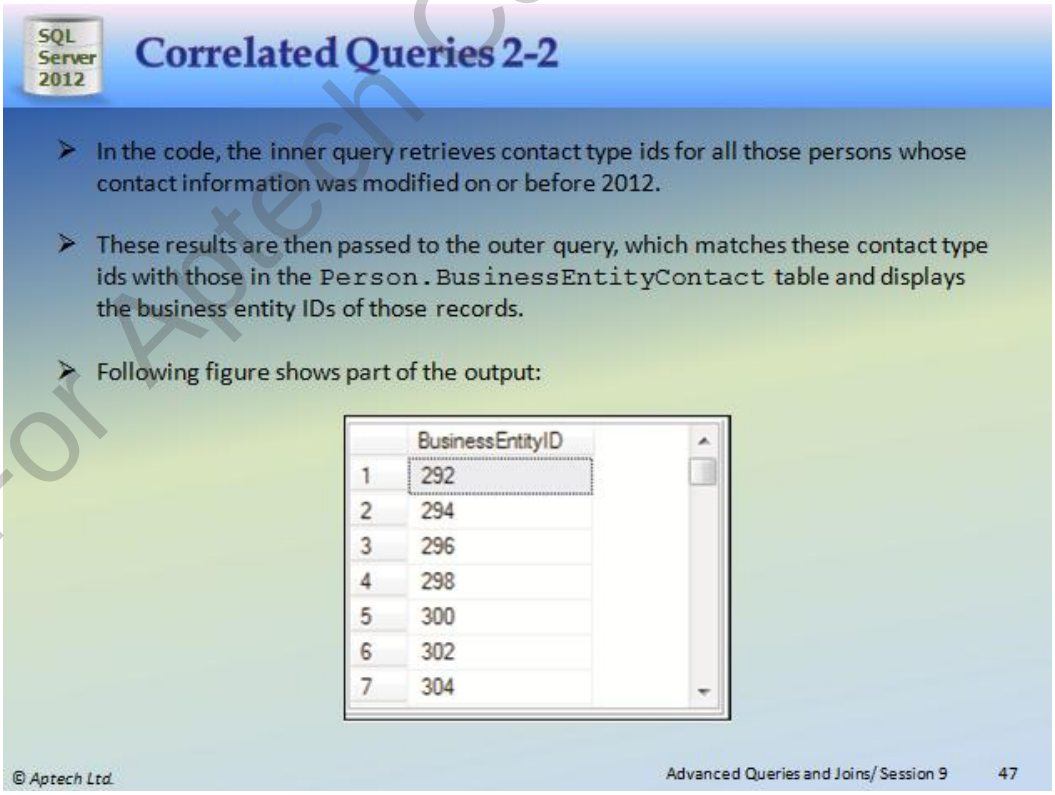
Using slide 45, explain the nested subqueries.

A subquery that is defined inside another subquery is called a nested subquery. Consider that you wanted to retrieve and display the names of persons from Canada. There is no direct way to retrieve this information since the Sales.SalesTerritory table is not related to Person.Person table. Hence, a nested subquery is used here as shown in code snippet. The output is shown in figure.

Correlated Queries

Slides 46 and 47





Using slides 46 and 47, explain the correlated queries.

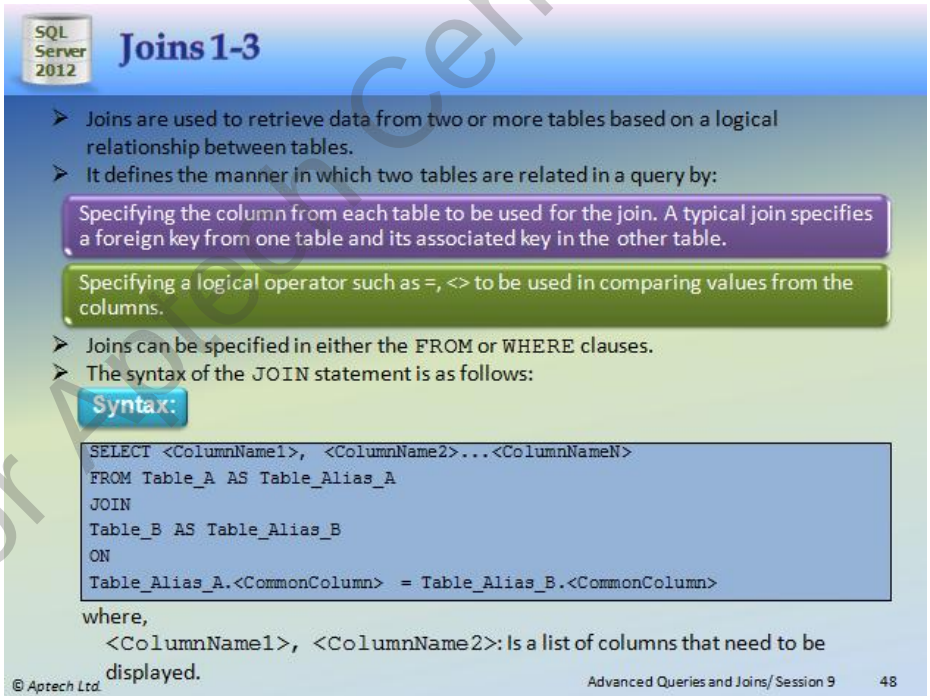
In many queries containing subqueries, the subquery needs to be evaluated only once to provide the values needed by the parent query. This is because in most of the queries, the subquery makes no reference to the parent query, so the value in the subquery remains constant.

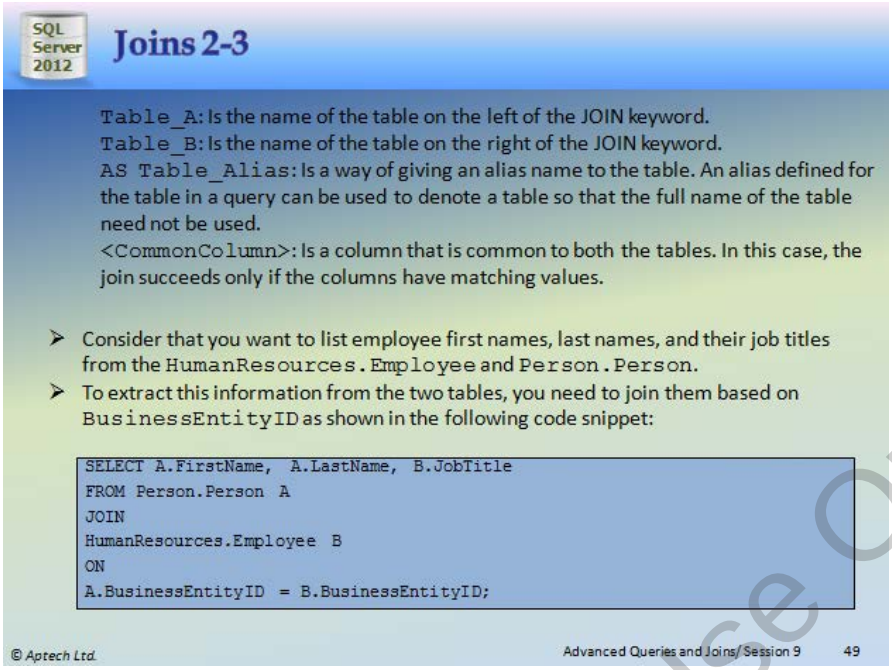
However, if the subquery refers to a parent query, the subquery needs to be revaluated for every iteration in the parent query. This is because the search criterion in the subquery is dependent upon the value of a particular record in the parent query.
When a subquery takes parameters from its parent query, it is known as Correlated subquery. Consider that you want to retrieve all the business entity ids of persons whose contact information was last modified not earlier than 2012. To do this, you can use a correlated subquery as shown in the code snippet.

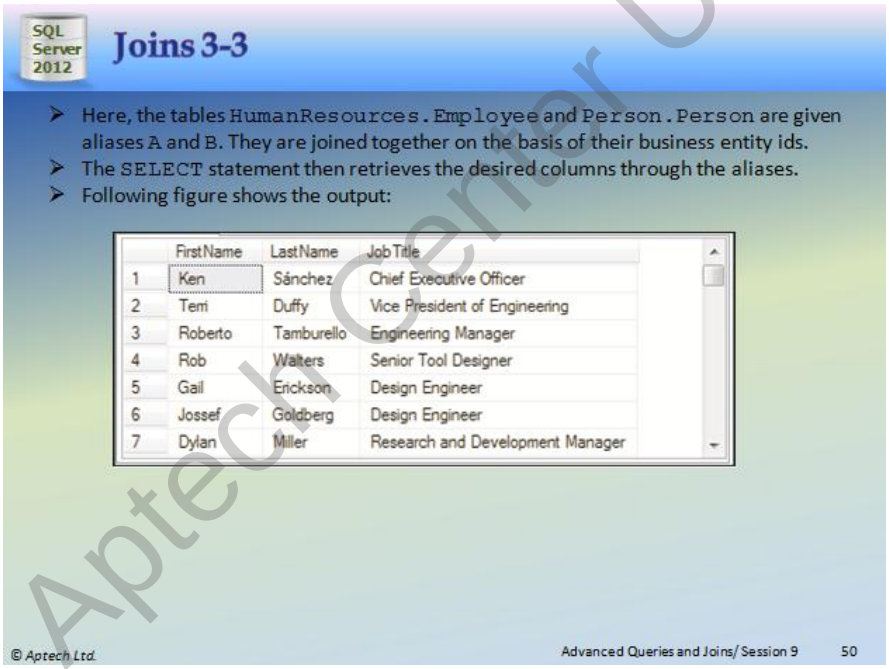
In the code snippet, the inner query retrieves contact type ids for all those persons whose contact information was modified on or before 2012. These results are then passed to the outer query, which matches these contact type ids with those in the Person.BusinessEntityContact table and displays the business entity IDs of those records. Figure shows part of the output.

Joins

Slides 48 to 50







Using slides 48 to 50, explain the JOIN statement.

Joins are used to retrieve data from two or more tables based on a logical relationship between tables. A join typically specifies foreign key relationship between the tables. It defines the manner in which two tables are related in a query by:

- Specifying the column from each table to be used for the join. A typical join specifies a foreign key from one table and its associated key in the other table.
- Specifying a logical operator such as =, <> to be used in comparing values from the columns.

Joins can be specified in either the FROM or WHERE clauses.
Explain the syntax of the JOIN statement.

Consider that you want to list employee first names, last names, and their job titles from the HumanResources.Employee and Person.Person. To extract this information from the two tables, you need to join them based on BusinessEntityID as shown in code snippet.
Here, the tables HumanResources.Employee and Person.Person are given aliases A and B. They are joined together on the basis of their business entity ids. The SELECT statement then retrieves the desired columns through the aliases.
Figure shows the output on slide 50.
There are three types of joins as follows:

- Inner Joins
- Outer Joins
- Self-Joins

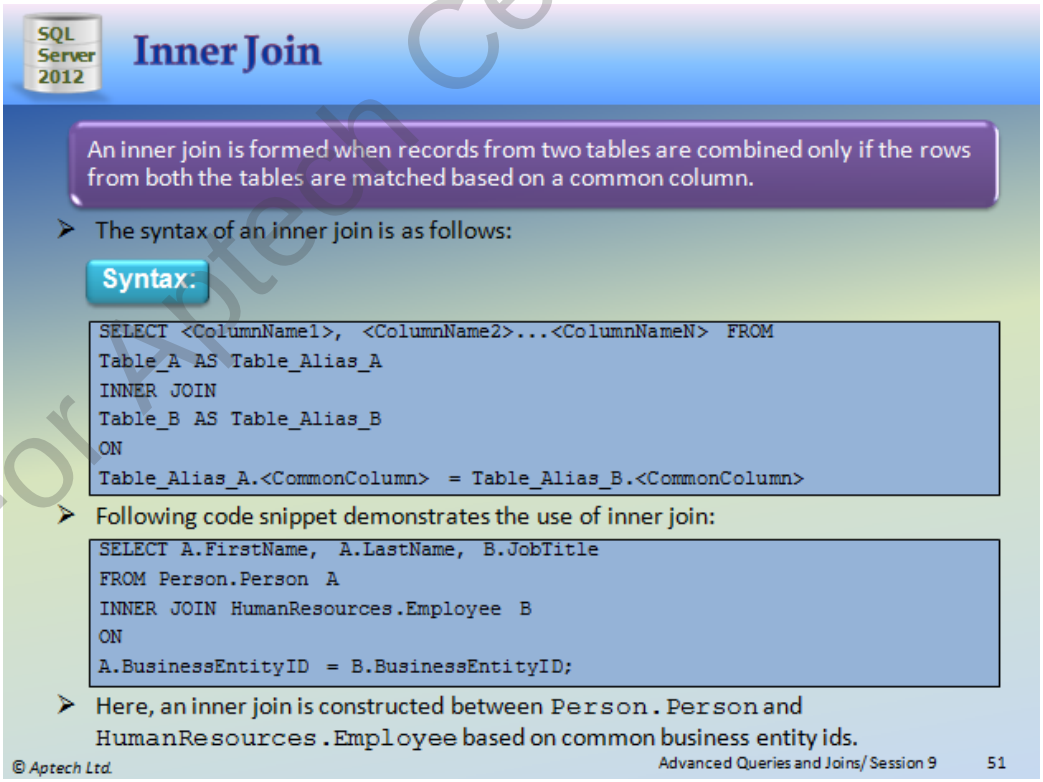
In-Class Question:

What is use of joins?

Answer:
Joins are used to retrieve data from two or more tables based on a logical relationship between tables.

Inner Join

Slide 51

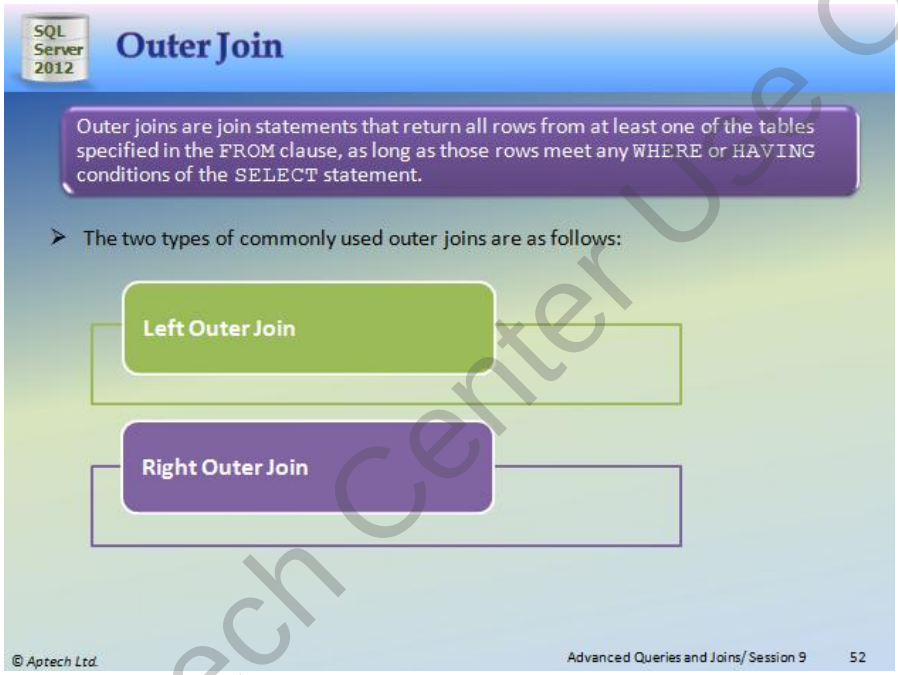


Using slide 51, explain the INNER JOIN.

An inner join is formed when records from two tables are combined only if the rows from both the tables are matched based on a common column. Explain the syntax of an inner join.
Explain the code snippet that demonstrates the use of inner join. The scenario for this is similar to code snippet. In the code snippet, an inner join is constructed between Person.Person and HumanResources.Employee based on common business entity ids. Here again, the two tables are given aliases of A and B respectively. The output is the same as shown in figure.

Outer Joins

Slide 52

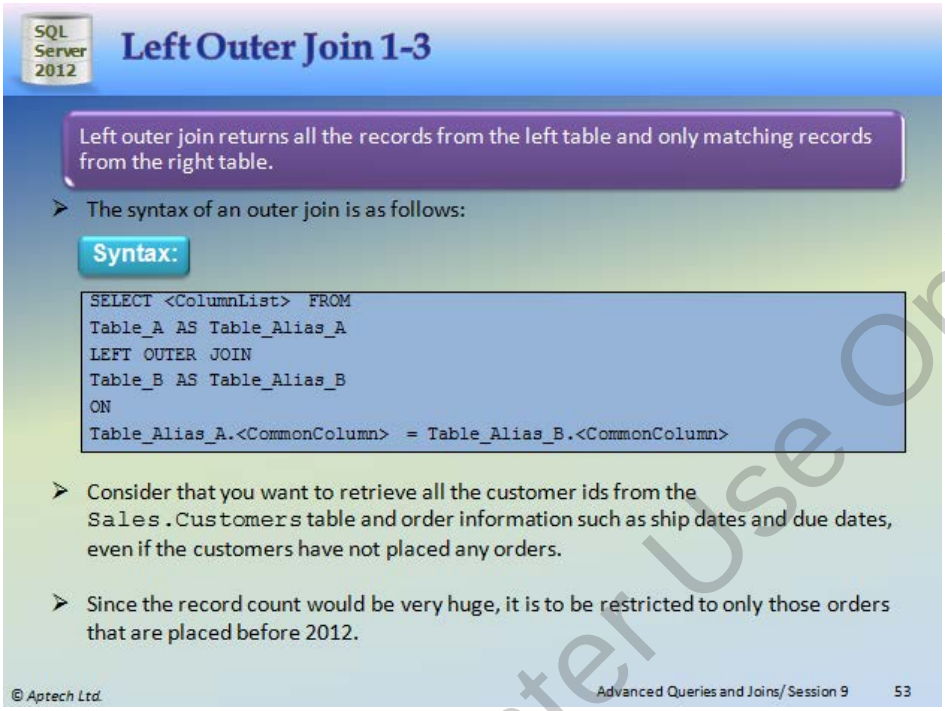


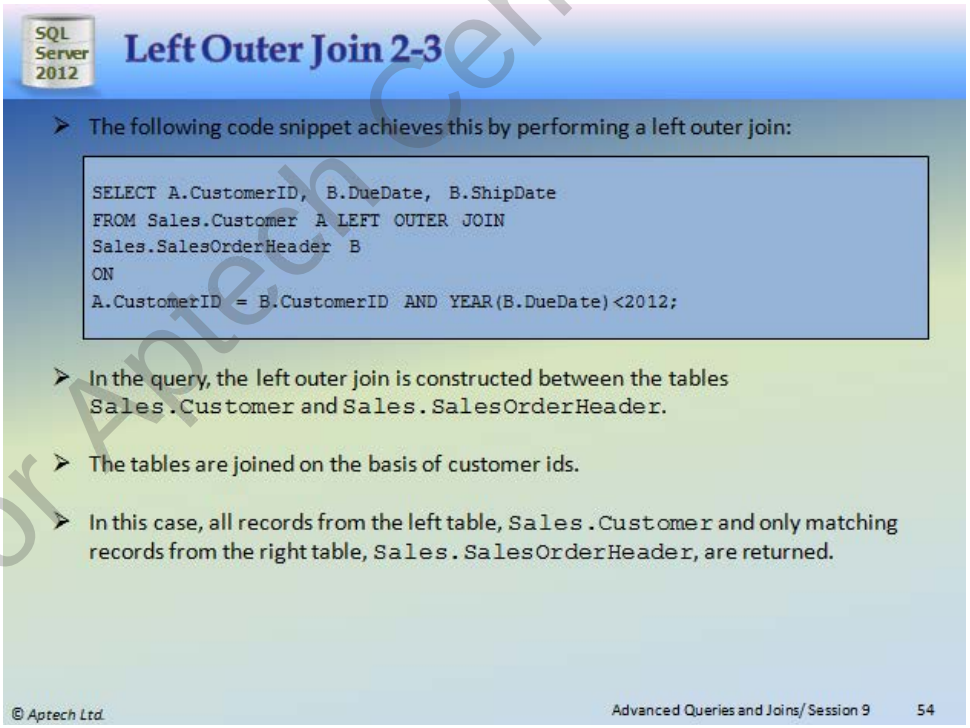
Using slide 52, explain the outer joins.
Outer joins are join statements that return all rows from at least one of the tables specified in the FROM clause, as long as those rows meet any WHERE or HAVING conditions of the SELECT statement. The two types of commonly used outer joins are as follows:

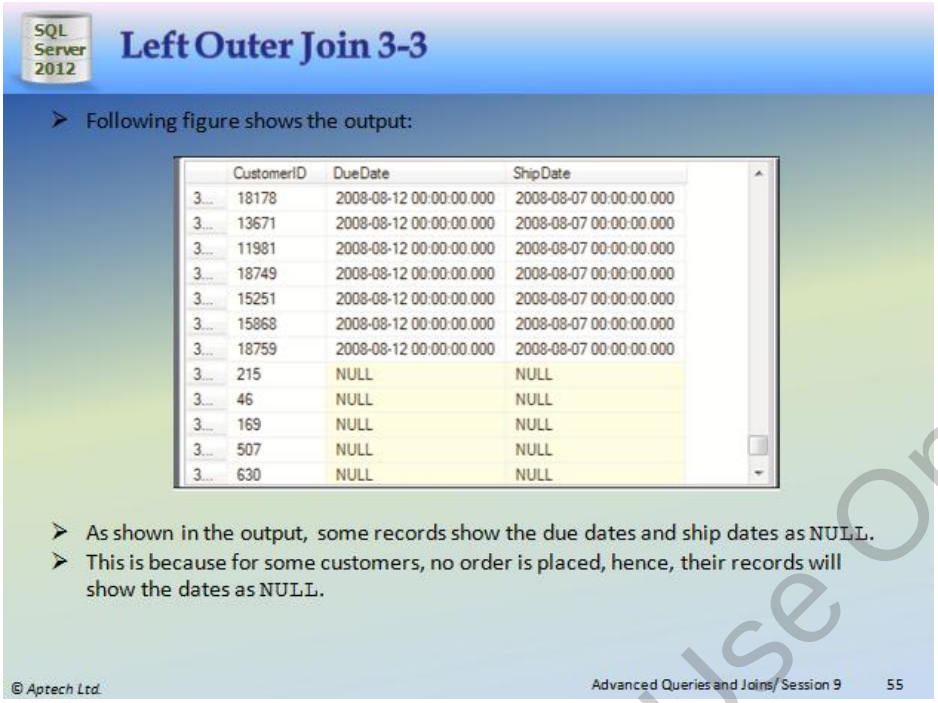
- Left Outer Join
- Right Outer Join

Left Outer Join

Slides 53 to 55







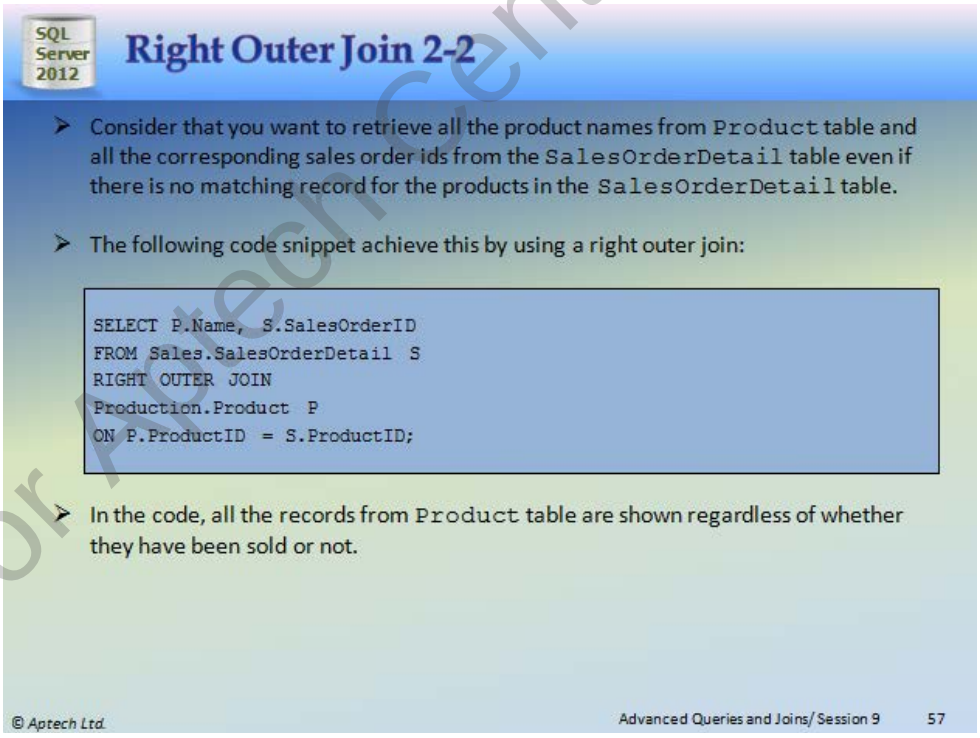
Using slides 53 to 55, explain the left outer join.

Left outer join returns all the records from the left table and only matching records from the right table. Explain the syntax of an outer join.
Consider that you want to retrieve all the customer ids from the Sales.Customers table and order information such as ship dates and due dates, even if the customers have not placed any orders. Since the record count would be very huge, it is to be restricted to only those orders that are placed before 2012. To achieve this, you perform a left outer join as shown in code snippet.

In the code snippet, the left outer join is constructed between the tables Sales.Customer and Sales.SalesOrderHeader. The tables are joined on the basis of customer ids. In this case, all records from the left table, Sales.Customer and only matching records from the right table, Sales.SalesOrderHeader, are returned. Figure shows the output.
As shown in the output, some records show the due dates and ship dates as NULL. This is because for some customers, no order is placed, hence, their records will show the dates as NULL.

Right Outer Join

Slides 56 and 57



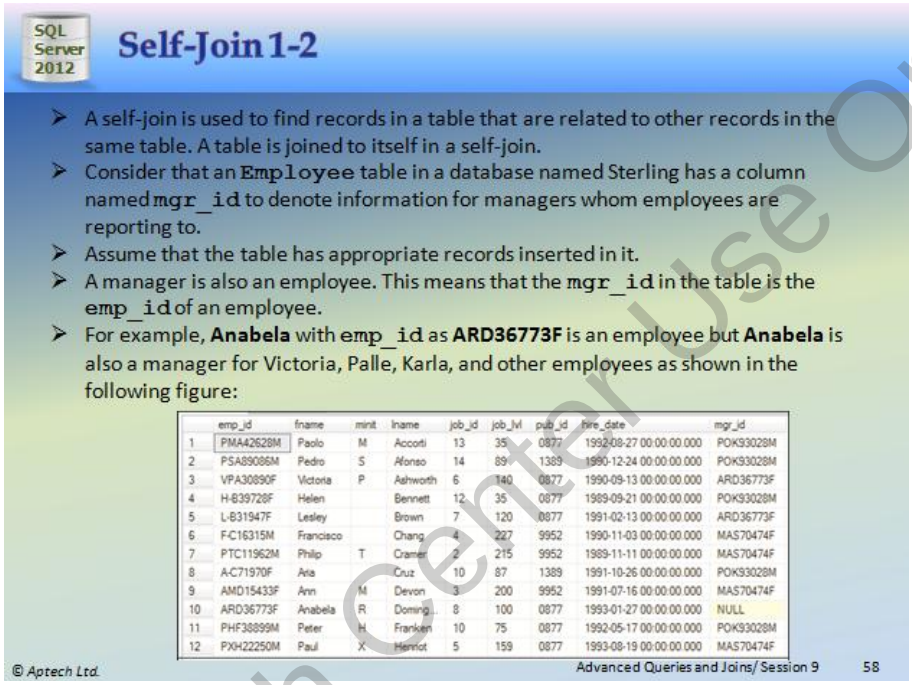
Using slides 56 and 57, explain the right outer join.

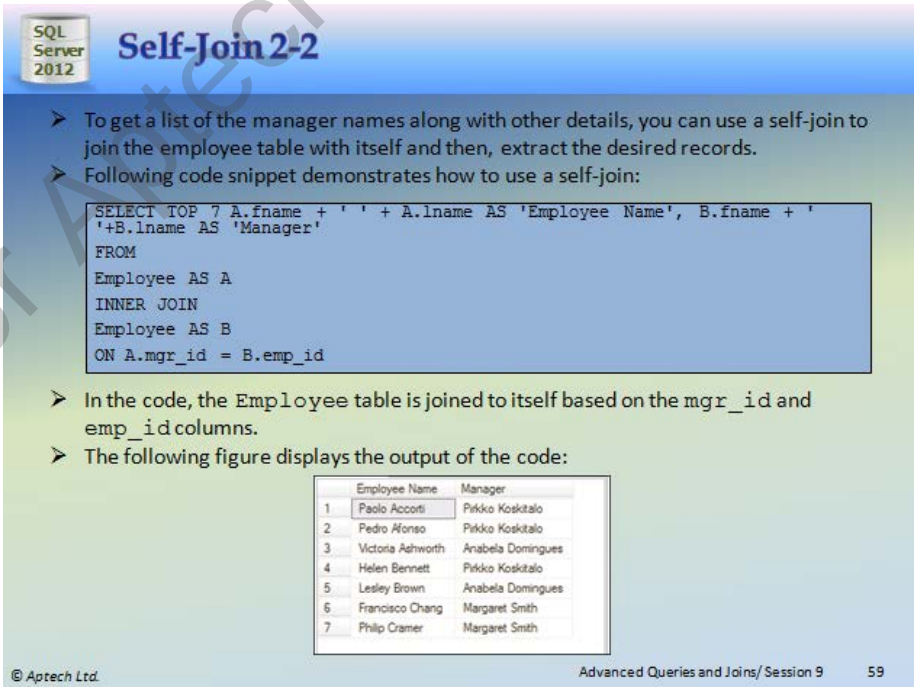
The right outer join retrieves all the records from the second table in the join regardless of whether there is matching data in the first table or not. Explain the syntax of a right outer join.

Consider that you want to retrieve all the product names from Product table and all the corresponding sales order ids from the SalesOrderDetail table even if there is no matching record for the products in the SalesOrderDetail table. To do this, you will use a right outer join as shown in code snippet.
In the code, all the records from Product table are shown regardless of whether they have been sold or not.

Self-Join

Slides 58 and 59





Using slides 58 and 59, explain the SELF JOIN.

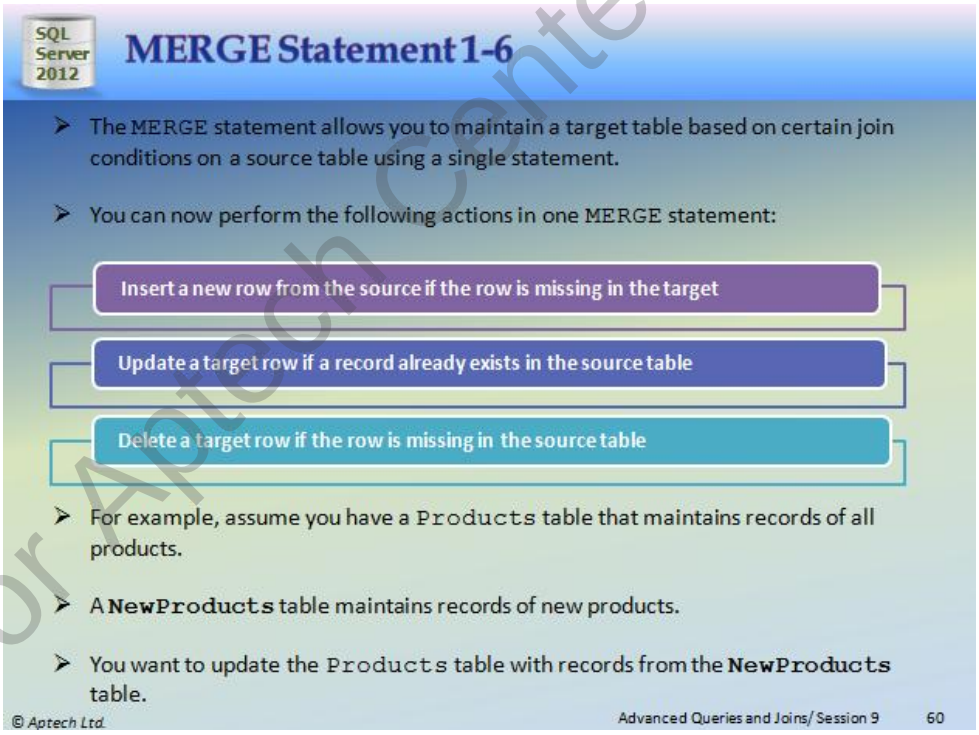
A self-join is used to find records in a table that are related to other records in the same table. A table is joined to itself in a self-join.
Consider that an Employee table in a database named, Sterling, has a column named, mgr_id to denote information for managers whom employees are reporting to. Assume that the table has appropriate records inserted in it.
A manager is also an employee. This means that the mgr_id in the table is the emp_id of an employee.

For example, Anabela with emp_id as ARD36773F is an employee but Anabela is also a manager for Victoria, Palle, Karla, and other employees as shown in figure.
To get a list of the manager names along with other details, you can use a self-join to join the employee table with itself and then, extract the desired records.
Explain the code snippet demonstrates how to use a self-join.

In the code snippet, the Employee table is joined to itself based on the mgr_id and emp_id columns.
The output of the code is shown in figure.

MERGE Statement

Slides 60 to 65



MERGE Statement 2-6

- Here, `NewProducts` table is the source table and `Products` is the target table.
- The `Products` table contains records of existing products with updated data and new products.
- Following figure shows the two tables:

Products

	ProductID	Name	Type	PurchaseDate
1	101	Rivets	Hardware	2012-12-01
2	102	Nuts	Hardware	2012-12-01
3	103	Washers	Hardware	2011-01-01
4	104	Rings	Hardware	2013-01-15
5	105	Paper Clips	Stationery	2013-01-01

NewProducts

	ProductID	Name	Type	PurchaseDate
1	102	Nuts	Hardware	2012-12-01
2	103	Washers	Hardware	2011-01-01
3	107	Rings	Hardware	2013-01-15
4	108	Paper Clips	Stationery	2013-01-01

MERGE Statement 3-6

- Consider that you want to:
  - Compare last and first names of customers from both source and target tables
  - Update customer information in target table if the last and first names match
  - Insert new records in target table if the last and first names in source table do not exist in target table
  - Delete existing records in target table if the last and first names do not match with those of source table
- `MERGE` also allows you to optionally display those records that were inserted, updated, or deleted by using an `OUTPUT` clause.
- The syntax of the `MERGE` statement is as follows:

```
MERGE target_table
USING source_table
ON match_condition
WHEN MATCHED THEN UPDATE SET Col1 = val1 [, Col2 = val2...]
WHEN [TARGET] NOT MATCHED THEN INSERT (Col1 [,Col2...] VALUES (Val1 [,
Val2...])
WHEN NOT MATCHED BY SOURCE THEN DELETE
[OUTPUT $action, Inserted.Col1, Deleted.Col1,...] ;
```

where,

`target_table`: is the table `WHERE` changes are being made.

MERGE Statement 4-6

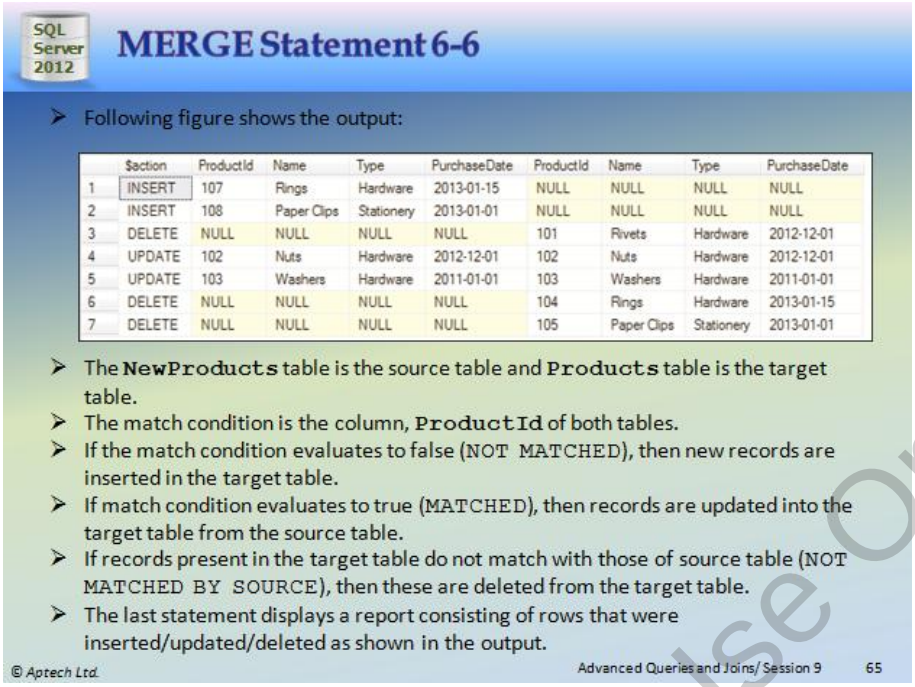
`source_table`: is the table from which rows will be inserted, updated, or deleted into the target table.
`match_conditions`: are the JOIN conditions and any other comparison operators.
`MATCHED`: true if a row in the `target_table` and `source_table` matches the `match_condition`.
`NOT MATCHED`: true if a row from the `source_table` does not exist in the `target_table`.
`SOURCE NOT MATCHED`: true if a row exists in the `target_table` but not in the `source_table`.
`OUTPUT`: An optional clause that allows to view those records that have been inserted/deleted/updated in `target_table`.

- MERGE statements are terminated with a semi-colon (;).

MERGE Statement 5-6

- Following code snippet shows how to use MERGE statement. It makes use of the **`Sterling`** database:

```
MERGE INTO Products AS P1
USING
NewProducts AS P2
ON P1.ProductId = P2.ProductId
WHEN MATCHED THEN
UPDATE SET
P1.Name = P2.Name,
P1.Type = P2.Type,
P1.PurchaseDate = P2.PurchaseDate
WHEN NOT MATCHED THEN
INSERT (ProductId, Name, Type, PurchaseDate)
VALUES (P2.ProductId, P2.Name, P2.Type, P2.PurchaseDate)
WHEN NOT MATCHED BY SOURCE THEN
DELETE
OUTPUT $action, Inserted.ProductId, Inserted.Name, Inserted.Type,
Inserted.PurchaseDate, Deleted.ProductId,Deleted.Name, Deleted.Type,
Deleted.PurchaseDate;
```



	$action	ProductId	Name	Type	PurchaseDate	ProductId	Name	Type	PurchaseDate
1	INSERT	107	Rings	Hardware	2013-01-15	NULL	NULL	NULL	NULL
2	INSERT	108	Paper Clips	Stationery	2013-01-01	NULL	NULL	NULL	NULL
3	DELETE	NULL	NULL	NULL	NULL	101	Rivets	Hardware	2012-12-01
4	UPDATE	102	Nuts	Hardware	2012-12-01	102	Nuts	Hardware	2012-12-01
5	UPDATE	103	Washers	Hardware	2011-01-01	103	Washers	Hardware	2011-01-01
6	DELETE	NULL	NULL	NULL	NULL	104	Rings	Hardware	2013-01-15
7	DELETE	NULL	NULL	NULL	NULL	105	Paper Clips	Stationery	2013-01-01

Using slides 60 to 65, explain the MERGE statement.

The MERGE statement allows you to maintain a target table based on certain join conditions on a source table using a single statement. You can now perform the following actions in one MERGE statement:

- Insert a new row from the source if the row is missing in the target
- Update a target row if a record already exists in the source table
- Delete a target row if the row is missing in the source table

For example, assume you have a Products table that maintains records of all products. A NewProducts table maintains records of new products. You want to update the Products table with records from the NewProducts table. Here, NewProducts table is the source table and Products is the target table. The Products table contains records of existing products with updated data and new products. Figure shows the two tables.

Consider that you want to:

- Compare last and first names of customers from both source and target tables
- Update customer information in target table if the last and first names match
- Insert new records in target table if the last and first names in source table do not exist in target table
- Delete existing records in target table if the last and first names do not match with those of source table

The MERGE statement accomplishes the tasks in a single statement. MERGE also allows you to optionally display those records that were inserted, updated, or deleted by using an OUTPUT clause.

Explain the syntax of the MERGE statement on slide 62.
MERGE statements are terminated with a semi-colon (;).

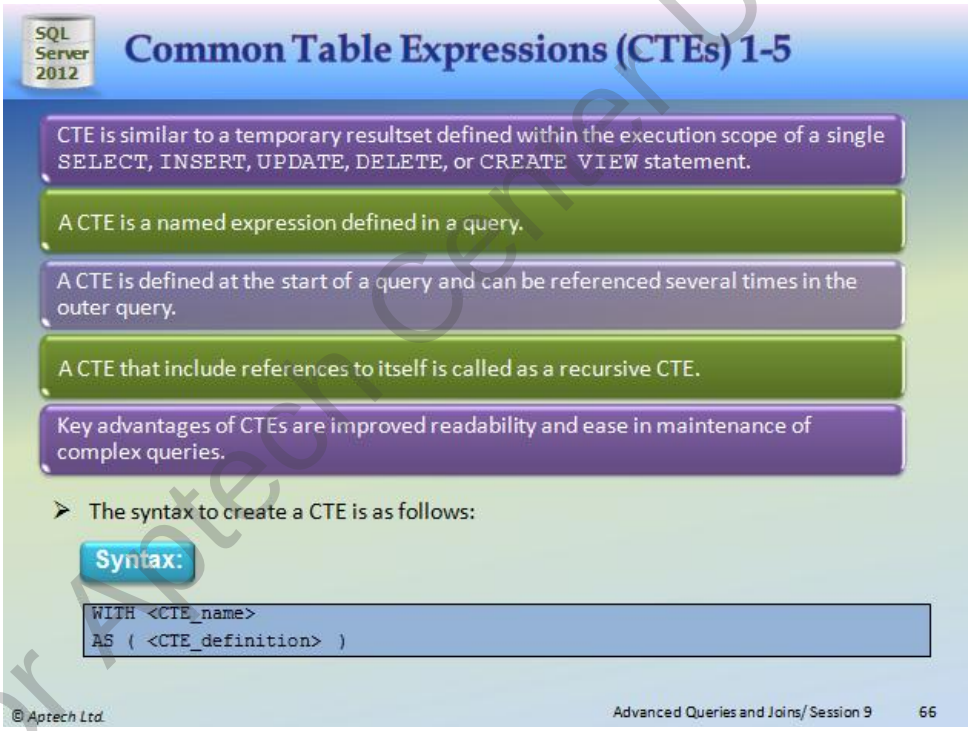
Explain the code snippet which shows how to use MERGE statement. It makes use of the Sterling database. Figure shows the output on slide 65.

The NewProducts table is the source table and Products table is the target table. The match condition is the column, ProductId of both tables. If the match condition evaluates to false (NOT MATCHED), then new records are inserted in the target table.
If match condition evaluates to true (MATCHED), then records are updated into the target table from the source table.

If records present in the target table do not match with those of source table (NOT MATCHED BY SOURCE), then these are deleted from the target table. The last statement displays a report consisting of rows that were inserted/updated/deleted as shown in the output.

Common Table Expressions

Slides 66 to 70



Common Table Expressions (CTEs) 2-5

- The following code snippet retrieves and displays the customer count year-wise for orders present in the `Sales.SalesOrderHeader` table:

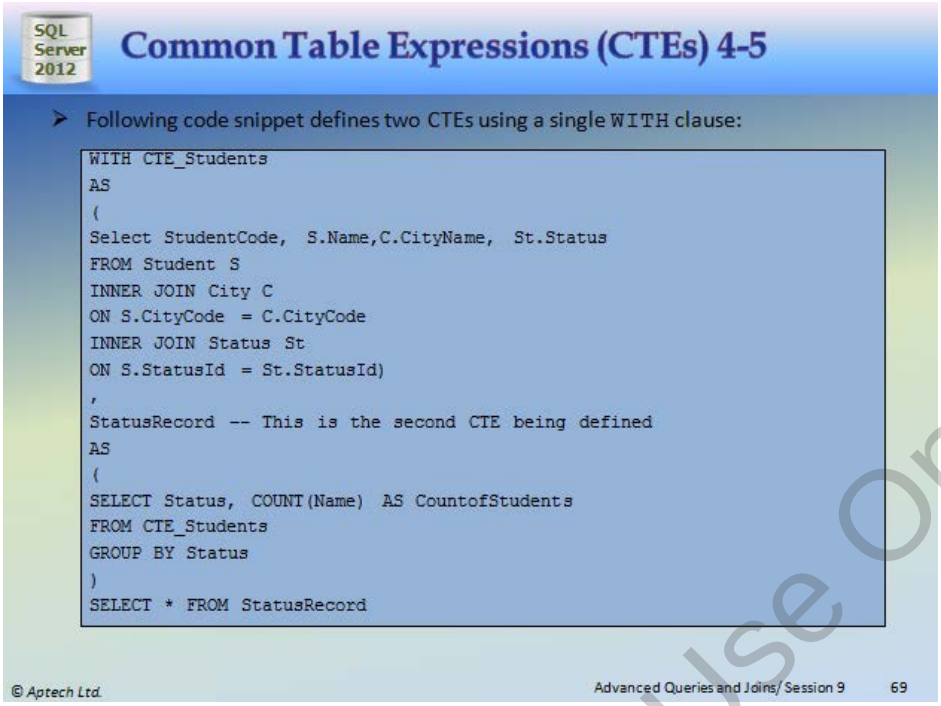
```
WITH CTE_OrderYear
AS
(
SELECT YEAR(OrderDate) AS OrderYear, CustomerID
FROM Sales.SalesOrderHeader
)
SELECT OrderYear, COUNT(DISTINCT CustomerID) AS CustomerCount
FROM CTE_OrderYear
GROUP BY OrderYear;
```

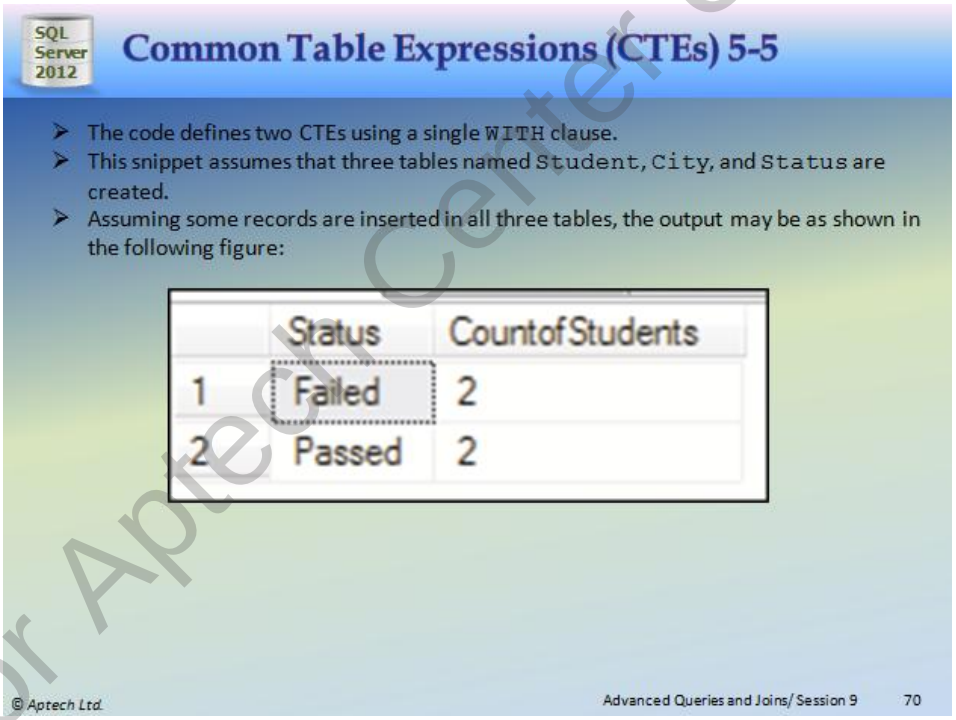
- Here, **`CTE_OrderYear`** is specified as the CTE name.
- The `WITH...AS` keywords begins the CTE definition.
- Then, the CTE is used in the `SELECT` statement to retrieve and display the desired results. Following figure shows the output:

	OrderYear	CustomerCount
1	2007	9864
2	2008	11844
3	2005	1216
4	2006	3094

Common Table Expressions (CTEs) 3-5

- The following guidelines need to be remembered while defining CTEs:
  - CTEs are limited in scope to the execution of the outer query. Hence, when the outer query ends, the lifetime of the CTE will end.
  - You need to define a name for a CTE and also, define unique names for each of the columns referenced in the SELECT clause of the CTE.
  - It is possible to use inline or external aliases for columns in CTEs.
  - A single CTE can be referenced multiple times in the same query with one definition.
  - Multiple CTEs can also be defined in the same WITH clause.





Using slides 66 to 70, explain the Common Table Expressions.

A Common Table Expression (CTE) is similar to a temporary resultset defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement. A CTE is a named expression defined in a query. A CTE is defined at the start of a query and can be referenced several times in the outer query. A CTE that include references to itself is called as a recursive CTE.

Key advantages of CTEs are improved readability and ease in maintenance of complex queries. Explain the syntax to create a CTE.

For example, to retrieve and display the customer count year-wise for orders present in the Sales.SalesOrderHeader table, the code will be as given in code snippet.
Here, CTE_OrderYear is specified as the CTE name. The WITH...AS keywords begins the CTE definition. Then, the CTE is used in the SELECT statement to retrieve and display the desired results. Figure shows the output.

The following guidelines need to be remembered while defining CTEs:

- CTEs are limited in scope to the execution of the outer query. Hence, when the outer query ends, the lifetime of the CTE will end.
- You need to define a name for a CTE and also, define unique names for each of the columns referenced in the SELECT clause of the CTE.
- It is possible to use inline or external aliases for columns in CTEs.
- A single CTE can be referenced multiple times in the same query with one definition.
- Multiple CTEs can also be defined in the same WITH clause. For example, consider the code snippet that defines two CTEs using a single WITH clause. This snippet assumes that three tables named Student, City, and Status are created.

Assuming some records are inserted in all three tables, the output may be as shown in figure on slide 70. Mention temporary tables in the SQL server starts with # symbol.

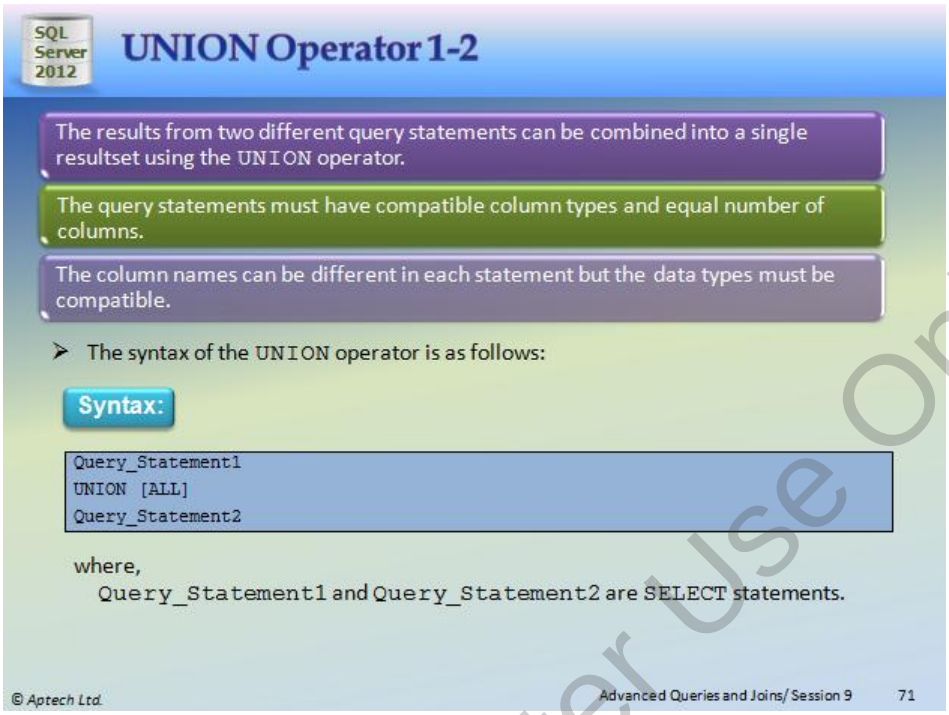
In-Class Question:

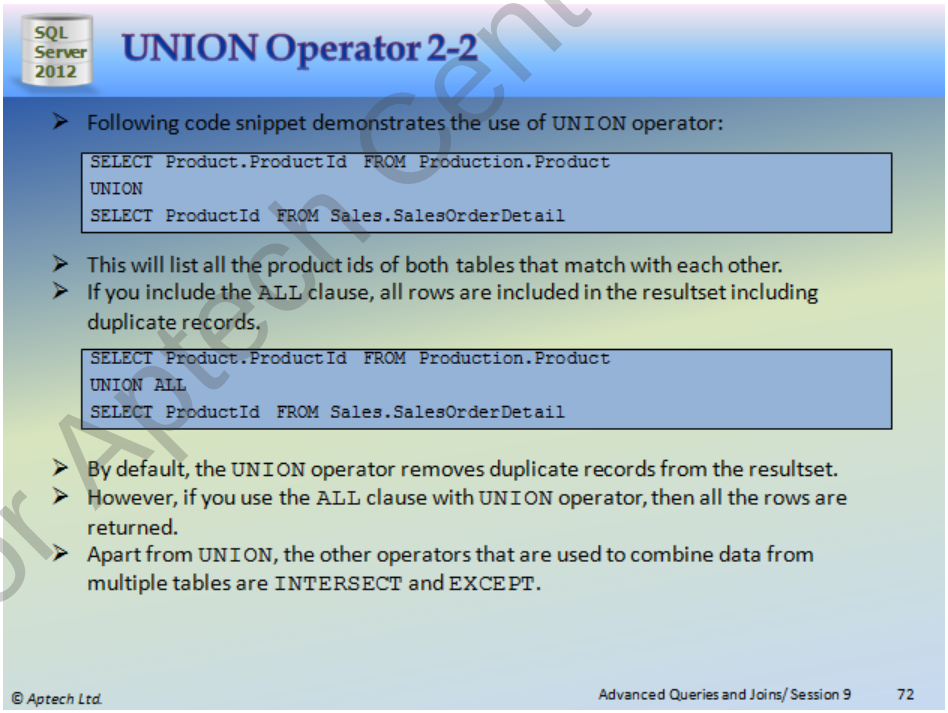
 What is a CTE?

Answer:
A Common Table Expression (CTE) is similar to a temporary resultset defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement.

UNION Operator

Slides 71 and 72





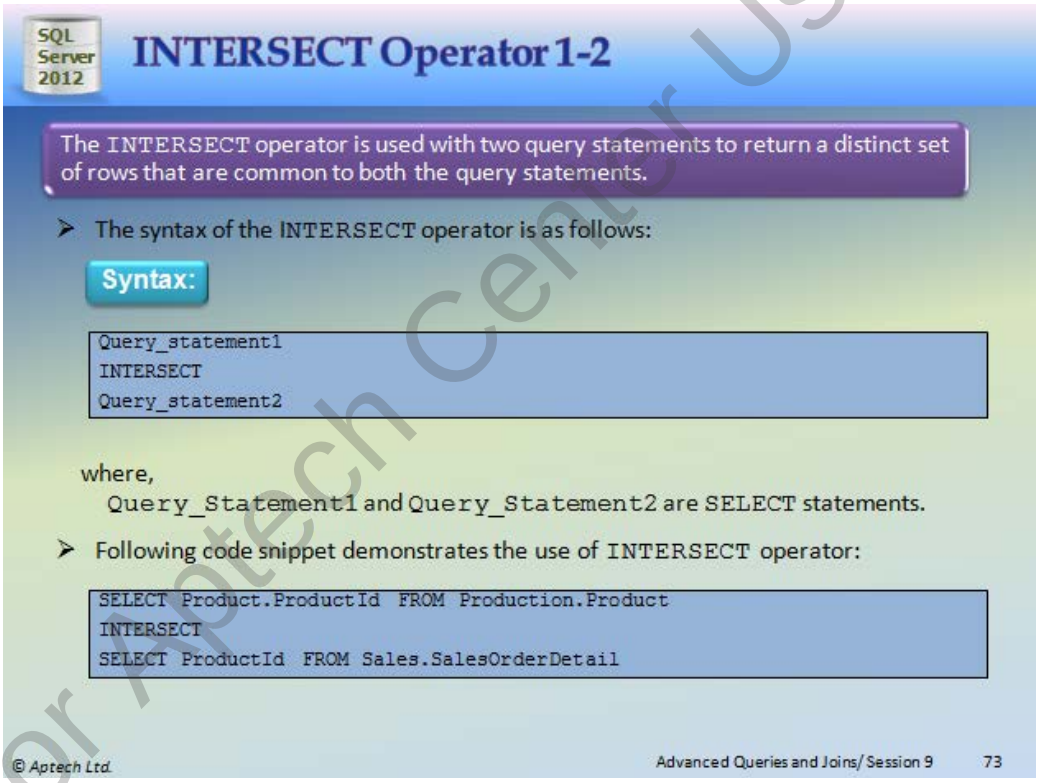
Using slides 71 and 72, explain the UNION operator.

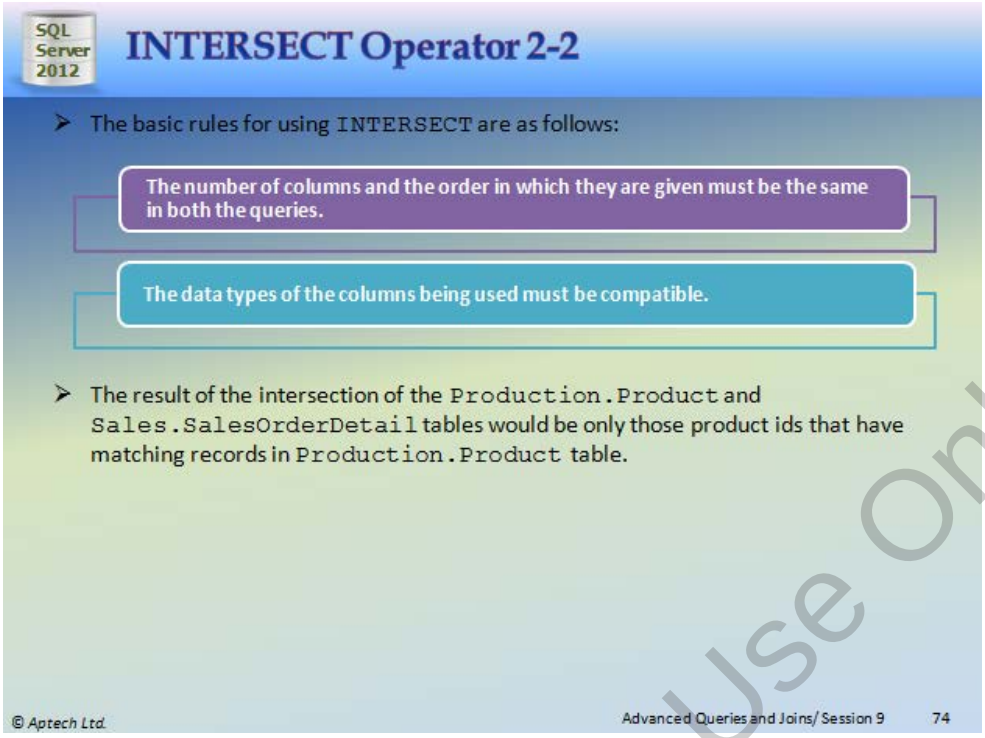
The results from two different query statements can be combined into a single resultset using the UNION operator. The query statements must have compatible column types and equal number of columns. The column names can be different in each statement but the data types must be compatible. By compatible data types, it means that it should be possible to convert the contents of one of the columns into another. For example, if one of

the query statements has an int data type and the other query statement has a money data type, they are compatible and a union can take place between them because the int data can be converted into money data.
Explain the syntax of the UNION operator.
Explain the code snippet which demonstrates the UNION operator.
This will list all the product ids of both tables that match with each other. If you include the ALL clause, all rows are included in the resultset including duplicate records.
Explain the code snippet which demonstrates the UNION ALL operator.
By default, the UNION operator removes duplicate records from the resultset. However, if you use the ALL clause with UNION operator, then all the rows are returned. Apart from UNION, the other operators that are used to combine data from multiple tables are INTERSECT and EXCEPT.

INTERSECT Operator

Slides 73 and 74





Using slides 73 and 74, explain the INTERSECT operator.

Consider again the two tables, Product and SalesOrderDetail present in AdventureWorks2012. Suppose you want to display only the rows that are common to both the tables. To do this, you will need to use an operator named, INTERSECT. The INTERSECT operator is used with two query statements to return a distinct set of rows that are common to both the query statements. Explain the syntax of the INTERSECT operator.

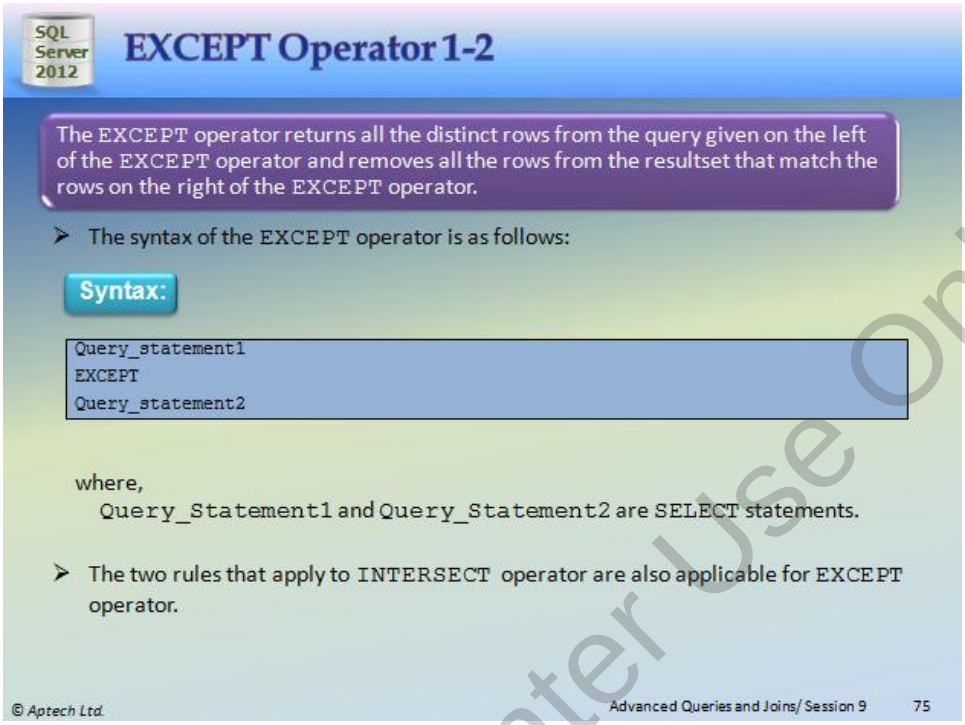
Explain the code snippet which demonstrates the INTERSECT operator.
The basic rules for using INTERSECT are as follows:

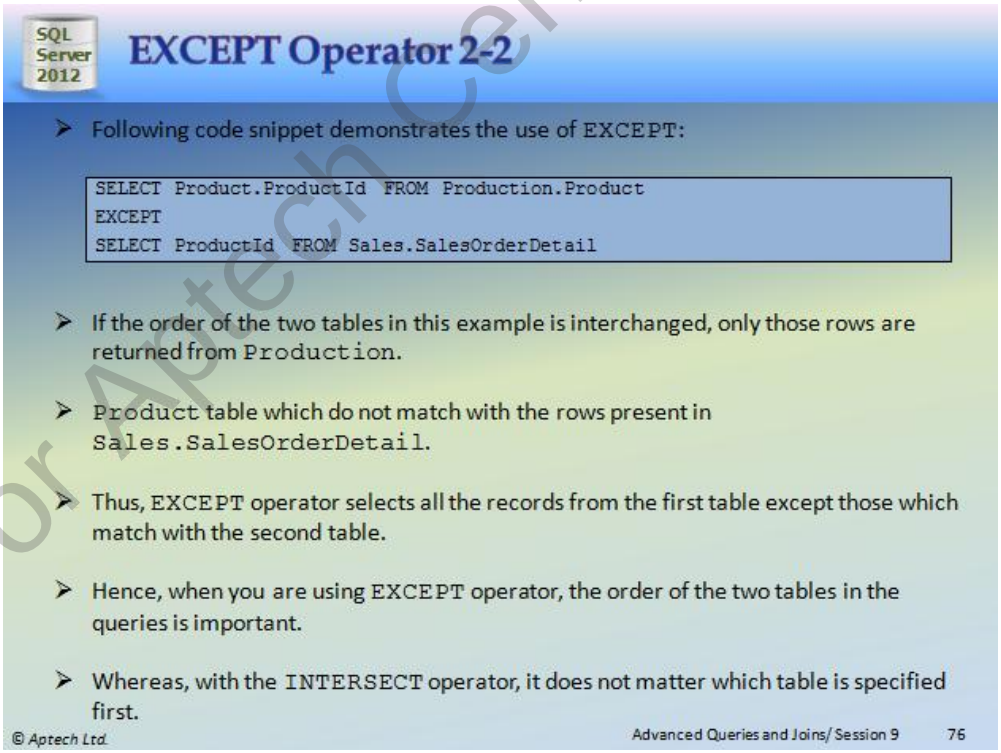
- The number of columns and the order in which they are given must be the same in both the queries.
- The data types of the columns being used must be compatible.

The result of the intersection of the Production.Product and Sales.SalesOrderDetail tables would be only those product ids that have matching records in Production.Product table. In large enterprises, there are huge volumes of data stored in databases. Instead of storing the entire data in a single table, it can be spread over several tables that are related to one another. When data is stored in such tables, there must be some means to combine and retrieve the data from those tables. Using SQL Server, there are a number of ways to combine data from multiple tables. The following sections explore these ways in detail.

EXCEPT Operator

Slides 75 and 76





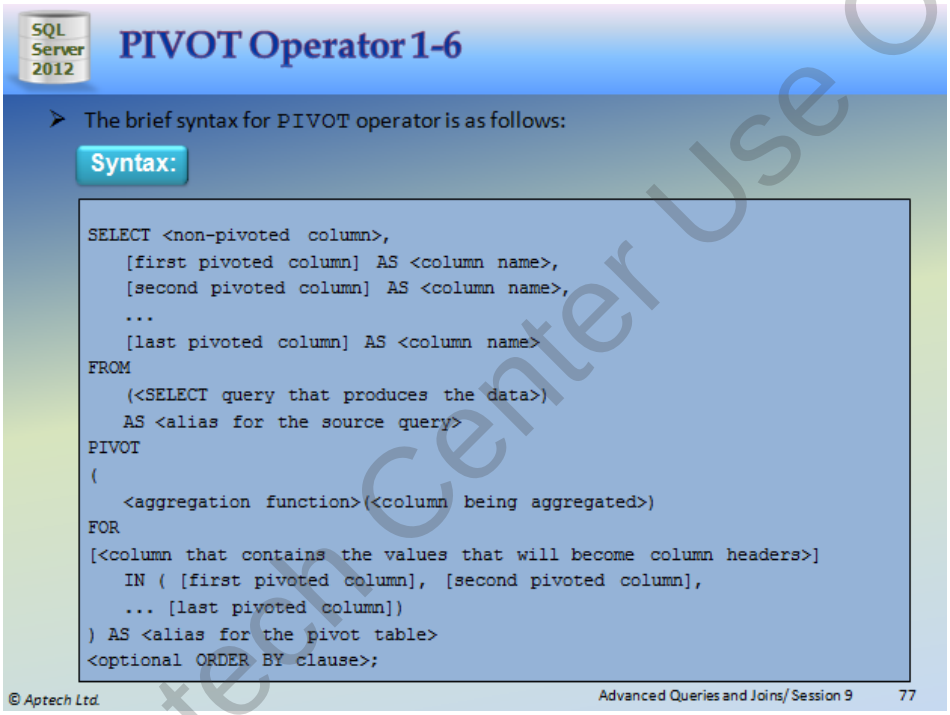
Using slides 75 and 76, explain EXCEPT operator.

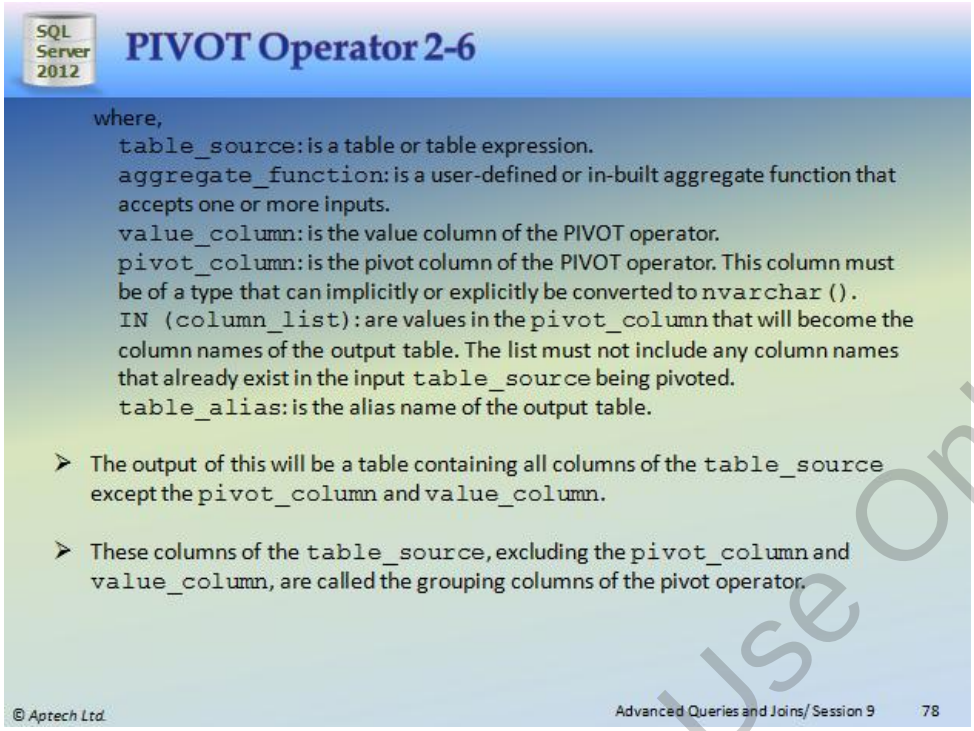
The EXCEPT operator returns all of the distinct rows from the query given on the left of the EXCEPT operator and removes all the rows from the resultset that match the rows on the right of the EXCEPT operator. Explain the syntax of the EXCEPT operator.

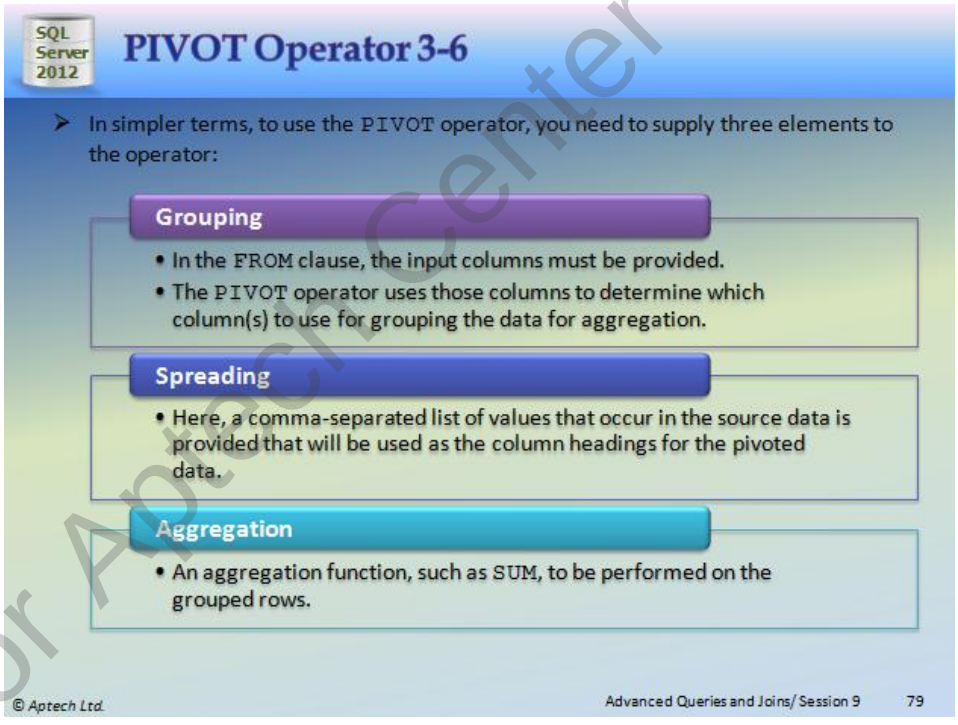
The two rules that apply to INTERSECT operator are also applicable for EXCEPT operator.
Explain the code snippet which demonstrates the EXCEPT operator.
If the order of the two tables in this example is interchanged, only those rows are returned from Production.Product table which do not match with the rows present in Sales.SalesOrderDetail. Thus, in simple terms, EXCEPT operator selects all the records from the first table except those which match with the second table. Hence, when you are using EXCEPT operator, the order of the two tables in the queries is important. Whereas, with the INTERSECT operator, it does not matter which table is specified first.

PIVOT Operator

Slides 77 to 82







PIVOT Operator 4-6

- Following code snippet is shown without the PIVOT operator and demonstrates a simple GROUP BY aggregation.
- As the number of records would be huge, the resultset is limited to 5 by specifying TOP 5.

```
SELECT TOP 5 SUM(SalesYTD) AS TotalSalesYTD, Name
FROM Sales.SalesTerritory
GROUP BY Name
```

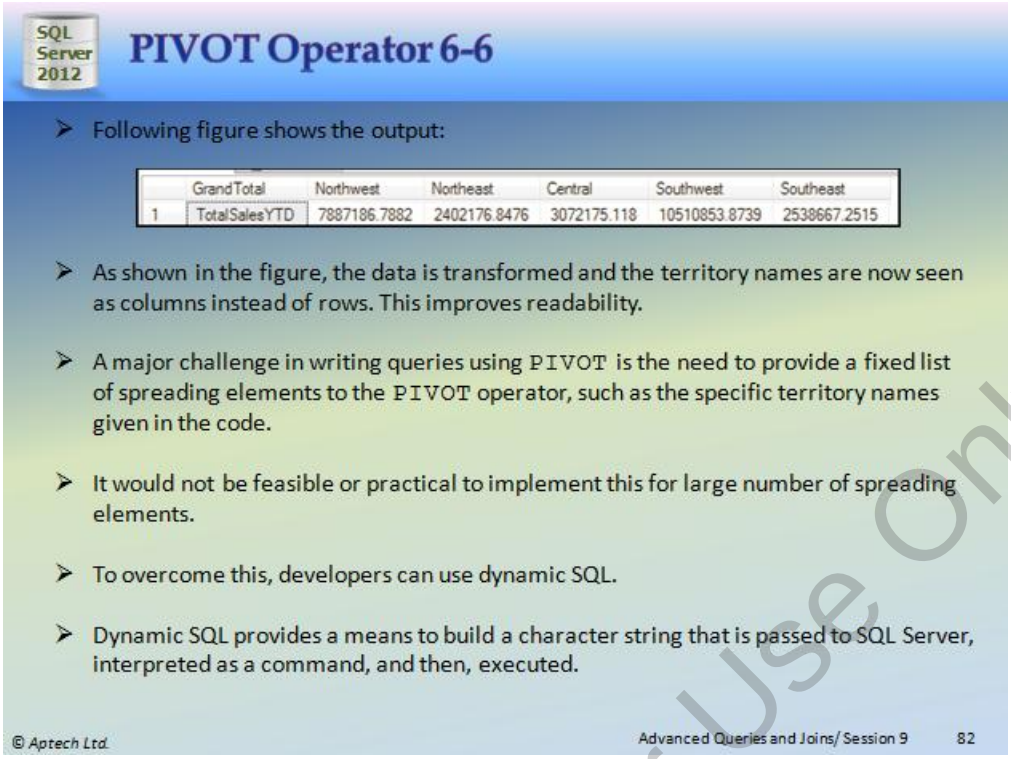
- Following figure shows the output:

	TotalSalesYTD	Name
1	7887186.7882	Northwest
2	2402176.8476	Northeast
3	3072175.118	Central
4	10510853.8739	Southwest
5	2538667.2515	Southeast

PIVOT Operator 5-6

- The top 5 year to date sales along with territory names grouped by territory names are displayed.
- The same query is rewritten in the following code snippet using a PIVOT so that the data is transformed from a row-based orientation to a column-based orientation:

```
-- Pivot table with one row and six columns
SELECT TOP 5 'TotalSalesYTD' AS GrandTotal,
[Northwest], [Northeast], [Central], [Southwest],[Southeast]
FROM
(SELECT TOP 5 Name, SalesYTD
FROM Sales.SalesTerritory
) AS SourceTable
PIVOT
(
SUM(SalesYTD)
FOR Name IN ([Northwest], [Northeast], [Central], [Southwest],
[Southeast])
) AS PivotTable;
```



Using slides 77 to 82, explain the PIVOT operator.

Consider a scenario where data needs to be displayed in a different orientation than it is stored in, in terms of row and column layout. The process of transforming data from a row-based orientation to a column-based orientation is called pivoting. The PIVOT and UNPIVOT operators of SQL Server help to change the orientation of data from column-oriented to row-oriented and vice versa. This is accomplished by consolidating values present in a column to a list of distinct values and then projecting that list in the form of column headings. Explain the syntax for PIVOT.
The output of this will be a table containing all columns of the table_source except the pivot_column and value_column. These columns of the table_source, excluding the pivot_column and value_column, are called the grouping columns of the pivot operator. In simpler terms, to use the PIVOT operator, you need to supply three elements to the operator:

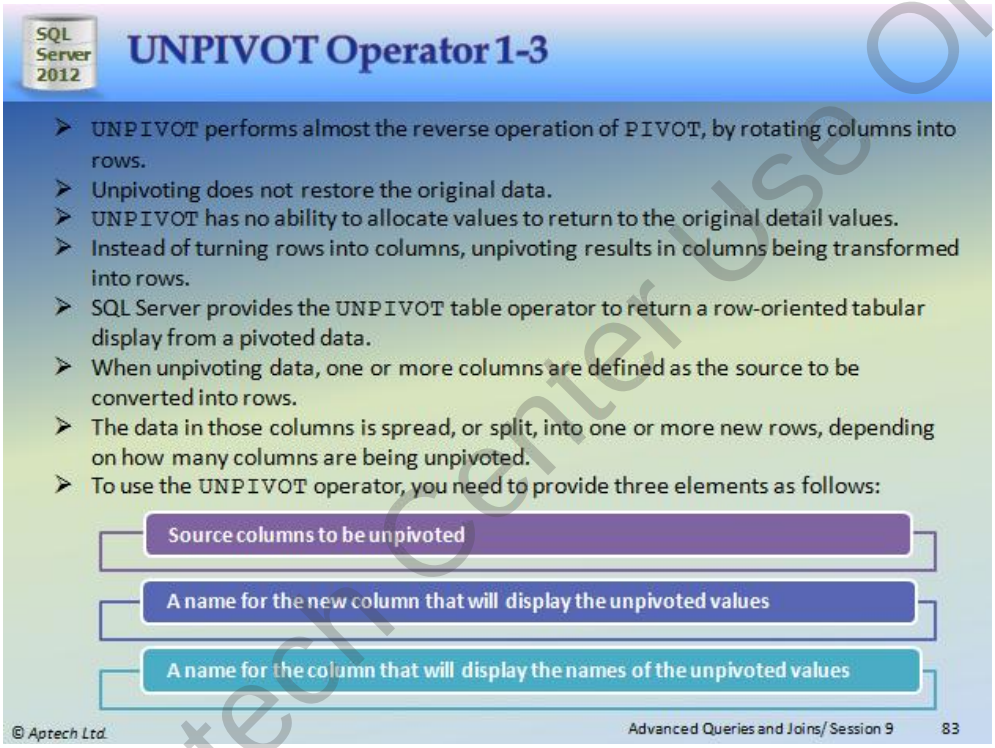
- **Grouping**: In the FROM clause, the input columns must be provided. The PIVOT operator uses those columns to determine which column(s) to use for grouping the data for aggregation.
- **Spreading**: Here, a comma-separated list of values that occur in the source data is provided that will be used as the column headings for the pivoted data.
- **Aggregation**: An aggregation function, such as SUM, to be performed on the grouped rows. Consider an example to understand the PIVOT operator. Code snippet is shown without the PIVOT operator and demonstrates a simple GROUP BY aggregation. As the number of records would be huge, the resultset is limited to 5 by specifying TOP 5.

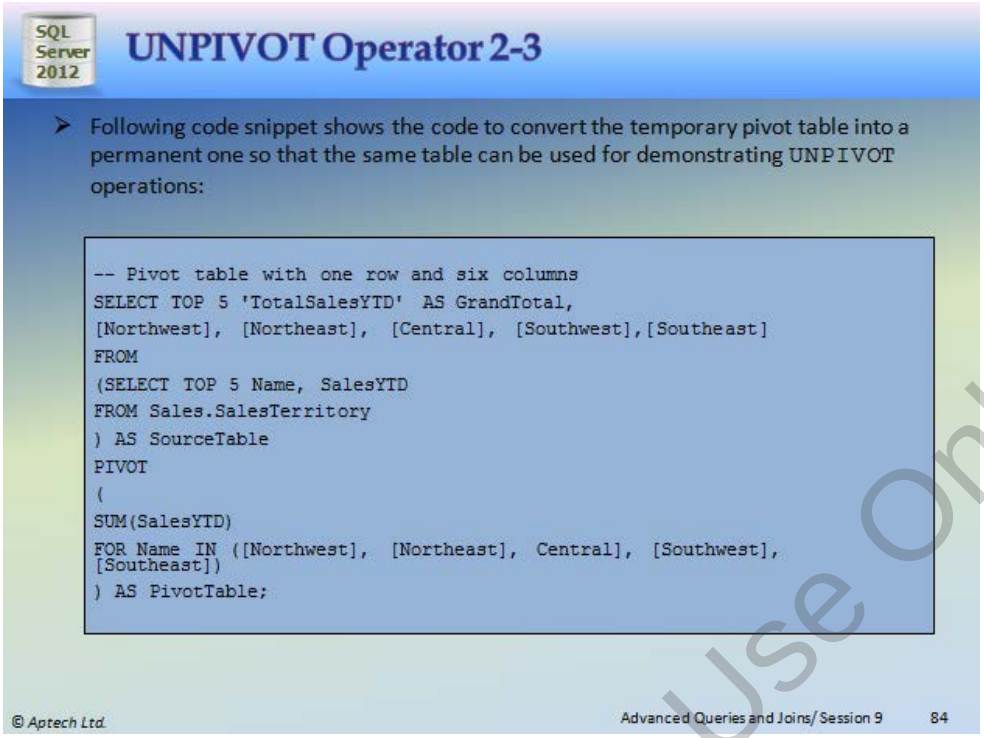
The top 5 year to date sales along with territory names grouped by territory names are displayed. Now, the same query is rewritten in code snippet using a PIVOT so that the data is transformed from a row-based orientation to a column-based orientation. Figure shows the output.

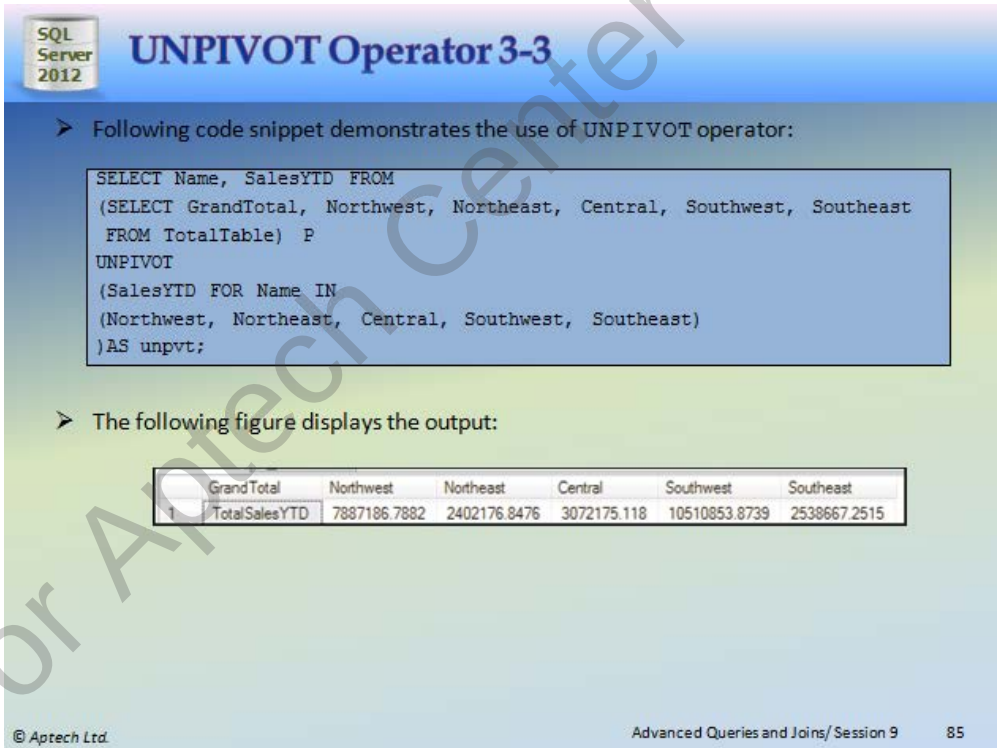
As shown in figure on slide 82, the data is transformed and the territory names are now seen as columns instead of rows. This improves readability. A major challenge in writing queries using PIVOT is the need to provide a fixed list of spreading elements to the PIVOT operator, such as the specific territory names given in code snippet. It would not be feasible or practical to implement this for large number of spreading elements. To overcome this, developers can use dynamic SQL. Dynamic SQL provides a means to build a character string that is passed to SQL Server, interpreted as a command and then, executed.

UNPIVOT Operator

Slides 83 to 85







Using slides 83 to 85, explain the UNPIVOT operator.

UNPIVOT performs almost the reverse operation of PIVOT, by rotating columns into rows. Unpivoting does not restore the original data. Detail-level data was lost during the aggregation process in the original pivot. UNPIVOT has no ability to allocate values to return to the original detail values. Instead of turning rows into columns, unpivoting results in columns being transformed into rows. SQL Server provides the UNPIVOT table operator to return a row-oriented tabular display from a pivoted data.

When unpivoting data, one or more columns are defined as the source to be converted into rows. The data in those columns is spread, or split, into one or more new rows, depending on how many columns are being unpivoted. To use the UNPIVOT operator, you need to provide three elements as follows:

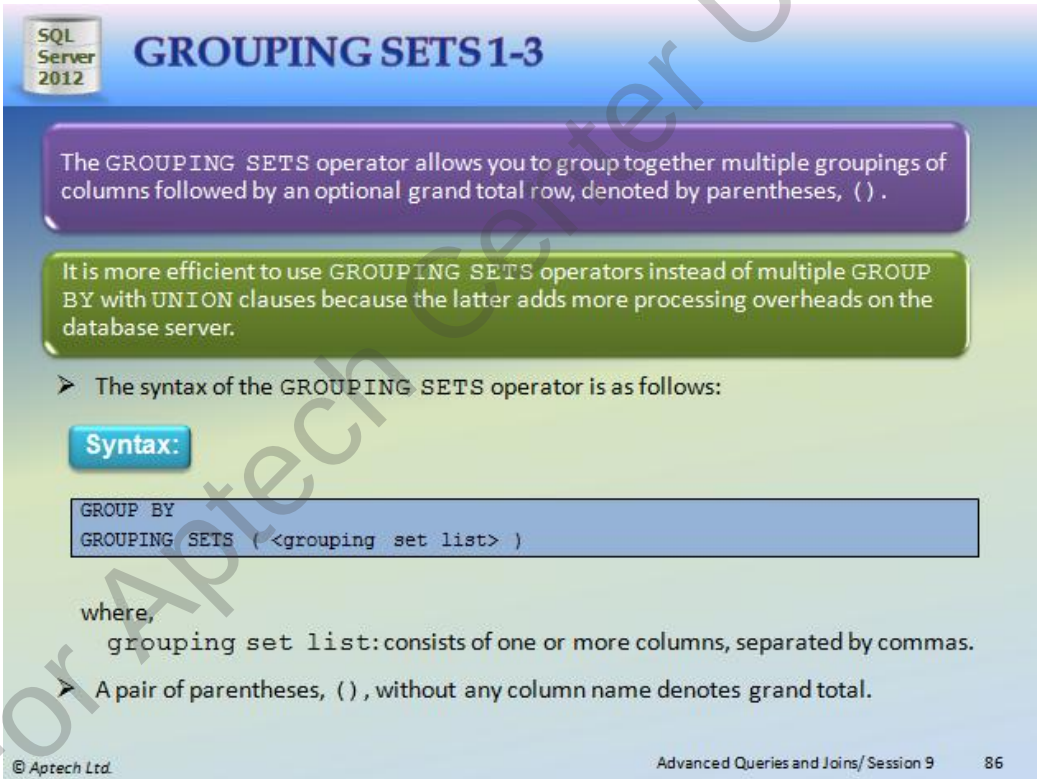
- Source columns to be unpivoted
- A name for the new column that will display the unpivoted values
- A name for the column that will display the names of the unpivoted values

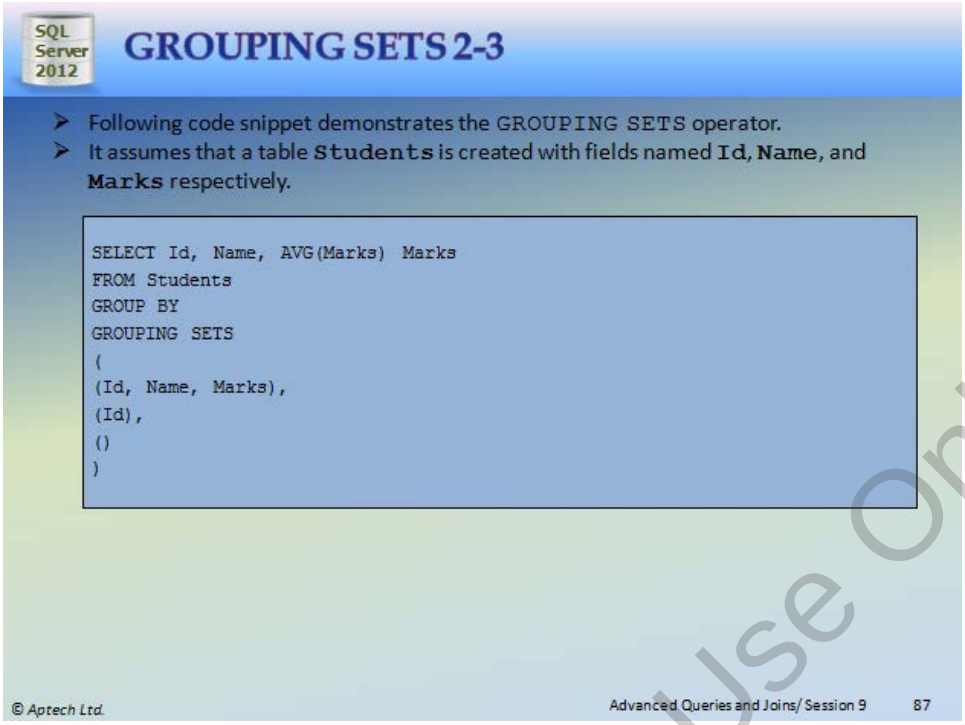
Consider the earlier scenario. Explain the code snippet which shows the code to convert the temporary pivot table into a permanent one so that the same table can be used for demonstrating UNPIVOT operations.

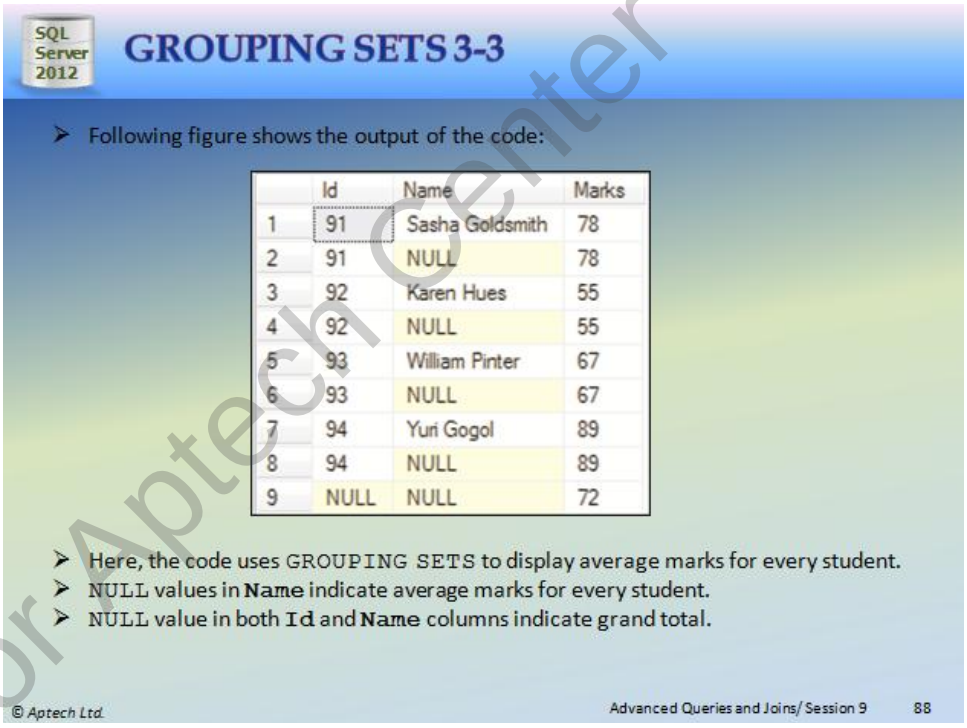
Explain the code snippet which demonstrates the UNPIVOT operator.

Grouping Sets

Slides 86 to 88







	Id	Name	Marks
1	91	Sasha Goldsmith	78
2	91	NULL	78
3	92	Karen Hues	55
4	92	NULL	55
5	93	William Pinter	67
6	93	NULL	67
7	94	Yuri Gogol	89
8	94	NULL	89
9	NULL	NULL	72

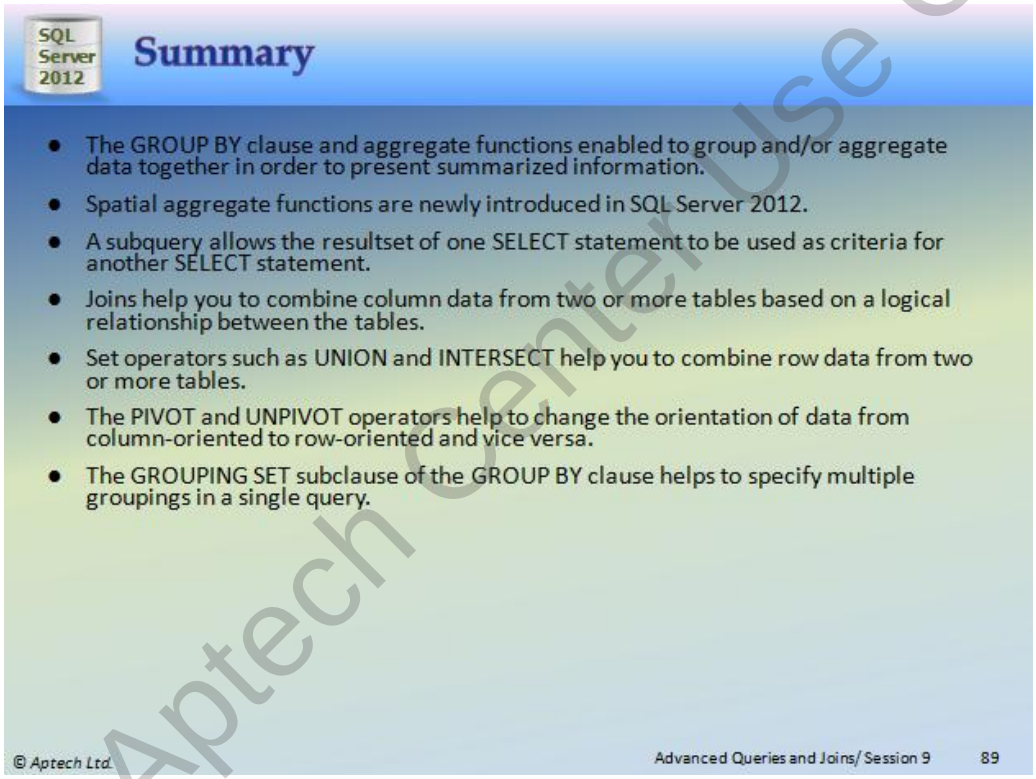
Using slides 86 to 88, explain the GROUPING sets.

The GROUPING SETS operator supports aggregation of multiple column groupings and an optional grand total. Consider that you need a report that groups several columns of a table. Further, you want aggregates of the columns. In earlier versions of SQL Server, you would have to write several distinct GROUP BY clauses followed by UNION clauses to achieve this. First introduced in SQL Server 2008, the GROUPING SETS operator allows you to group together multiple groupings of columns followed by an optional grand total row, denoted by

parentheses, (). It is more efficient to use GROUPING SETS operators instead of multiple GROUP BY with UNION clauses because the latter adds more processing overheads on the database server. Explain the syntax for the GROUPING SETS operator.
Explain the code snippet which demonstrates the GROUPING SETS operator. It assumes that a table Students is created with fields named Id, Name, and Marks respectively.
Figure shows the output of the code.
Here, the code uses GROUPING SETS to display average marks for every student. NULL values in Name indicate average marks for every student. NULL value in both Id and Name columns indicate grand total.

Summarize Session

Slide 89



Using slide 89, summarize the session. End the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- The GROUP BY clause and aggregate functions enabled to group and/or aggregate data together in order to present summarized information.
- Spatial aggregate functions are newly introduced in SQL Server 2012.
- A subquery allows the resultset of one SELECT statement to be used as criteria for another SELECT statement.
- Joins help you to combine column data from two or more tables based on a logical relationship between the tables.

- Set operators such as UNION and INTERSECT help you to combine row data from two or more tables.
- The PIVOT and UNPIVOT operators help to change the orientation of data from column-oriented to row-oriented and vice versa.
- The GROUPING SET subclause of the GROUP BY clause helps to specify multiple groupings in a single query.

9.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Views, Stored Procedures, and Querying Metadata topic offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 10 – Using Views, Stored Procedures, and Querying Metadata

10.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from the previous session. Prepare a question or two which will be a key point to relate the current session objectives.

10.1.1 Objectives

By the end of this session, the learners will be able to:

- Define views
- Describe the technique to create, alter, and drop views
- Define stored procedures
- Explain the types of stored procedures
- Describe the procedure to create, alter, and execute stored procedures
- Describe nested stored procedures
- Describe querying SQL Server metadata
    - System Catalog views and functions
    - Querying Dynamic Management Objects

10.1.2 Teaching Skills

To teach this session, you should be well-versed with views and creating, altering, and dropping views and stored procedures. The session also covers an explanation of the techniques to query metadata.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

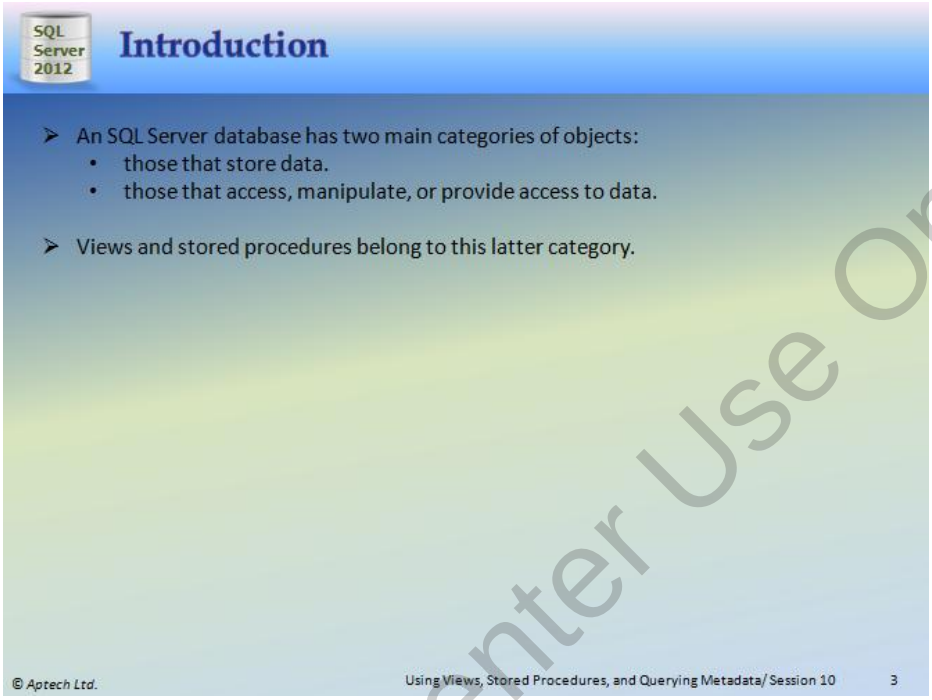
Overview of the Session:

Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces views and describes creating, altering, and dropping views. They will also know about stored procedures in detail and the techniques to query metadata.

10.2 In-Class Explanations

Introduction

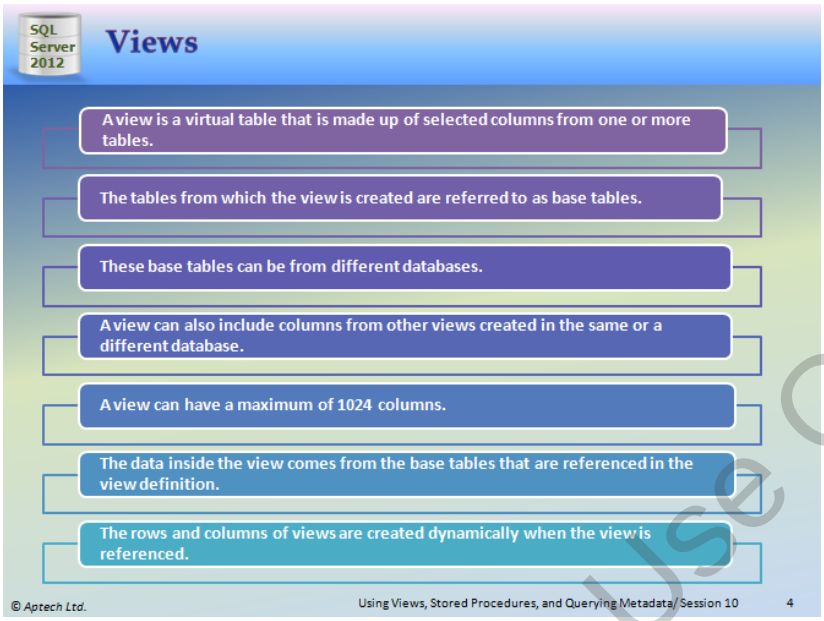
Slide 3



Using slide 3, explain the introduction.
An SQL Server database has two main categories of objects: those that store data and those that access, manipulate, or provide access to data. Views and stored procedures belong to this latter category.

Views

Slide 4



Using slide 4, explain the views.
A view is a virtual table that is made up of selected columns from one or more tables. The tables from which the view is created are referred to as base tables. These base tables can be from different databases. A view can also include columns from other views created in the same or a different database. A view can have a maximum of 1024 columns.

The data inside the view comes from the base tables that are referenced in the view definition. The rows and columns of views are created dynamically when the view is referenced.

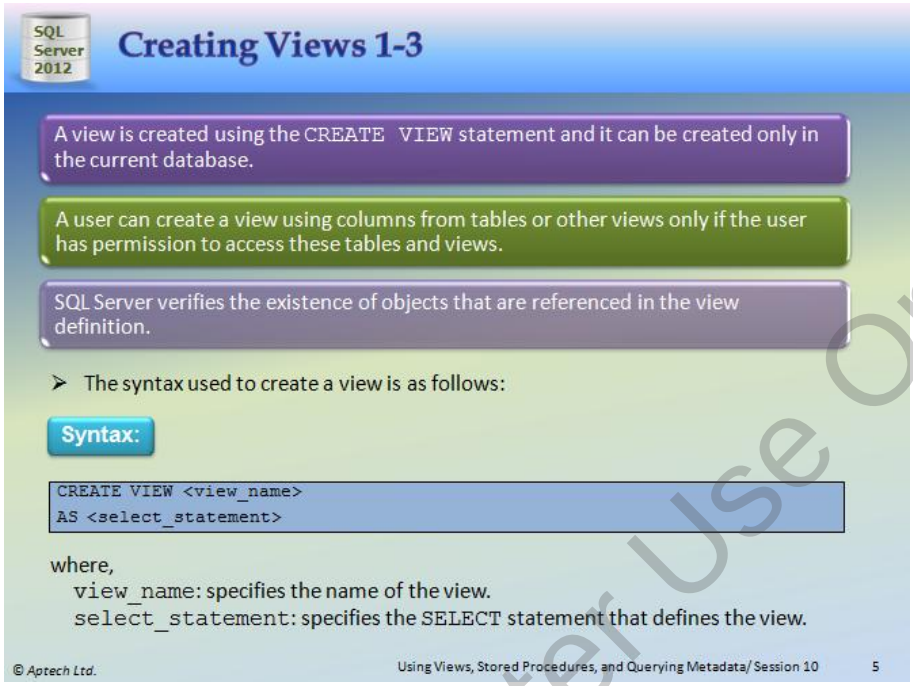
In-Class Question:

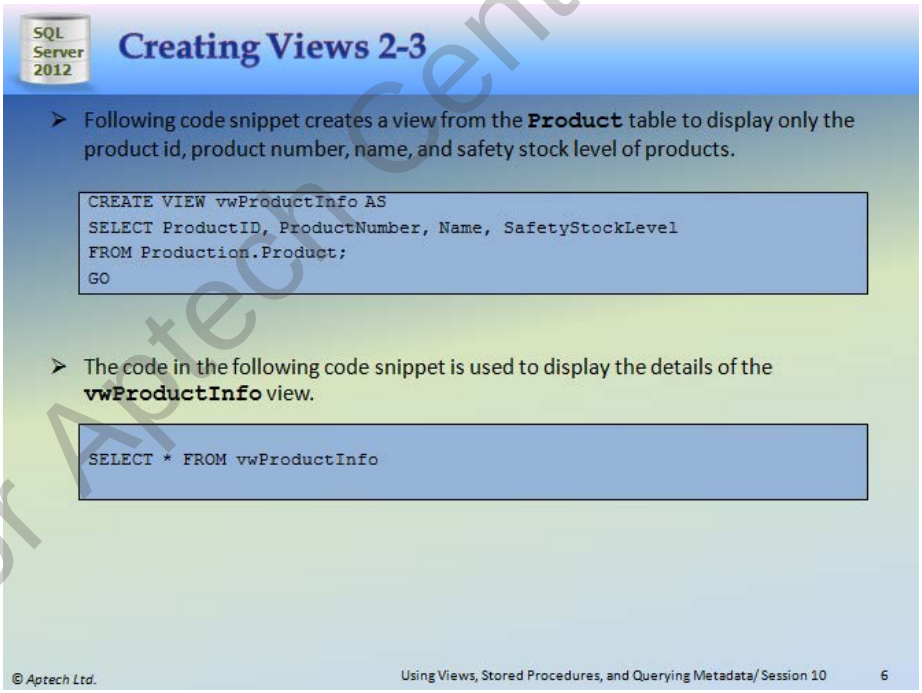
What is a view?

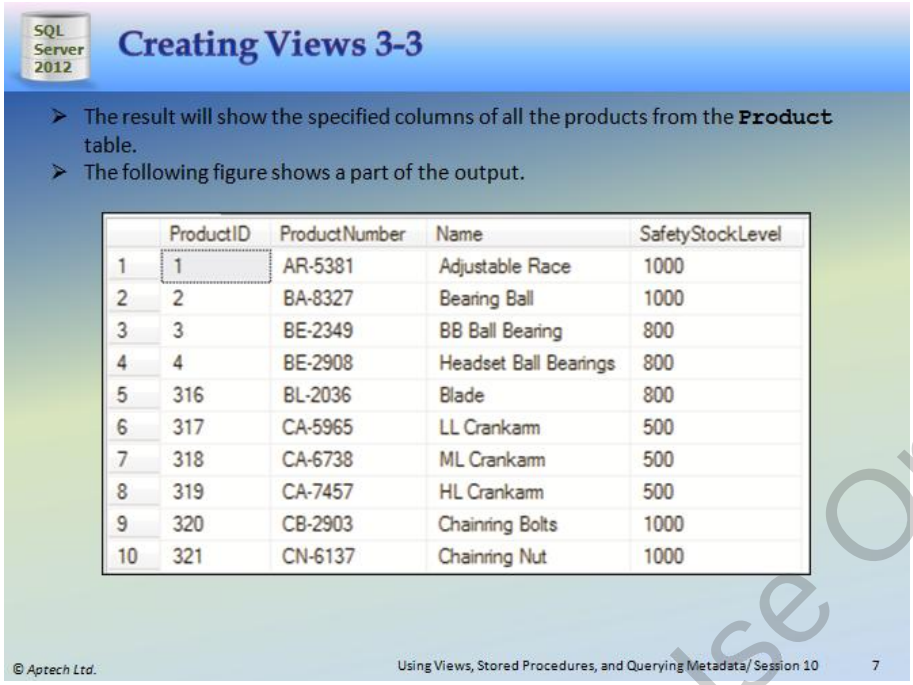
Answer:
A view is a virtual table that is made up of selected columns from one or more tables.

Creating Views

Slides 5 to 7







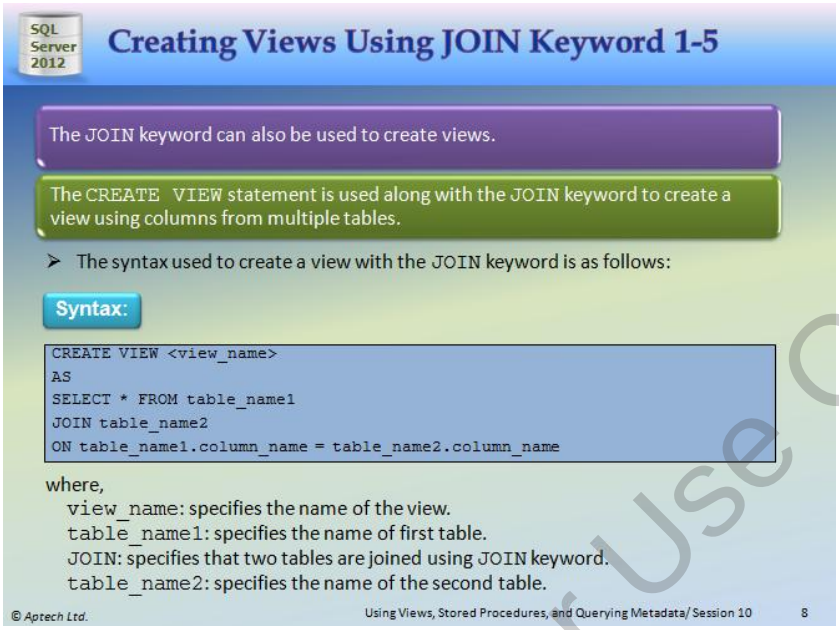
Using slides 5 to 7, explain how to create views.
A user can create a view using columns from tables or other views only if the user has permission to access these tables and views. A view is created using the CREATE VIEW statement and it can be created only in the current database.
SQL Server verifies the existence of objects that are referenced in the view definition.
While creating a view, test the SELECT statement that defines the view to make sure that SQL Server returns the expected result. The view can be created only after the SELECT statement is tested and the resultset has been verified.
Explain the syntax to create a view.
Explain the Code Snippet which creates a view from the Product table to display only the product id, product number, name, and safety stock level of products.
The word vw is prefixed to a view name as per the recommended coding conventions.

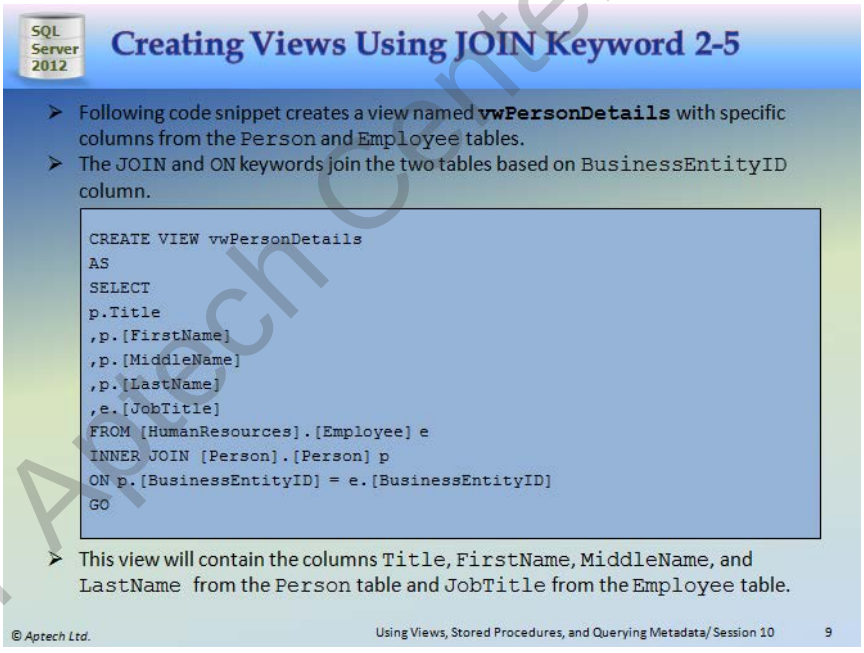
Explain the Code Snippet that is used to display the details of the vwProductInfo view.

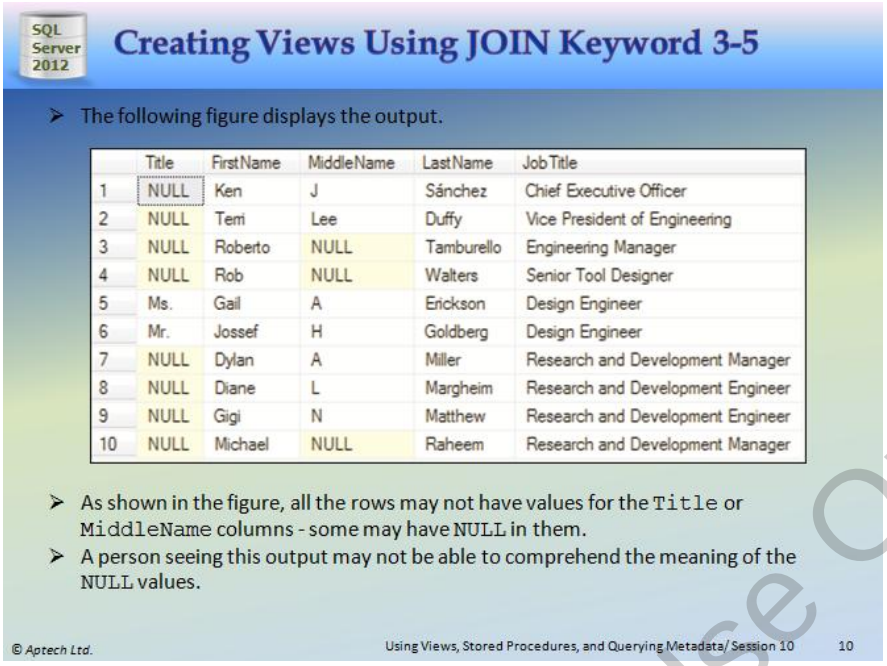
The result will show the specified columns of all the products from the Product table. A part of the output is shown in figure on slide 7.

Creating Views Using JOIN Keyword

Slides 8 to 12





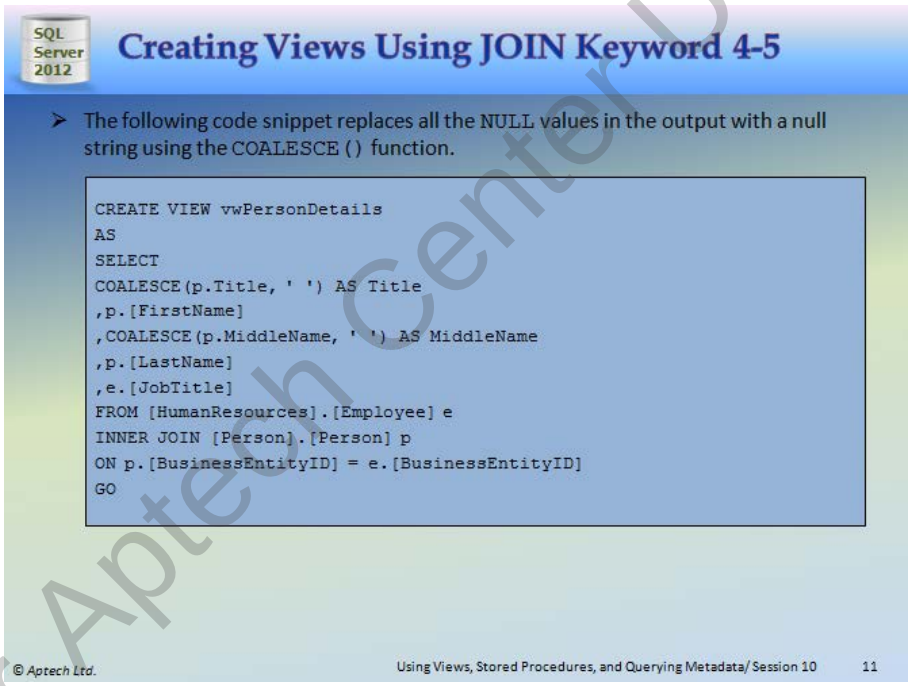


Creating Views Using JOIN Keyword 3-5

- The following figure displays the output.

	Title	FirstName	MiddleName	LastName	JobTitle
1	NULL	Ken	J	Sánchez	Chief Executive Officer
2	NULL	Terri	Lee	Duffy	Vice President of Engineering
3	NULL	Roberto	NULL	Tamburello	Engineering Manager
4	NULL	Rob	NULL	Walters	Senior Tool Designer
5	Ms.	Gail	A	Erickson	Design Engineer
6	Mr.	Jossef	H	Goldberg	Design Engineer
7	NULL	Dylan	A	Miller	Research and Development Manager
8	NULL	Diane	L	Margheim	Research and Development Engineer
9	NULL	Gigi	N	Matthew	Research and Development Engineer
10	NULL	Michael	NULL	Raheem	Research and Development Manager

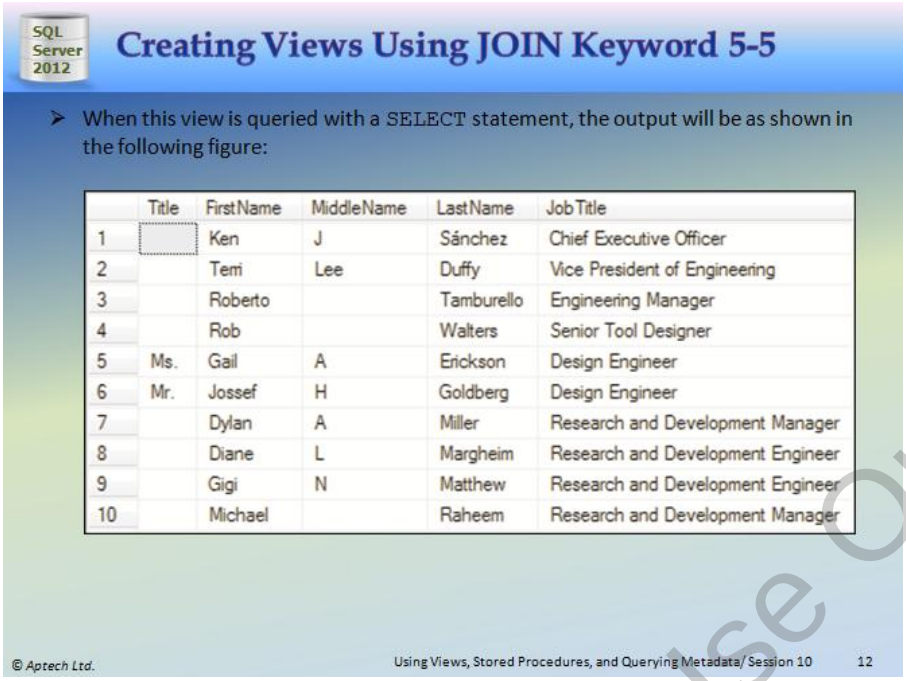
- As shown in the figure, all the rows may not have values for the `Title` or `MiddleName` columns - some may have `NULL` in them.
- A person seeing this output may not be able to comprehend the meaning of the `NULL` values.



Creating Views Using JOIN Keyword 4-5

- The following code snippet replaces all the `NULL` values in the output with a null string using the `COALESCE()` function.

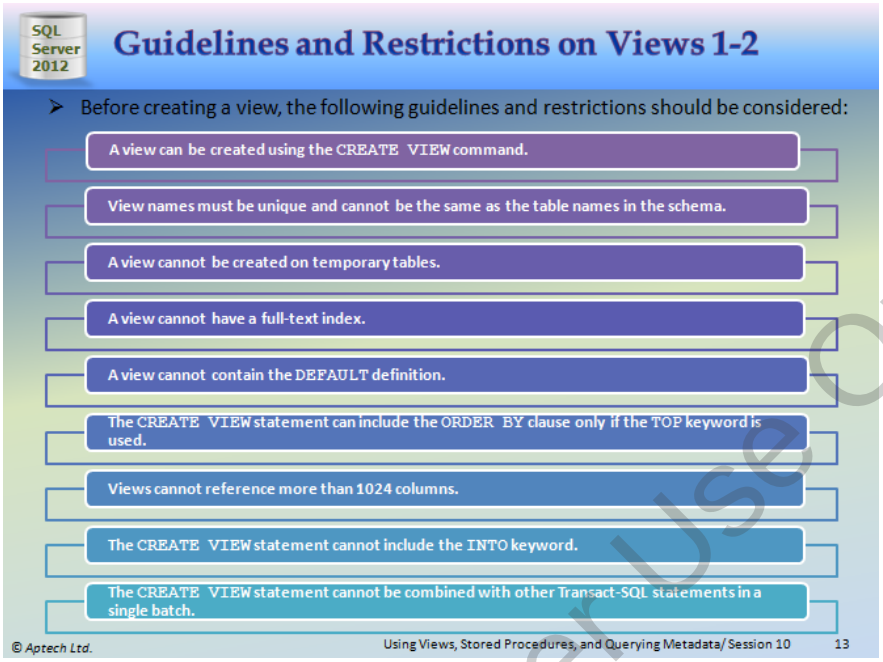
```
CREATE VIEW vwPersonDetails
AS
SELECT
COALESCE(p.Title, ' ') AS Title
,p.[FirstName]
,COALESCE(p.MiddleName, ' ') AS MiddleName
,p.[LastName]
,e.[JobTitle]
FROM [HumanResources].[Employee] e
INNER JOIN [Person].[Person] p
ON p.[BusinessEntityID] = e.[BusinessEntityID]
GO
```

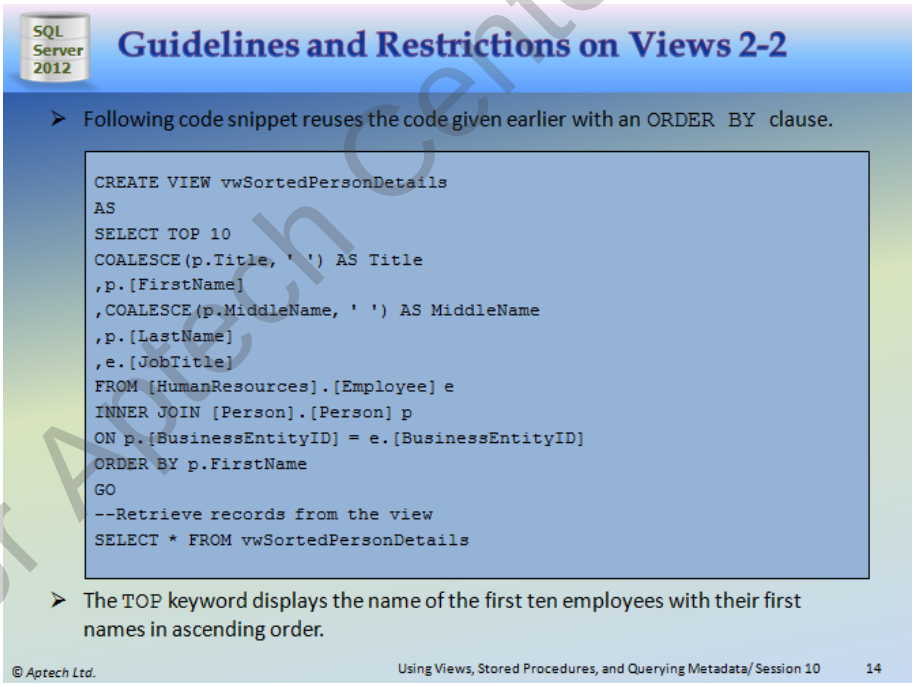


Using slides 8 to 12, explain how to create views using join keyword.
The JOIN keyword can also be used to create views. The CREATE VIEW statement is used along with the JOIN keyword to create a view using columns from multiple tables.
Explain the syntax is used to create a view with the JOIN keyword.
Explain the Code Snippet which creates a view named vwPersonDetails with specific columns from the Person and Employee tables. The JOIN and ON keywords join the two tables based on BusinessEntityID column.
This view will contain the columns Title, FirstName, MiddleName, and LastName from the Person table and JobTitle from the Employee table. Once the view is created, you can retrieve records from it, and manipulate and modify records as well.
Explain the Code Snippet which shows executing a query on this view.
The output will be as shown in figure on slide 10.
As shown in figure on slide 10, all the rows may not have values for the Title or MiddleName columns - some may have NULL in them. A person seeing this output may not be able to comprehend the meaning of the NULL values. Hence, to replace all the NULL values in the output with a null string, the COALESCE() function can be used as shown in Code Snippet on slide 11.
When this view is queried with a SELECT statement, the output will be as shown in figure on slide 12.

Guidelines and Restrictions on Views

Slides 13 and 14





Using slides 13 and 14, explain the guidelines and restrictions on views.

A view can be created using the CREATE VIEW command. Before creating a view, the following guidelines and restrictions should be considered:

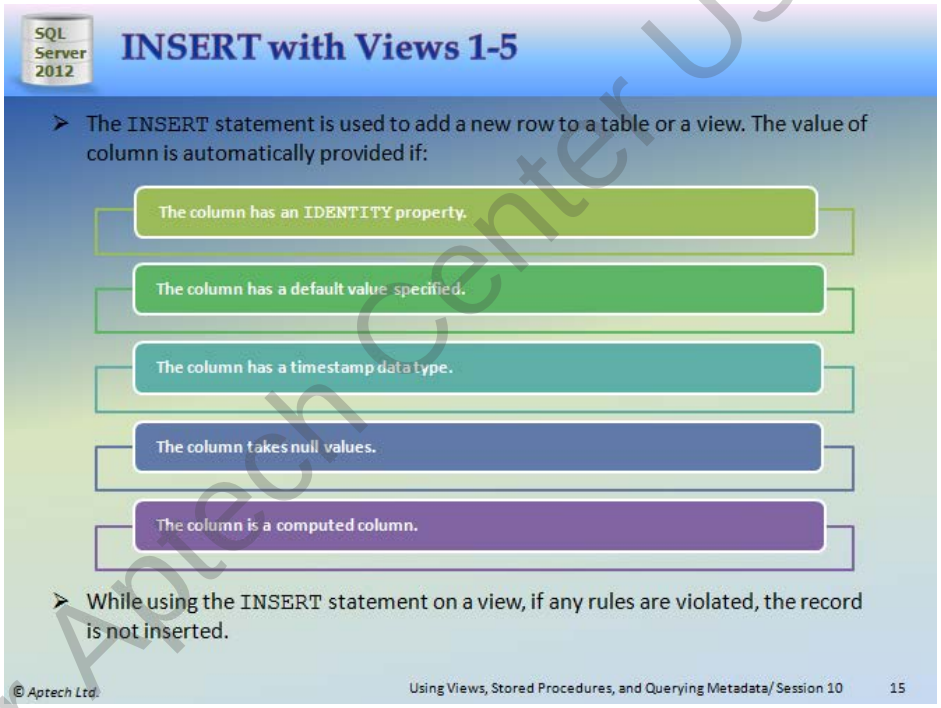
- A view is created only in the current database. The base tables and views from which the view is created can be from other databases or servers.
- View names must be unique and cannot be the same as the table names in the schema.
- A view cannot be created on temporary tables.
- A view cannot have a full-text index.

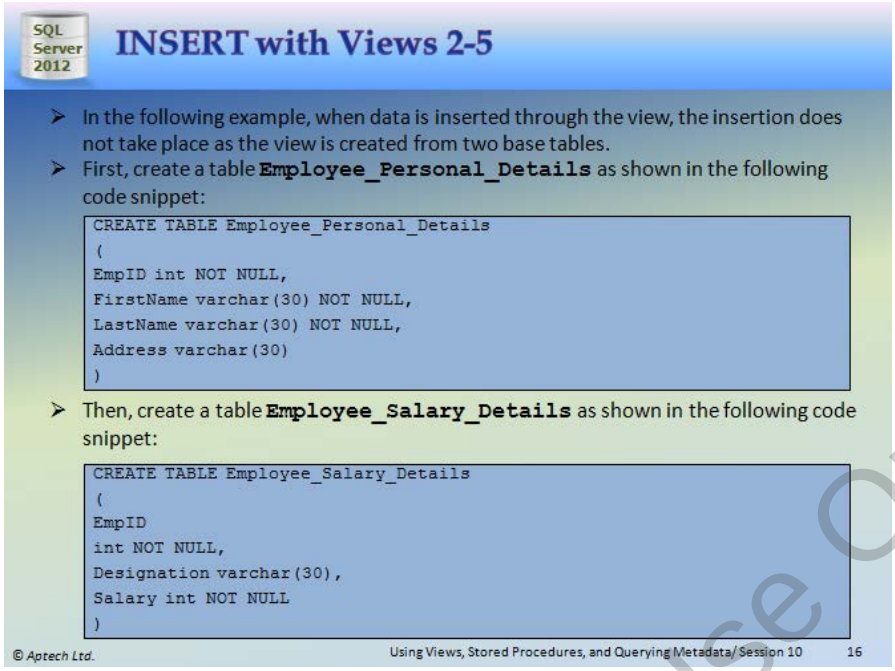
- A view cannot contain the DEFAULT definition.
- The CREATE VIEW statement can include the ORDER BY clause only if the TOP keyword is used.
- Views cannot reference more than 1024 columns.
- The CREATE VIEW statement cannot include the INTO keyword.
- The CREATE VIEW statement cannot be combined with other Transact-SQL statements in a single batch.

If a view contains columns that have values derived from an expression, such columns have to be given alias names. Also, if a view contains similarly-named columns from different tables, to distinguish these columns, alias names must be specified.
In the Code Snippet on slide 14, TOP keyword displays the name of the first ten employees with their first names in ascending order.

INSERT with Views

Slides 15 to 19





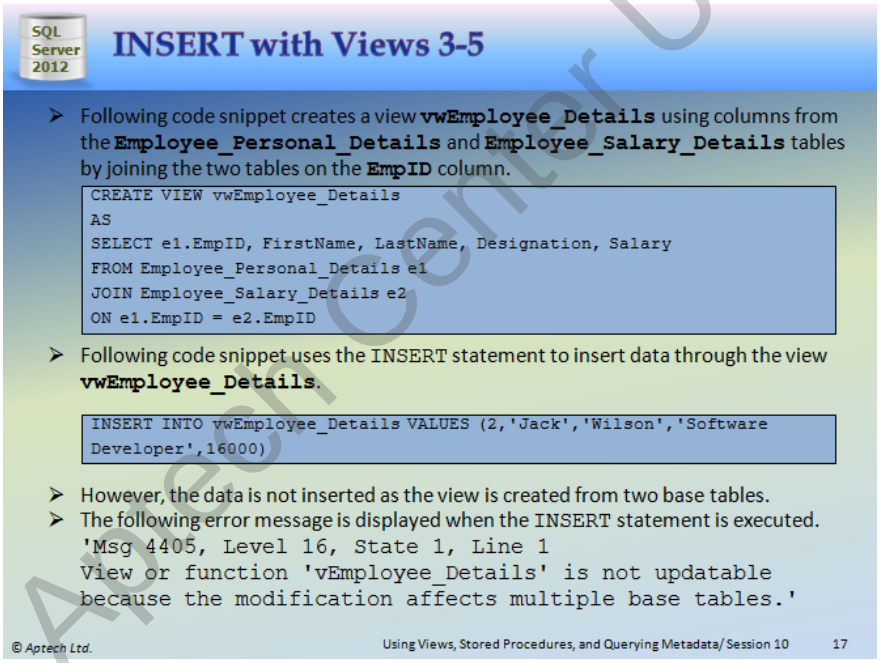
INSERT with Views 2-5

- In the following example, when data is inserted through the view, the insertion does not take place as the view is created from two base tables.
- First, create a table **Employee_Personal_Details** as shown in the following code snippet:

```
CREATE TABLE Employee_Personal_Details
(
EmpID int NOT NULL,
FirstName varchar(30) NOT NULL,
LastName varchar(30) NOT NULL,
Address varchar(30)
)
```

- Then, create a table **Employee_Salary_Details** as shown in the following code snippet:

```
CREATE TABLE Employee_Salary_Details
(
EmpID
int NOT NULL,
Designation varchar(30),
Salary int NOT NULL
)
```

INSERT with Views 3-5

- Following code snippet creates a view **vwEmployee_Details** using columns from the **Employee_Personal_Details** and **Employee_Salary_Details** tables by joining the two tables on the **EmpID** column.

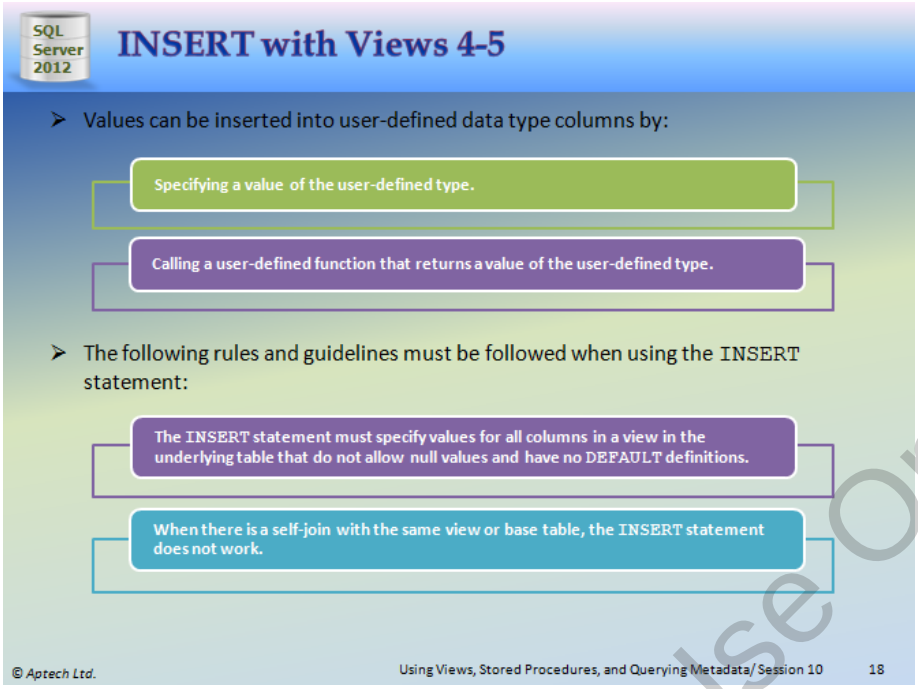
```
CREATE VIEW vwEmployee_Details
AS
SELECT e1.EmpID, FirstName, LastName, Designation, Salary
FROM Employee_Personal_Details e1
JOIN Employee_Salary_Details e2
ON e1.EmpID = e2.EmpID
```

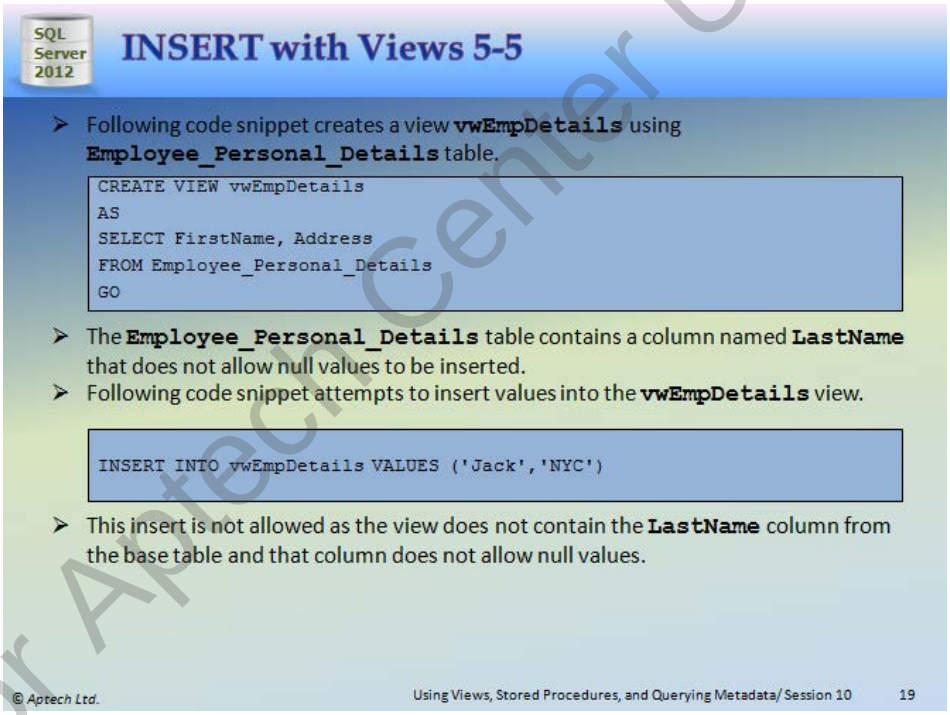
- Following code snippet uses the INSERT statement to insert data through the view **vwEmployee_Details**.

```
INSERT INTO vwEmployee_Details VALUES (2,'Jack','Wilson','Software
Developer',16000)
```

- However, the data is not inserted as the view is created from two base tables.
- The following error message is displayed when the INSERT statement is executed.

```
'Msg 4405, Level 16, State 1, Line 1
View or function 'vEmployee_Details' is not updatable
because the modification affects multiple base tables.'
```



Using slides 15 to 19, explain the INSERT with Views.

The INSERT statement is used to add a new row to a table or a view. During the execution of the statement, if the value for a column is not provided, the SQL Server Database Engine must provide a value based on the definition of the column. If the Database Engine is unable to provide this value, then the new row will not be added.
The value for the column is provided automatically if:

- The column has an IDENTITY property.
- The column has a default value specified.
- The column has a timestamp data type.

- The column takes null values.
- The column is a computed column.

While using the INSERT statement on a view, if any rules are violated, the record is not inserted.
In the following example, when data is inserted through the view, the insertion does not take place as the view is created from two base tables.
First, create a table Employee_Personal_Details as shown in Code Snippet on slide 16.
Then, create a table Employee_Salary_Details as shown in Code Snippet on slide 16.

Explain the Code Snippet that creates a view vwEmployee_Details using columns from the Employee_Personal_Details and Employee_Salary_Details tables by joining the two tables on the EmpID column.

Explain the Code Snippet that uses the INSERT statement to insert data through the view vwEmployee_Details. However, the data is not inserted as the view is created from two base tables.

Explain the error message is displayed when the INSERT statement is executed.

Values can be inserted into user-defined data type columns by:

- Specifying a value of the user-defined type.
- Calling a user-defined function that returns a value of the user-defined type.

The following rules and guidelines must be followed when using the INSERT statement: The INSERT statement must specify values for all columns in a view in the underlying table that do not allow null values and have no DEFAULT definitions. When there is a self-join with the same view or base table, the INSERT statement does not work.

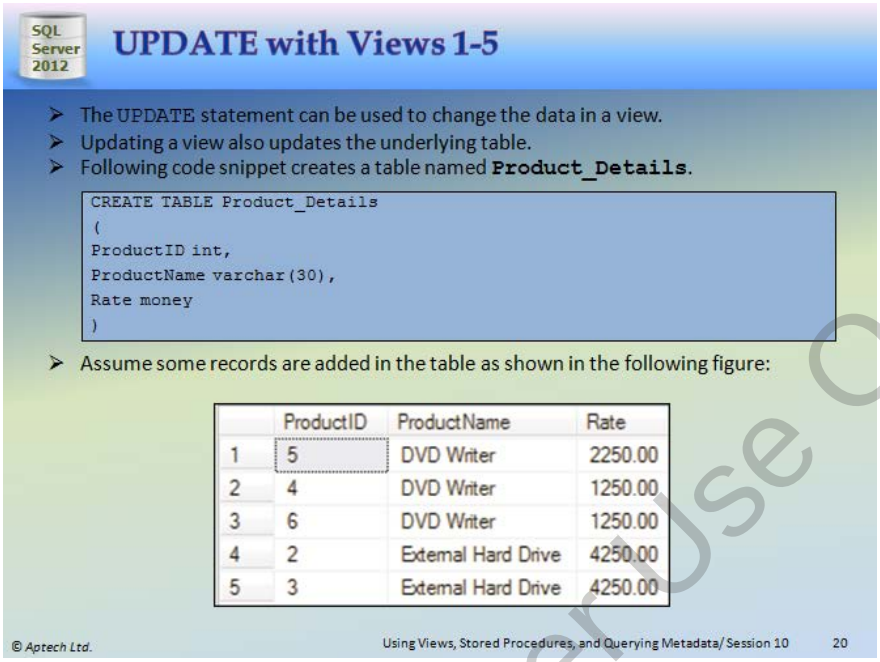
Explain the Code Snippet that creates a view vwEmpDetails using Employee_Personal_Details table. The Employee_Personal_Details table contains a column named LastName that does not allow null values to be inserted.

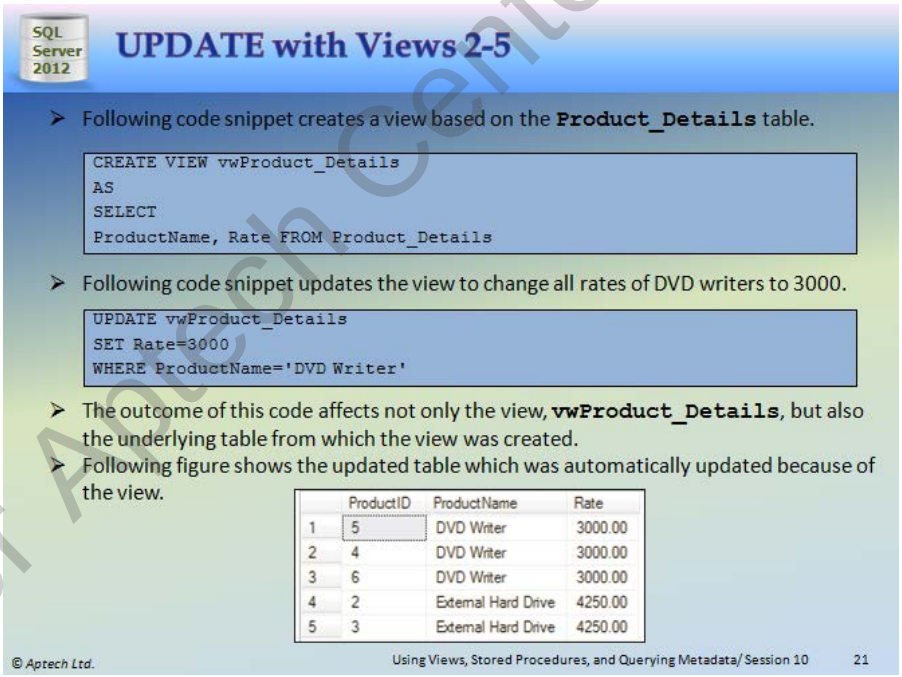
Explain the Code Snippet that attempts to insert values into the vwEmpDetails view.

This insert is not allowed as the view does not contain the LastName column from the base table and that column does not allow null values.

UPDATE with Views

Slides 20 to 24





UPDATE with Views 3-5

- Large value data types include `varchar(max)`, `nvarchar(max)`, and `varbinary(max)`.
- To update data having large value data types, the `.WRITE` clause is used.
- The `.WRITE` clause specifies that a section of the value in a column is to be modified.
- The `.WRITE` clause cannot be used to update a `NULL` value in a column.
- Also, it cannot be used to set a column value to `NULL`.

Syntax:

```
column_name .WRITE (expression, @Offset, @Length)
```

where,

`column_name`: specifies the name of the large value data-type column.
`Expression`: specifies the value that is copied to the column.
`@Offset`: specifies the starting point in the value of the column at which the expression is written.
`@Length`: specifies the length of the section in the column.

- `@Offset` and `@Length` are specified in bytes for `varbinary` and `varchar` data types and in characters for the `nvarchar` data type.

UPDATE with Views 4-5

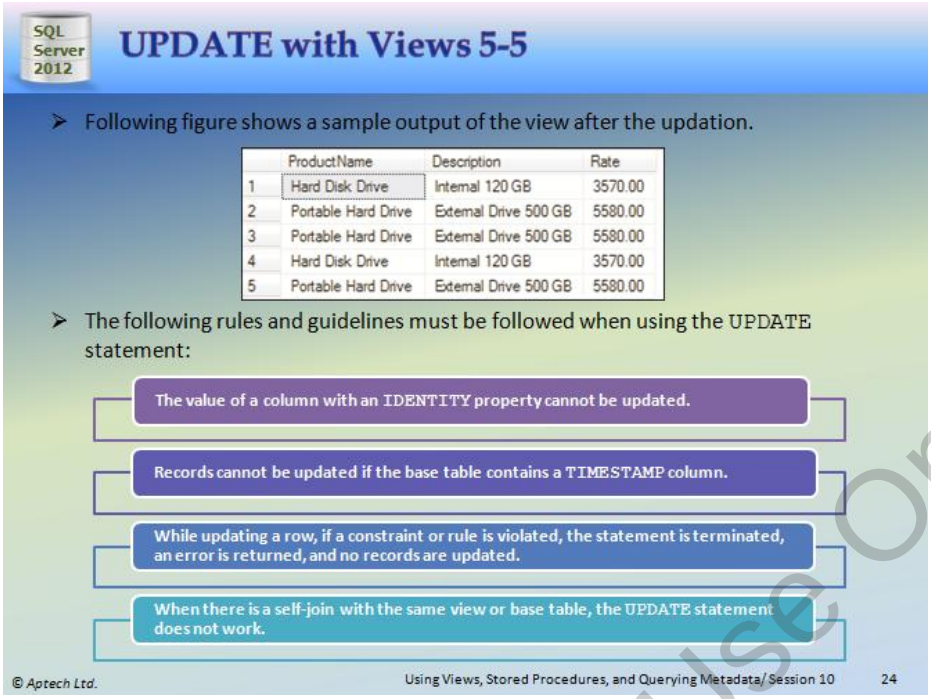
- Assume that the table **`Product_Details`** is modified to include a column **`Description`** having data type `nvarchar(max)`.
- The following code snippet creates a view based on this table, having the columns **`ProductName`**, **`Description`**, and **`Rate`**.

```
CREATE VIEW vwProduct_Details
AS
SELECT
ProductName,
Description,
Rate FROM Product_Details
```

- Following code snippet uses the `UPDATE` statement on the view **`vwProduct_Details`**.
- The `.WRITE` clause is used to change the value of Internal in the **`Description`** column to External.

```
UPDATE vwProduct_Details
SET Description .WRITE(N'Ex',0,2)
WHERE ProductName='Portable Hard Drive'
```

- All the rows in the view that had 'Portable Hard Drive' as product name will be updated with `External` instead of Internal in the **`Description`** column.



Using slides 20 to 24, explain how to update with view.

The UPDATE statement can be used to change the data in a view. Updating a view also updates the underlying table. Consider an example. Explain the Code Snippet that creates a table named Product_Details.

Assume some records are added in the table as shown in figure on slide 20.

Explain the Code Snippet that creates a view based on the Product_Details table.

Explain the Code Snippet that updates the view to change all rates of DVD writers to 3000.

The outcome of this code affects not only the view, vwProduct_Details, but also the underlying table from which the view was created. Figure on slide 21 shows the updated table which was automatically updated because of the view.

Large value data types include varchar(max), nvarchar(max), and varbinary(max). To update data having large value data types, the .WRITE clause is used. The .WRITE clause specifies that a section of the value in a column is to be modified. The .WRITE clause cannot be used to update a NULL value in a column. Also, it cannot be used to set a column value to NULL. @Offset and @Length are specified in bytes for varbinary and varchar data types and in characters for the nvarchar data type.

Assume that the table Product_Details is modified to include a column Description having data type nvarchar(max).

A view is created based on this table, having the columns ProductName, Description, and Rate as shown in Code Snippet on slide 23.

Explain the Code Snippet that uses the UPDATE statement on the view vwProduct_Details.

The .WRITE clause is used to change the value of Internal in the Description column to External.

As a result of the code, all the rows in the view that had 'Portable Hard Drive' as product name will be updated with External instead of Internal in the Description column.

Figure on slide 24 shows a sample output of the view after the updation.

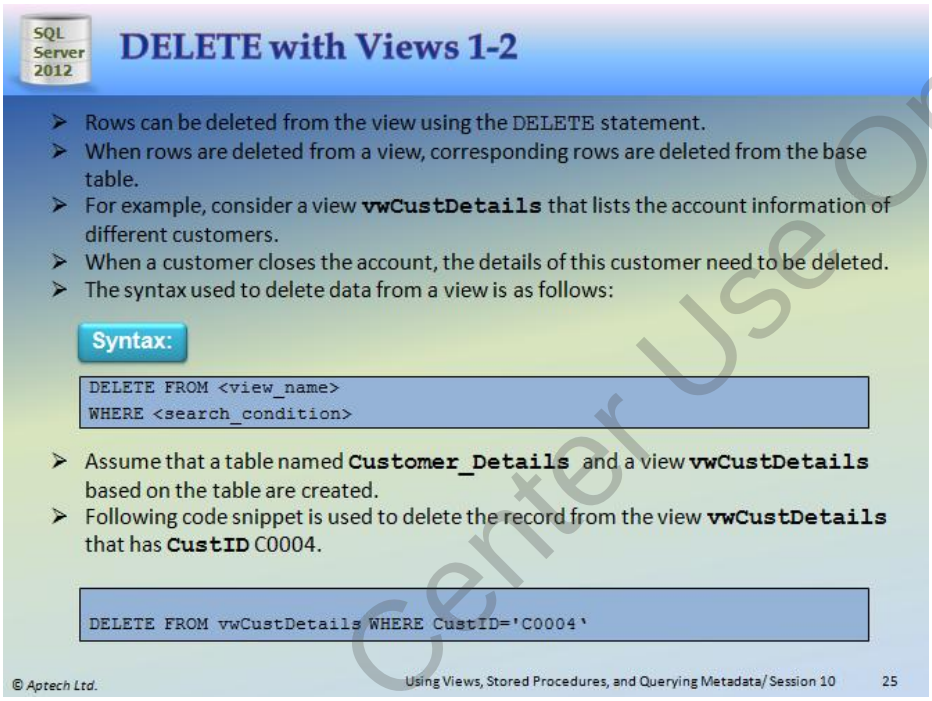
Following rules and guidelines must be followed when using the UPDATE statement:

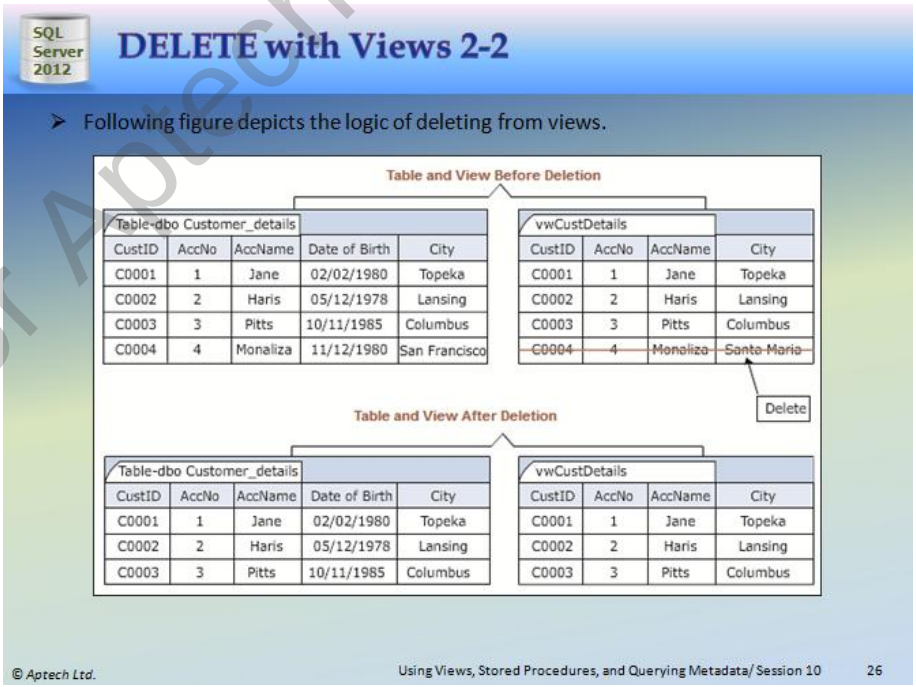
- The value of a column with an IDENTITY property cannot be updated. Records cannot be updated if the base table contains a TIMESTAMP column.

- While updating a row, if a constraint or rule is violated, the statement is terminated, an error is returned, and no records are updated.
- When there is a self-join with the same view or base table, the UPDATE statement does not work.

DELETE with Views

Slides 25 and 26





Using slides 25 and 26, explain how to delete with views.

SQL Server enables you to delete rows from a view. Rows can be deleted from the view using the DELETE statement. When rows are deleted from a view, corresponding rows are deleted from the base table. For example, consider a view vwCustDetails that lists the account information of different customers. When a customer closes the account, the details of this customer need to be deleted. This is done using the DELETE statement.

Explain the syntax used to delete data from a view.

Assume that a table named Customer_Details and a view vwCustDetails based on the table are created.

Explain the Code Snippet used to delete the record from the view vwCustDetails that has CustID C0004.

Figure on slide 26 depicts the logic of deleting from views.

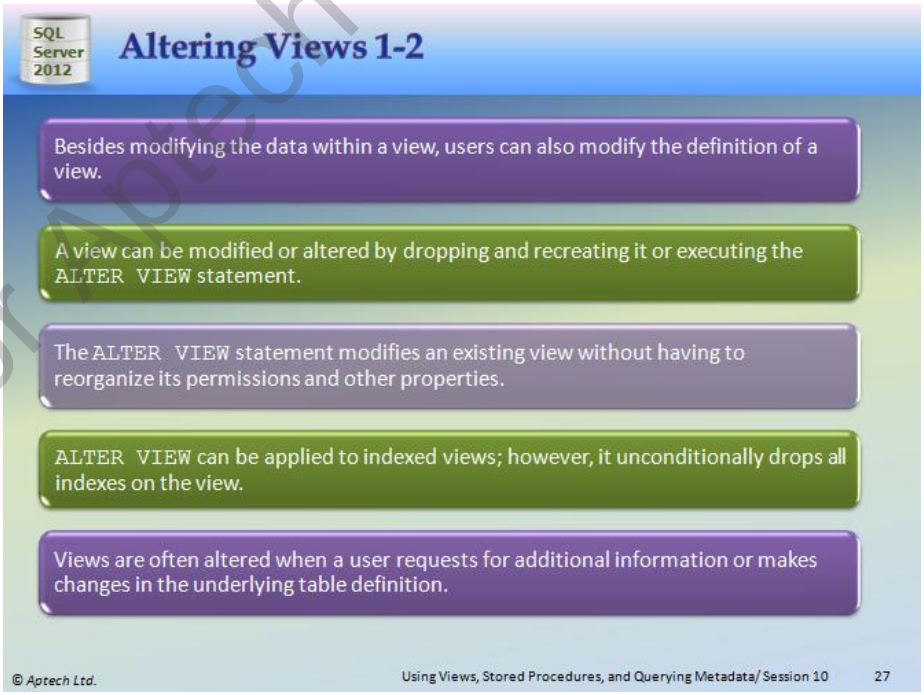
In-Class Question:

What is used to delete a row from a view?

Answer:
Rows can be deleted from the view using the DELETE statement.

Let us Understand how to Alter a View

Slides 27 and 28



Using slides 27 and 28, explain how to alter a view.

Besides modifying the data within a view, users can also modify the definition of a view. A view can be modified or altered by dropping and recreating it or executing the ALTER VIEW statement. The ALTER VIEW statement modifies an existing view without having to reorganize its permissions and other properties. ALTER VIEW can be applied to indexed views; however, it unconditionally drops all indexes on the view.

Views are often altered when a user requests for additional information or makes changes in the underlying table definition.

Explain the syntax used to alter a view.

Explain the Code Snippet alters the view, vwProductInfo to include the ReOrderPoint column.

Dropping Views

Slide 29

Using slide 29, explain how to drop a view.

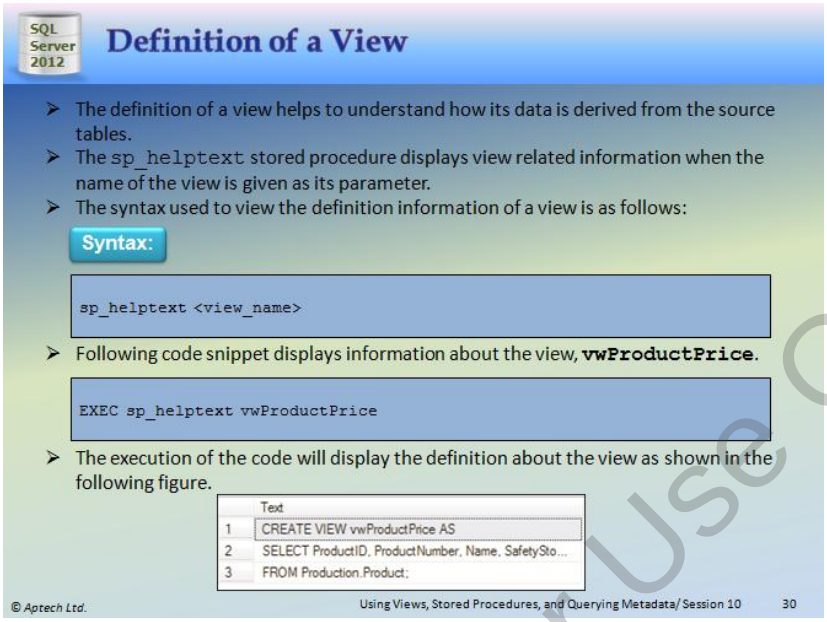
A view can be removed from the database if it is no longer needed. This is done using the DROP VIEW statement. When a view is dropped, the data in the base tables remains unaffected. The definition of the view and other information associated with the view is deleted from the system catalog. All permissions for the view are also deleted. If a user queries any view that references the dropped view, the user receives an error message.

The following syntax is used to drop a view.

Explain the Code Snippet that deletes the view, vwProductInfo.

Definition of View

Slide 30



Using slide 30, explain the definition of view.
The definition of a view helps to understand how its data is derived from the source tables.
There are certain system stored procedures that help to retrieve view definitions.
The sp_helptext stored procedure displays view related information when the name of the view is given as its parameter. Information about the definition of a view can be obtained if such information is not encrypted.
Explain the syntax used to view the definition information of a view.
Explain the Code Snippet that displays information about the view, vwProductPrice.
The execution of the code will display the definition about the view as shown in the figure on slide 30.

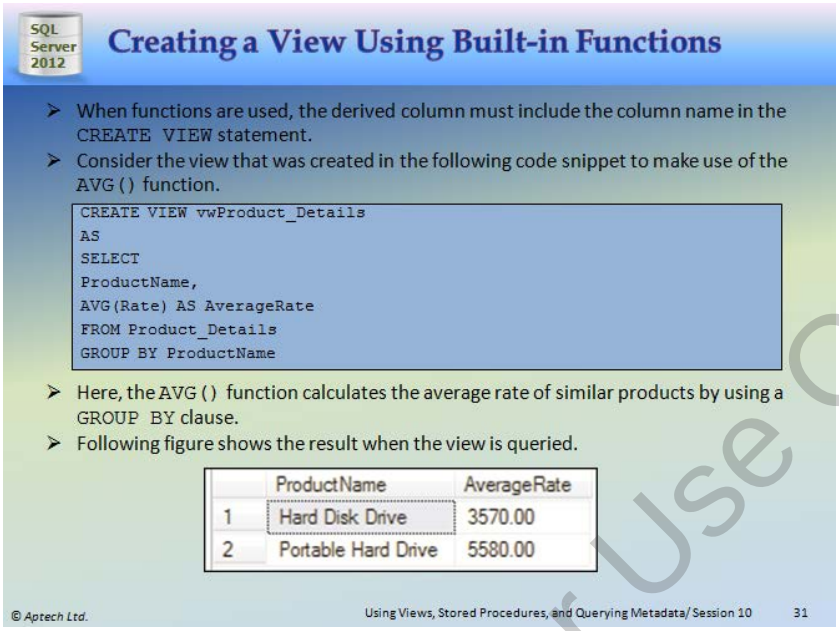
In-Class Question:

Which stored procedure displays view related information when the name of the view is given as its parameter?

Answer:
The sp_helptext stored procedure displays view related information when the name of the view is given as its parameter.

Create a View Using Built-in Functions

Slide 31



Using slide 31, explain how to create view using built-in functions.

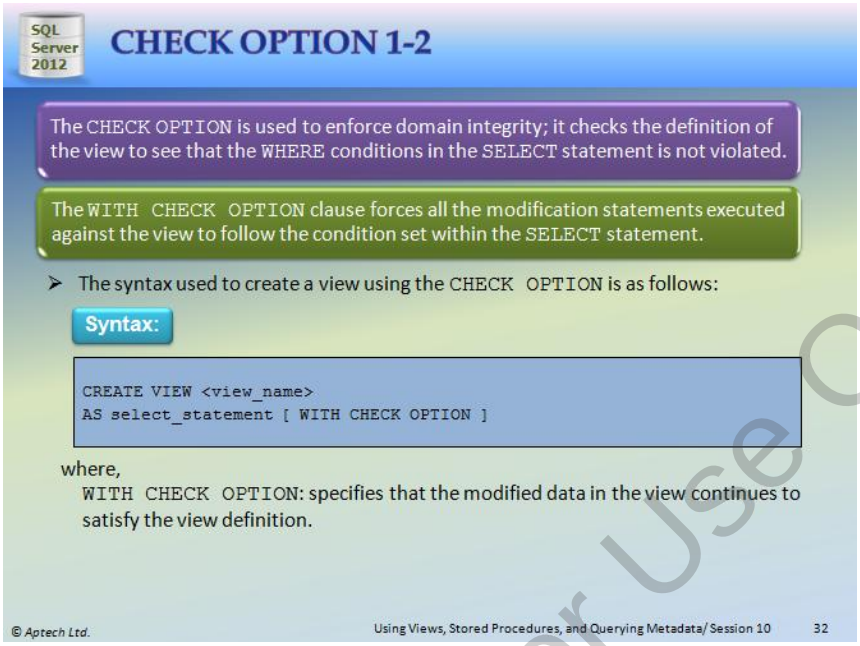
Views can be created using built-in functions of SQL Server. When functions are used, the derived column must include the column name in the CREATE VIEW statement.

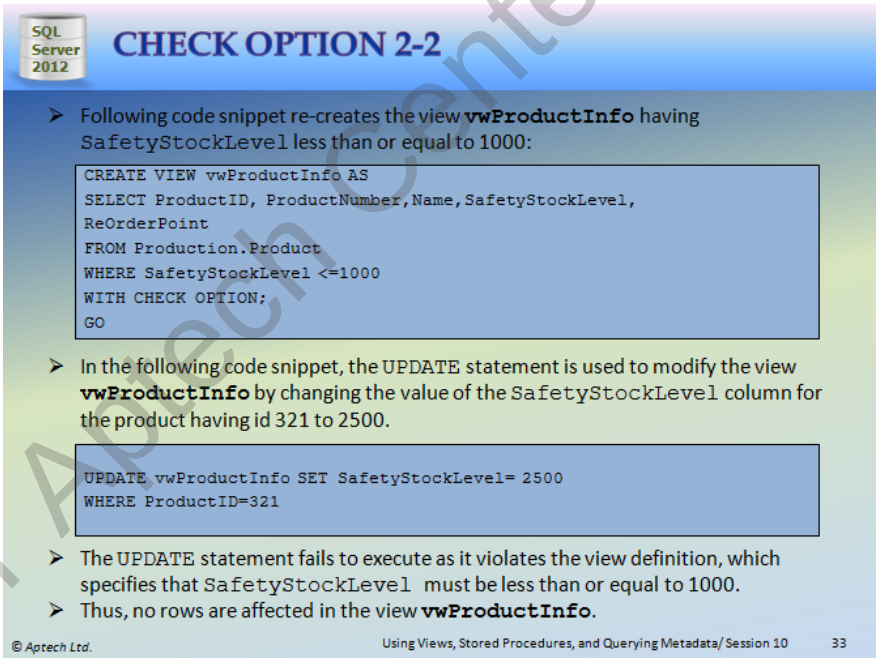
Consider the view that was created in Code Snippet on slide 23. It has been re-created in the Code Snippet on slide 31 to make use of the AVG() function.

Here, the AVG() function calculates the average rate of similar products by using a GROUP BY clause. Figure on slide 31 shows the result when the view is queried.

CHECK OPTION

Slides 32 and 33





Using slides 32 and 33, explain the CHECK OPTION.

The CHECK OPTION is an option associated with the CREATE VIEW statement. It is used to ensure that all the updates in the view satisfy the conditions mentioned in the view definition. If the conditions are not satisfied, the database engine returns an error. Thus, the CHECK OPTION is used to enforce domain integrity; it checks the definition of the view to see that the WHERE conditions in the SELECT statement is not violated.

The WITH CHECK OPTION clause forces all the modification statements executed against the view to follow the condition set within the SELECT statement. When a row is modified, the WITH CHECK OPTION makes sure that the data remains visible through the view.
Explain the syntax which creates a view using the CHECK OPTION.
Explain the Code Snippet that re-creates the view vwProductInfo having SafetyStockLevel less than or equal to 1000.
In Code Snippet, the UPDATE statement is used to modify the view vwProductInfo by changing the value of the SafetyStockLevel column for the product having id 321 to 2500. The UPDATE statement fails to execute as it violates the view definition, which specifies that SafetyStockLevel must be less than or equal to 1000. Thus, no rows are affected in the view vwProductInfo.

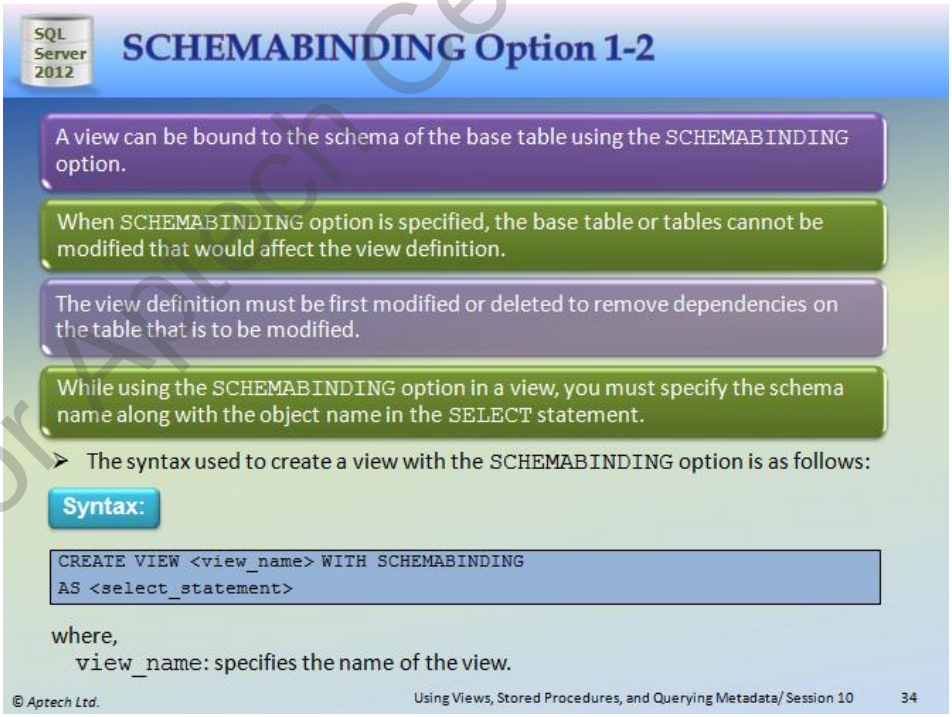
In-Class Question:

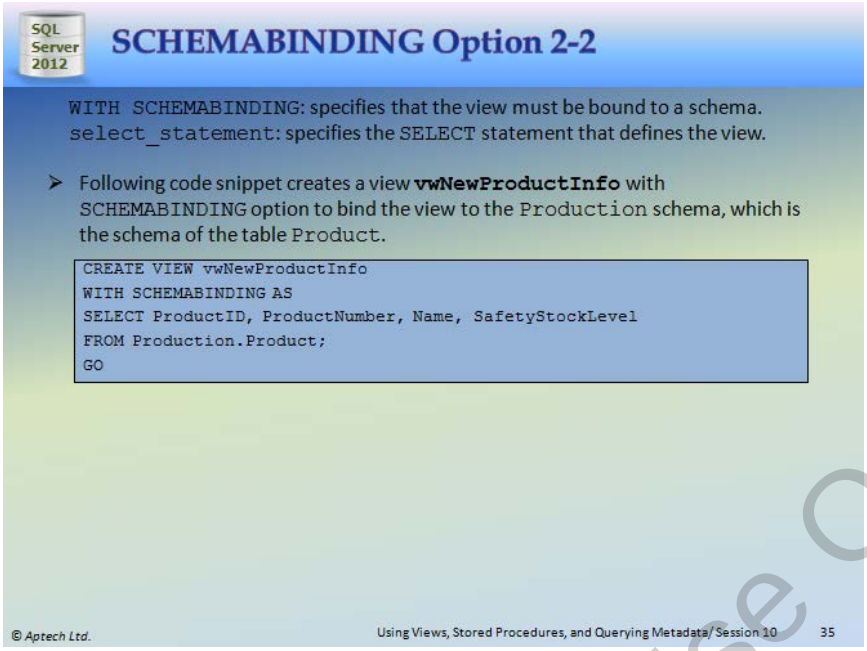
What is the use of CHECK OPTION?

Answer:
The CHECK OPTION is used to ensure that all the updates in the view satisfy the conditions mentioned in the view definition.

SCHEMABINDING Option

Slides 34 and 35





Using slides 34 and 35, explain the SCHEMABINDING option.

A view can be bound to the schema of the base table using the SCHEMABINDING option. This option can be used with CREATE VIEW or ALTER VIEW statements. When SCHEMABINDING option is specified, the base table or tables cannot be modified that would affect the view definition. The view definition must be first modified or deleted to remove dependencies on the table that is to be modified.

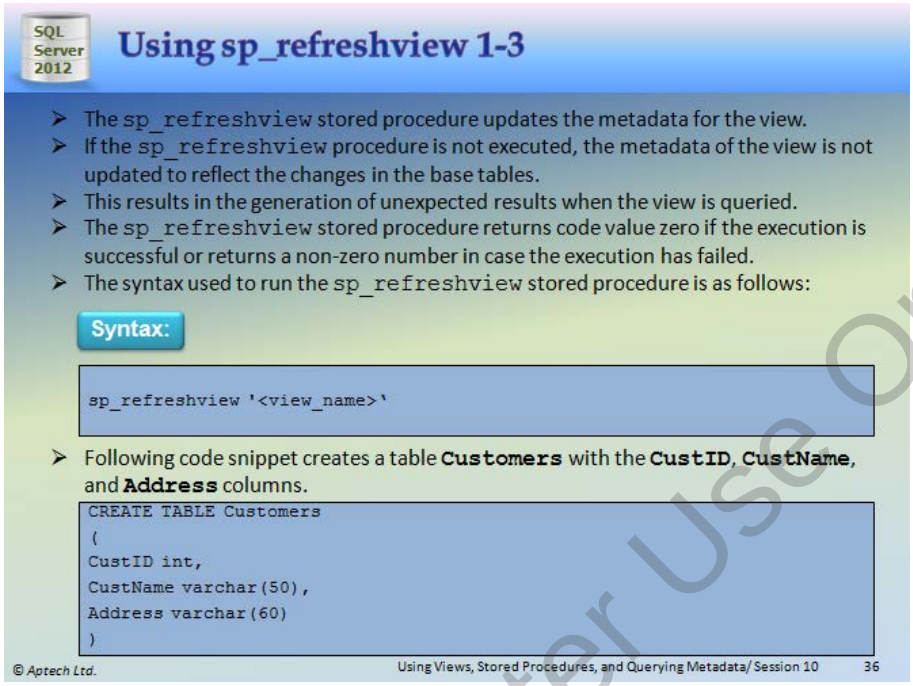
While using the SCHEMABINDING option in a view, you must specify the schema name along with the object name in the SELECT statement.

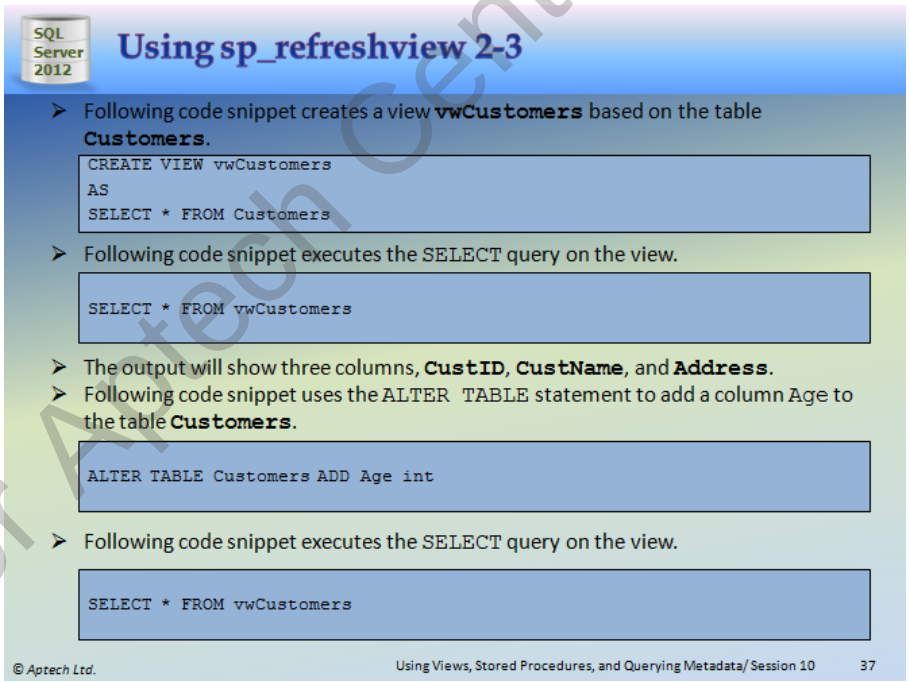
Explain the syntax used to create a view with the SCHEMABINDING option.

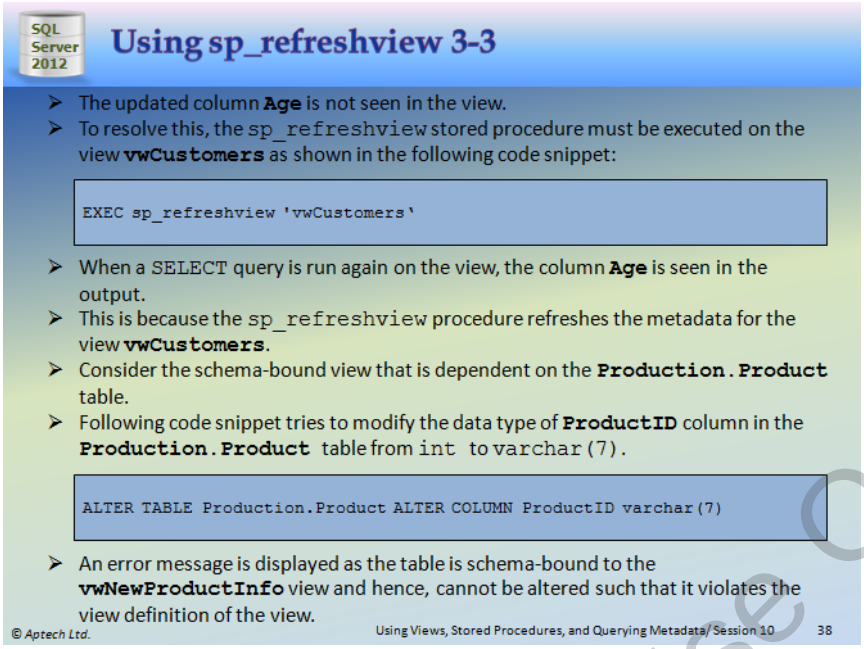
Explain the Code Snippet that creates a view vwNewProductInfo with SCHEMABINDING option to bind the view to the Production schema, which is the schema of the table Product.

Using sp_refreshview

Slides 36 to 38







Using slides 36 to 38, explain how to use sp_refreshview.
During the creation of a view, the SCHEMABINDING option is used to bind the view to the schema of the tables that are included in the view. However, a view can also be created without selecting the SCHEMABINDING option. In such a case, if changes are made to the underlying objects (tables or views) on which the view depends, the sp_refreshview stored procedure should be executed. The sp_refreshview stored procedure updates the metadata for the view. If the sp_refreshview procedure is not executed, the metadata of the view is not updated to reflect the changes in the base tables. This results in the generation of unexpected results when the view is queried.

The sp_refreshview stored procedure returns code value zero if the execution is successful or returns a non-zero number in case the execution has failed.

Explain the syntax used to run the sp_refreshview stored procedure.

Explain the Code Snippet that creates a table Customers with the CustID, CustName, and Address columns.

Explain the Code Snippet that creates a view vwCustomers based on the table Customers and executes the SELECT query on the view.

The output of Code Snippet shows the three columns, CustID, CustName, and Address.

Explain the Code Snippet that uses the ALTER TABLE statement to add a column Age to the table Customers.

Explain the Code Snippet that executes the SELECT query on the view.

The updated column Age is not seen in the view. To resolve this, the sp_refreshview stored procedure must be executed on the view vwCustomers as shown in the Code Snippet on slide 38.

When a SELECT query is run again on the view, the column Age is seen in the output. This is because the sp_refreshview procedure refreshes the metadata for the view vwCustomers.

Tables that are schema-bound to a view cannot be dropped unless the view is dropped or changed such that it no longer has schema binding. If the view is not dropped or changed and you attempt to drop the table, the Database Engine returns an error message.

Also, when an ALTER TABLE statement affects the view definition of a schema-bound view, the ALTER TABLE statement fails.

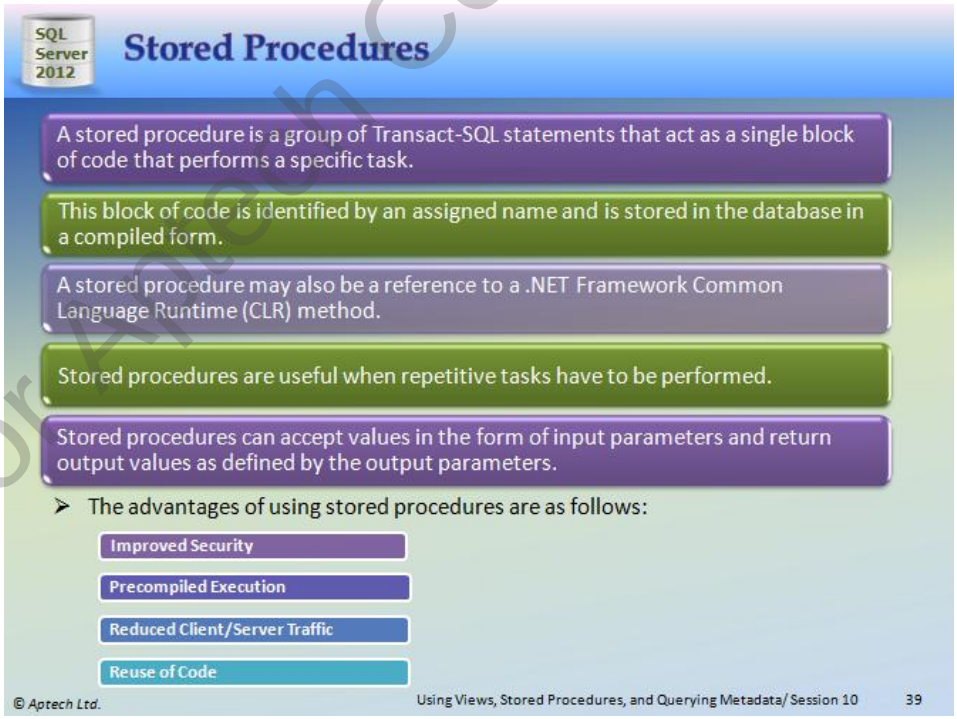
Consider the schema-bound view that was created earlier. It is dependent on the Production.Product table.

Explain the Code Snippet that tries to modify the data type of ProductID column in the Production.Product table from int to varchar(7).

The Database Engine returns an error message as the table is schema-bound to the vwNewProductInfo view and hence, cannot be altered such that it violates the view definition of the view.

Stored Procedures

Slide 39



Using slide 39, explain the stored procedures.
A stored procedure is a group of Transact-SQL statements that act as a single block of code that performs a specific task. This block of code is identified by an assigned name and is

stored in the database in a compiled form. A stored procedure may also be a reference to a .NET Framework Common Language Runtime (CLR) method.
Stored procedures are useful when repetitive tasks have to be performed. This eliminates the need for repetitively typing out multiple Transact-SQL statements and then repetitively compiling them.
Stored procedures can accept values in the form of input parameters and return output values as defined by the output parameters.
Using stored procedures offers numerous advantages over using Transact-SQL statements. These are as follows:

- Improved Security: The database administrator can improve the security by associating database privileges with stored procedures. Users can be given permission to execute a stored procedure even if the user does not have permission to access the tables or views.
- Precompiled Execution: Stored procedures are compiled during the first execution. For every subsequent execution, SQL Server reuses this precompiled version. This reduces the time and resources required for compilation.
- Reduced Client/Server Traffic: Stored procedures help in reducing network traffic. When Transact-SQL statements are executed individually, there is network usage separately for execution of each statement. When a stored procedure is executed, the Transact-SQL statements are executed together as a single unit. Network path is not used separately for execution of each individual statement. This reduces network traffic.
- Reuse of code: Stored procedures can be used multiple times. This eliminates the need to repetitively type out hundreds of Transact-SQL statements every time a similar task is to be performed.

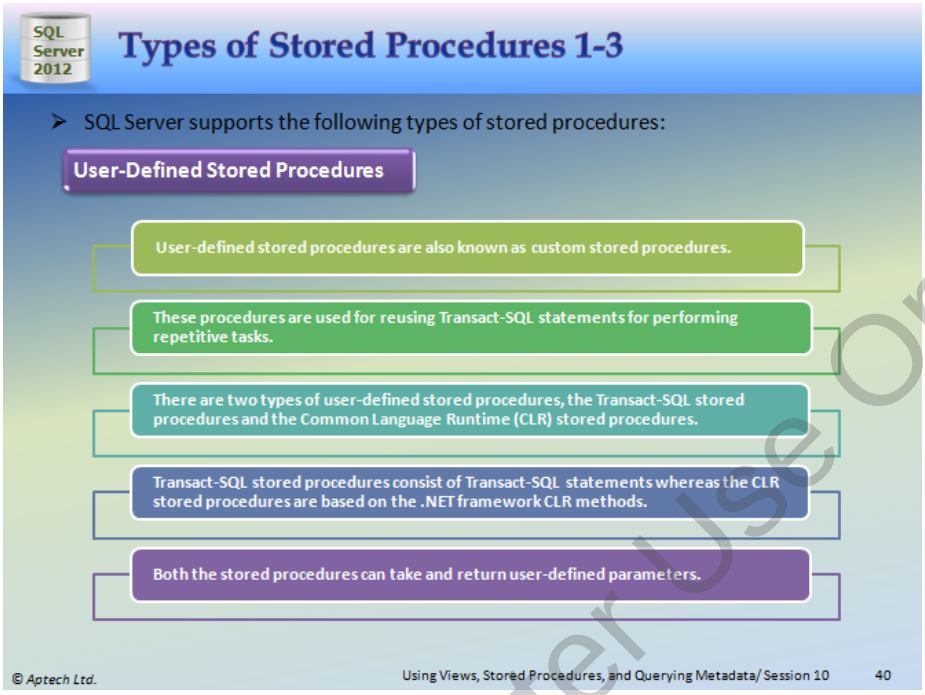
In-Class Question:

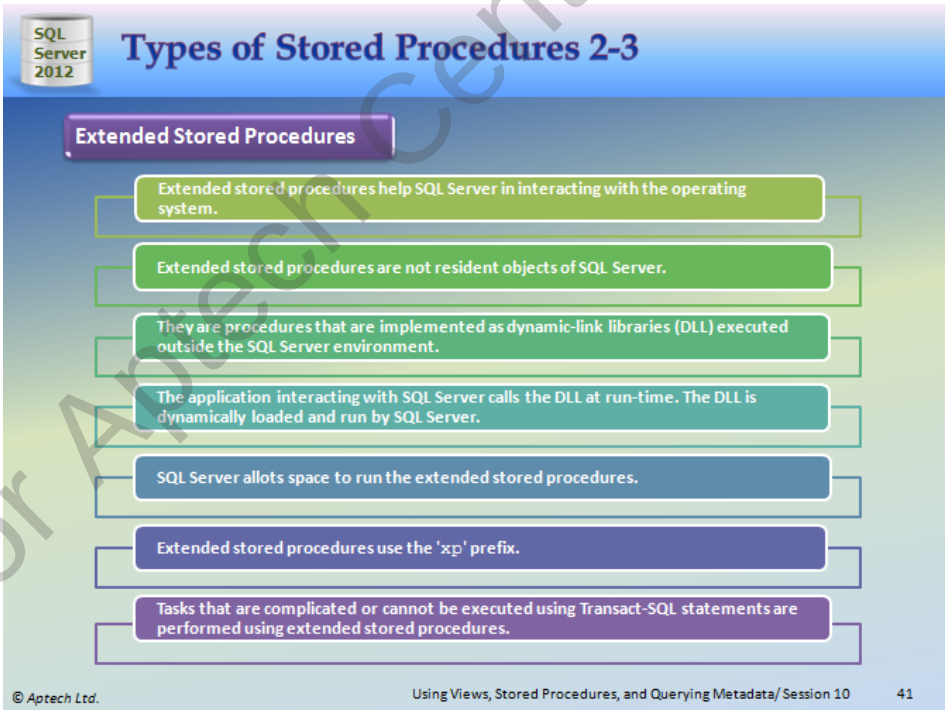
 What is a stored procedure?

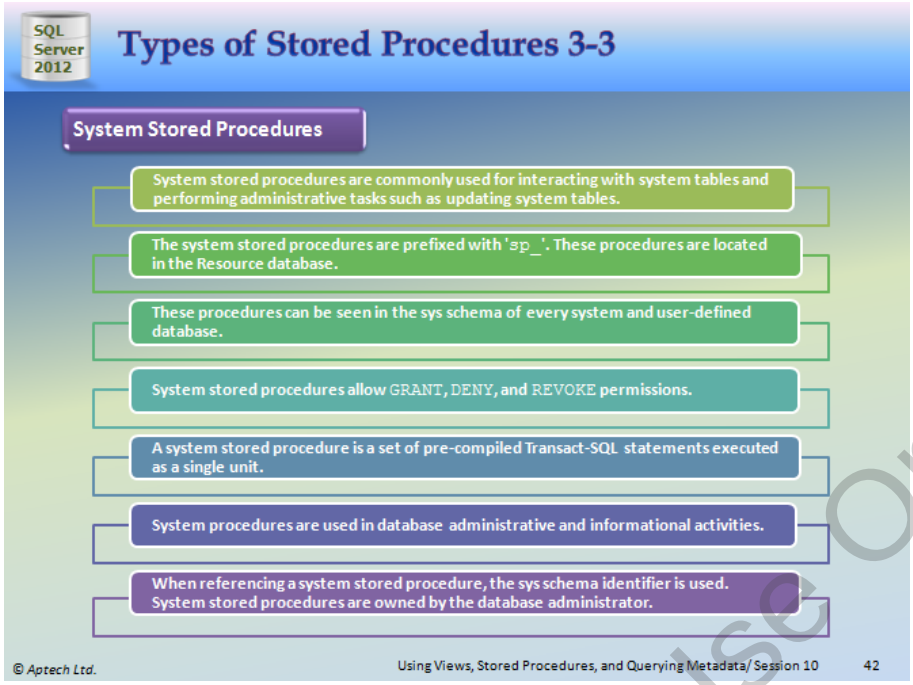
Answer:
A stored procedure is a group of Transact-SQL statements that act as a single block of code that performs a specific task.

Types of Stored Procedures

Slides 40 to 42







Using slides 40 to 42, explain the types of stored procedures.

SQL Server supports various types of stored procedures. These are described as follows:

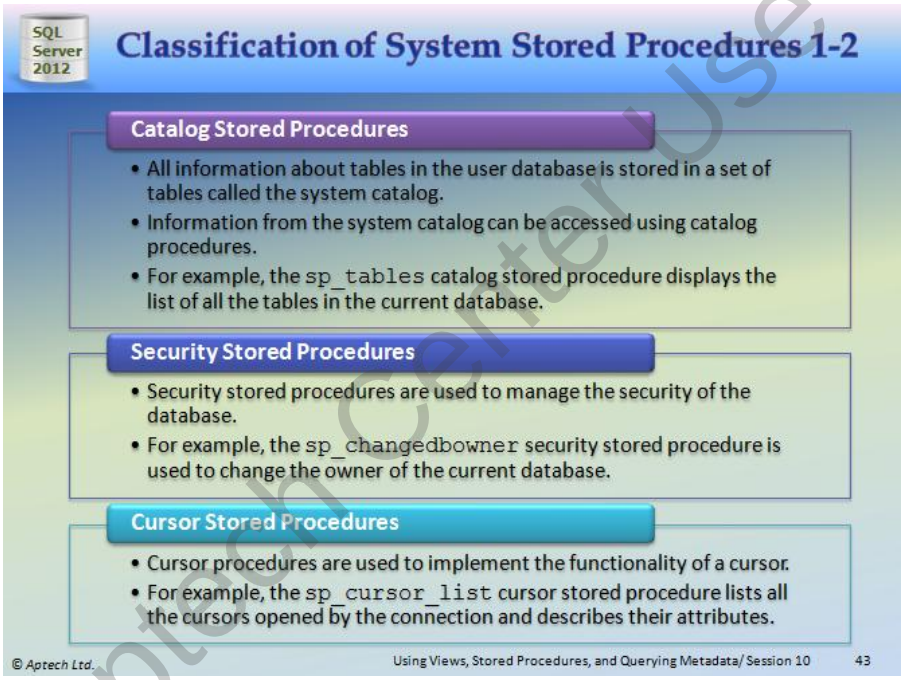
- **User-Defined Stored Procedures**: User-defined stored procedures are also known as custom stored procedures. These procedures are used for reusing Transact-SQL statements for performing repetitive tasks. There are two types of user-defined stored procedures, the Transact-SQL stored procedures and the Common Language Runtime (CLR) stored procedures.
  Transact-SQL stored procedures consists of Transact-SQL statements whereas the CLR stored procedures are based on the .NET framework CLR methods. Both the stored procedures can take and return user-defined parameters.
- **Extended Stored Procedures**: Extended stored procedures help SQL Server in interacting with the operating system. Extended stored procedures are not resident objects of SQL Server. They are procedures that are implemented as Dynamic-link Libraries (DLL) executed outside the SQL Server environment. The application interacting with SQL Server calls the DLL at run-time. The DLL is dynamically loaded and run by SQL Server. SQL Server allots space to run the extended stored procedures. Extended stored procedures use the 'xp' prefix. Tasks that are complicated or cannot be executed using Transact-SQL statements are performed using extended stored procedures.
- **System Stored Procedures**: System stored procedures are commonly used for interacting with system tables and performing administrative tasks such as updating system tables. The system stored procedures are prefixed with 'sp_'. These procedures are located in the Resource database. These procedures can be seen in the sys schema of every system and user-defined database. System stored procedures allow GRANT, DENY, and REVOKE permissions.
  A system stored procedure is a set of pre-compiled Transact-SQL statements executed as a single unit. System procedures are used in database administrative and informational activities. These procedures provide easy access to the metadata information about database objects such as system tables, user-defined tables, views, and indexes.

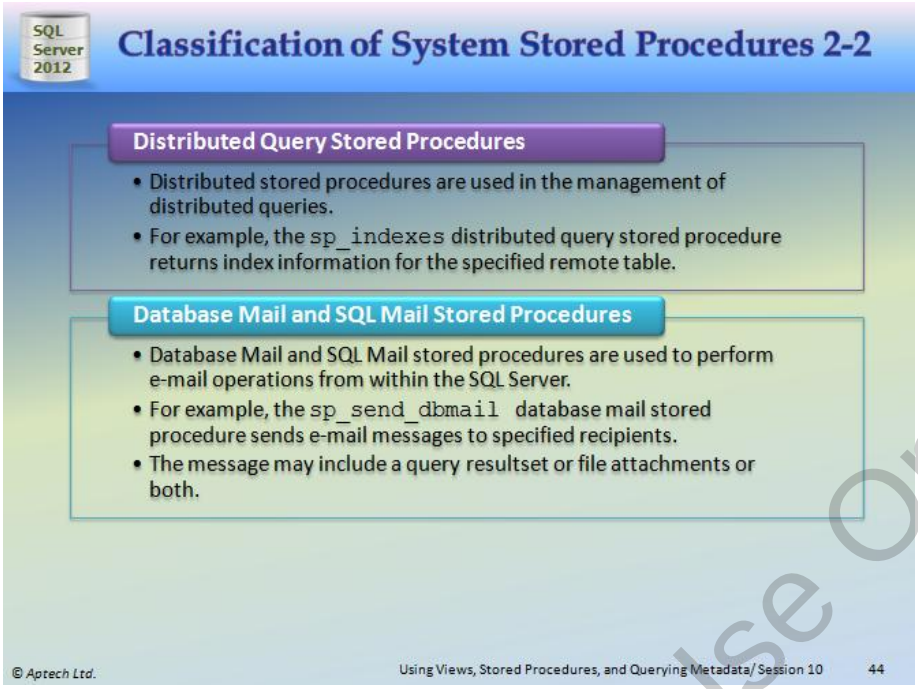
System stored procedures logically appear in the sys schema of system and user-defined databases. When referencing a system stored procedure, the sys schema identifier is used. The system stored procedures are stored physically in the hidden Resource database and have the sp_ prefix. System stored procedures are owned by the database administrator.

Mention system tables are created by default at the time of creating a new database. These tables store the metadata information about user-defined objects such as tables and views. Users cannot access or update the system tables using system stored procedures except through permissions granted by a database administrator.

Classification of System Stored Procedures

Slides 43 and 44





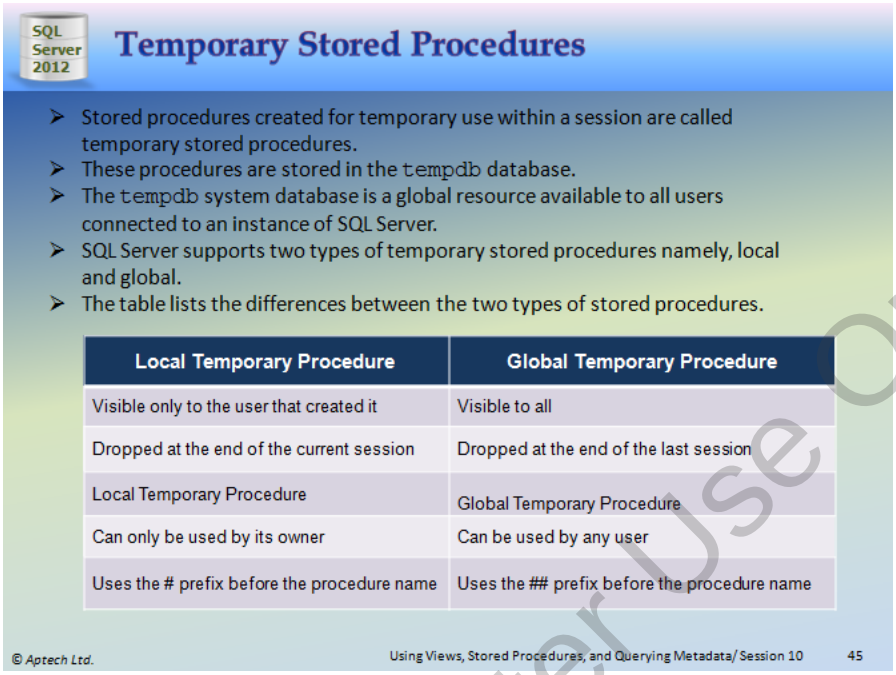
Using slides 43 and 44, explain the classification of system stored procedures.

System stored procedures can be classified into different categories depending on the tasks they perform. Some of the important categories are as follows:

- **Catalog Stored Procedures**: All information about tables in the user database is stored in a set of tables called the system catalog. Information from the system catalog can be accessed using catalog procedures. For example, the sp_tables catalog stored procedure displays the list of all the tables in the current database.
- **Security Stored Procedures**: Security stored procedures are used to manage the security of the database. For example, the sp_changedbowner security stored procedure is used to change the owner of the current database.
-
- **Cursor Stored Procedures**: Cursor procedures are used to implement the functionality of a cursor. For example, the sp_cursor_list cursor stored procedure lists all the cursors opened by the connection and describes their attributes.
- **Distributed Query Stored Procedures**: Distributed stored procedures are used in the management of distributed queries. For example, the sp_indexes distributed query stored procedure returns index information for the specified remote table.
- **Database Mail and SQL Mail Stored Procedures**: Database Mail and SQL Mail stored procedures are used to perform e-mail operations from within the SQL Server. For example, the sp_send_dbmail database mail stored procedure sends e-mail messages to specified recipients. The message may include a query resultset or file attachments or both.

Temporary Stored Procedures

Slide 45



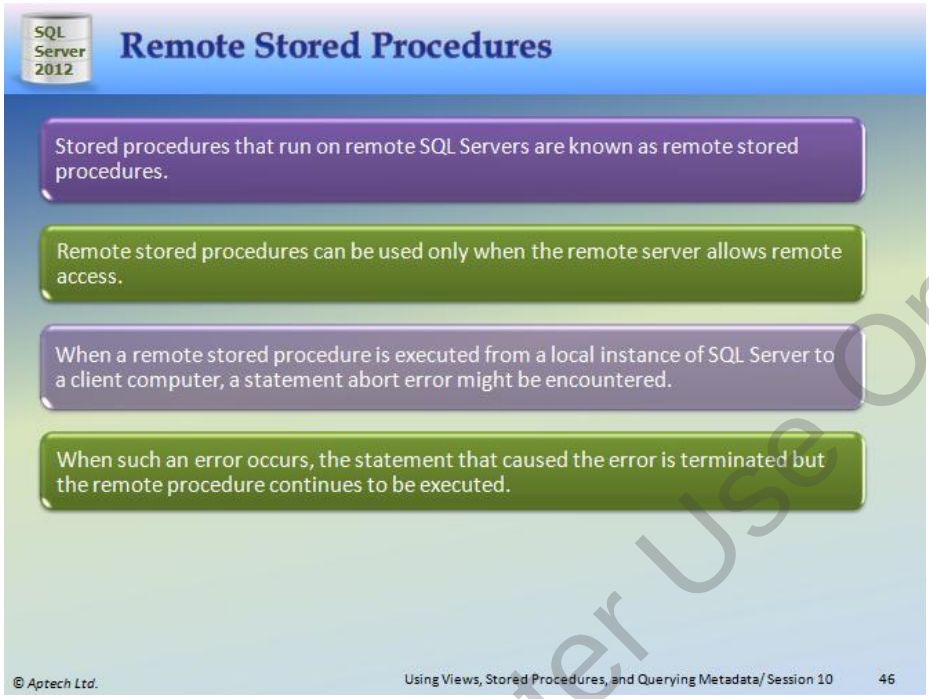
Using slide 45, explain the temporary stored procedures.

Stored procedures created for temporary use within a session are called temporary stored procedures. These procedures are stored in the tempdb database. The tempdb system database is a global resource available to all users connected to an instance of SQL Server. It holds all temporary tables and temporary stored procedures.

SQL Server supports two types of temporary stored procedures namely, local and global. The differences between the two types are given in table.

Remote Stored Procedures

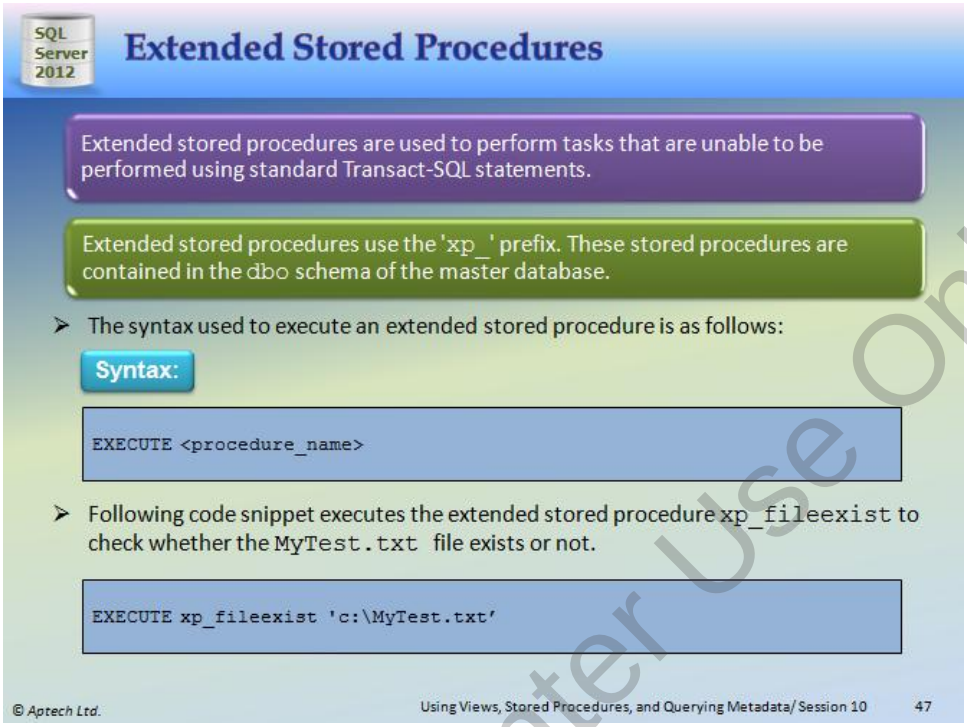
Slide 46



Using slide 46, explain the remote stored procedures.
Stored procedures that run on remote SQL Servers are known as remote stored procedures. Remote stored procedures can be used only when the remote server allows remote access. When a remote stored procedure is executed from a local instance of SQL Server to a client computer, a statement abort error might be encountered. When such an error occurs, the statement that caused the error is terminated but the remote procedure continues to be executed.

Extended Stored Procedures

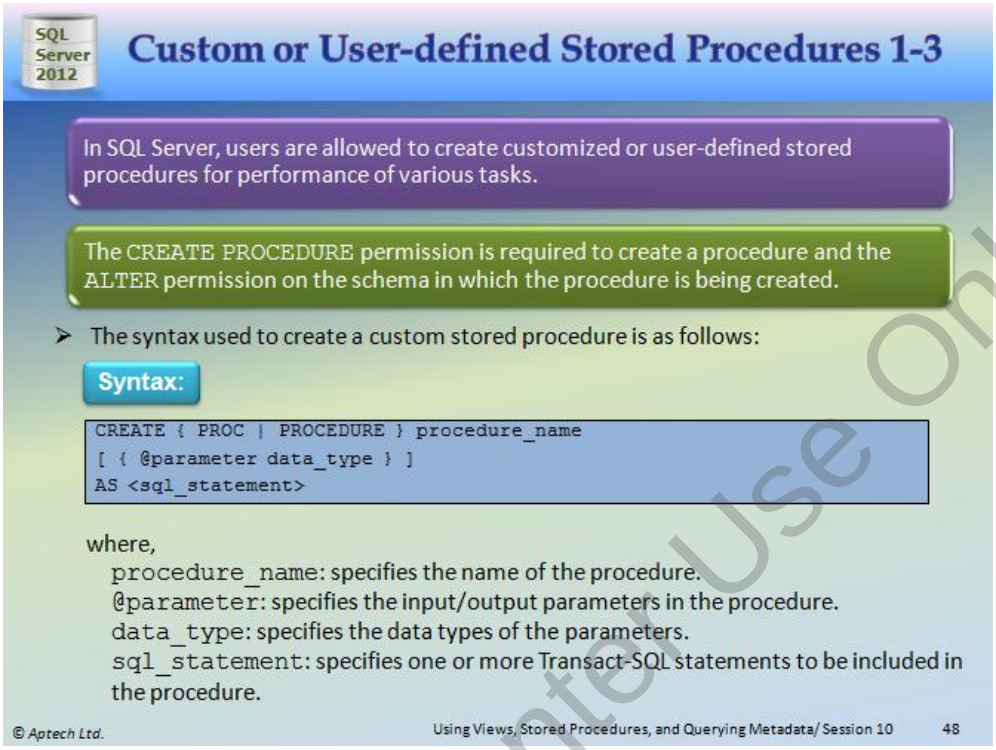
Slide 47

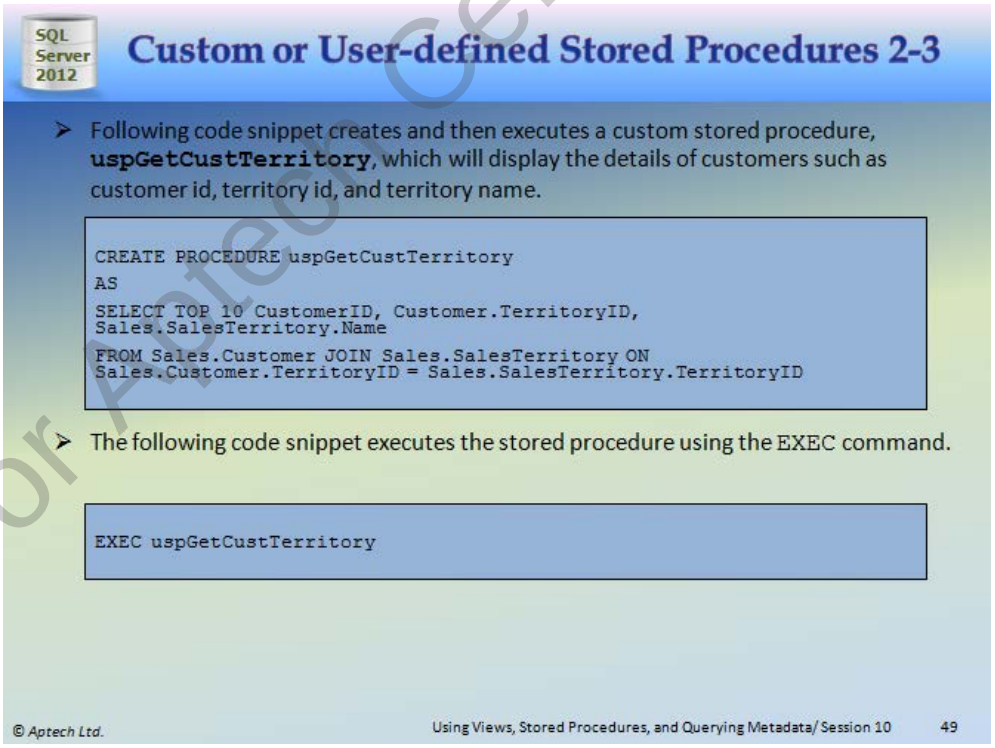


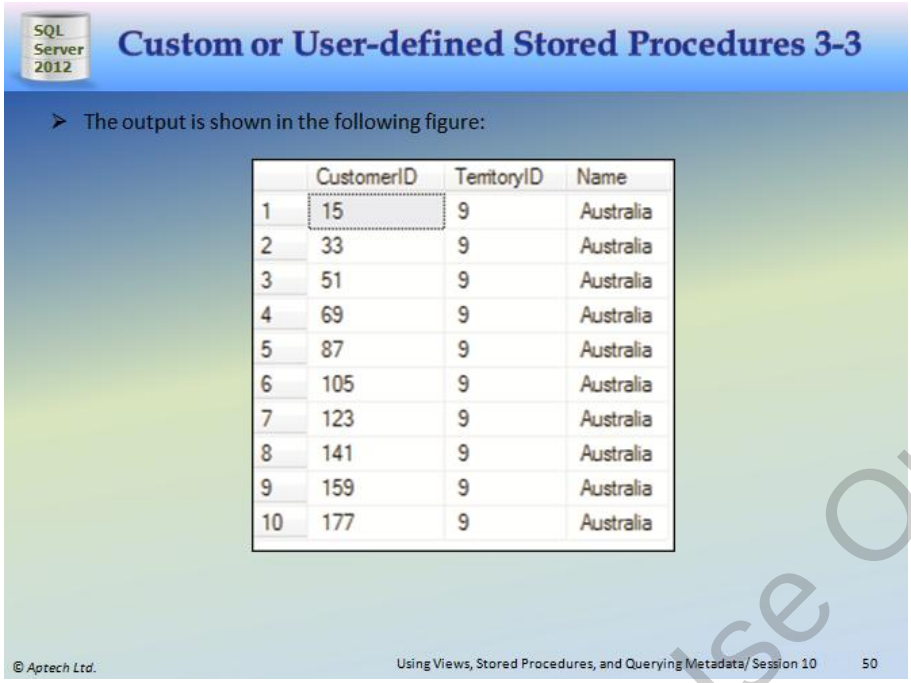
Using slide 47, explain the remote stored procedures.
Extended stored procedures are used to perform tasks that are unable to be performed using standard Transact-SQL statements. Extended stored procedures use the 'xp_' prefix. These stored procedures are contained in the dbo schema of the master database.
The following syntax is used to execute an extended stored procedure.
Explain the Code Snippet that executes the extended stored procedure xp_file exist to check whether the MyTest.txt file exists or not.
Mention when an extended stored procedure is executed, either in a batch or in a module, qualify the stored procedure name with master.dbo.

Custom or User-defined Stored Procedures

Slides 48 to 50







Using slides 48 to 50, explain the custom or user-defined stored procedures.

In SQL Server, users are allowed to create customized stored procedures for performance of various tasks. Such stored procedures are referred to as user-defined or custom stored procedures.

For example, consider a table Customer_Details that stores the details about all the customers. You would need to type out Transact-SQL statements every time you wished to view the details about the customers. Instead, you could create a custom stored procedure that would display these details whenever the procedure is executed.

Creating a custom stored procedure requires CREATE PROCEDURE permission in the database and ALTER permission on the schema in which the procedure is being created.

Explain the syntax used to create a custom stored procedure.

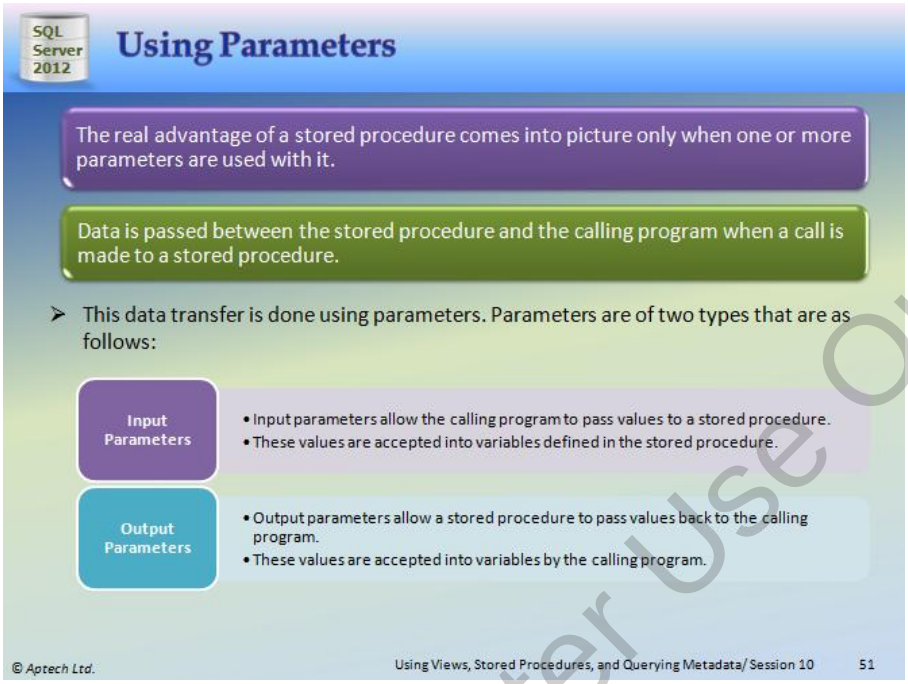
Explain the Code Snippet that creates and then executes a custom stored procedure, uspGetCustTerritory, which will display the details of customers such as customer id, territory id, and territory name.

To execute the stored procedure, the EXEC command is used as shown in the Code Snippet on side 49.

The output is shown in the figure on slide 50.

Using Parameters

Slide 51



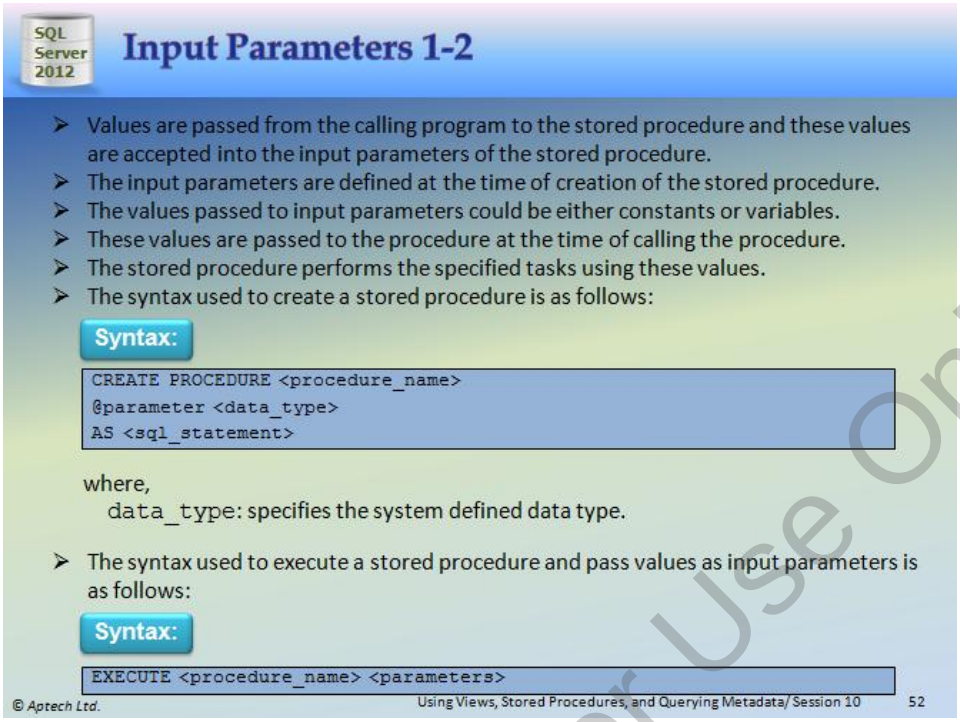
Using slide 51, explain how to use parameters.

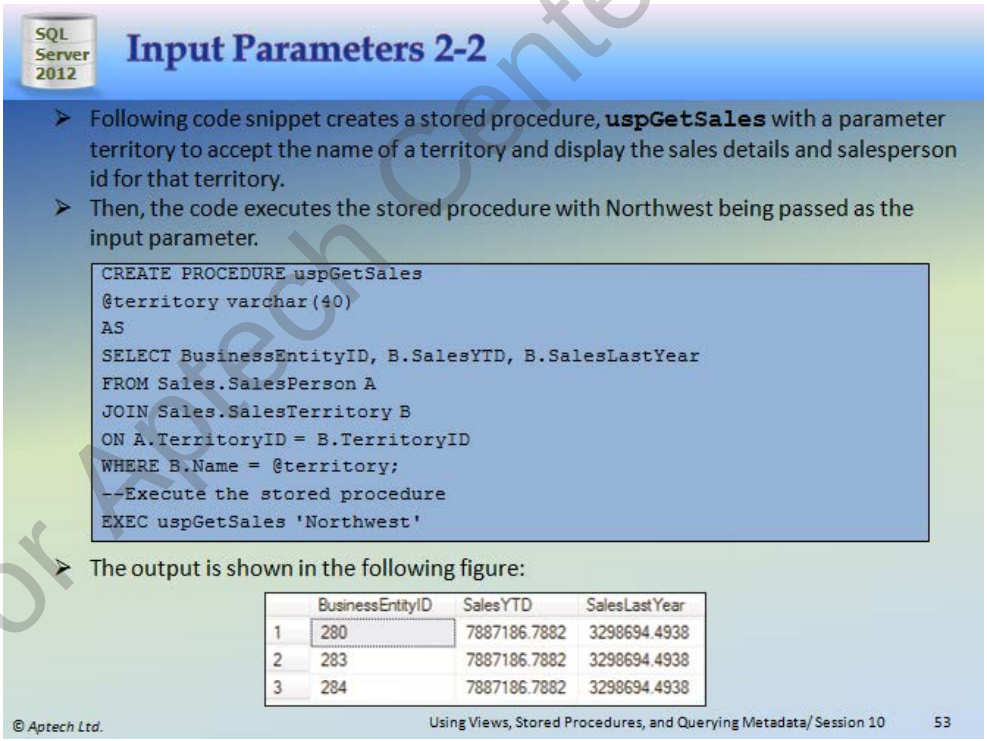
The real advantage of a stored procedure comes into picture only when one or more parameters are used with it. Data is passed between the stored procedure and the calling program when a call is made to a stored procedure. This data transfer is done using parameters. Parameters are of two types that are as follows:

- Input Parameters: Input parameters allow the calling program to pass values to a stored procedure. These values are accepted into variables defined in the stored procedure.
- Output Parameters: Output parameters allow a stored procedure to pass values back to the calling program. These values are accepted into variables by the calling program.

Input Parameters

Slides 52 and 53





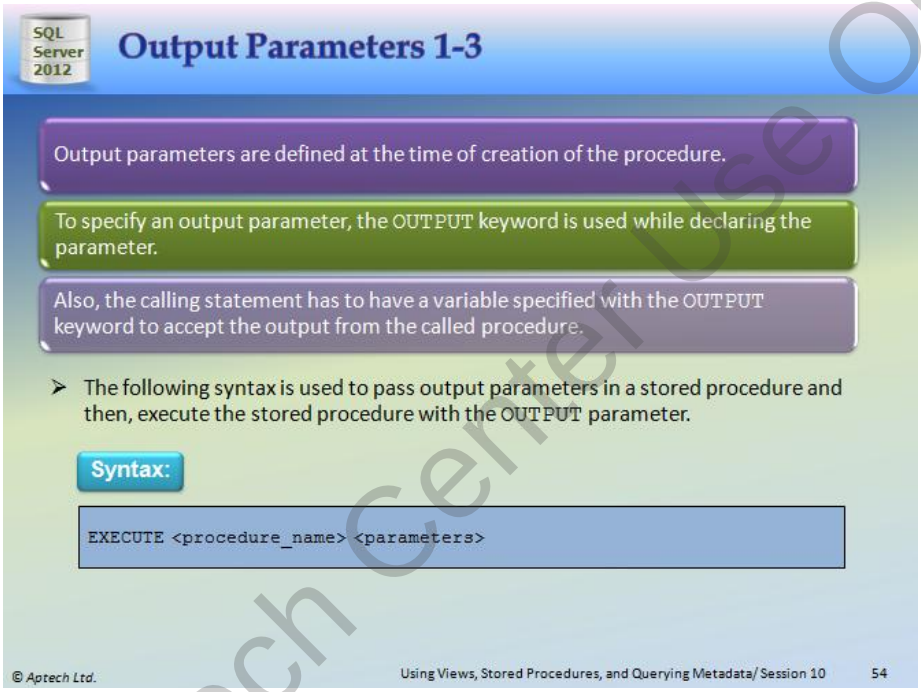
Using slides 52 and 53, explain the input parameters.

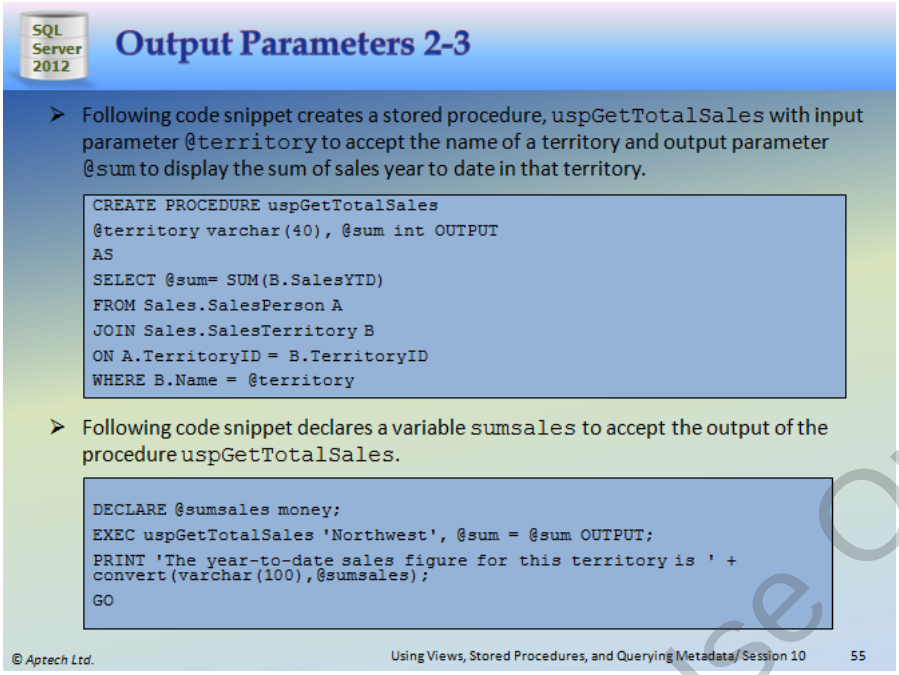
Values are passed from the calling program to the stored procedure and these values are accepted into the input parameters of the stored procedure. The input parameters are defined at the time of creation of the stored procedure. The values passed to input parameters could be either constants or variables. These values are passed to the procedure at the time of calling the procedure. The stored procedure performs the specified tasks using these values.

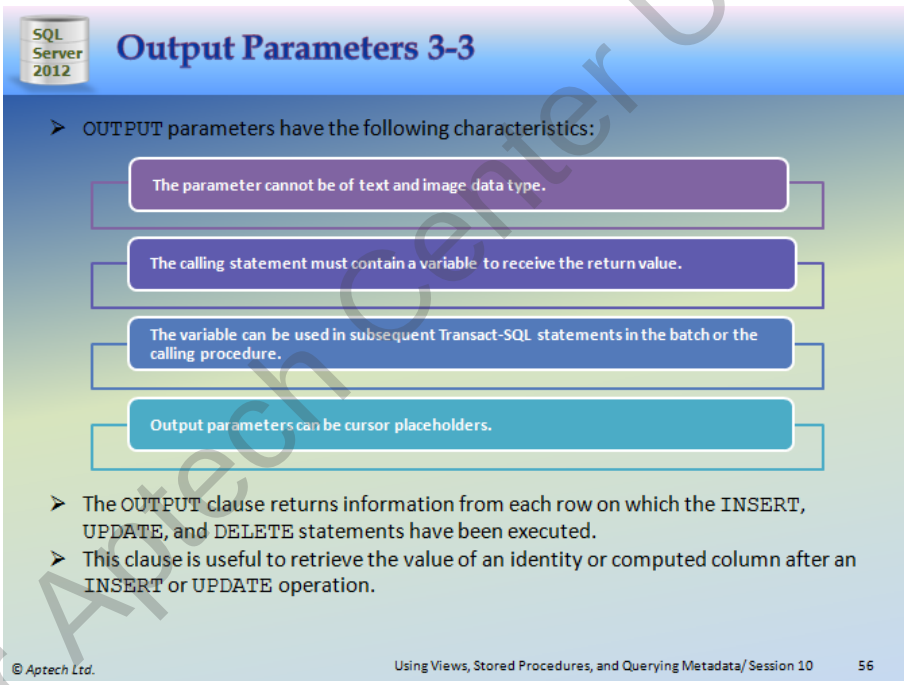
Explain the used to create a stored procedure.
Explain the syntax used to execute a stored procedure and pass values as input parameters.
Explain the Code Snippet that creates a stored procedure, uspGetSales with a parameter territory to accept the name of a territory and display the sales details and salesperson id for that territory. Then, the code executes the stored procedure with Northwest being passed as the input parameter.
The output is shown in the figure on slide 53.

Output Parameters

Slides 54 to 56







Using slides 54 to 56, explain the output parameters.

Stored procedures occasionally need to return output back to the calling program. This transfer of data from the stored procedure to the calling program is performed using output parameters.

Output parameters are defined at the time of creation of the procedure. To specify an output parameter, the OUTPUT keyword is used while declaring the parameter. Also, the calling statement has to have a variable specified with the OUTPUT keyword to accept the output from the called procedure.

The following syntax is used to pass output parameters in a stored procedure and then, execute the stored procedure with the OUTPUT parameter specified.

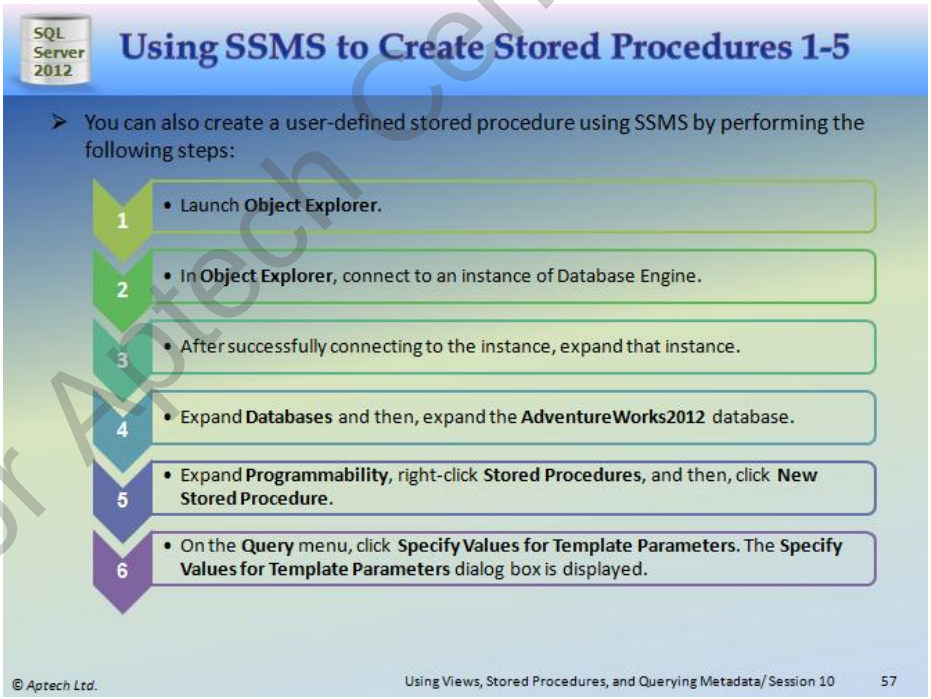
Explain the Code Snippet that creates a stored procedure, uspGetTotalSales with input parameter @territory to accept the name of a territory and output parameter @sum to display the sum of sales year to date in that territory.
Explain the Code Snippet that declares a variable sumsales to accept the output of the procedure uspGetTotalSales.
The code passes Northwest as the input to the uspGetTotalSales stored procedure and accepts the output in the variable sumsales. The output is printed using the PRINT command.
OUTPUT parameters have the following characteristics:

- The parameter cannot be of text and image data type.
- The calling statement must contain a variable to receive the return value.
- The variable can be used in subsequent Transact-SQL statements in the batch or the calling procedure.
- Output parameters can be cursor placeholders.

The OUTPUT clause returns information from each row on which the INSERT, UPDATE, and DELETE statements have been executed. This clause is useful to retrieve the value of an identity or computed column after an INSERT or UPDATE operation.

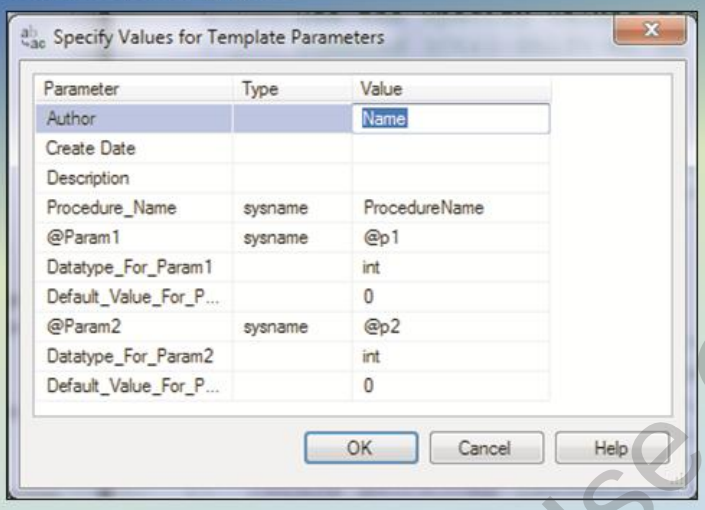
Using SSMS to Create Stored Procedures

Slides 57 to 61



Using SSMS to Create Stored Procedures 2-5

- This is shown in the following figure:

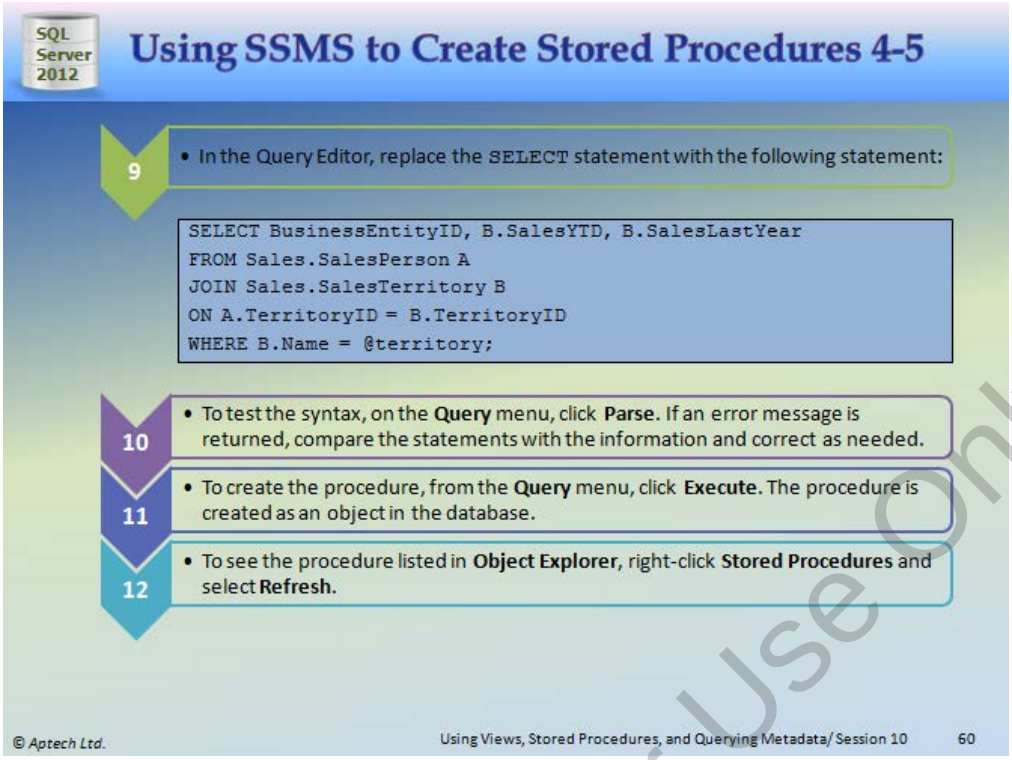


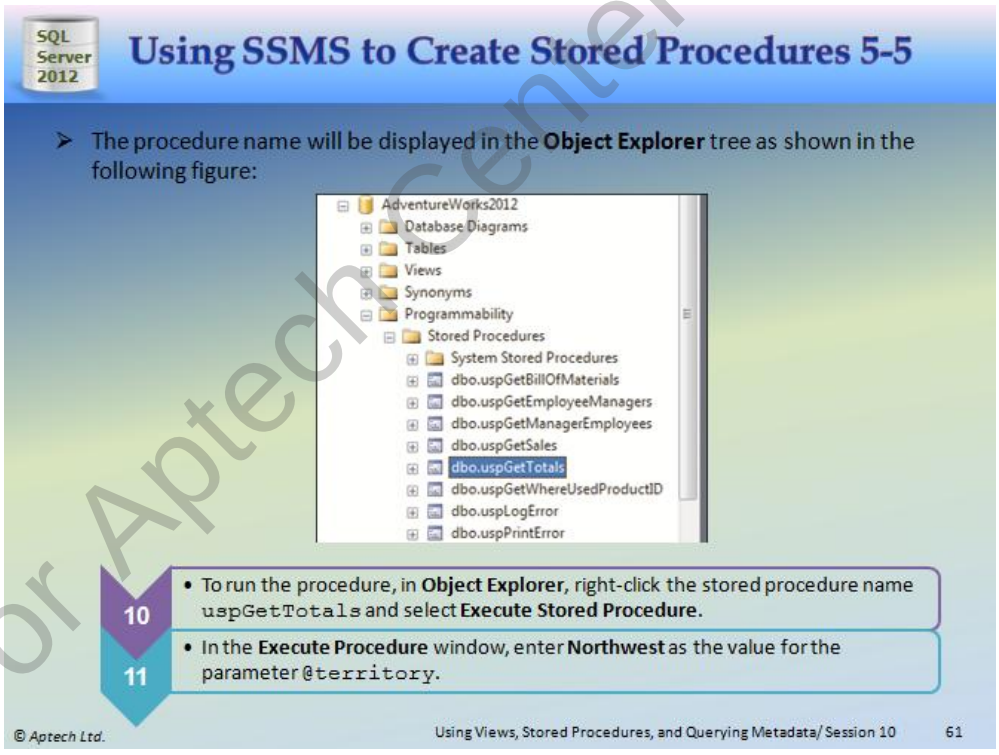
Using SSMS to Create Stored Procedures 3-5

7. In the **Specify Values for Template Parameters** dialog box, enter the values for the parameters as shown in the following table:

Parameter	Value
Author	Your name
Create Date	Today's date
Description	Returns year to sales data for a territory
Procedure_Name	uspGetTotals
@Param1	@territory
@Datatype_For_Param1	varchar(50)
Default_Value_For_Param1	NULL
@Param2	
@Datatype_For_Param2	
Default_Value_For_Param2	

8. After entering these details, click **OK**.

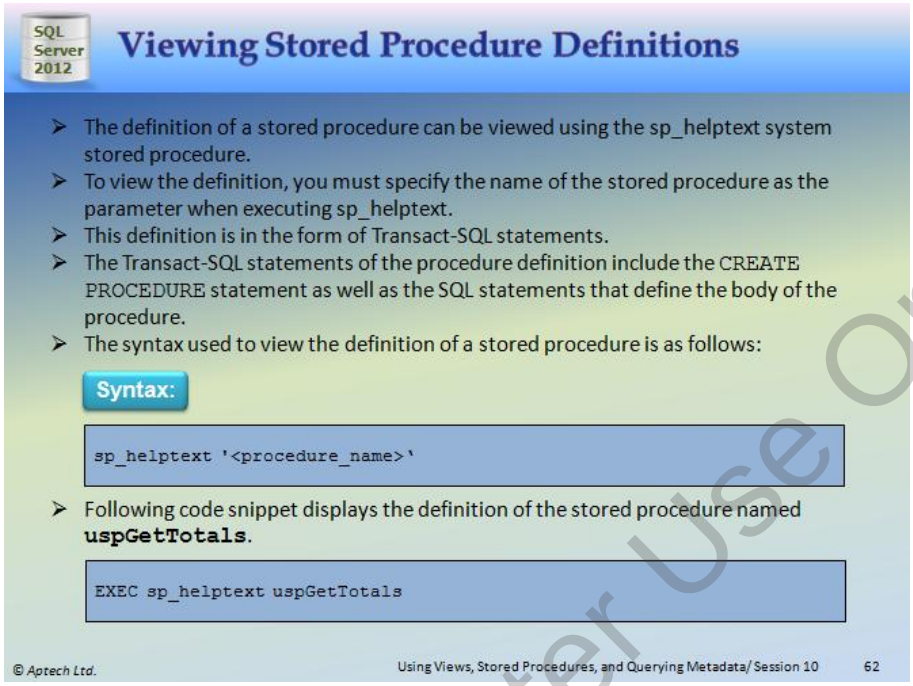




Using slides 57 to 61, explain the steps to create stored procedure using SSMS in details. Explain the values for the parameters as shown in table on slide 59.
The procedure name will be displayed in the Object Explorer tree as shown in figure on slide 61.

Viewing Stored Procedure Definitions

Slide 62



Using slide 62, explain how to view stored procedure definitions.

The definition of a stored procedure can be viewed using the sp_helptext system stored procedure. To view the definition, you must specify the name of the stored procedure as the parameter when executing sp_helptext. This definition is in the form of Transact-SQL statements.

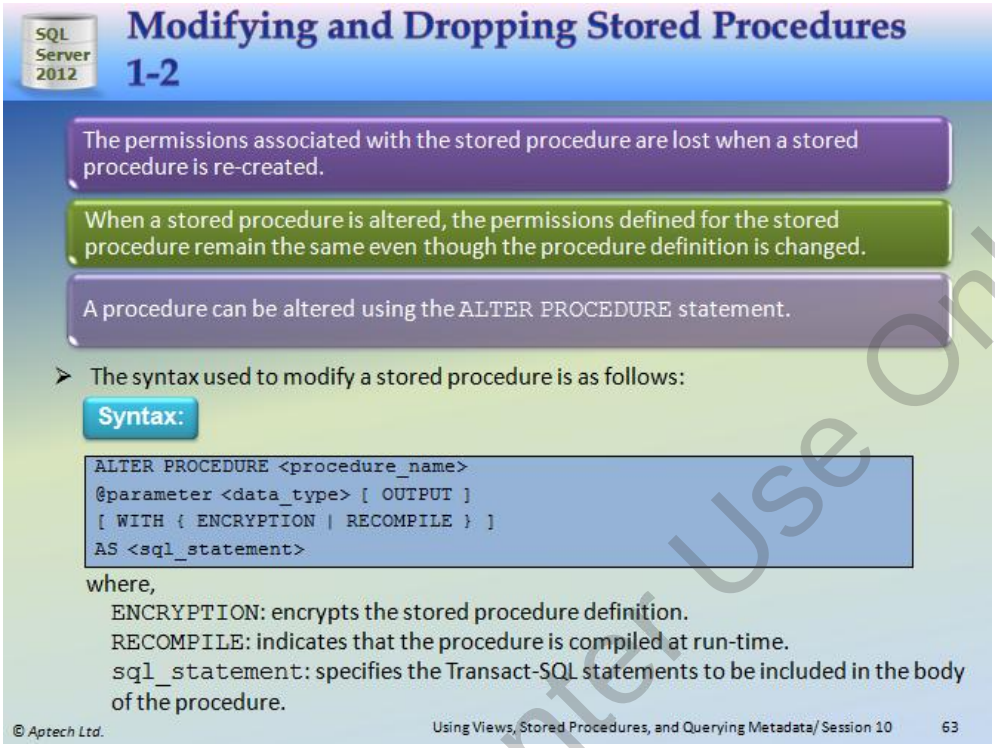
The Transact-SQL statements of the procedure definition include the CREATE PROCEDURE statement as well as the SQL statements that define the body of the procedure.

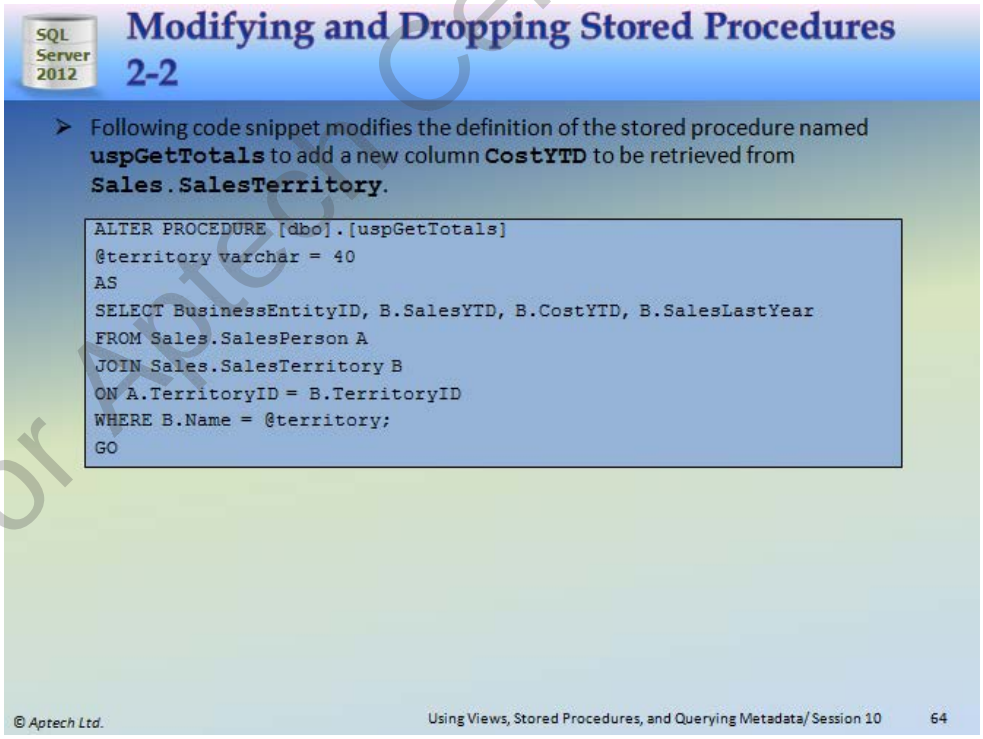
Explain the syntax used to view the definition of a stored procedure.

Explain the Code Snippet that displays the definition of the stored procedure named uspGetTotals.

Modifying and Dropping Stored Procedures

Slides 63 and 64



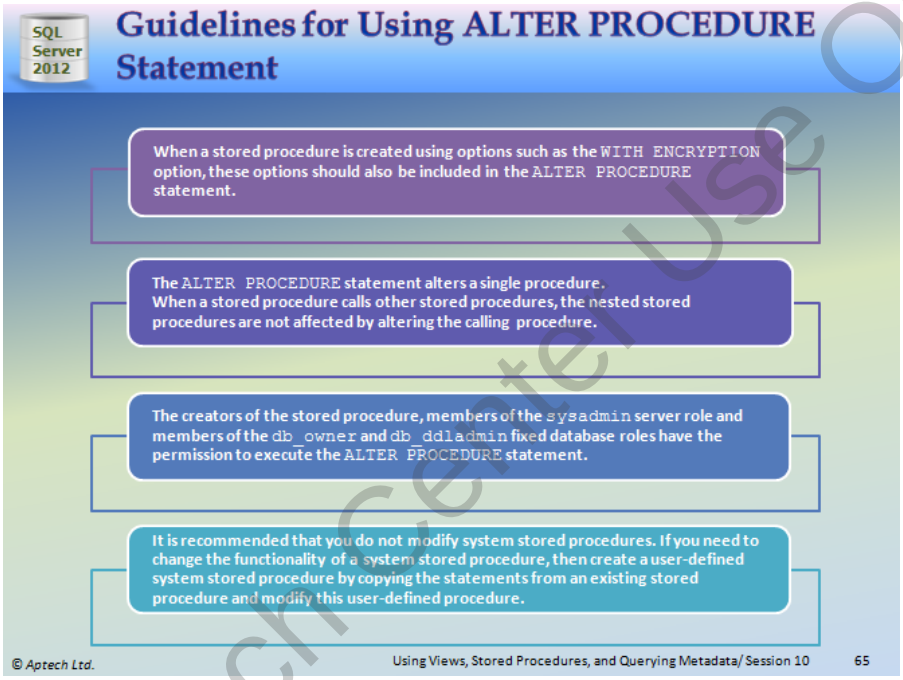


Using slides 63 and 64, explain how to modify and drop stored procedures.
The permissions associated with the stored procedure are lost when a stored procedure is re-created. However, when a stored procedure is altered, the permissions defined for the stored procedure remains the same even though the procedure definition is changed.

A procedure can be altered using the ALTER PROCEDURE statement.
Explain the syntax is used to modify a stored procedure.
Explain the Code Snippet that modifies the definition of the stored procedure named uspGetTotals to add a new column CostYTD to be retrieved from Sales.SalesTerritory.
Mention when you change the definition of a stored procedure, the dependent objects may fail when executed. This happens if the dependent objects are not updated to reflect the changes made to the stored procedure.

Guidelines for Using ALTER PROCEDURE Statement

Slide 65

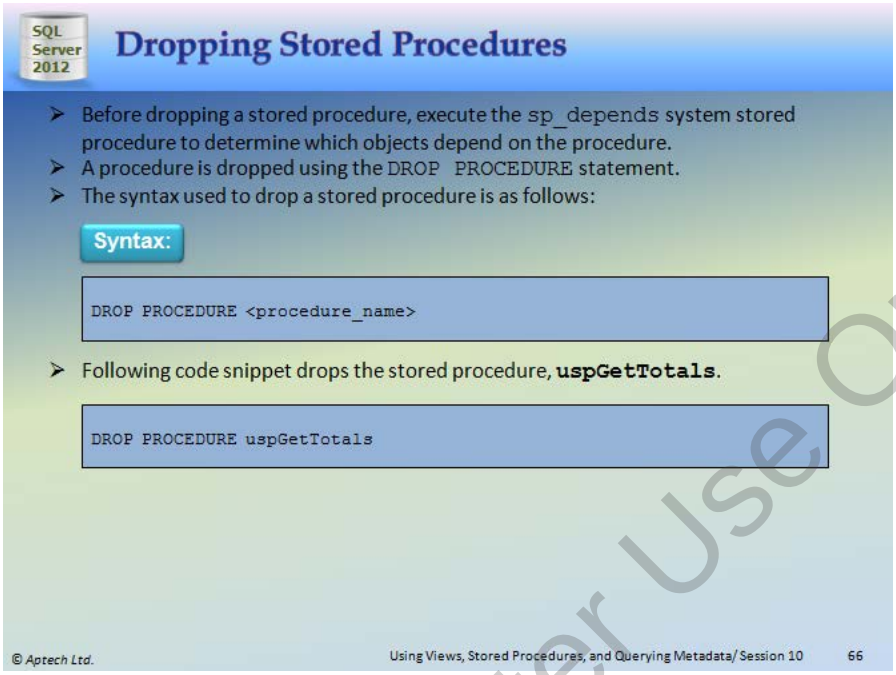


Using slide 65, explain the guidelines for using alter procedure statement.
Stored procedures are altered using the ALTER PROCEDURE statement. Following facts have to be considered while using the ALTER PROCEDURE statement:

- When a stored procedure is created using options such as the WITH ENCRYPTION option, these options should also be included in the ALTER PROCEDURE statement.
- The ALTER PROCEDURE statement alters a single procedure. When a stored procedure calls other stored procedures, the nested stored procedures are not affected by altering the calling procedure.
- The creators of the stored procedure, members of the sysadmin server role and members of the db_owner and db_ddladmin fixed database roles have the permission to execute the ALTER PROCEDURE statement.
- It is recommended that you do not modify system stored procedures. If you need to change the functionality of a system stored procedure, then create a user-defined system stored procedure by copying the statements from an existing stored procedure and modify this user-defined procedure.

Dropping Stored Procedures

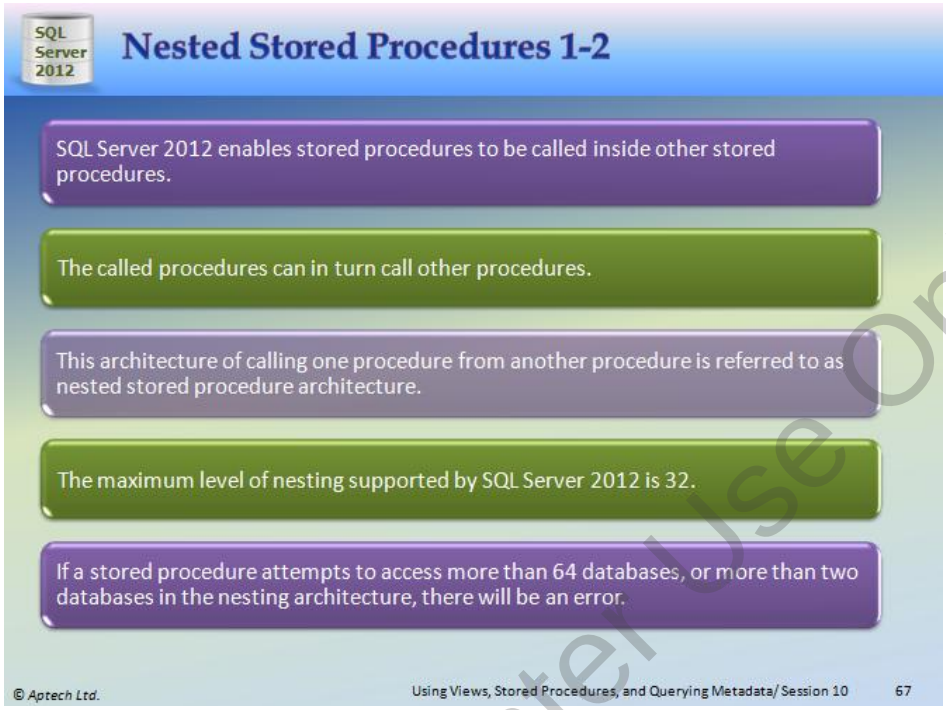
Slide 66

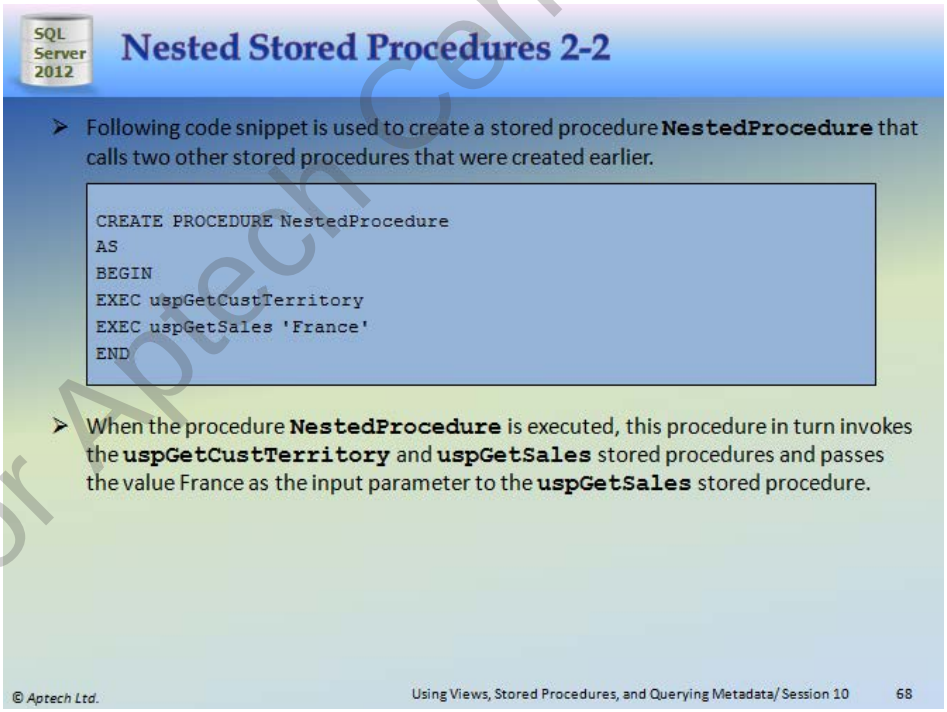


Using slide 66, explain how to drop stored procedures.
Stored procedures can be dropped if they are no longer needed. If another stored procedure calls a deleted procedure, an error message is displayed.
If a new procedure is created using the same name as well as the same parameters as the dropped procedure, all calls to the dropped procedure will be executed successfully. This is because they will now refer to the new procedure, which has the same name and parameters as the deleted procedure.
Before dropping a stored procedure, execute the sp_depends system stored procedure to determine which objects depend on the procedure.
A procedure is dropped using the DROP PROCEDURE statement.
Explain the syntax used to drop a stored procedure.
Explain the Code Snippet that drops the stored procedure, uspGetTotals.

Nested Stored Procedures

Slides 67 and 68





Using slides 67 and 68, explain the nested stored procedures.

SQL Server 2012 enables stored procedures to be called inside other stored procedures. The called procedures can in turn call other procedures. This architecture of calling one procedure from another procedure is referred to as nested stored procedure architecture. When a stored procedure calls another stored procedure, the level of nesting is said to be increased by one. Similarly, when a called procedure completes its execution and passes

control back to the calling procedure, the level of nesting is said to be decreased by one. The maximum level of nesting supported by SQL Server 2012 is 32. If the level of nesting exceeds 32, the calling process fails. Also, note that if a stored procedure attempts to access more than 64 databases, or more than two databases in the nesting architecture, there will be an error.

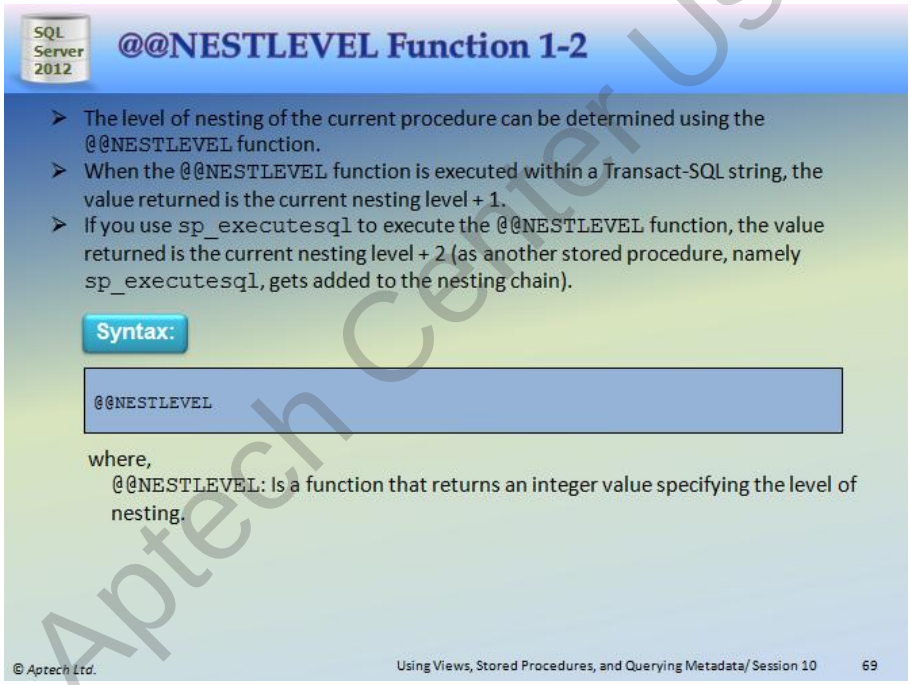
Explain the Code Snippet that is used to create a stored procedure NestedProcedure that calls two other stored procedures.

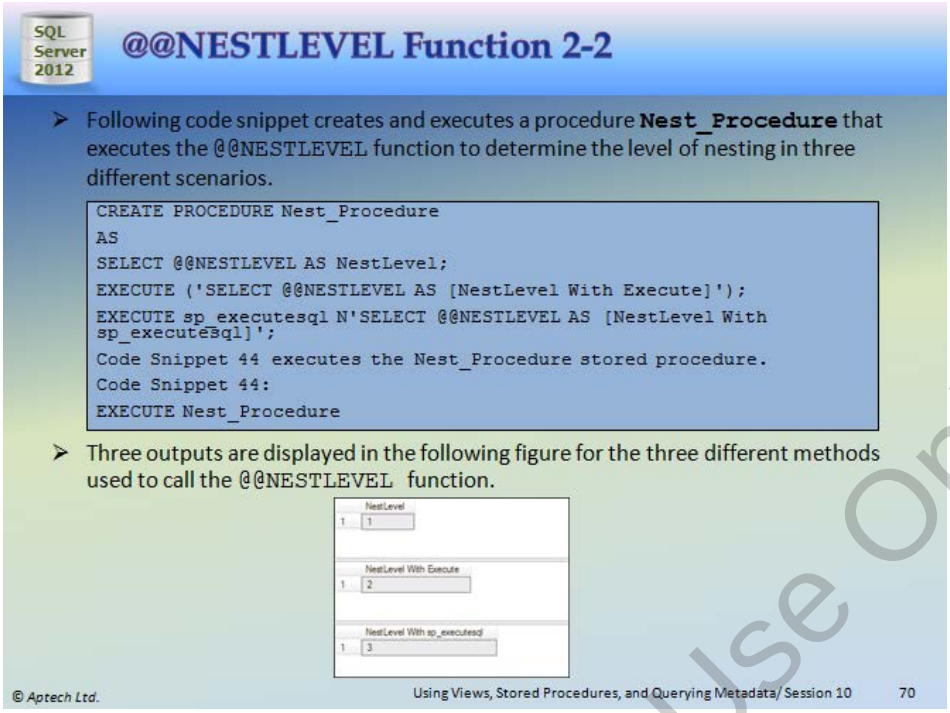
When the procedure NestedProcedure is executed, this procedure in turn invokes the uspGetCustTerritory and uspGetSales stored procedures and passes the value France as the input parameter to the uspGetSales stored procedure.

Mention, although there can be a maximum of 32 levels of nesting, there is no limit as to the number of stored procedure that can be called from a given stored procedure.

@@NESTLEVEL Function

Slides 69 and 70





Using slides 69 and 70, explain the @@nestlevel function.

The level of nesting of the current procedure can be determined using the @@NESTLEVEL function. When the @@NESTLEVEL function is executed within a Transact-SQL string, the value returned is the current nesting level + 1.

If you use sp_executesql to execute the @@NESTLEVEL function, the value returned is the current nesting level + 2 (as another stored procedure, namely, sp_executesql, gets added to the nesting chain).

Explain the Code Snippet that creates and executes a procedure Nest_Procedure that executes the @@NESTLEVEL function to determine the level of nesting in three different scenarios.

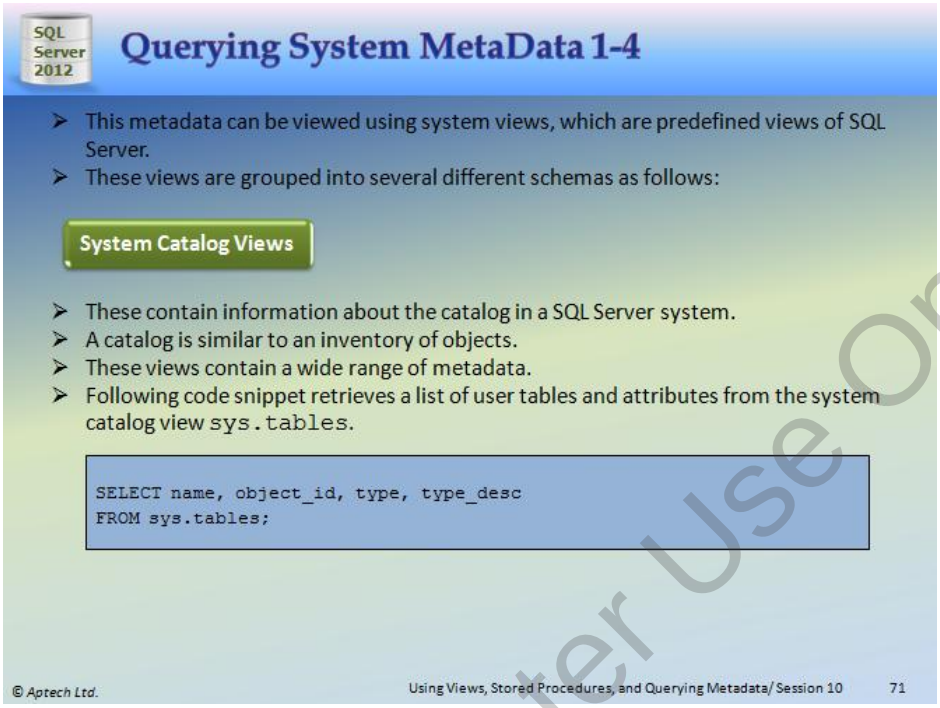
Explain the Code Snippet that executes the Nest_Procedure stored procedure.

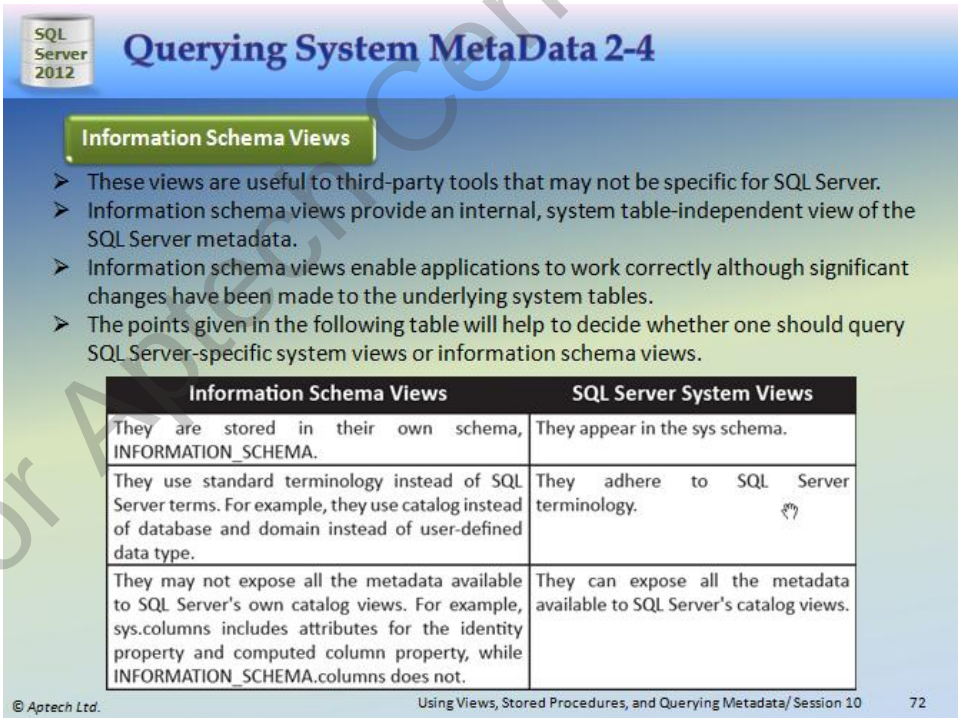
Three outputs are displayed in the figure on slide 70 for the three different methods used to call the @@NESTLEVEL function.

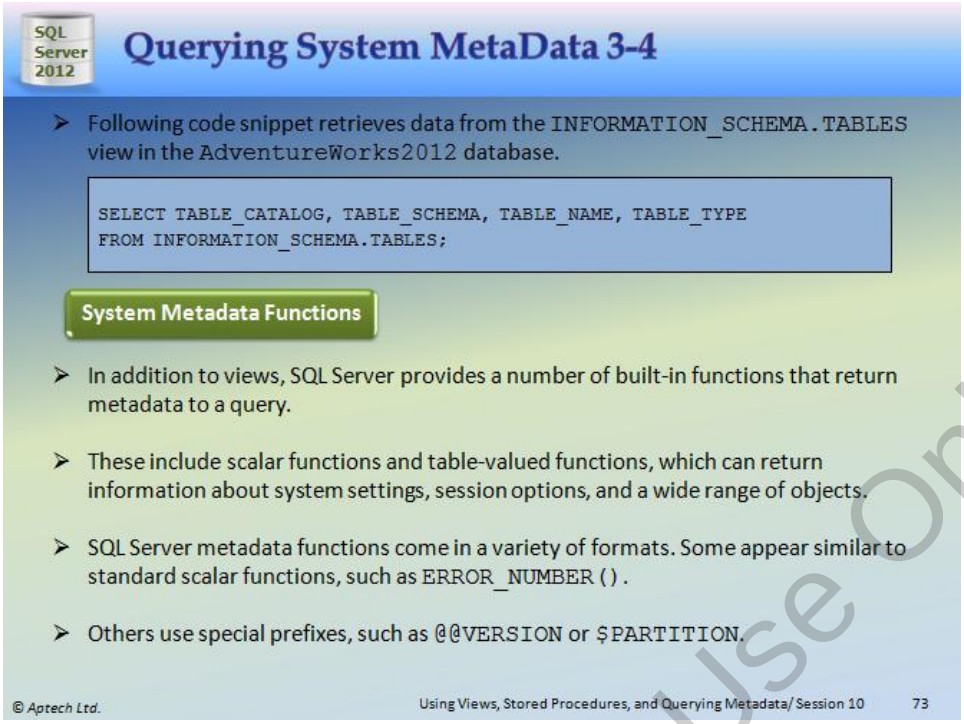
Mention, the sp_executesql stored procedure can also be used to execute Transact-SQL statements.

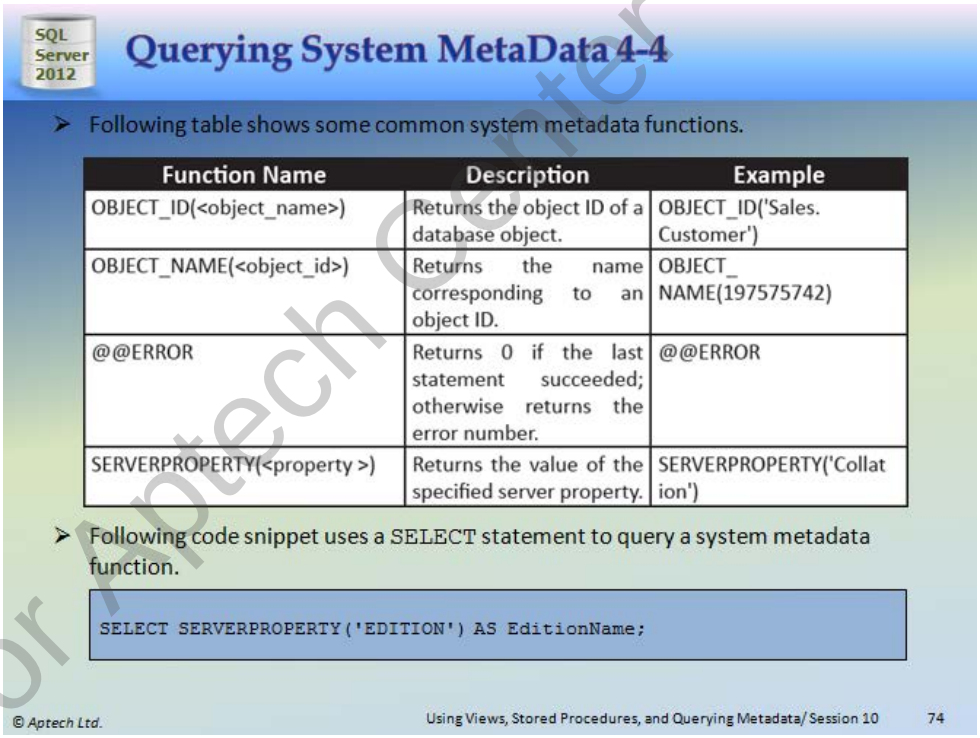
Querying System MetaData

Slides 71 to 74









Using slides 71 to 74, explain how to query system metadata.

The properties of an object such as a table or a view are stored in special system tables. These properties are referred to as metadata. All SQL objects produce metadata. This metadata can be viewed using system views, which are predefined views of SQL Server. There are over 230 different system views and these are automatically inserted into the user created database. These views are grouped into several different schemas.

- **System Catalog Views**: These contain information about the catalog in a SQL Server system. A catalog is similar to an inventory of objects. These views contain a wide range of metadata. In earlier versions of SQL Server, users were required to query a large

number of system tables, system views, and system functions. In SQL Server 2012, all user-accessible catalog metadata can easily be found by querying just the catalog views.
Explain the Code Snippet that retrieves a list of user tables and attributes from the system catalog view sys.tables.

- **Information Schema**: View users can query information schema views to return system metadata. These views are useful to third-party tools that may not be specific for SQL Server. Information schema views provide an internal, system table-independent view of the SQL Server metadata. Information schema views enable applications to work correctly although significant changes have been made to the underlying system tables. The points given in table will help to decide whether one should query SQL Server-specific system views or information schema views.
  Explain the Code Snippet that retrieves data from the INFORMATION_SCHEMA.TABLES view in the AdventureWorks2012 database.
- **System Metadata Functions**: In addition to views, SQL Server provides a number of built-in functions that return metadata to a query. These include scalar functions and table-valued functions, which can return information about system settings, session options, and a wide range of objects.
  SQL Server metadata functions come in a variety of formats. Some appear similar to standard scalar functions, such as ERROR_NUMBER(). Others use special prefixes, such as @@VERSION or $PARTITION. Table shows some common system metadata functions.
  Explain the Code Snippet that uses a SELECT statement to query a system metadata function.

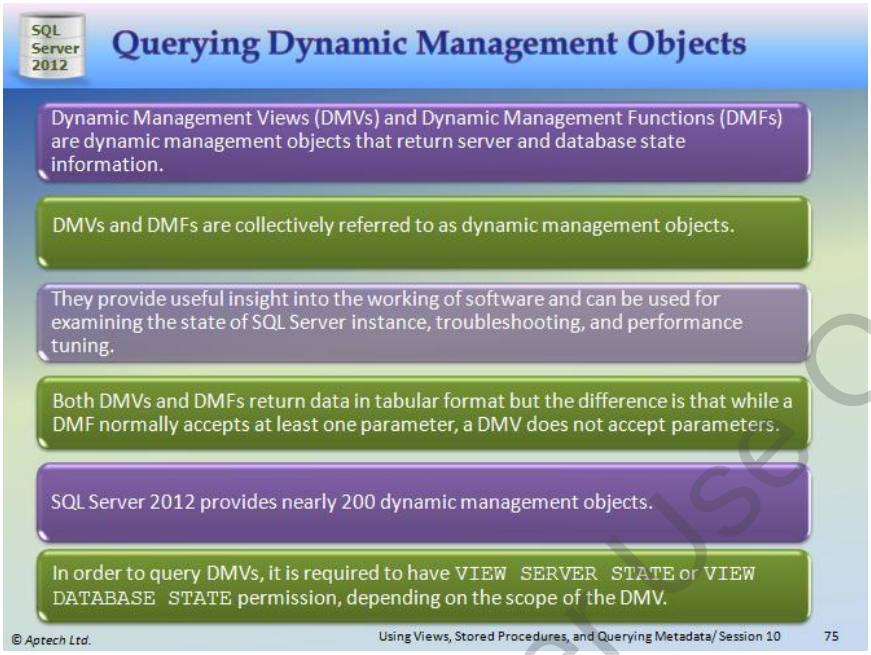
In-Class Question:

 What is the use of @@error?

Answer:
@@Error is used to detect the error in the execution of statements. It returns 0 if the last statement was executed successfully or else return the error number.

Querying Dynamic Management Objects

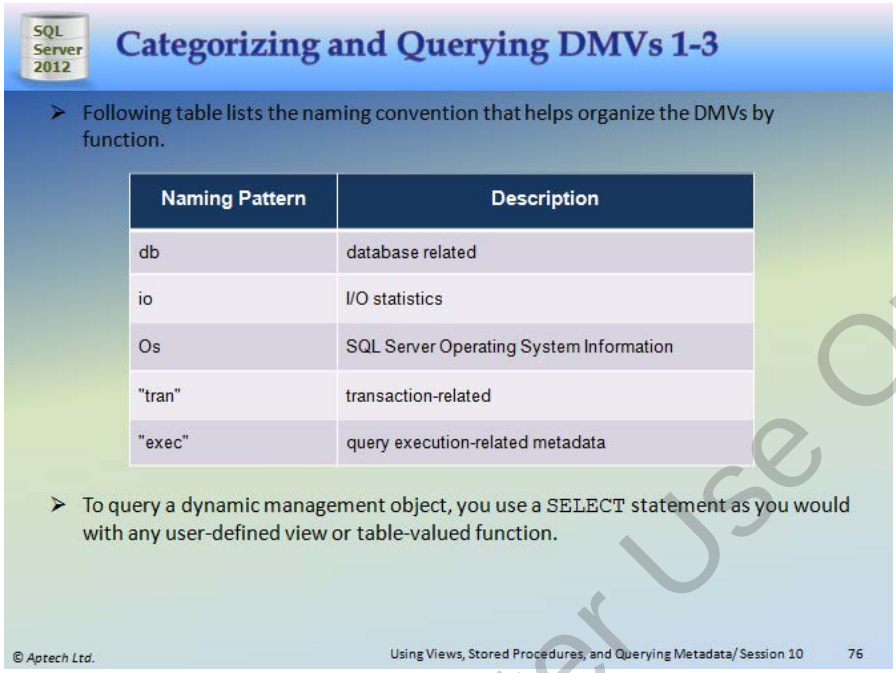
Slide 75

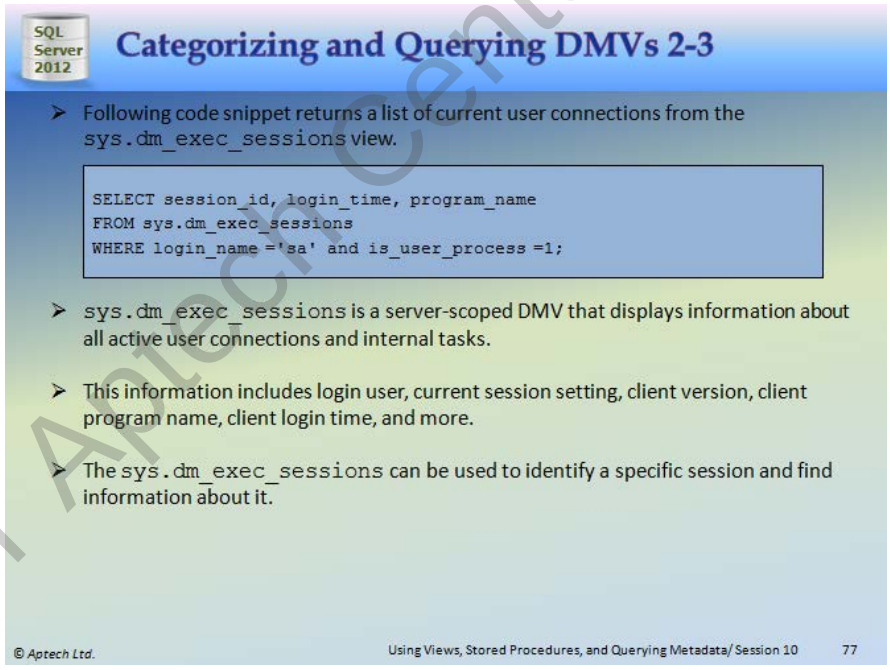


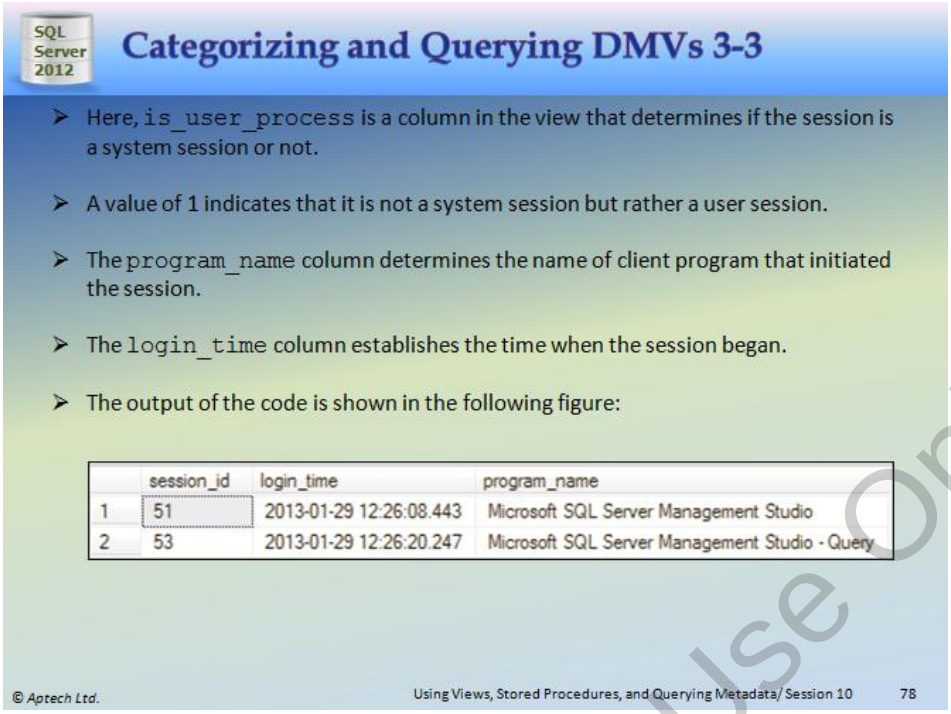
Using slide 75, explain how to query the dynamic management objects.
First introduced in SQL Server 2005, Dynamic Management Views (DMVs) and Dynamic Management Functions (DMFs) are dynamic management objects that return server and database state information. DMVs and DMFs are collectively referred to as dynamic management objects. They provide useful insight into the working of software and can be used for examining the state of SQL Server instance, troubleshooting, and performance tuning.
Both DMVs and DMFs return data in tabular format but the difference is that while a DMF normally accepts at least one parameter, a DMV does not accept parameters. SQL Server 2012 provides nearly 200 dynamic management objects. In order to query DMVs, it is required to have VIEW SERVER STATE or VIEW DATABASE STATE permission, depending on the scope of the DMV.

Categorize and Query DMVs

Slides 76 to 78







Using slides 76 to 78, explain how to categorize and query DMVs.

The table on slide 76 lists the naming convention that helps organize the DMVs by function.

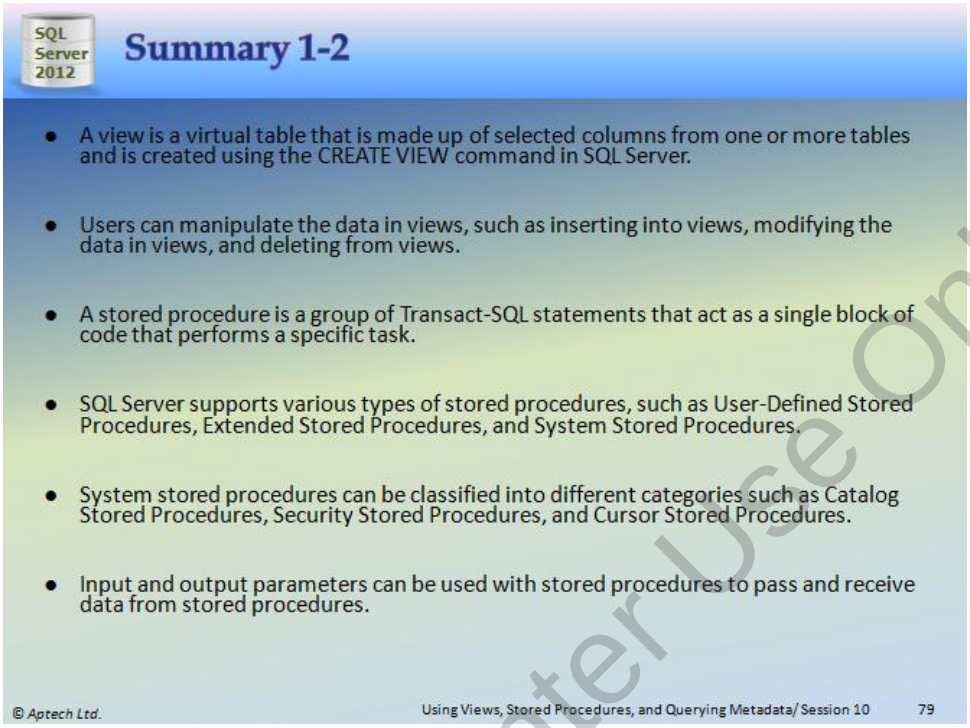
To query a dynamic management object, you use a SELECT statement as you would with any user-defined view or table-valued function. For example, explain the Code Snippet that returns a list of current user connections from the sys.dm_exec_sessions view.

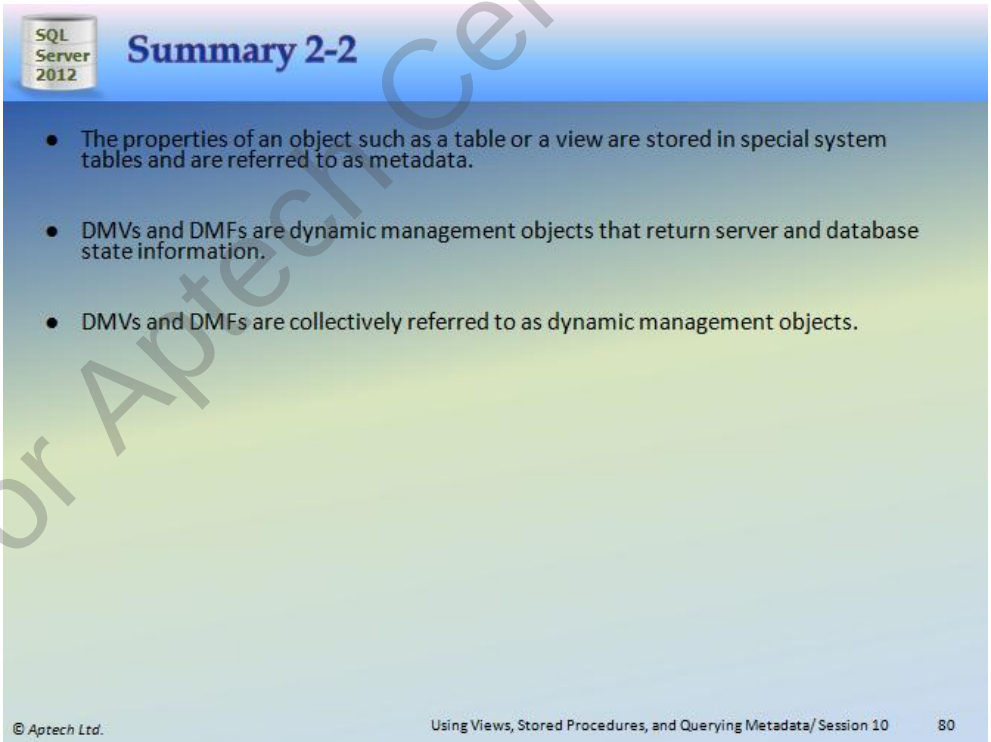
sys.dm_exec_sessions is a server-scoped DMV that displays information about all active user connections and internal tasks. This information includes login user, current session setting, client version, client program name, client login time, and more. The sys.dm_exec_sessions can be used to identify a specific session and find information about it.

Here, is_user_process is a column in the view that determines if the session is a system session or not. A value of 1 indicates that it is not a system session but rather a user session. The program_name column determines the name of client program that initiated the session. The login_time column establishes the time when the session began. The output of Code Snippet is shown in the figure on slide 78.

Summarize Session

Slides 79 and 80





Using slides 79 and 80 , summarize the session. End the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- A view is a virtual table that is made up of selected columns from one or more tables and is created using the CREATE VIEW command in SQL Server.
- Users can manipulate the data in views, such as inserting into views, modifying the data in views, and deleting from views.
- A stored procedure is a group of Transact-SQL statements that act as a single block of code that performs a specific task.
- SQL Server supports various types of stored procedures, such as User-Defined Stored Procedures, Extended Stored Procedures, and System Stored Procedures.
- System stored procedures can be classified into different categories such as Catalog Stored Procedures, Security Stored Procedures, and Cursor Stored Procedures.
- Input and output parameters can be used with stored procedures to pass and receive data from stored procedures.
- The properties of an object such as a table or a view are stored in special system tables and are referred to as metadata.
- DMVs and DMFs are dynamic management objects that return server and database state information.
- DMVs and DMFs are collectively referred to as dynamic management objects.

10.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Indexes topic that is offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 11 – Indexes

11.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from the previous session. Prepare a question or two which will be a key point to relate the current session objectives.

11.1.1 Objectives

By the end of this session, the learners will be able to:

- Define and explain indexes
- Describe the performance of indexes
- Explain clustered indexes
- Explain nonclustered indexes
- Explain partitioning of data
- Explain the steps to display query performance data using indexes

11.1.2 Teaching Skills

To teach this session, you should be well-versed with indexes and their performance. Along with this also prepare yourself with different types of indexes. You should also know the procedure to query performance data using indexes.
You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

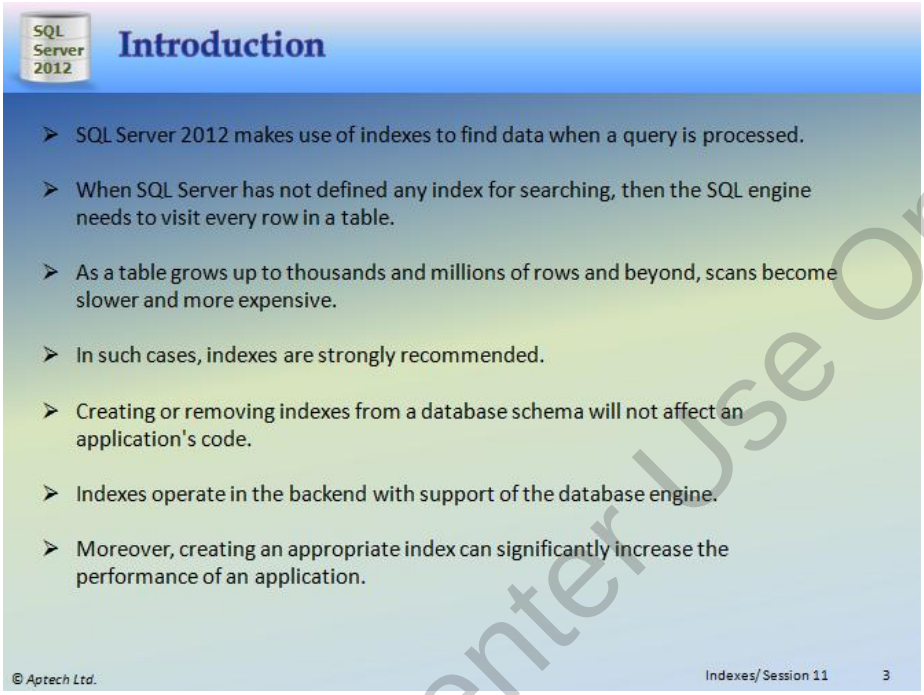
In-Class Activities:
Follow the order given here during In-Class activities.

Overview of the Session:
Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces indexes, their performance, and different types of indexes. They will also know about procedure to query performance data using indexes.

11.2 In-Class Explanations

Introduction

Slide 3



Using slide 3, explain the introduction.

SQL Server 2012 makes use of indexes to find data when a query is processed. The SQL Server engine uses an index in the similar way as a student uses a book index. For example, consider that you need to find all references to INSERT statements in a SQL book. The immediate approach taken will be to scan each page of the book beginning from the starting page. You mark each time the word INSERT is found, until the end of the book is reached. This approach is time consuming and laborious. The second way is to use the index in the back of the book to find the page numbers for each occurrence of the INSERT statements. The second way produces the same results as the first, but by tremendously saving time.

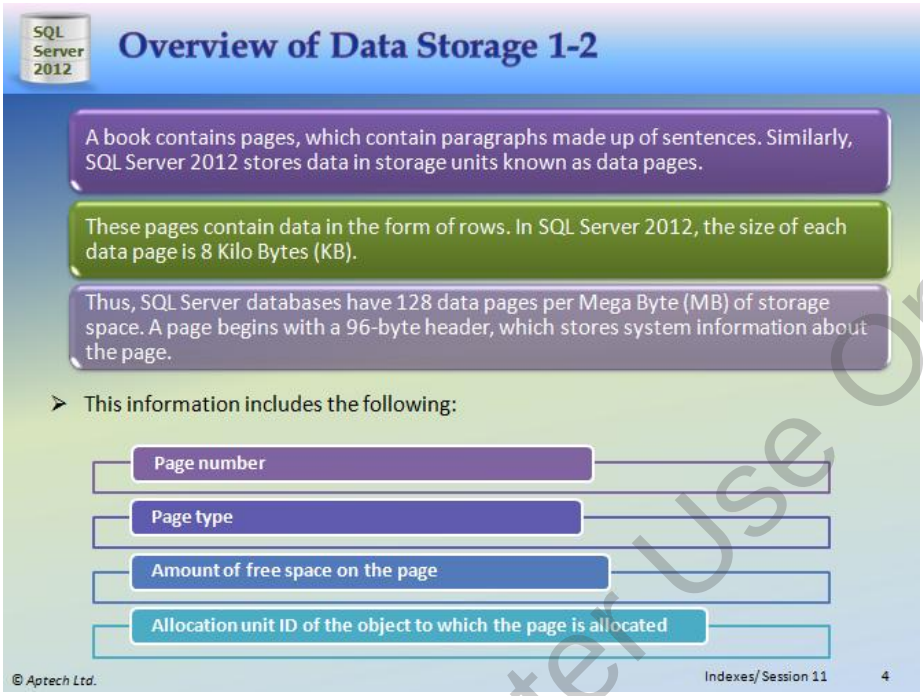
When SQL Server has not defined any index for searching, then the process is similar to the first way in the example; the SQL engine needs to visit every row in a table. In database terminology, this behavior is called table scan, or just scan.

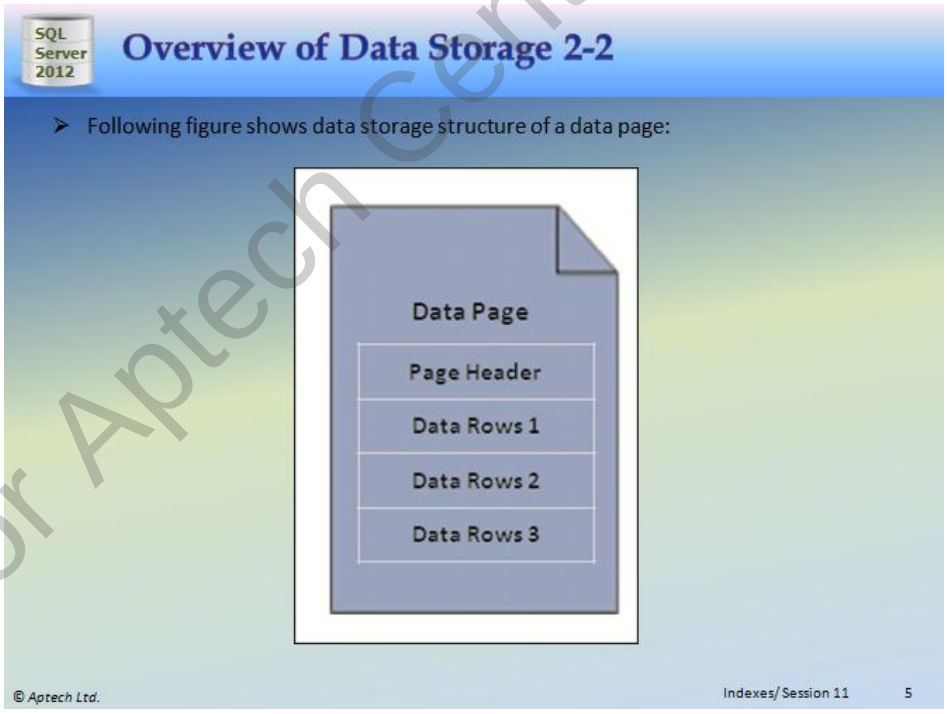
A table scan is not always troublesome, but it is sometimes unavoidable. However, as a table grows up to thousands and millions of rows and beyond, scans become slower and more expensive. In such cases, indexes are strongly recommended.

Creating or removing indexes from a database schema will not affect an application's code. Indexes operate in the backend with support of the database engine. Moreover, creating an appropriate index can significantly increase the performance of an application.

Overview of Data Storage

Slides 4 and 5





Using slides 4 and 5, explain the overview of data storage.

A book contains pages, which contain paragraphs made up of sentences. Similarly, SQL Server 2012 stores data in storage units known as data pages. These pages contain data in the form of rows.

In SQL Server 2012, the size of each data page is 8 Kilo Bytes (KB). Thus, SQL Server databases have 128 data pages per Mega Byte (MB) of storage space.

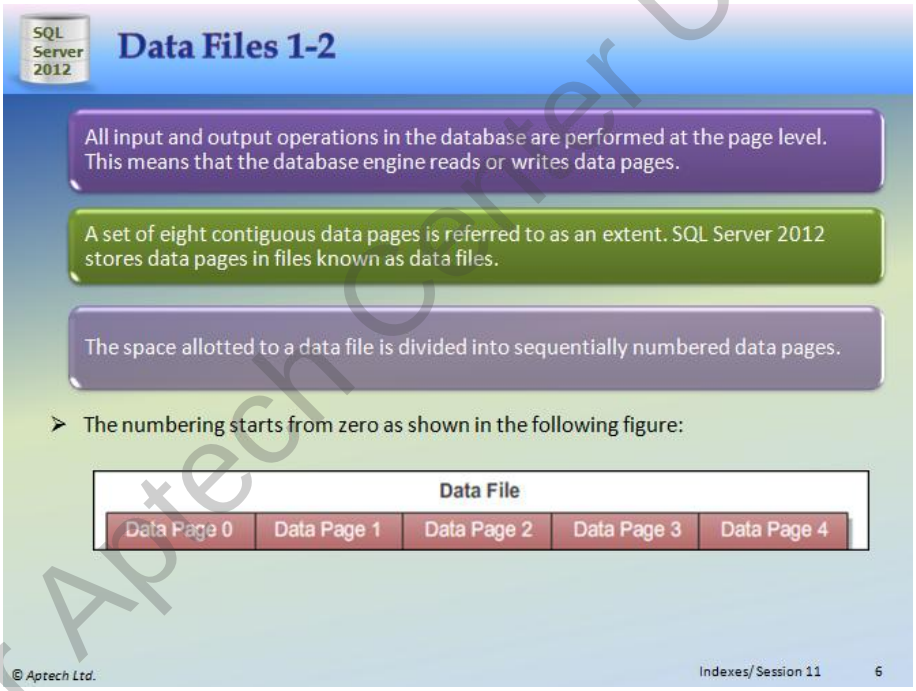
A page begins with a 96-byte header, which stores system information about the page. This information includes the following:

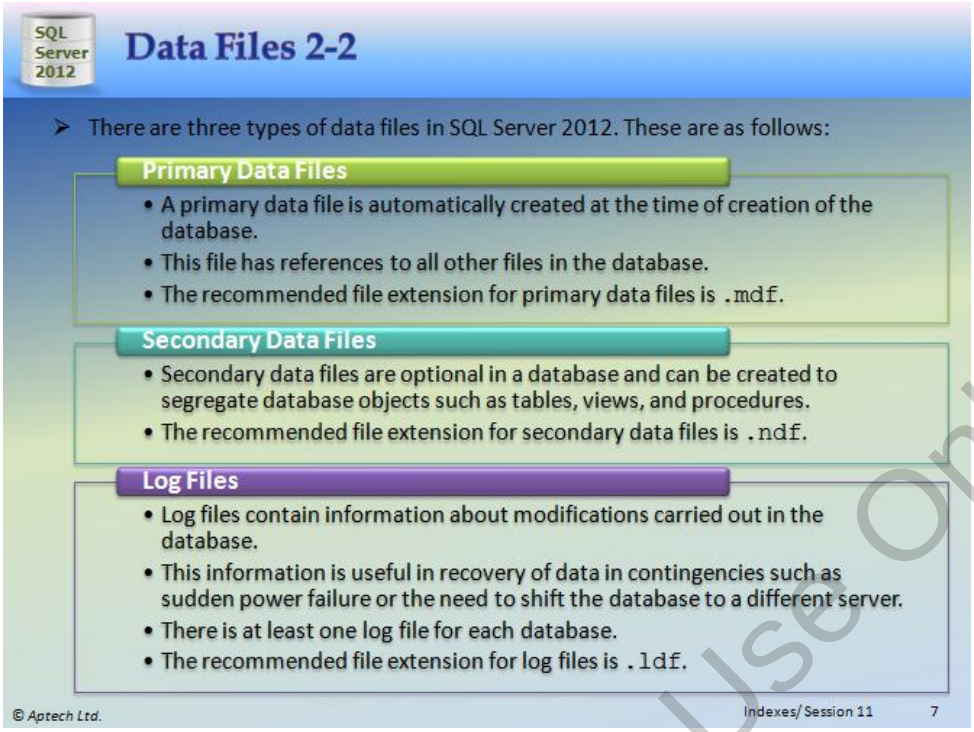
- Page number
- Page type
- Amount of free space on the page
- Allocation unit ID of the object to which the page is allocated

Figure on slide 5 shows data storage structure of a data page.
Mention a data page is the smallest unit of data storage. An allocation unit is a collection of data pages grouped together based on the page type. This grouping is done for efficient management of data.

Data Files

Slides 6 and 7





Using slides 6 and 7, explain the data files.

All input and output operations in the database are performed at the page level. This means that the database engine reads or writes data pages. A set of eight contiguous data pages is referred to as an extent.

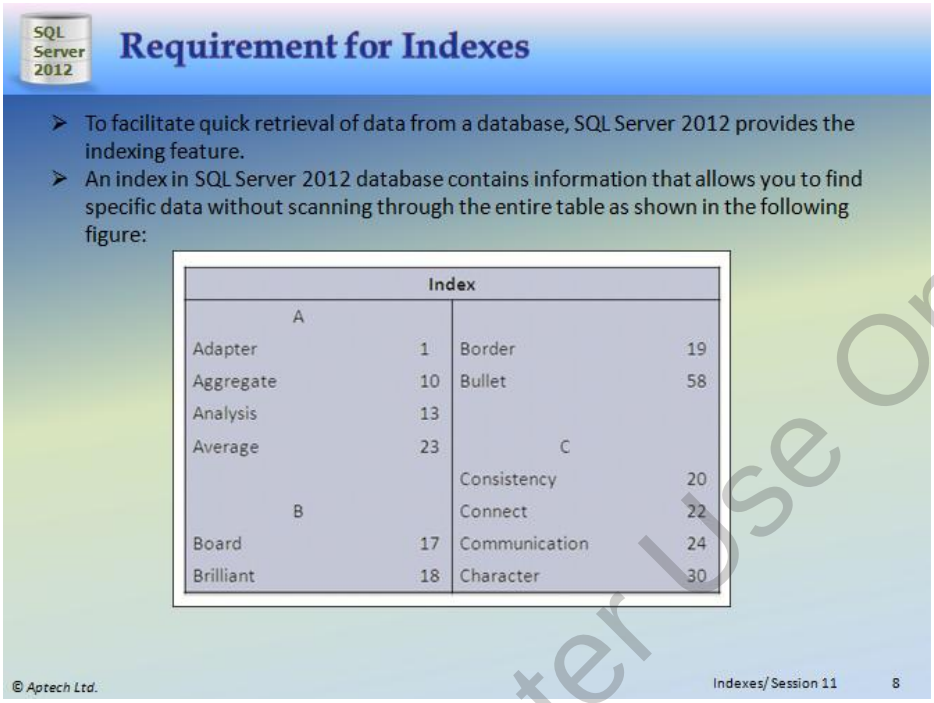
SQL Server 2012 stores data pages in files known as data files. The space allotted to a data file is divided into sequentially numbered data pages. The numbering starts from zero as shown in the figure on slide 6.

There are three types of data files in SQL Server 2012. These are as follows:

- Primary Data Files: A primary data file is automatically created at the time of creation of the database. This file has references to all other files in the database. The recommended file extension for primary data files is .mdf.
- Secondary Data Files: Secondary data files are optional in a database and can be created to segregate database objects such as tables, views, and procedures. The recommended file extension for secondary data files is .ndf.
- Log Files: Log files contain information about modifications carried out in the database. This information is useful in recovery of data in contingencies such as sudden power failure or the need to shift the database to a different server. There is at least one log file for each database. The recommended file extension for log files is .ldf.

Requirement for Indexes

Slide 8

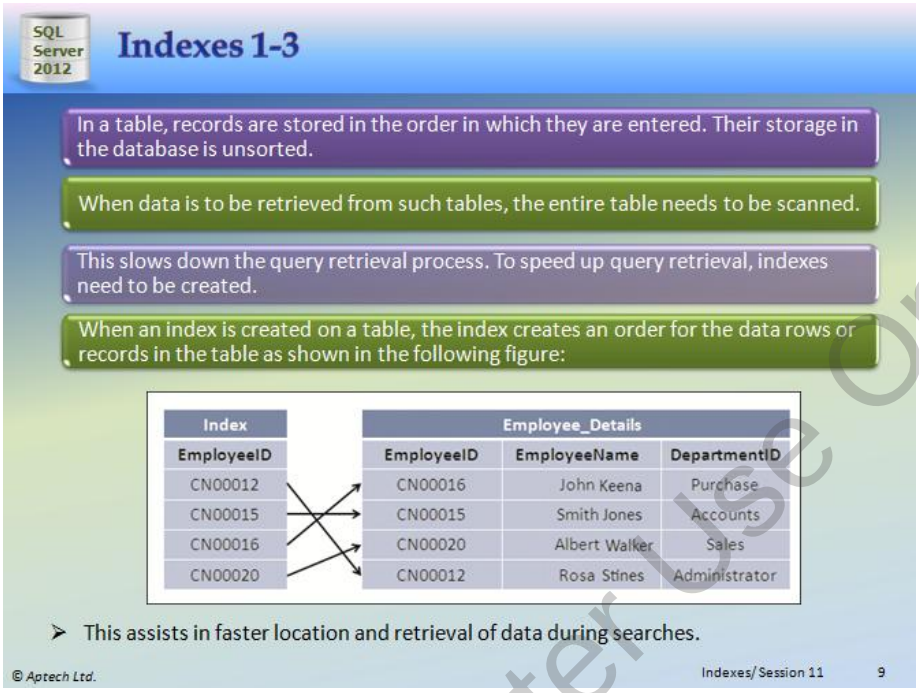


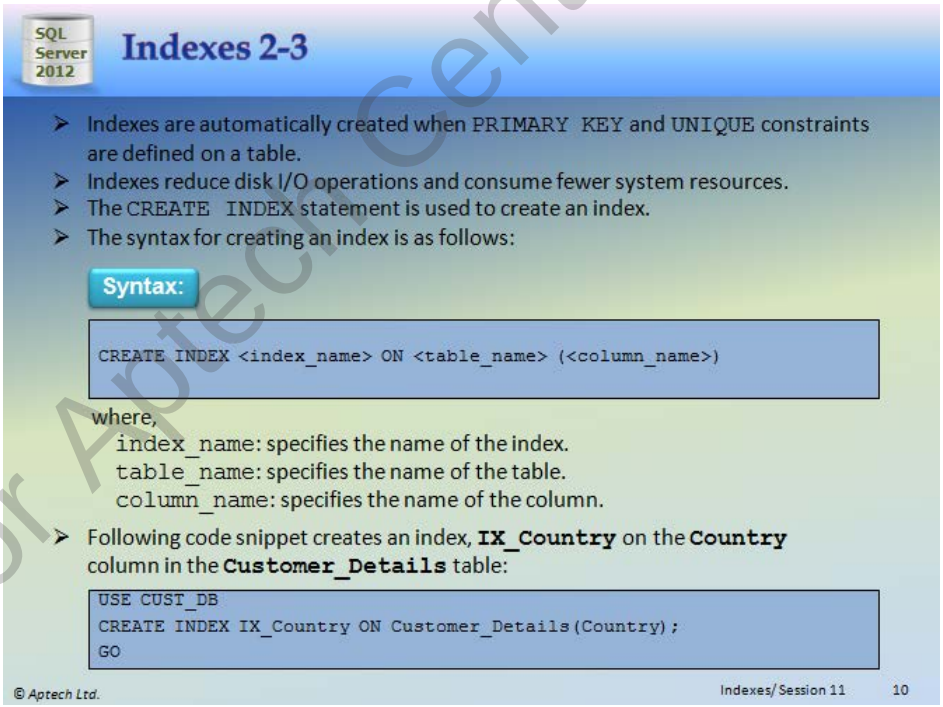
Using slide 8, explain the requirement for indexes.

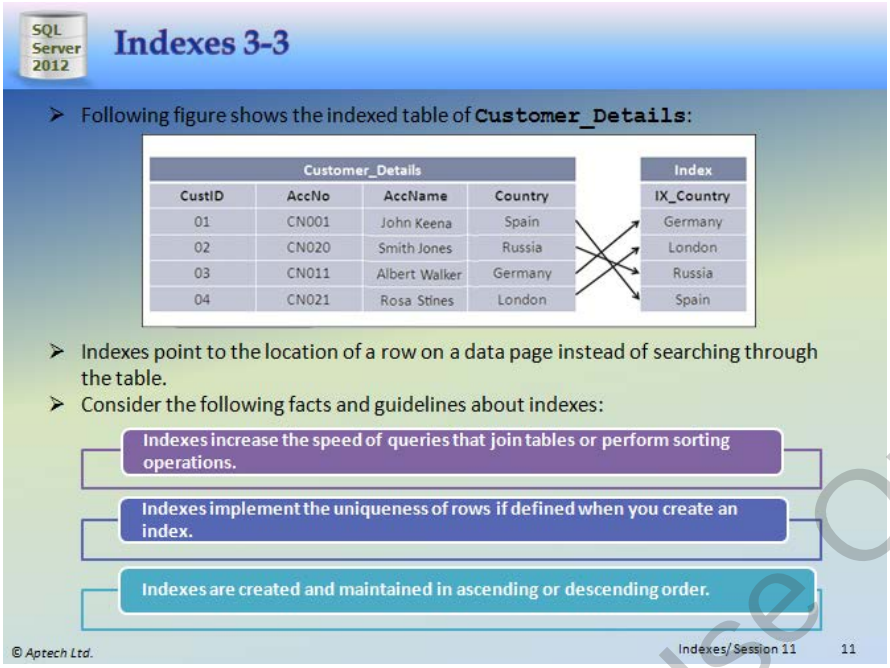
To facilitate quick retrieval of data from a database, SQL Server 2012 provides the indexing feature. Similar to an index in a book, an index in SQL Server 2012 database contains information that allows you to find specific data without scanning through the entire table as shown in the figure.

Indexes

Slides 9 to 11







Using slides 9 to 11, explain the Indexes.
In a table, records are stored in the order in which they are entered. Their storage in the database is unsorted. When data is to be retrieved from such tables, the entire table needs to be scanned. This slows down the query retrieval process. To speed up query retrieval, indexes need to be created.

When an index is created on a table, the index creates an order for the data rows or records in the table as shown in figure on slide 9. This assists in faster location and retrieval of data during searches.

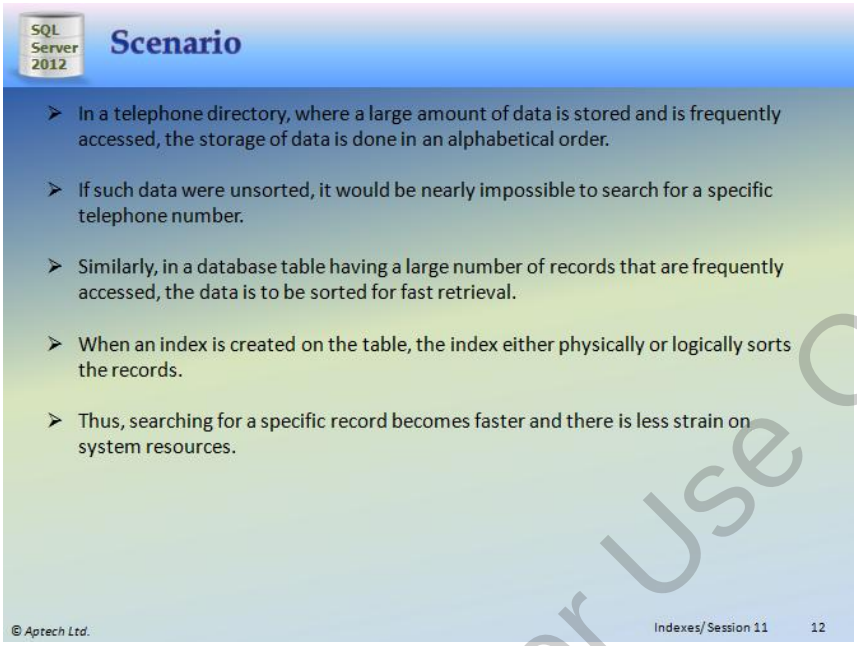
Indexes are automatically created when PRIMARY KEY and UNIQUE constraints are defined on a table. Indexes reduce disk I/O operations and consume fewer system resources.
Mention, multiple indexes can be created on a single table.
The CREATE INDEX statement is used to create an index. Explain the syntax for this statement.

Explain the Code Snippet which creates an index, IX_Country on the Country column in the Customer_Details table.
Figure on slide 11 shows the indexed table of Customer_Details.
Indexes point to the location of a row on a data page instead of searching through the table.
Consider the following facts and guidelines about indexes:

- Indexes increase the speed of queries that join tables or perform sorting operations.
- Indexes implement the uniqueness of rows if defined when you create an index.
- Indexes are created and maintained in ascending or descending order.

Scenario

Slide 12



Using slide 12, explain the scenario.

In a telephone directory, where a large amount of data is stored and is frequently accessed, the storage of data is done in an alphabetical order. If such data were unsorted, it would be nearly impossible to search for a specific telephone number. Similarly, in a database table having a large number of records that are frequently accessed, the data is to be sorted for fast retrieval. When an index is created on the table, the index either physically or logically sorts the records. Thus, searching for a specific record becomes faster and there is less strain on system resources.

In-Class Question:

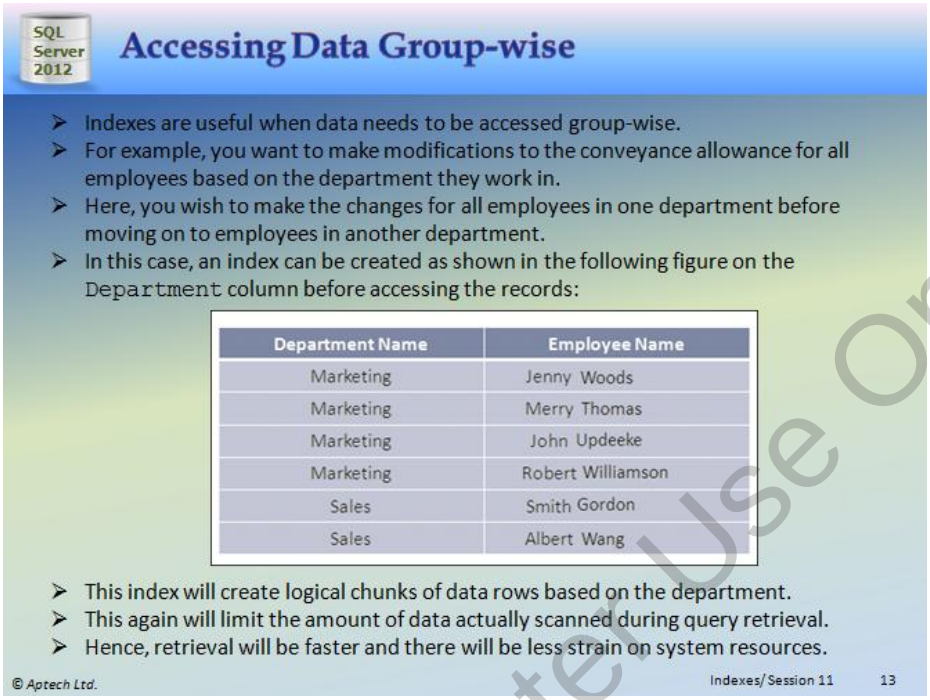
What is HAVING clause used for?

Answer:

The HAVING clause acts as a WHERE clause in places where the WHERE clause cannot be used against aggregate functions such as SUM().

Accessing Data Group-wise

Slide 13



Using slide 13, explain accessing data group-wise.

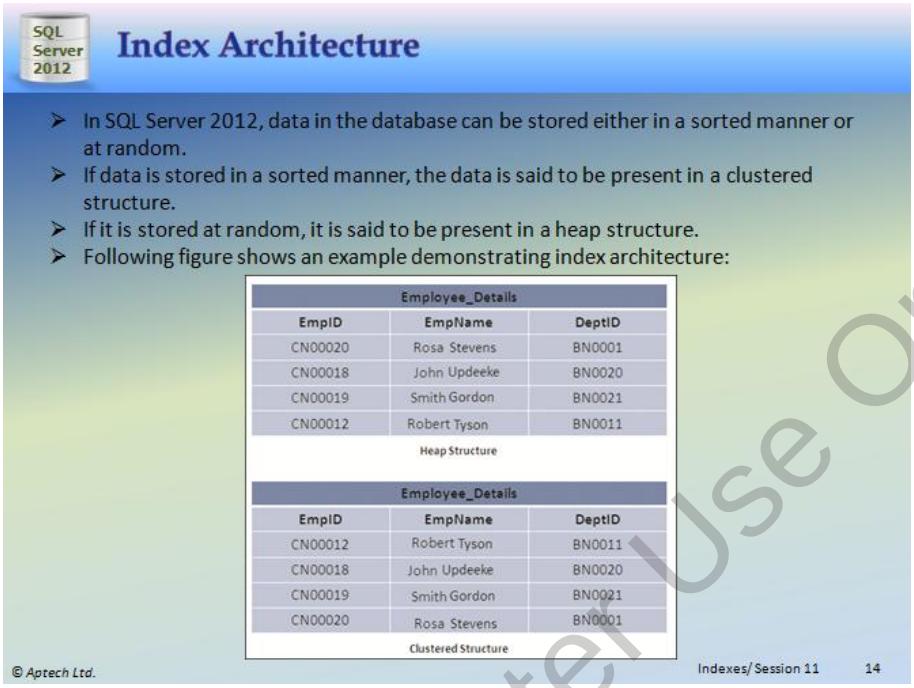
Indexes are useful when data needs to be accessed group-wise. For example, you want to make modifications to the conveyance allowance for all employees based on the department they work in. Here, you wish to make the changes for all employees in one department before moving on to employees in another department. In this case, an index can be created as shown in the figure on the Department column before accessing the records.

This index will create logical chunks of data rows based on the department. This again will limit the amount of data actually scanned during query retrieval.

Hence, retrieval will be faster and there will be less strain on system resources.

Index Architecture

Slide 14



Using slide 14, explain the index architecture.
In SQL Server 2012, data in the database can be stored either in a sorted manner or at random. If data is stored in a sorted manner, the data is said to be present in a clustered structure. If it is stored at random, it is said to be present in a heap structure. Figure shows an example demonstrating index architecture.

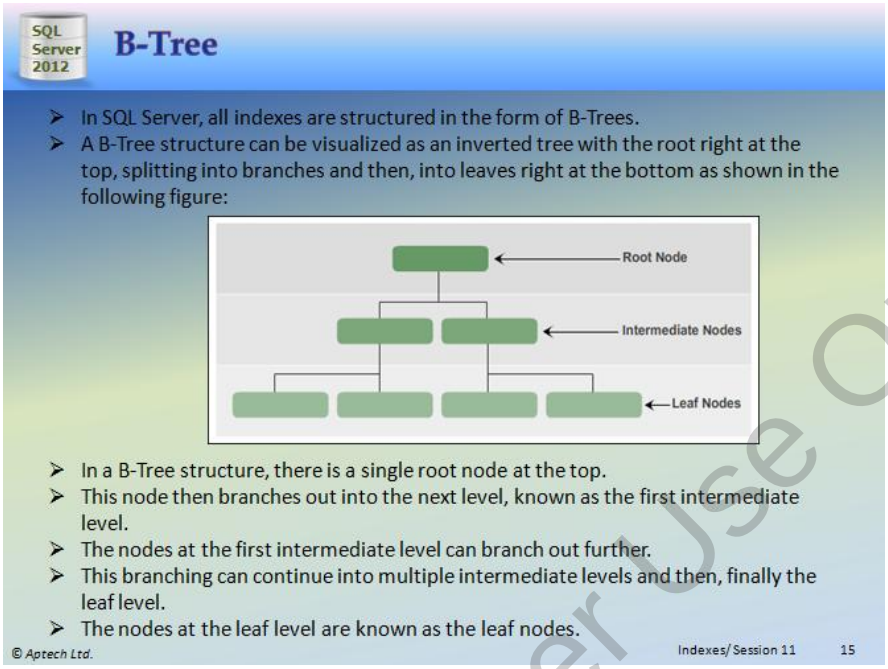
In-Class Question:

What is used to add summary rows in the sorted result set ?

Answer:
ROLLUP operator is used to add summary rows in the sorted result set.

B-Tree

Slide 15



Using slide 15, explain the B-Tree.

In SQL Server, all indexes are structured in the form of B-Trees. A B-Tree structure can be visualized as an inverted tree with the root right at the top, splitting into branches and then, into leaves right at the bottom as shown in the figure.

In a B-Tree structure, there is a single root node at the top. This node then branches out into the next level, known as the first intermediate level. The nodes at the first intermediate level can branch out further. This branching can continue into multiple intermediate levels and then, finally the leaf level. The nodes at the leaf level are known as the leaf nodes.

In-Class Question:

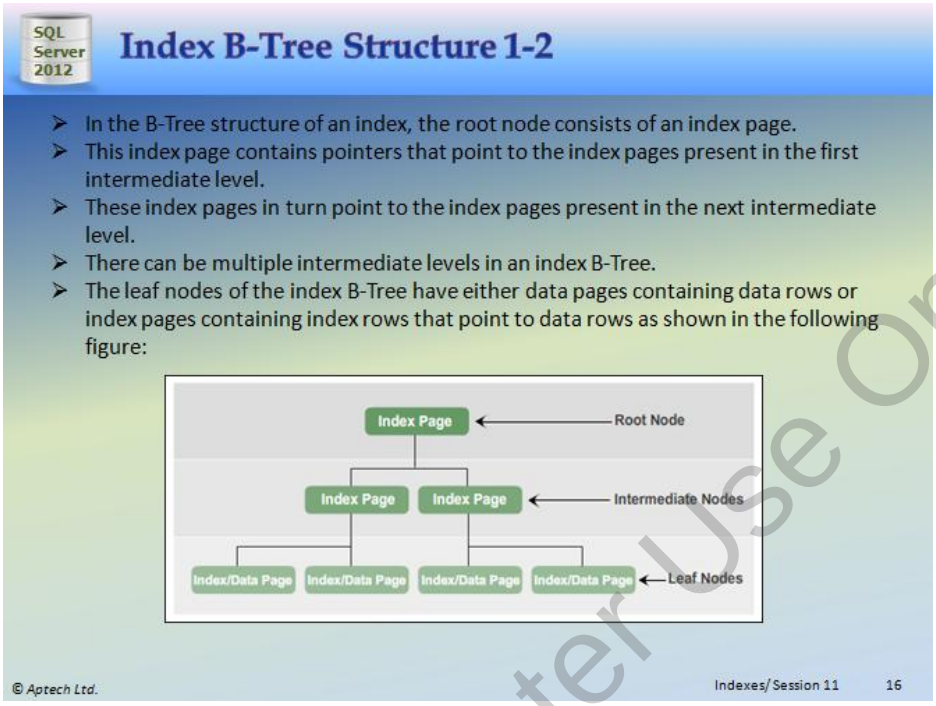
Which function is used to count the number of rows in a table?

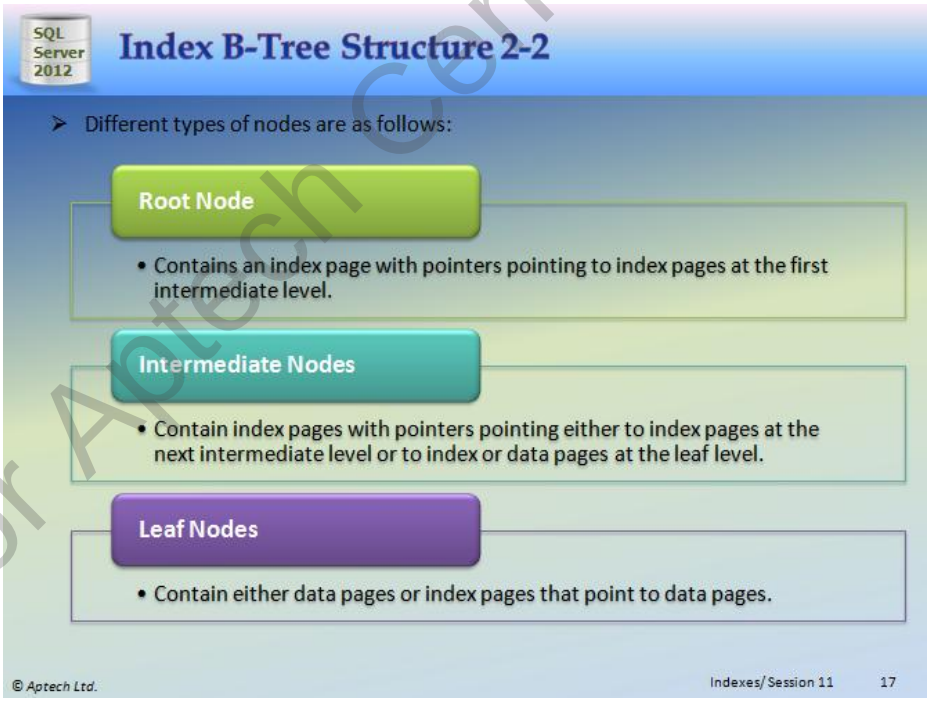
Answer:

The count(*) function is used to count the number of rows in a table.

Index B-Tree Structure

Slides 16 and 17





Using slides 16 and 17, explain the index B-Tree structure.

In the B-Tree structure of an index, the root node consists of an index page. This index page contains pointers that point to the index pages present in the first intermediate level. These index pages in turn point to the index pages present in the next intermediate level. There can be multiple intermediate levels in an index B-Tree. The leaf nodes of the index B-Tree

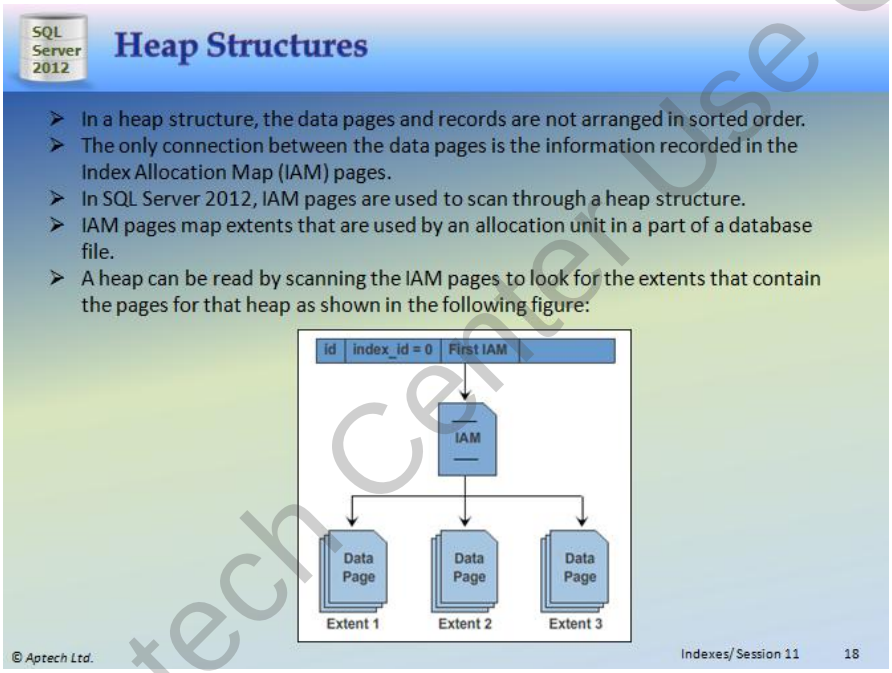
have either data pages containing data rows or index pages containing index rows that point to data rows as shown in the figure on slide 16.
Different types of nodes are as follows:

- **Root Node** - Contains an index page with pointers pointing to index pages at the first intermediate level.
- **Intermediate Nodes** - Contain index pages with pointers pointing either to index pages at the next intermediate level or to index or data pages at the leaf level.
- **Leaf Nodes** - Contain either data pages or index pages that point to data pages.

Mention a data page containing index entries is called an index page.

Heap Structures

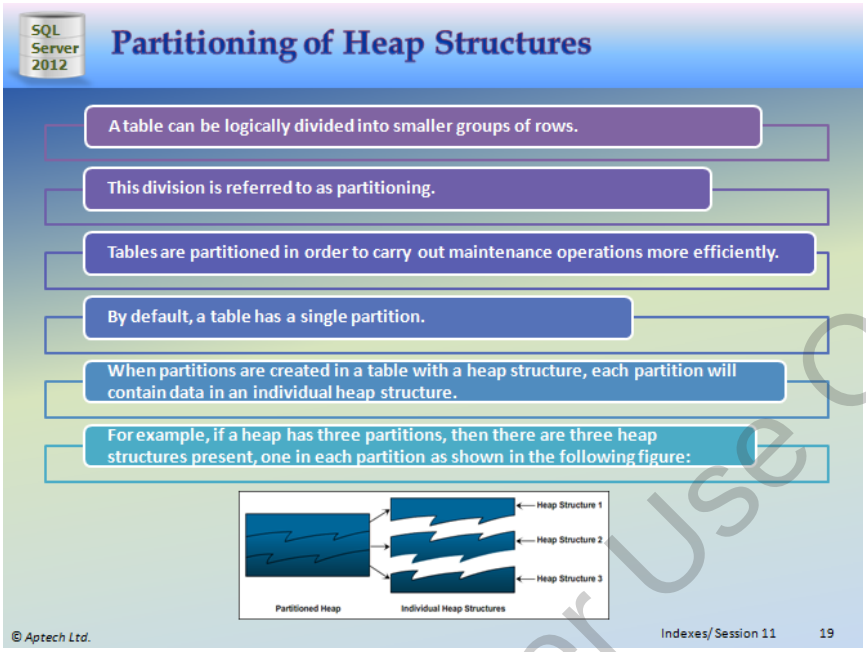
Slide 18



Using slide 18, explain the heap structures.
In a heap structure, the data pages and records are not arranged in sorted order. The only connection between the data pages is the information recorded in the Index Allocation Map (IAM) pages.
In SQL Server 2012, IAM pages are used to scan through a heap structure. IAM pages map extents that are used by an allocation unit in a part of a database file.
A heap can be read by scanning the IAM pages to look for the extents that contain the pages for that heap as shown in the figure.
If an allocation unit contains extents from more than one file, there will be multiple IAM pages linked together in an IAM chain to map these extents.

Partitioning of Heap Structures

Slide 19



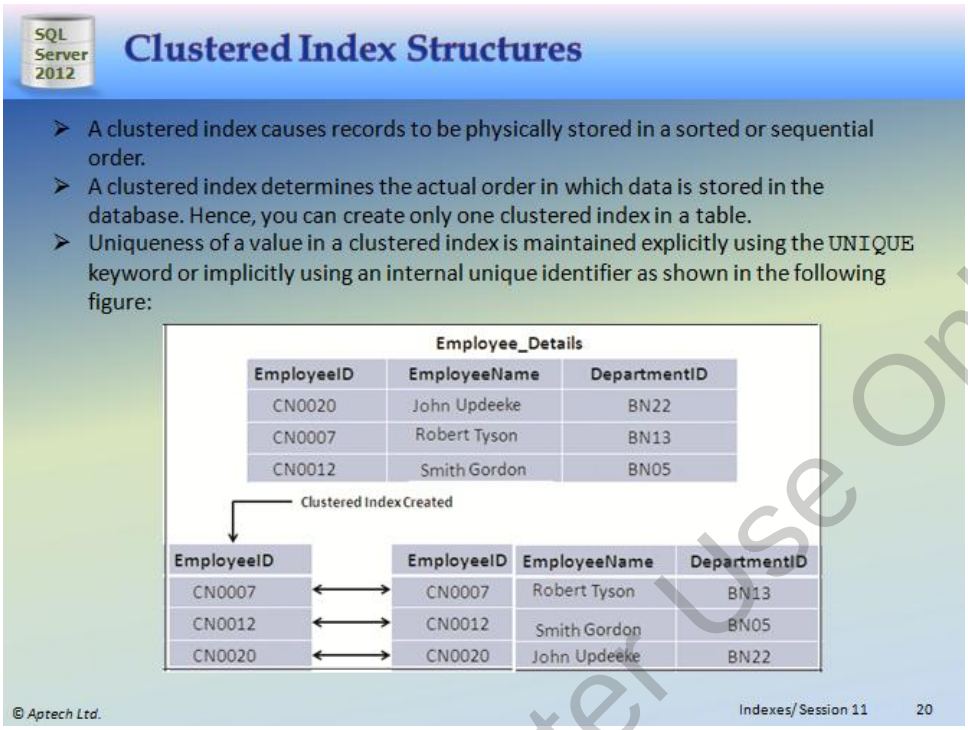
Using slide 19, explain the partitioning of heap structure.

A table can be logically divided into smaller groups of rows. This division is referred to as partitioning. Tables are partitioned in order to carry out maintenance operations more efficiently. By default, a table has a single partition.

When partitions are created in a table with a heap structure, each partition will contain data in an individual heap structure. For example, if a heap has three partitions, then there are three heap structures present, one in each partition as shown in the figure.

Clustered Index Structures

Slide 20



Using slide 20, explain the clustered index structures.

A clustered index causes records to be physically stored in a sorted or sequential order. A clustered index determines the actual order in which data is stored in the database. Hence, you can create only one clustered index in a table.

Uniqueness of a value in a clustered index is maintained explicitly using the UNIQUE keyword or implicitly using an internal unique identifier as shown in the figure.

In-Class Question:

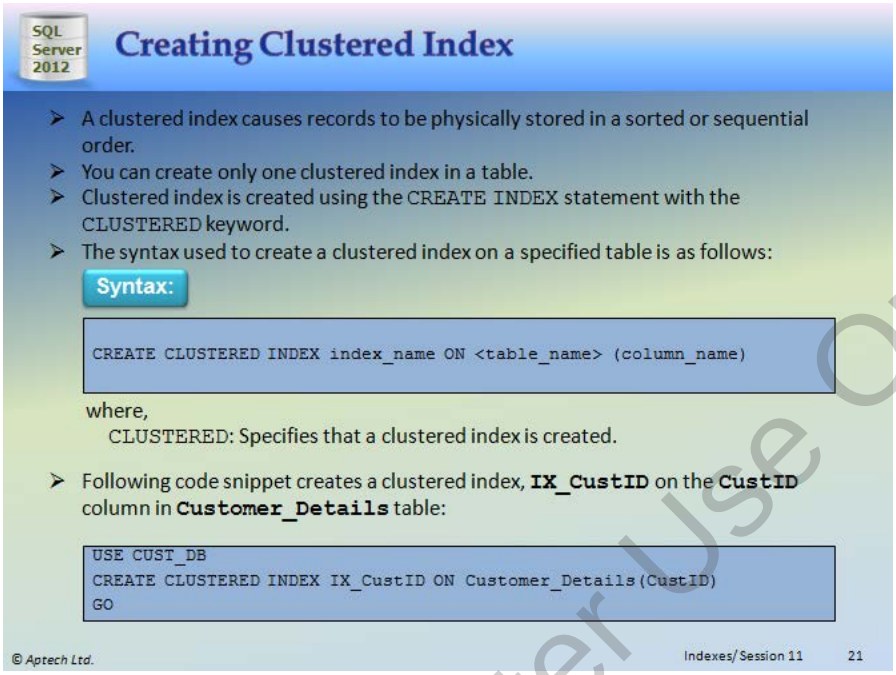
What is result for the geography type object?

Answer:

Circular object, which loosely bounds the selected input objects is result for the geography type object.

Creating Clustered Index

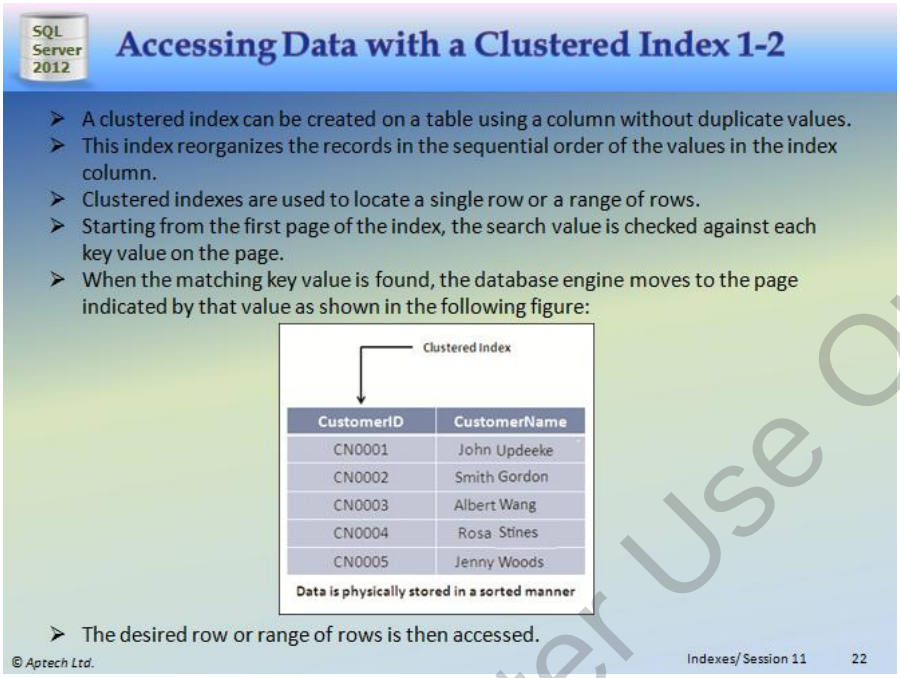
Slide 21

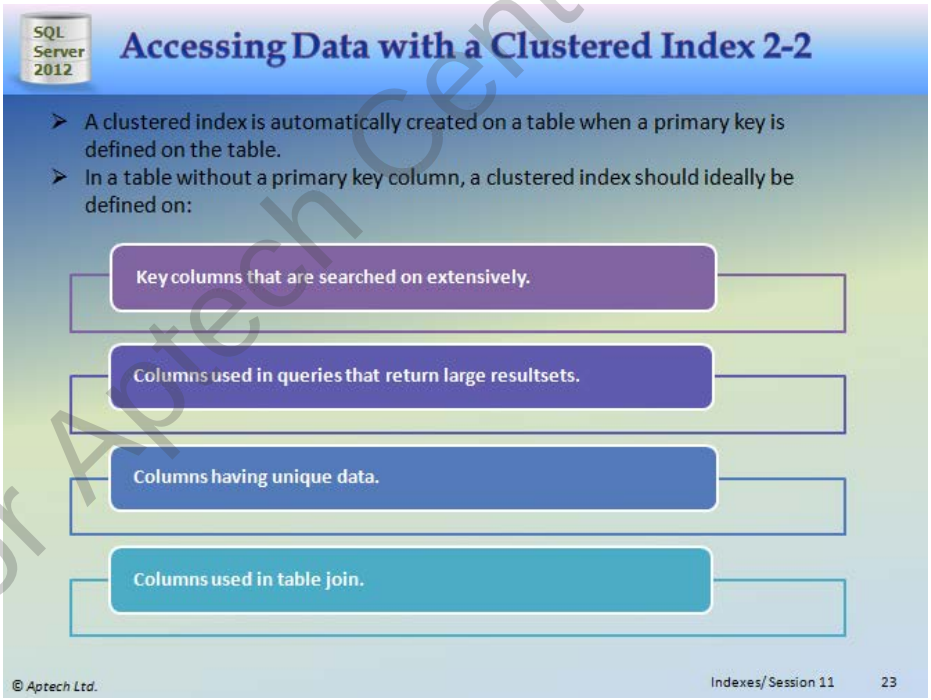


Using slide 21, explain how to create index.
A clustered index causes records to be physically stored in a sorted or sequential order. Thus, a clustered index determines the actual order in which data is stored in the database. Hence, you can create only one clustered index in a table. Clustered index is created using the CREATE INDEX statement with the CLUSTERED keyword. Explain the syntax that creates a clustered index on a specified table.
Explain the Code Snippet that creates a clustered index, IX_CustID on the CustID column in Customer_Details table.
Mention, before you create a clustered index, you need to make sure the free space in your system is at least 1.2 times the amount of data in the table.

Accessing Data with Clustered Index

Slides 22 and 23





Using slides 22 and 23, explain how to access data with clustered index.

A clustered index can be created on a table using a column without duplicate values. This index reorganizes the records in the sequential order of the values in the index column. Clustered indexes are used to locate a single row or a range of rows. Starting from the first page of the index, the search value is checked against each key value on the page. When the matching key value is found, the database engine moves to the page indicated by that value as shown in the figure on slide 22. The desired row or range of rows is then accessed.

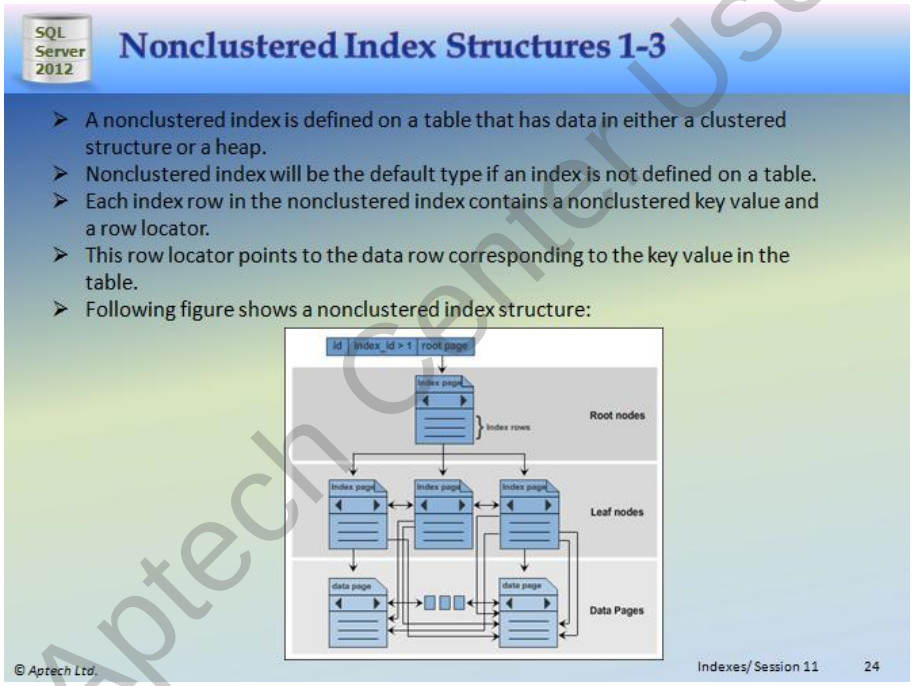
Clustered indexes are useful for columns that are searched frequently for key values or are accessed in sorted order.
A clustered index is automatically created on a table when a primary key is defined on the table. In a table without a primary key column, a clustered index should ideally be defined on:

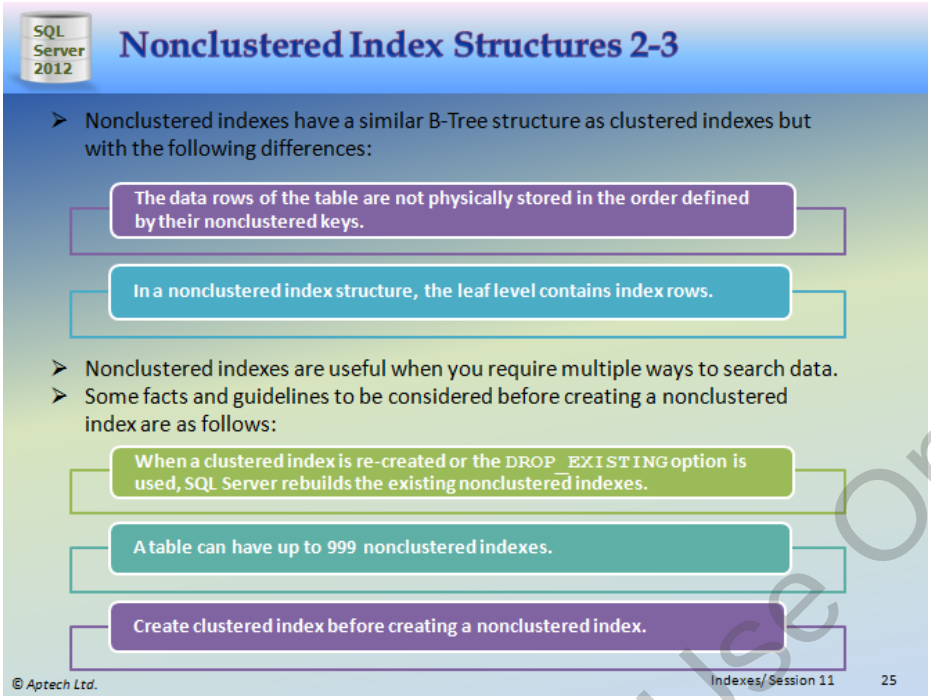
- Key columns that are searched on extensively.
- Columns used in queries that return large resultsets.
- Columns having unique data.
- Columns used in table join.

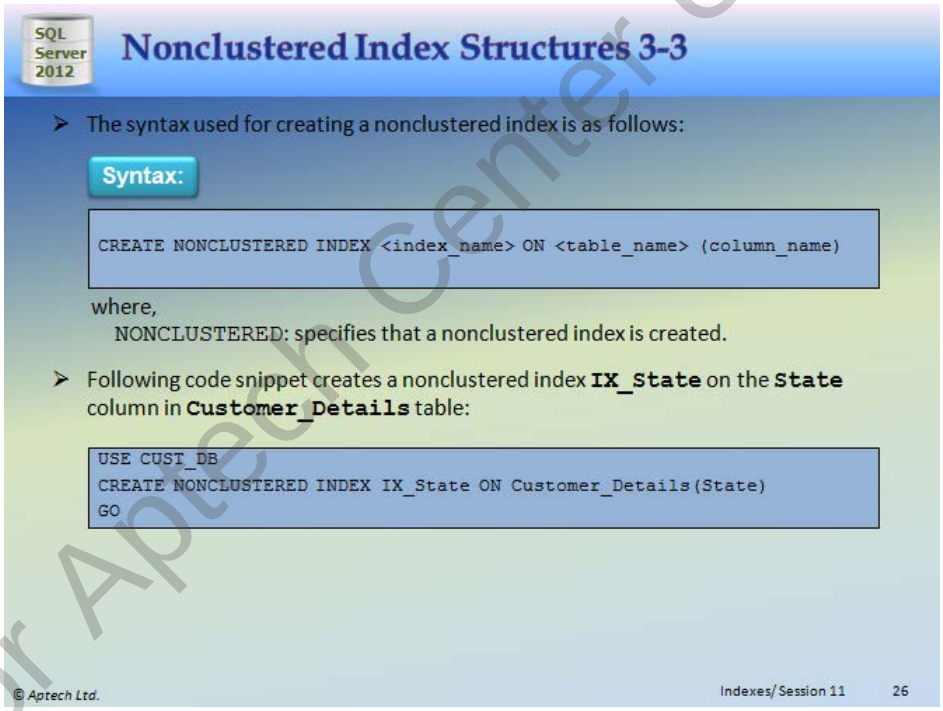
Mention, two or more tables can be logically joined through columns that are common to the tables. Data can then be retrieved from these tables as if they were a single table.

Nonclustered Index Structures

Slides 24 to 26







Using slides 24 to 26, explain the Nonclustered index structures.

A nonclustered index is defined on a table that has data in either a clustered structure or a heap. Nonclustered index will be the default type if an index is not defined on a table. Each index row in the nonclustered index contains a nonclustered key value and a row locator. This row locator points to the data row corresponding to the key value in the table.

Nonclustered indexes have a similar B-Tree structure as clustered indexes but with the following differences:

- The data rows of the table are not physically stored in the order defined by their nonclustered keys.

- In a nonclustered index structure, the leaf level contains index rows.

Figure on slide 24 shows a nonclustered index structure.

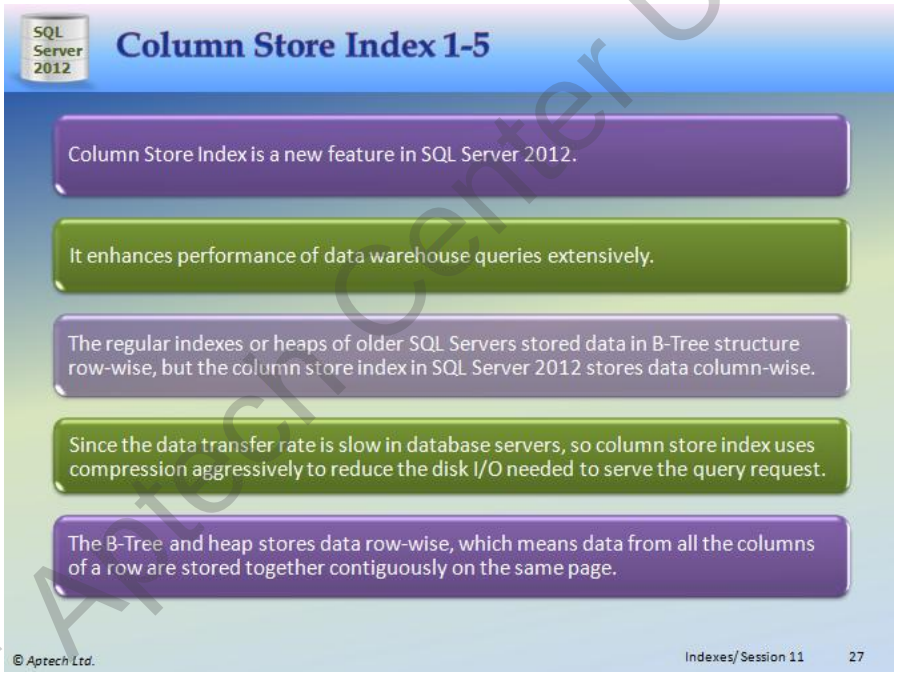
Nonclustered indexes are useful when you require multiple ways to search data. Some facts and guidelines to be considered before creating a nonclustered index are as follows:

- When a clustered index is re-created or the DROP_EXISTING option is used, SQL Server rebuilds the existing nonclustered indexes.
- A table can have up to 999 nonclustered indexes.
- Create clustered index before creating a nonclustered index.

Explain the syntax to create a nonclustered index.
Explain the Code Snippet on slide 26 which creates a nonclustered index IX_State on the State column in the Customer_Details.

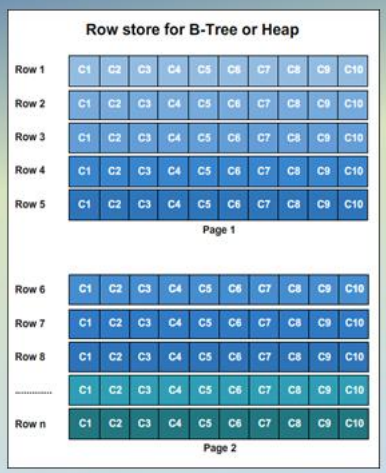
Column Store Index

Slides 27 to 31



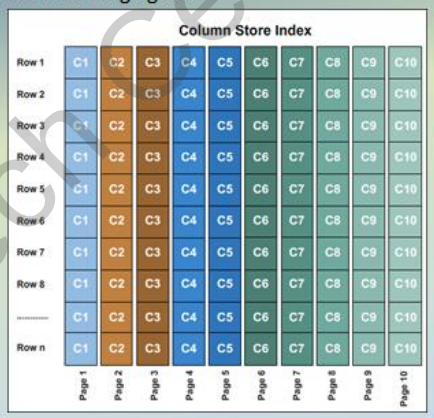
Column Store Index 2-5

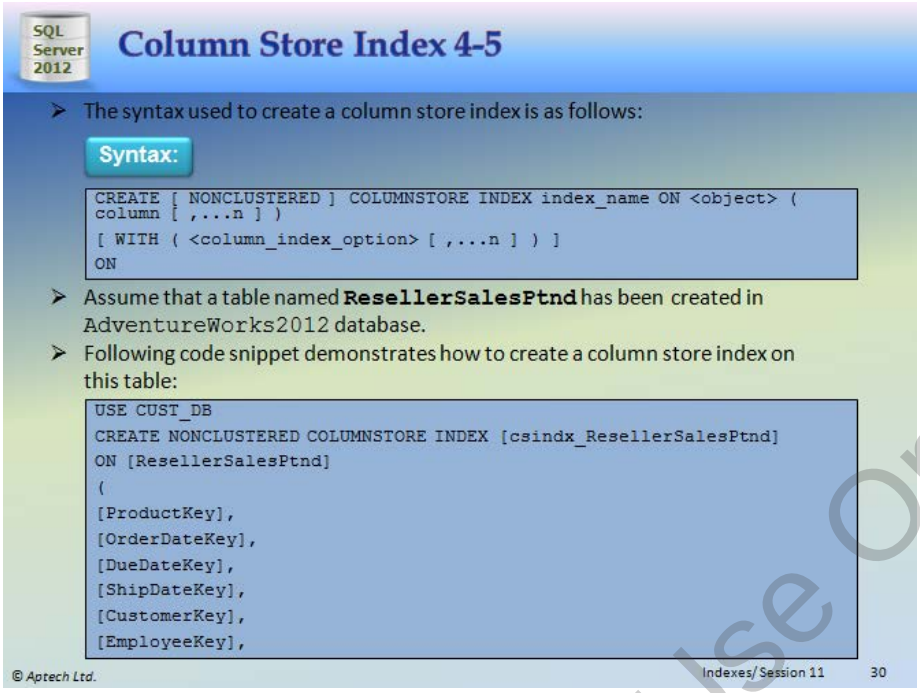
- For example, if there is a table with ten columns (C1 to C10), the data of all the ten columns from each row gets stored together contiguously on the same page as shown in the following figure:

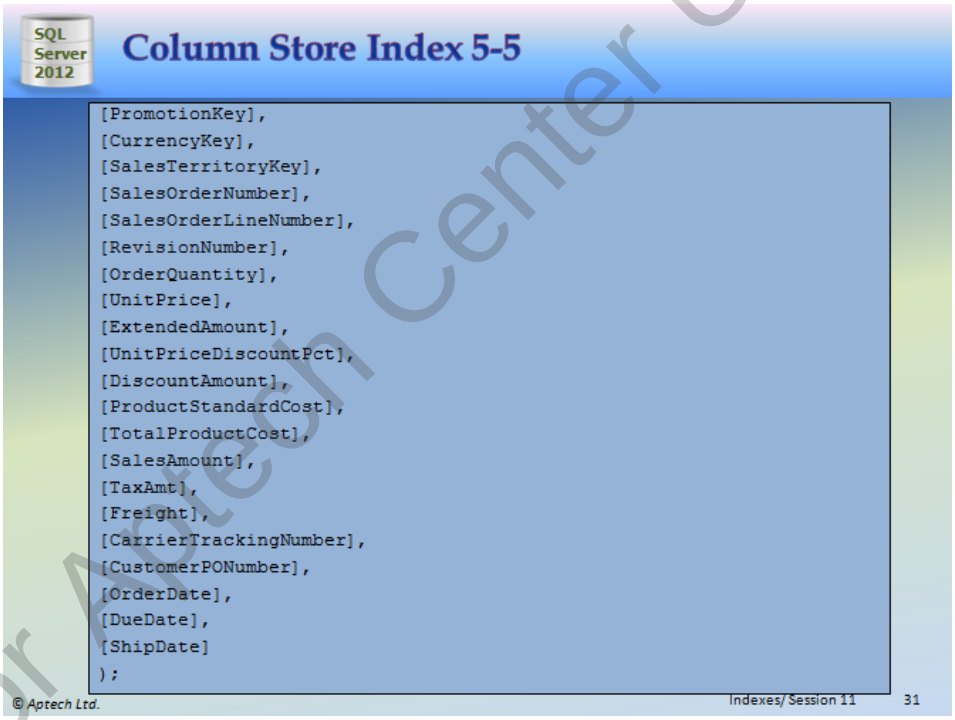


Column Store Index 3-5

- When column store index is created, the data is stored column-wise, which means data of each individual column from each rows is stored together on same page.
- For example, the data of column C1 of all the rows gets stored together on one page and the data for column C2 of all the rows gets stored on another page and so on as shown in the following figure:







Using slides 27 to 31, explain the column store index.

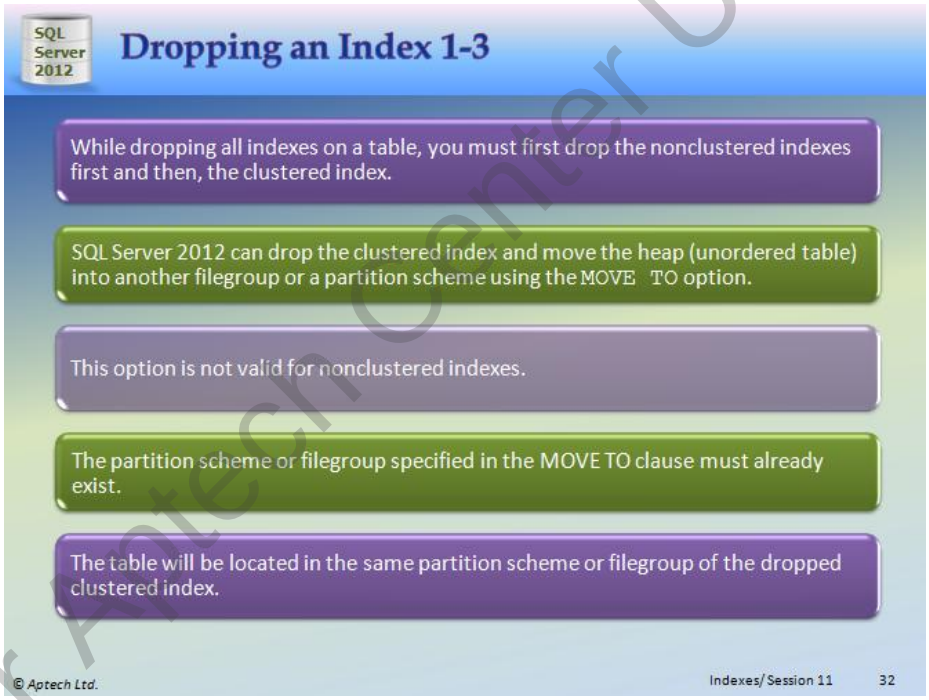
Column Store Index is a new feature in SQL Server 2012. It enhances performance of data warehouse queries extensively. The regular indexes or heaps of older SQL Servers stored data in B-Tree structure row-wise, but the column store index in SQL Server 2012 stores data column-wise. Since the data transfer rate is slow in database servers, so column store index uses compression aggressively to reduce the disk I/O needed to serve the query request.

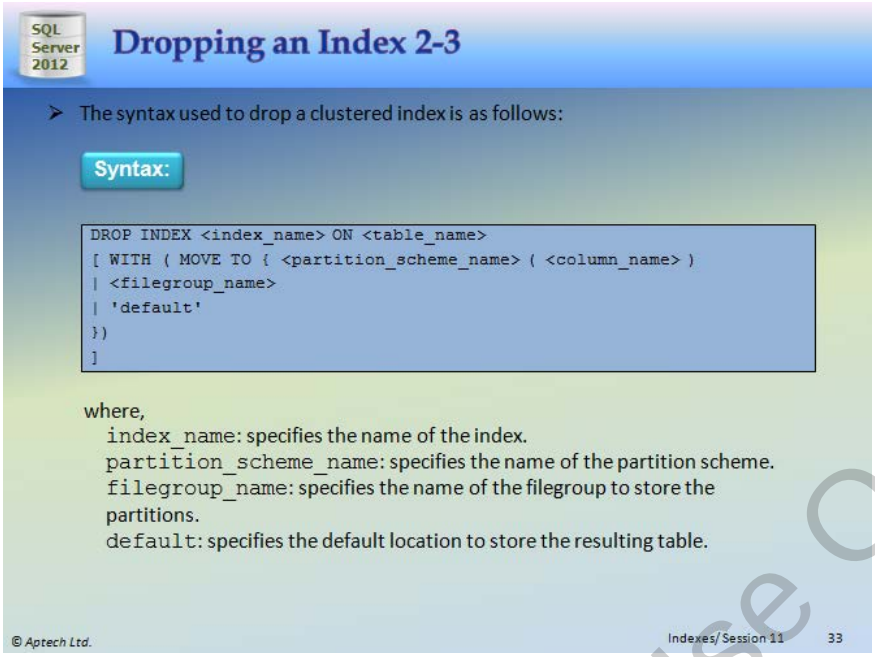
The B-Tree and heap stores data row-wise, which means data from all the columns of a row are stored together contiguously on the same page.

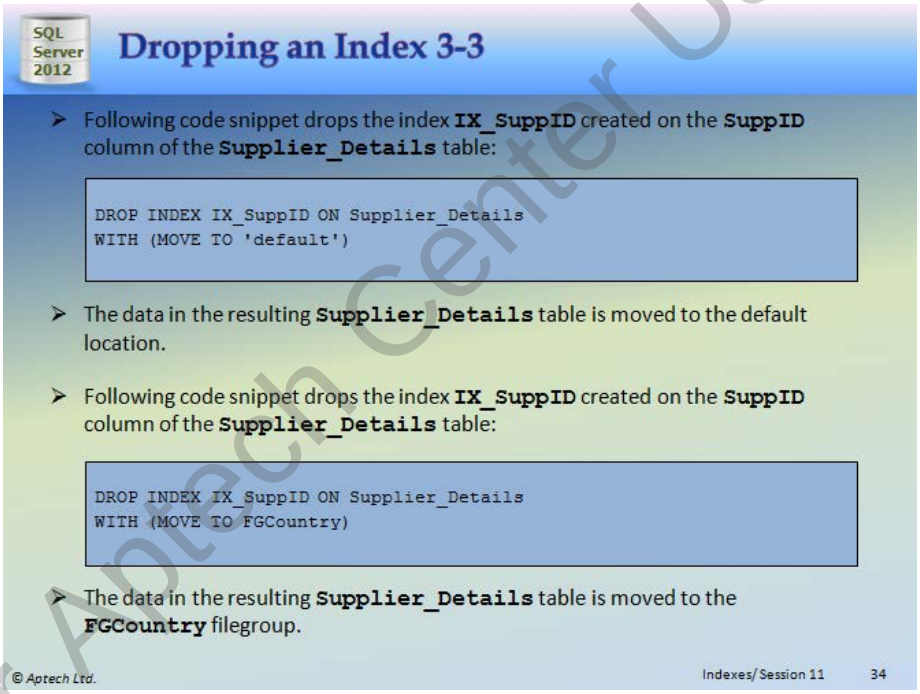
For example, if there is a table with ten columns (C1 to C10), the data of all the ten columns from each row gets stored together contiguously on the same page as shown in figure on slide 28.
When column store index is created, the data is stored column-wise, which means data of each individual column from each rows is stored together on same page.
For example, the data of column C1 of all the rows gets stored together on one page and the data for column C2 of all the rows gets stored on another page and so on as shown in the figure on slide 29.
Explain the syntax to create a column store index on slide 30.
Assume that a table named ResellerSalesPtnd has been created in AdventureWorks2012 database. Explain the Code Snippet on slide 31 that demonstrates how to create a column store index on this table.
Mention, COLUMNSTORE INDEX works only on enterprise edition of SQL Server 2012.

Dropping an Index

Slides 32 to 34







Using slides 32 to 34, explain how to drop a index.

On dropping a clustered index, the rows in the leaf level of the clustered index are copied to a heap. All the nonclustered indexes on the table should then point to the heap in which the data is stored. This is done by rebuilding nonclustered indexes when the clustered index is dropped. Thus, dropping the clustered index is a time-consuming process. Therefore, while dropping all indexes on a table, you must first drop the nonclustered indexes first and then, the clustered index.

SQL Server 2012 can drop the clustered index and move the heap (unordered table) into another filegroup or a partition scheme using the MOVE TO option.

- This option is not valid for nonclustered indexes.
- The partition scheme or filegroup specified in the MOVE TO clause must already exist.

- The table will be located in the same partition scheme or filegroup of the dropped clustered index.

Explain the syntax to drop a clustered index using slide 33.

Explain the Code Snippet on slide 34 which drops the index IX_SuppID created on the SuppID column of the Supplier_Details table.

The data in the resulting Supplier_Details table is moved to the default location.

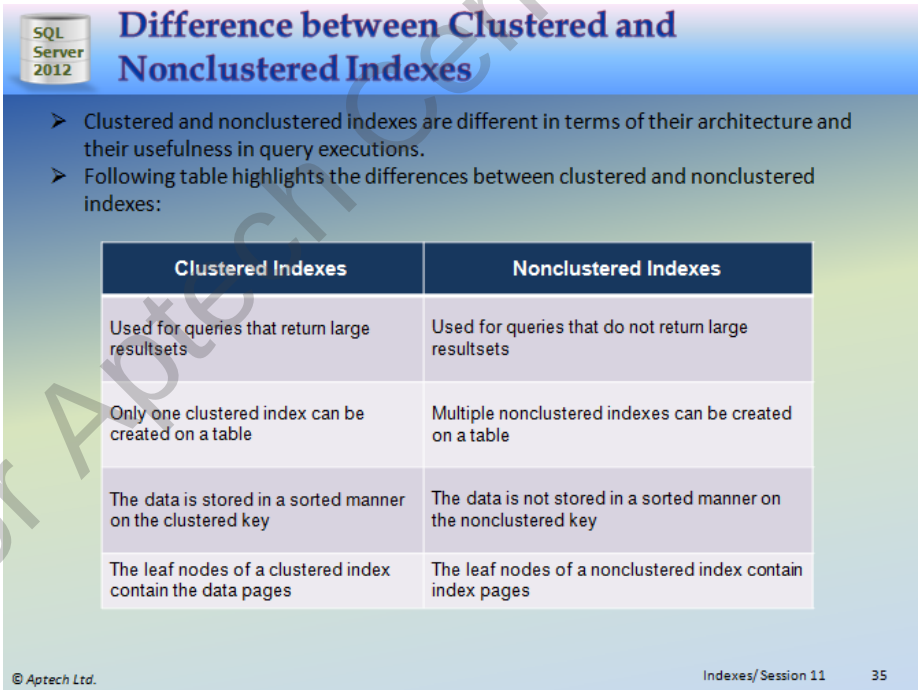
In-Class Question:


What is a self-contained query?

Answer:
Self-contained subqueries are written as standalone queries, without any dependencies on the outer query.

Difference between Clustered and Nonclustered Indexes

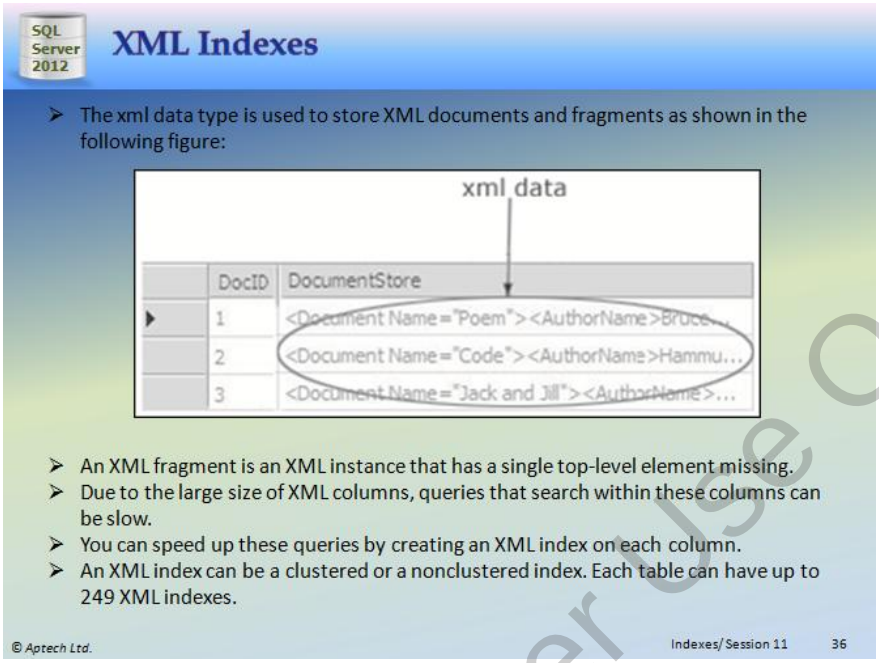
Slide 35



Using slide 35, explain the difference between clustered and nonclustered indexes in detail.

XML Indexes

Slide 36



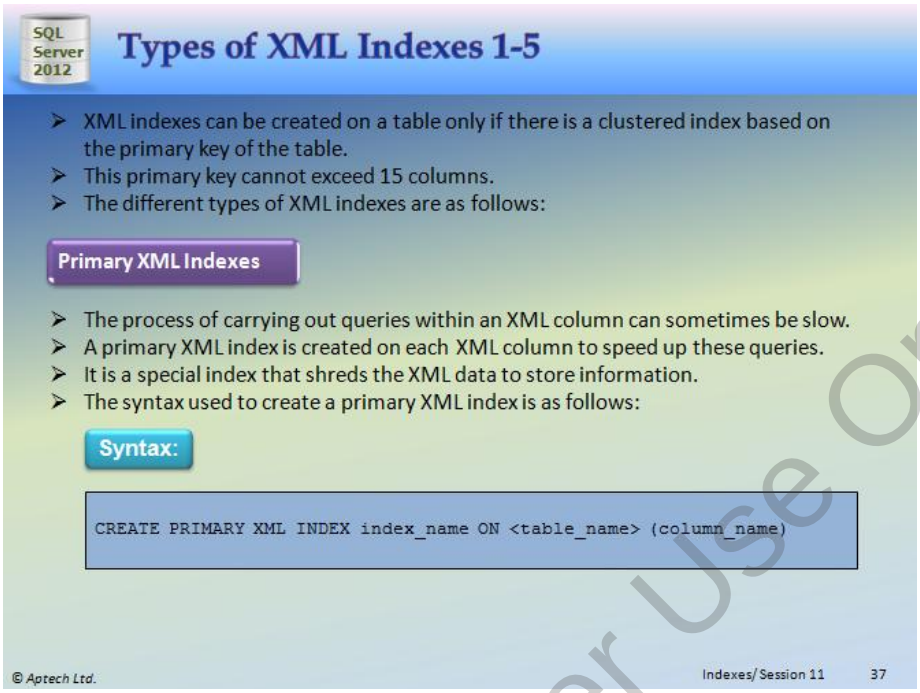
Using slide 36, explain the XML indexes.

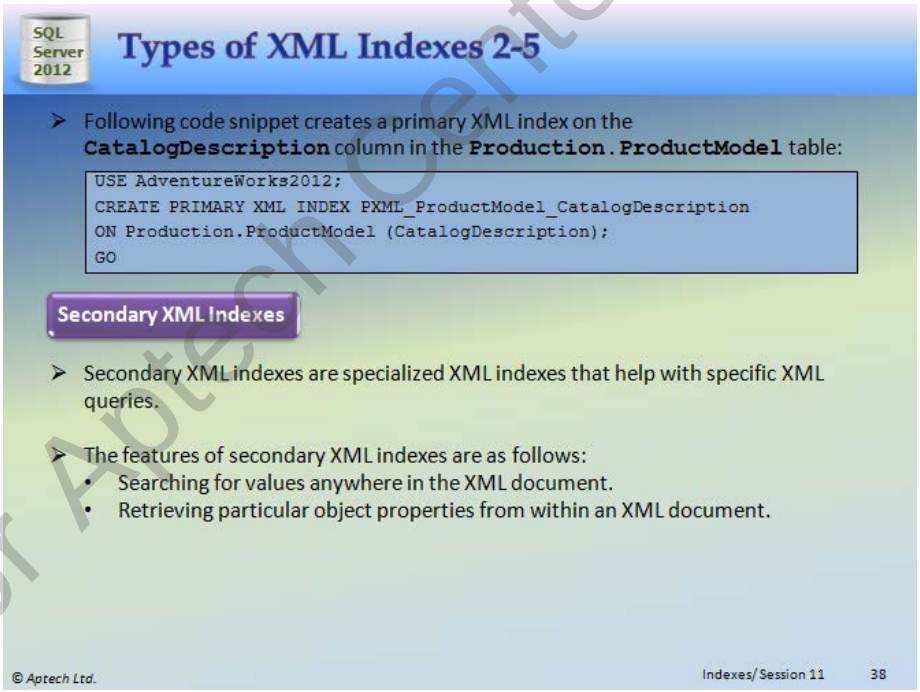
The xml data type is used to store XML documents and fragments as shown in the figure. An XML fragment is an XML instance that has a single top-level element missing.

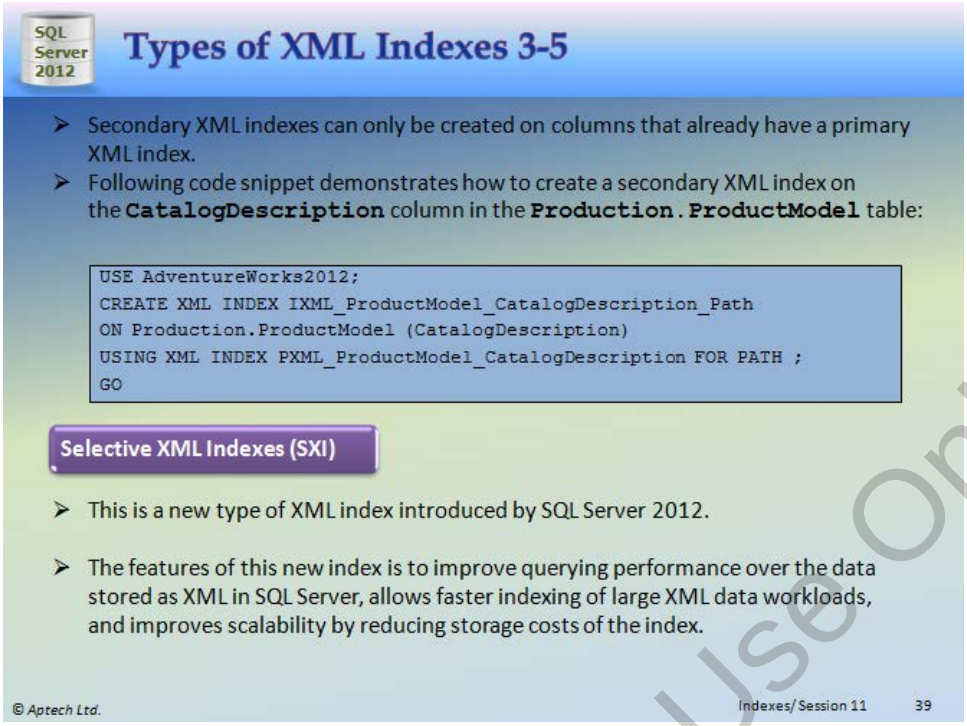
Due to the large size of XML columns, queries that search within these columns can be slow. You can speed up these queries by creating an XML index on each column. An XML index can be a clustered or a nonclustered index. Each table can have up to 249 XML indexes.

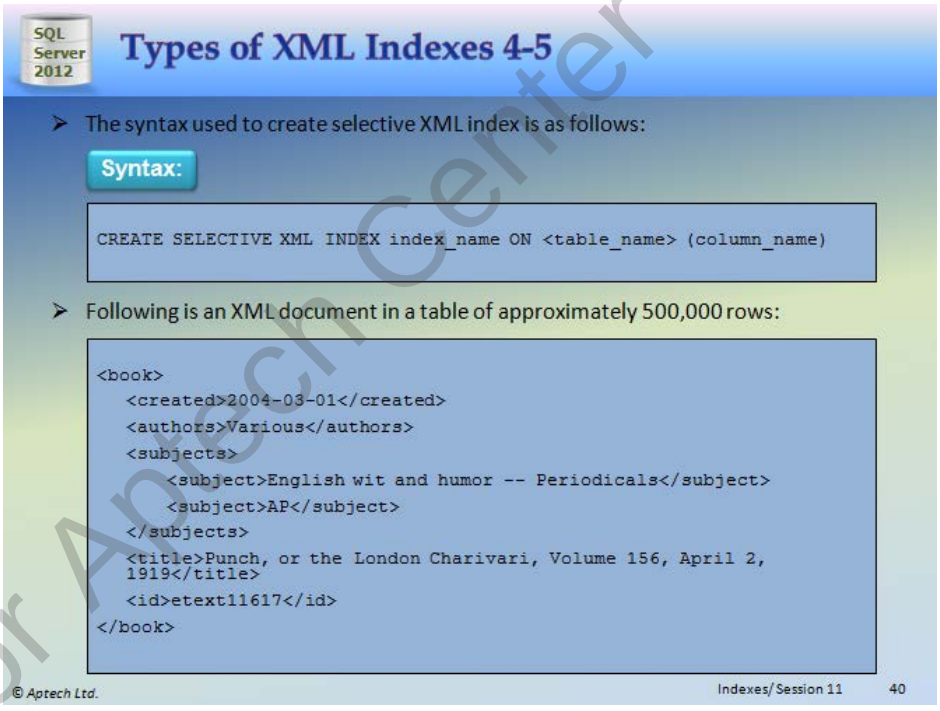
Types of XML Indexes

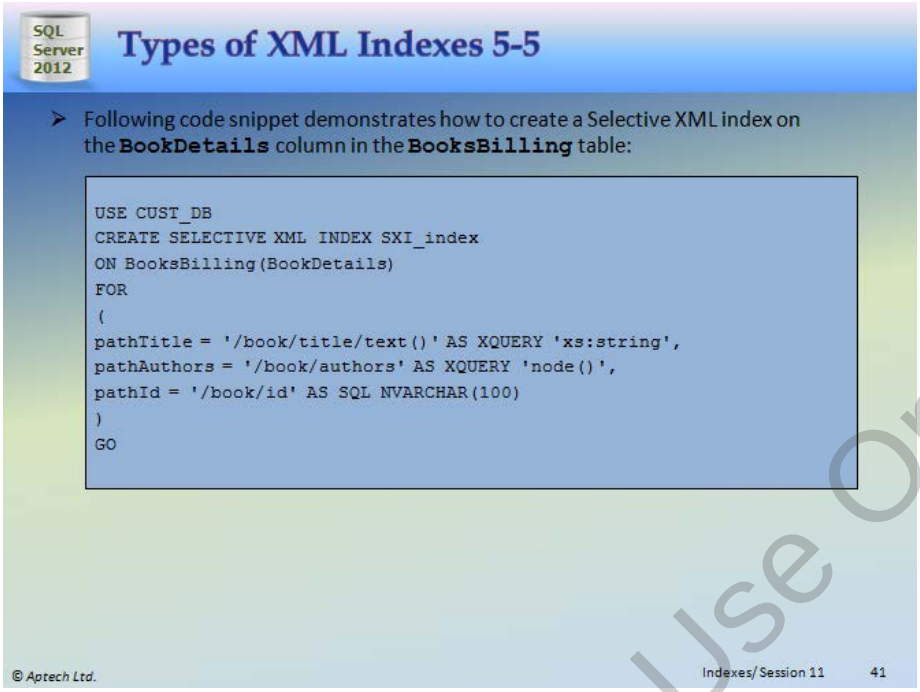
Slides 37 to 41











Using slides 37 to 41, explain the types of XML indexes.
XML indexes can be created on a table only if there is a clustered index based on the primary key of the table. This primary key cannot exceed 15 columns.

The different types of XML indexes are as follows:
Primary XML Indexes - The process of carrying out queries within an XML column can sometimes be slow. A primary XML index is created on each XML column to speed up these queries. It is a special index that shreds the XML data to store information. Explain the syntax to create a primary XML index.
Explain the Code Snippet that creates a primary XML index on the CatalogDescription column in the Production.ProductModel table.

Secondary XML Indexes - Secondary XML indexes are specialized XML indexes that help with specific XML queries. The features of secondary XML indexes are as follows:

- Searching for values anywhere in the XML document.
- Retrieving particular object properties from within an XML document.

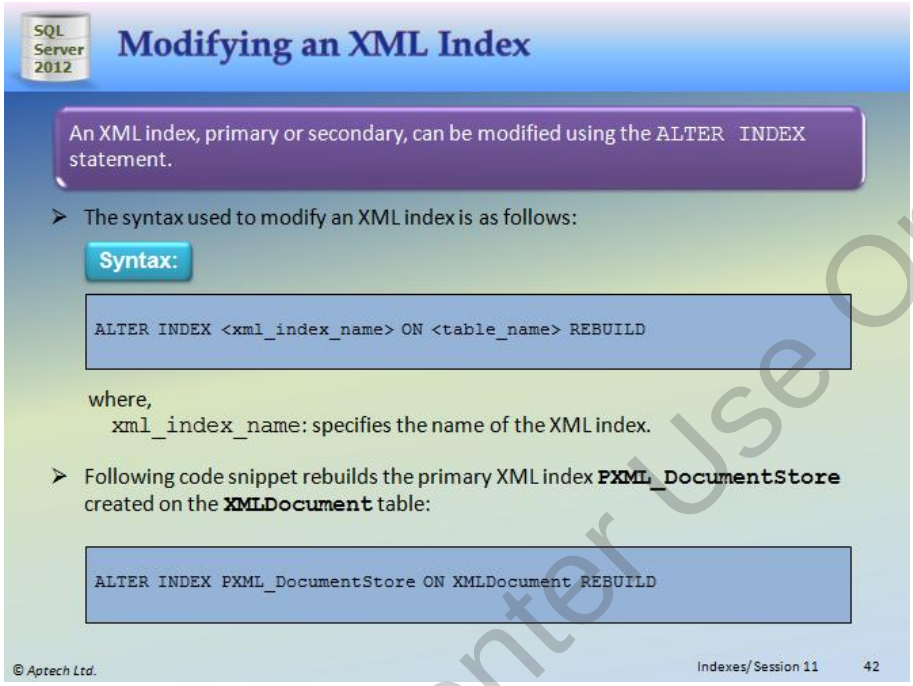
Secondary XML indexes can only be created on columns that already have a primary XML index. Explain the Code Snippet that demonstrates how to create a secondary XML index on the CatalogDescription column in the Production.ProductModel table.

Selective XML Indexes (SXI) – This is a new type of XML index introduced by SQL Server 2012. The features of this new index is to improve querying performance over the data stored as XML in SQL Server, allows faster indexing of large XML data workloads, and improves scalability by reducing storage costs of the index. Explain the syntax to create selective XML index.
Explain the Code Snippet that demonstrates how to create a Selective XML index on the BookDetails column in the BooksBilling table.

Mention SELECTIVE XML INDEX will work only in enterprise edition of SQL Server 2012.

Modifying an XML Index

Slide 42



Using slide 42, explain how to modify an XML index.
An XML index, primary or secondary, can be modified using the ALTER INDEX statement.
Explain the Code Snippet that rebuilds the primary XML index PXML_DocumentStore created on the XMLDocument table.

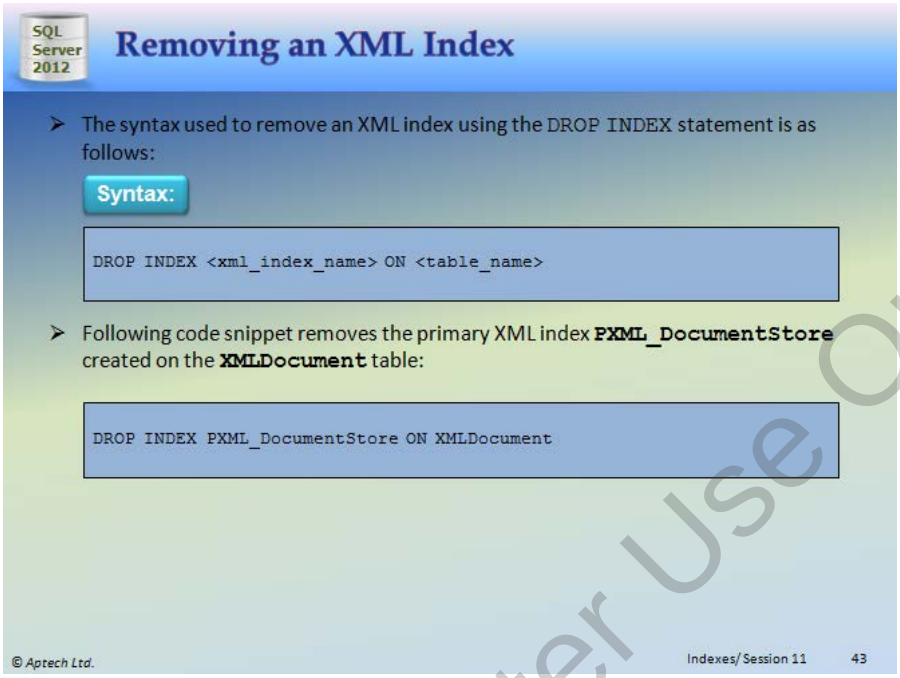
In-Class Question:

What is use of joins?

Answer:
Joins are used to retrieve data from two or more tables based on a logical relationship between tables.

Removing an XML Index

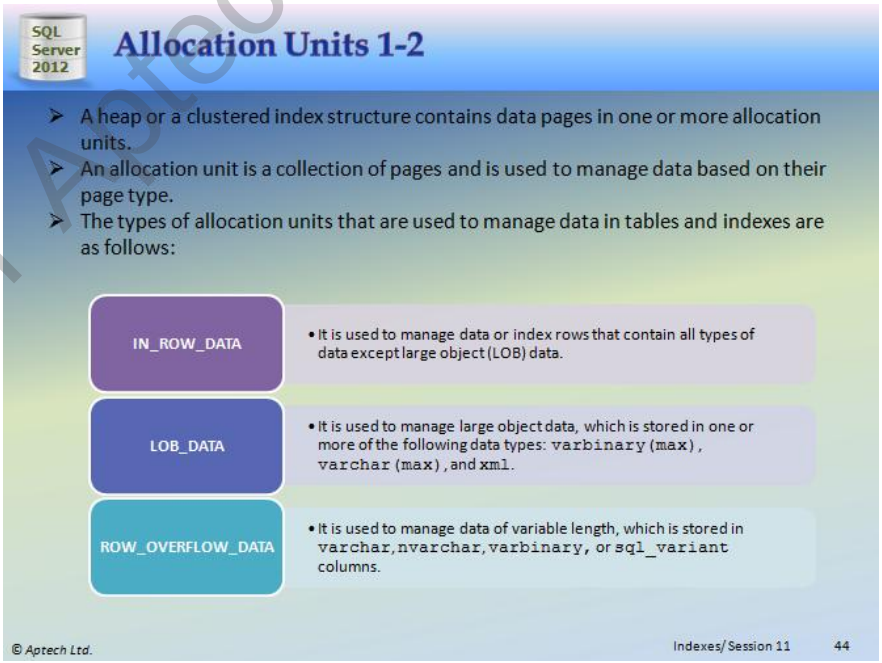
Slide 43

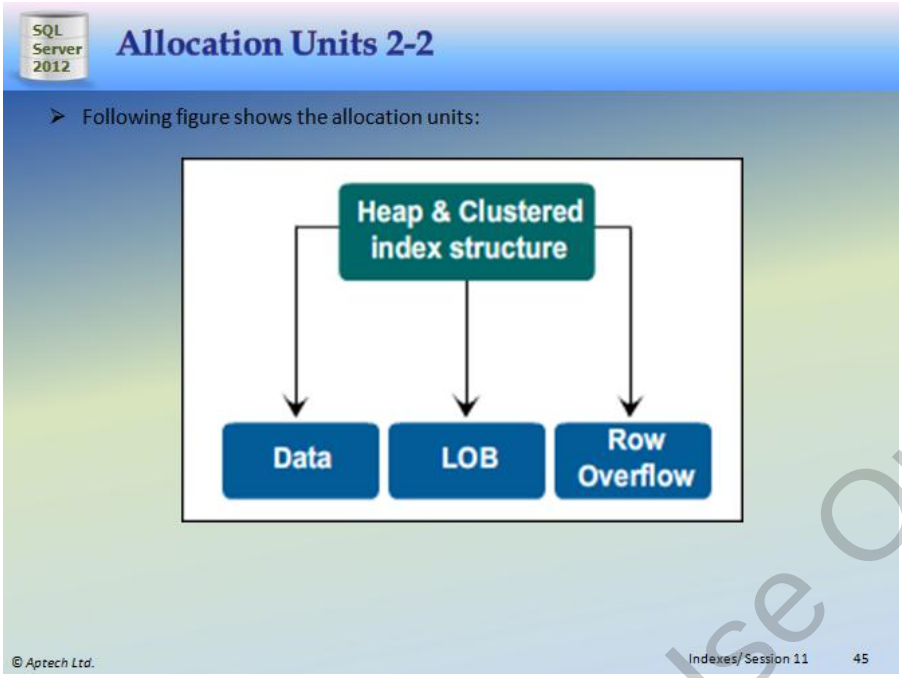


Using slide 43, explain how to remove an XML index.
Explain the syntax to remove an XML index using the DROP INDEX statement.
Explain the Code Snippet that removes the primary XML index PXML_DocumentStore created on the XMLDocument table.

Allocation Units

Slides 44 and 45





Using slides 44 and 45, explain the outer joins.

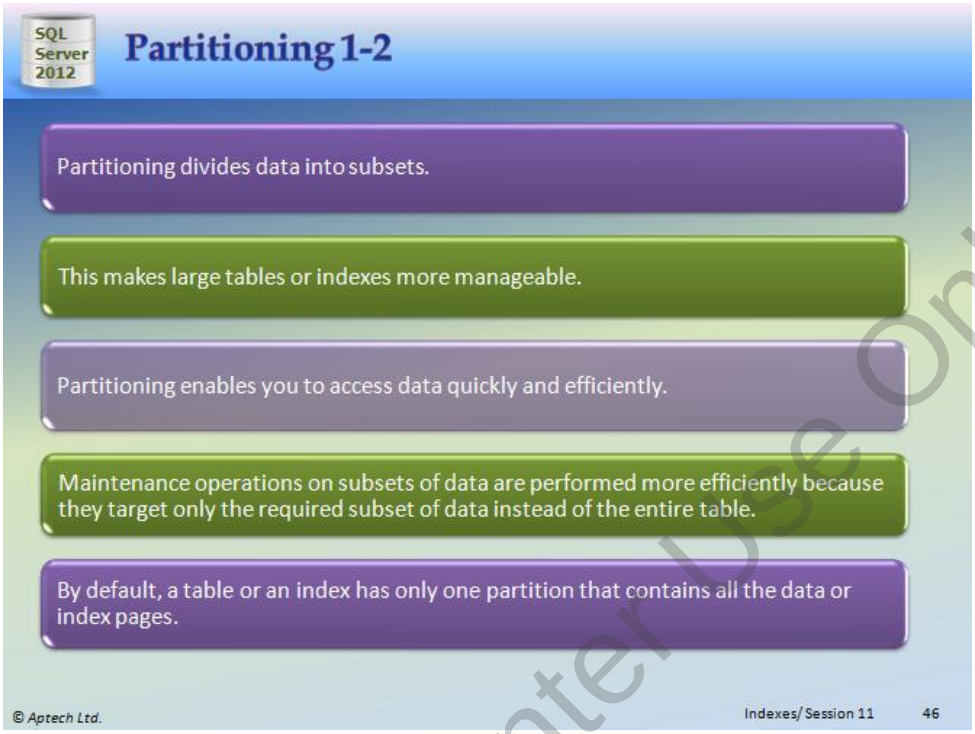
A heap or a clustered index structure contains data pages in one or more allocation units. An allocation unit is a collection of pages and is used to manage data based on their page type. The types of allocation units that are used to manage data in tables and indexes are as follows:
IN_ROW_DATA: It is used to manage data or index rows that contain all types of data except large object (LOB) data.
LOB_DATA: It is used to manage large object data, which is stored in one or more of the following data types: varbinary(max), varchar(max), and xml.
ROW_OVERFLOW_DATA: It is used to manage data of variable length, which is stored in varchar, nvarchar, varbinary, or sql_variant columns.

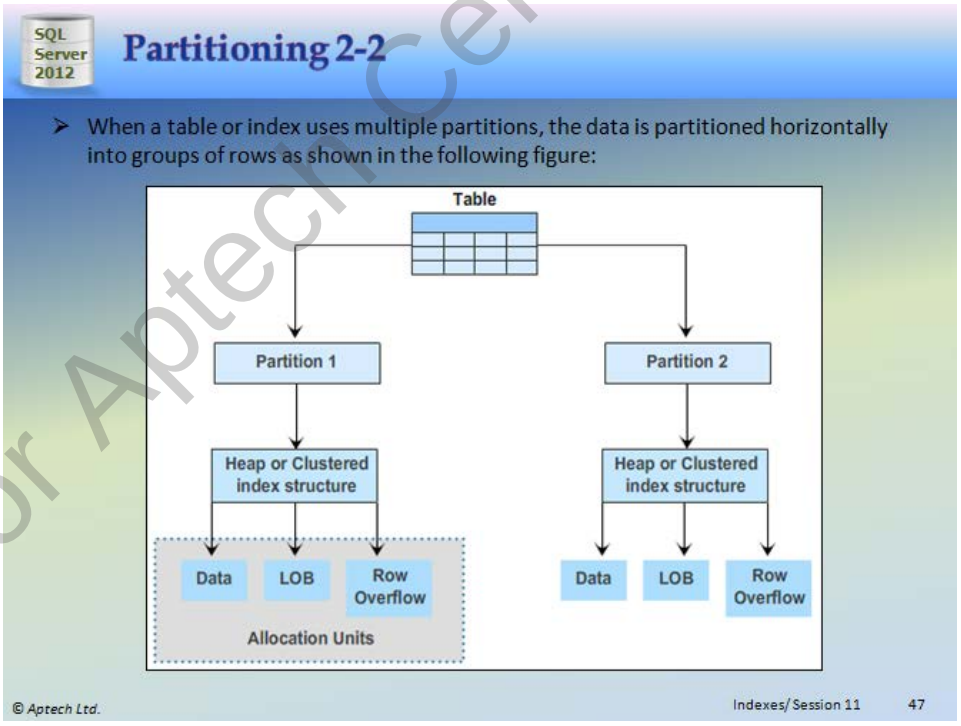
Figure on slide 45 shows the allocation of units.

Mention, a heap can have only one allocation unit of each type in a particular partition of a table.

Partitioning

Slides 46 and 47





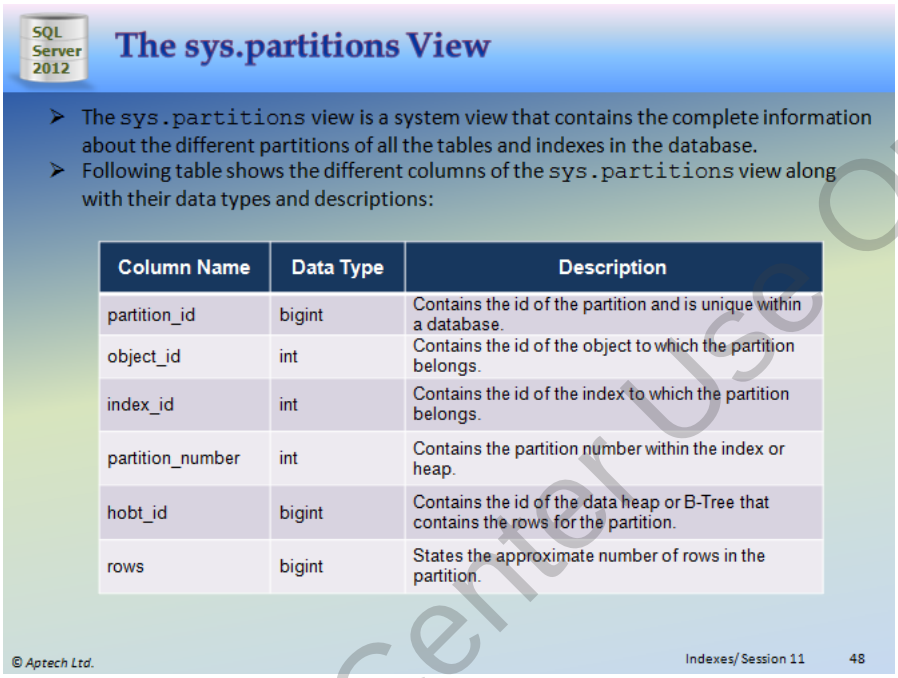
Using slides 46 and 47, explain the concept of partitioning.

Partitioning divides data into subsets. This makes large tables or indexes more manageable. Partitioning enables you to access data quickly and efficiently. Maintenance operations on subsets of data are performed more efficiently because they target only the required subset of data instead of the entire table.

By default, a table or an index has only one partition that contains all the data or index pages. When a table or index uses multiple partitions, the data is partitioned horizontally into groups of rows as shown in the figure on slide 47.

The sys.partitions View

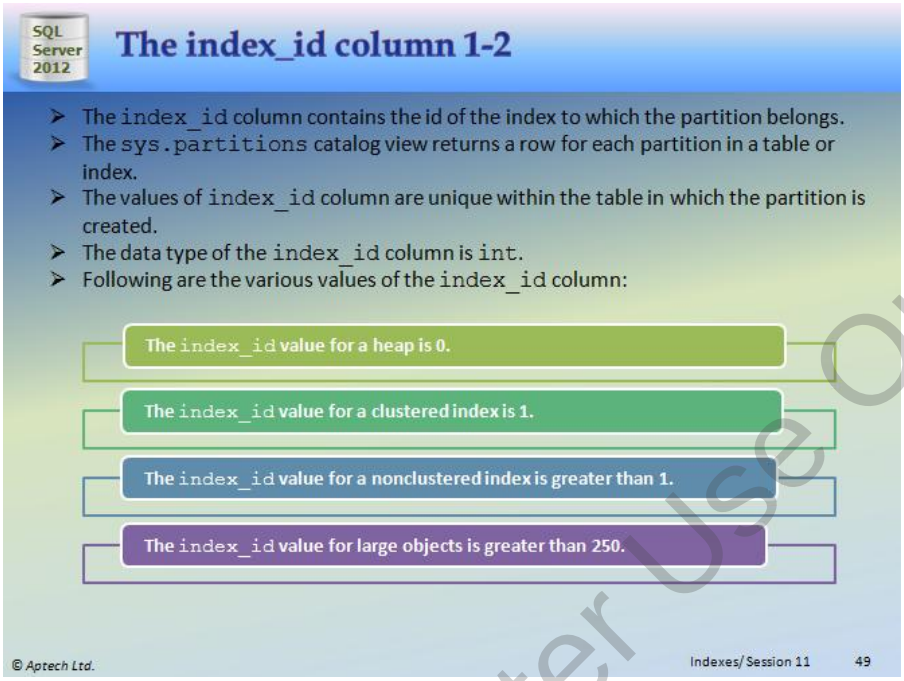
Slide 48

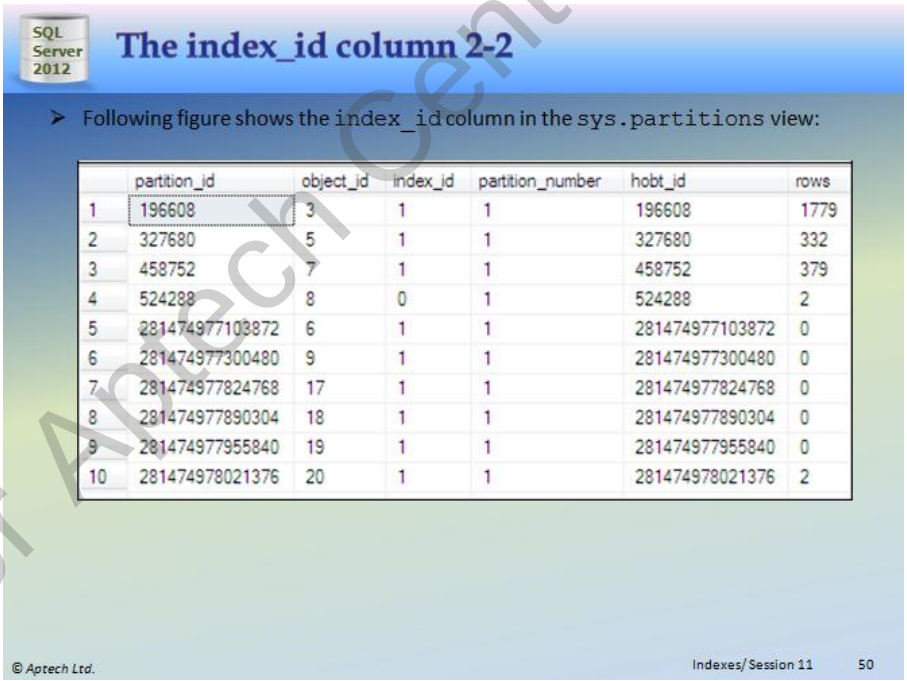


Using slide 48, explain the sys.partitions view.
The sys.partitions view is a system view that contains the complete information about the different partitions of all the tables and indexes in the database.
Table shows the different columns of the sys.partitions view along with their data types and descriptions.

The index_id column

Slides 49 and 50





	partition_id	object_id	index_id	partition_number	hobt_id	rows
1	196608	3	1	1	196608	1779
2	327680	5	1	1	327680	332
3	458752	7	1	1	458752	379
4	524288	8	0	1	524288	2
5	281474977103872	6	1	1	281474977103872	0
6	281474977300480	9	1	1	281474977300480	0
7	281474977824768	17	1	1	281474977824768	0
8	281474977890304	18	1	1	281474977890304	0
9	281474977955840	19	1	1	281474977955840	0
10	281474978021376	20	1	1	281474978021376	2

Using slides 49 and 50, explain the index_id column.
The index_id column contains the id of the index to which the partition belongs. The sys.partitions catalog view returns a row for each partition in a table or index. The values of index_id column are unique within the table in which the partition is created. The data type of the index_id column is int.

Following are the various values of the index_id column:

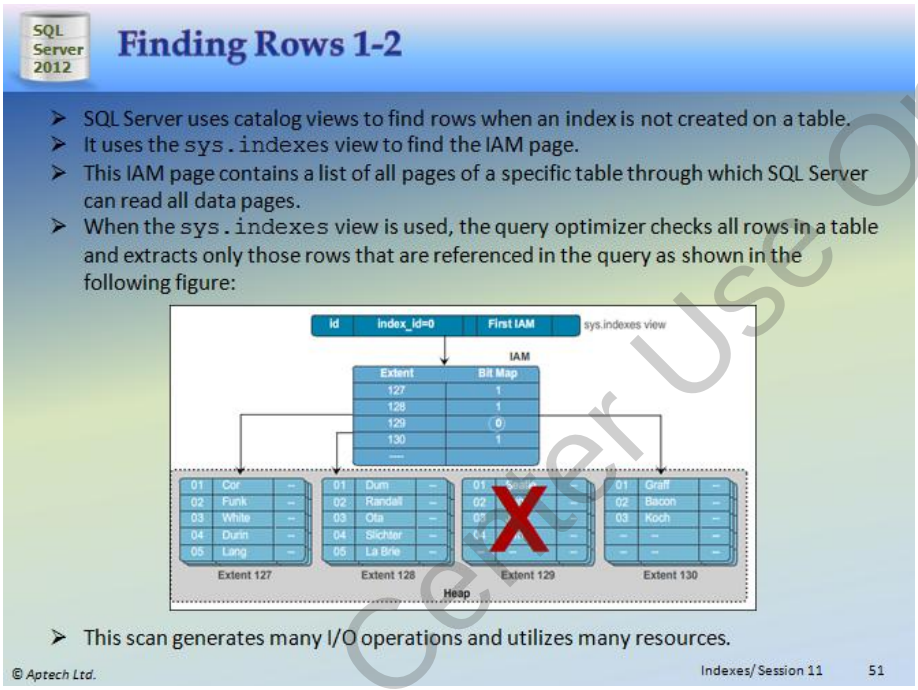
- The index_id value for a heap is 0.

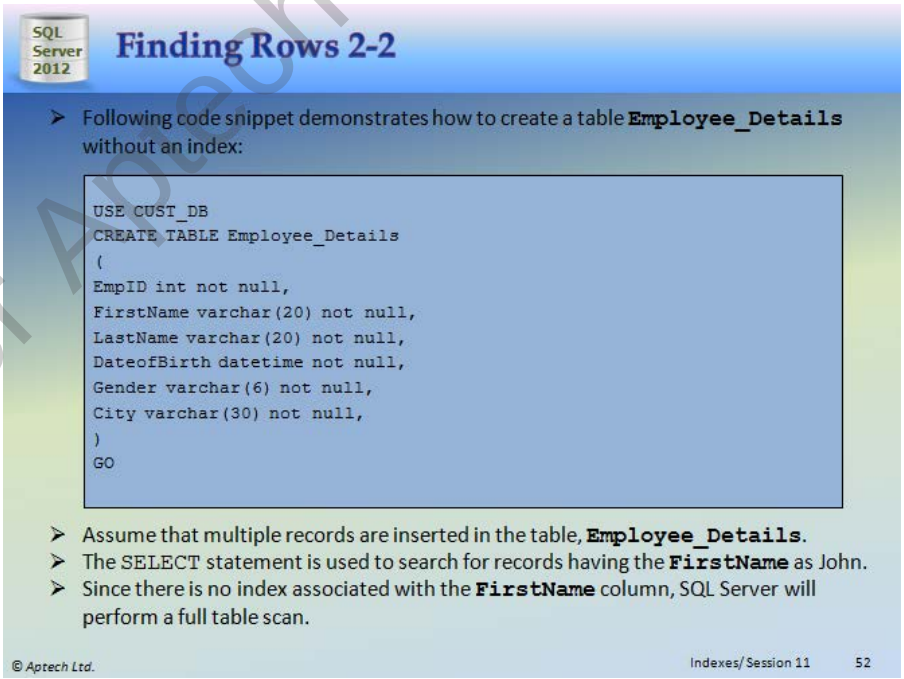
- The index_id value for a clustered index is 1.
- The index_id value for a nonclustered index is greater than 1.
- The index_id value for large objects is greater than 250.

Figure on slide 50 shows the index_id column in the sys.partitions view.

Finding Rows

Slides 51 and 52



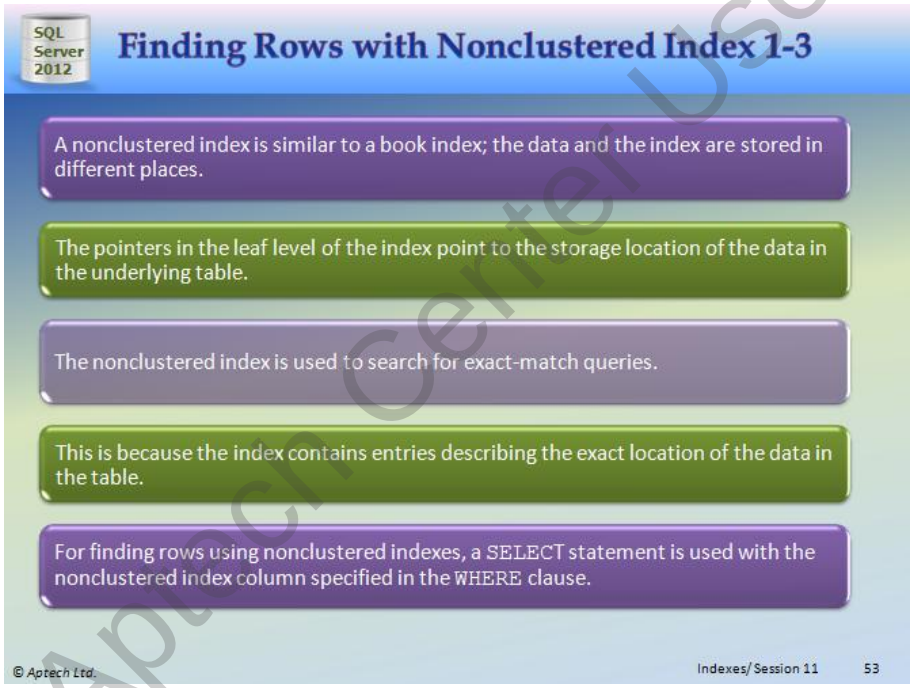


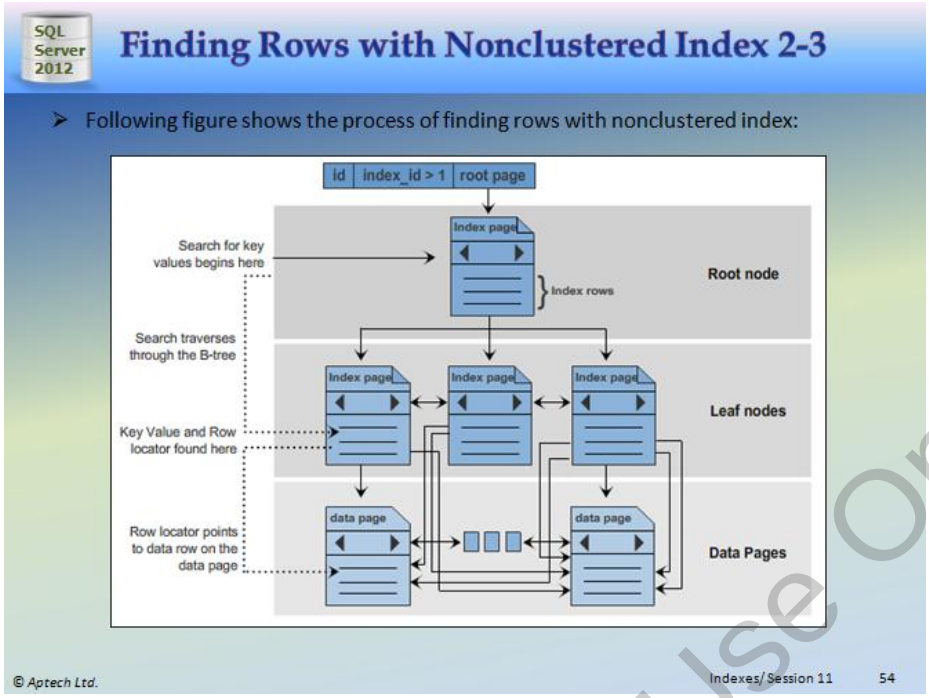
Using slides 51 and 52, explain how to find rows.

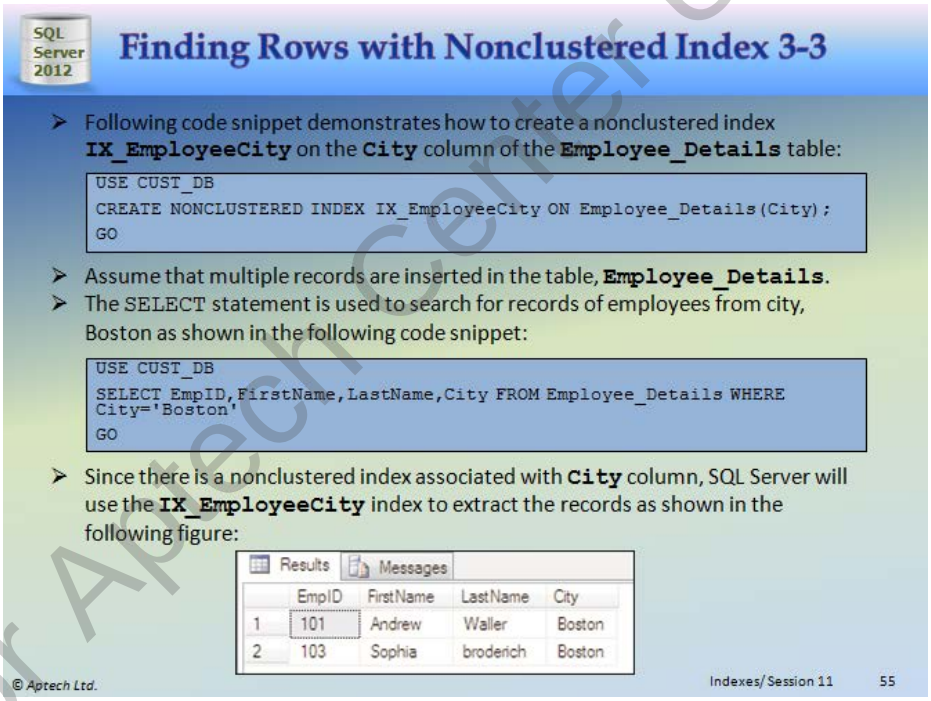
SQL Server uses catalog views to find rows when an index is not created on a table. It uses the sys.indexes view to find the IAM page. This IAM page contains a list of all pages of a specific table through which SQL Server can read all data pages.
When the sys.indexes view is used, the query optimizer checks all rows in a table and extracts only those rows that are referenced in the query as shown in the figure on slide 51. This scan generates many I/O operations and utilizes many resources.
Explain the Code Snippet that demonstrates how to create a table Employee_Details without an index.
Assume that multiple records are inserted in the table, Employee_Details. The SELECT statement is used to search for records having the FirstName as John. Since there is no index associated with the FirstName column, SQL Server will perform a full table scan.

Finding Rows with Nonclustered Index

Slides 53 to 55







Using slides 53 to 55, explain how to find rows with nonclustered index.

A nonclustered index is similar to a book index; the data and the index are stored in different places. The pointers in the leaf level of the index point to the storage location of the data in the underlying table. The nonclustered index is used to search for exact-match queries. This is because the index contains entries describing the exact location of the data in the table.

For finding rows using nonclustered indexes, a SELECT statement is used with the nonclustered index column specified in the WHERE clause.

Figure on slide 54 shows the process of finding rows with nonclustered index.

Explain the Code Snippet that demonstrates how to create a nonclustered index IX_EmployeeCity on the City column of the Employee_Details table.

Assume that multiple records are inserted in the table, Employee_Details. The SELECT statement is used to search for records of employees from city, Boston as shown in the Code Snippet on slide 55.
Since there is a nonclustered index associated with City column, SQL Server will use the IX_EmployeeCity index to extract the records as shown in the figure on slide 55.

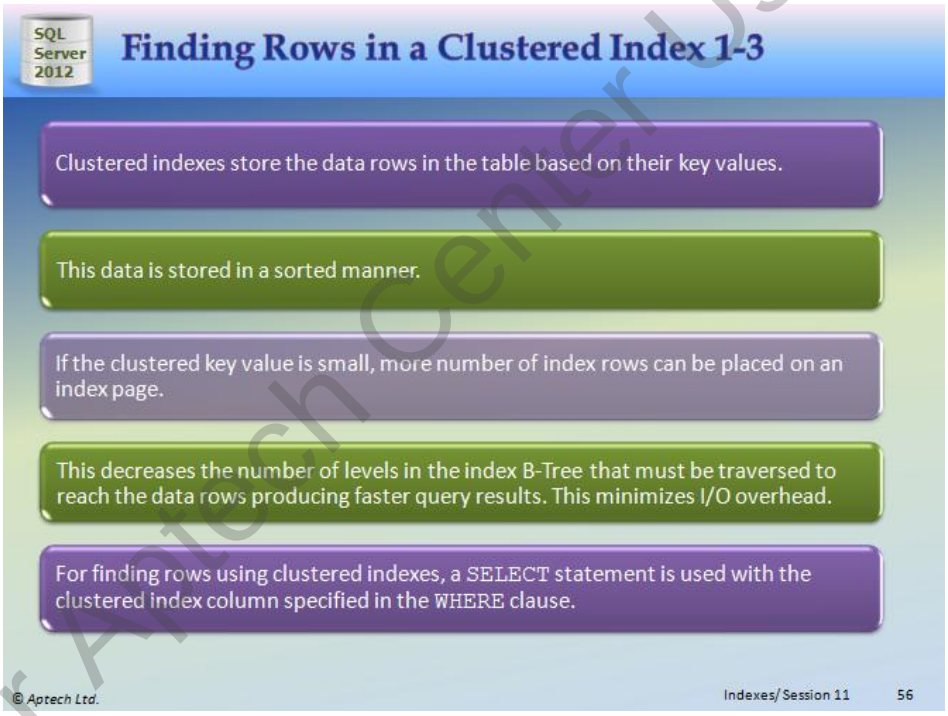
In-Class Question:

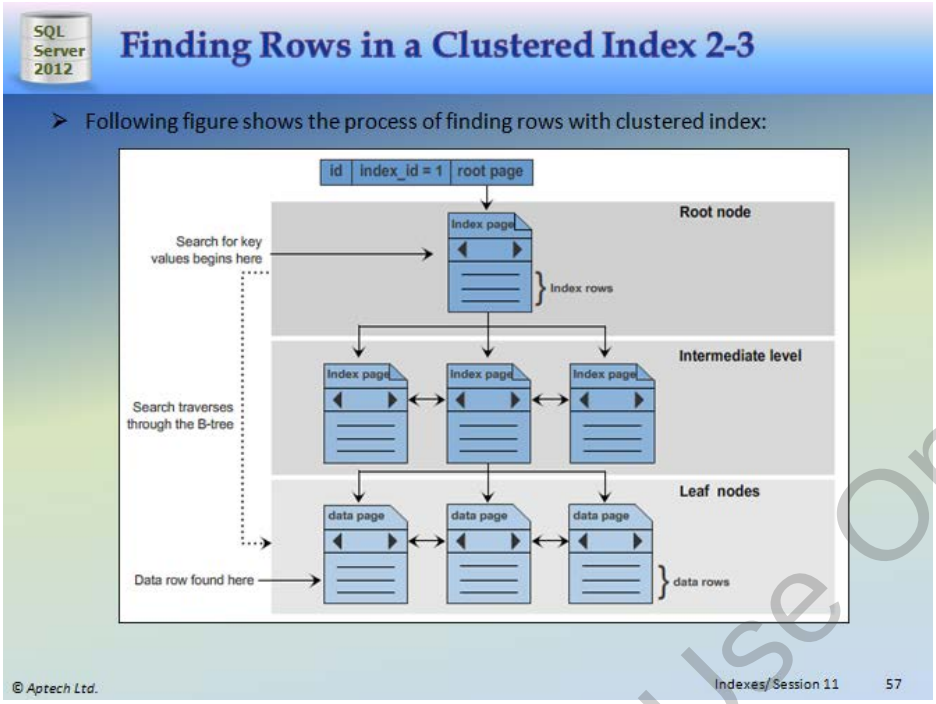
What is used to delete a row from a view?

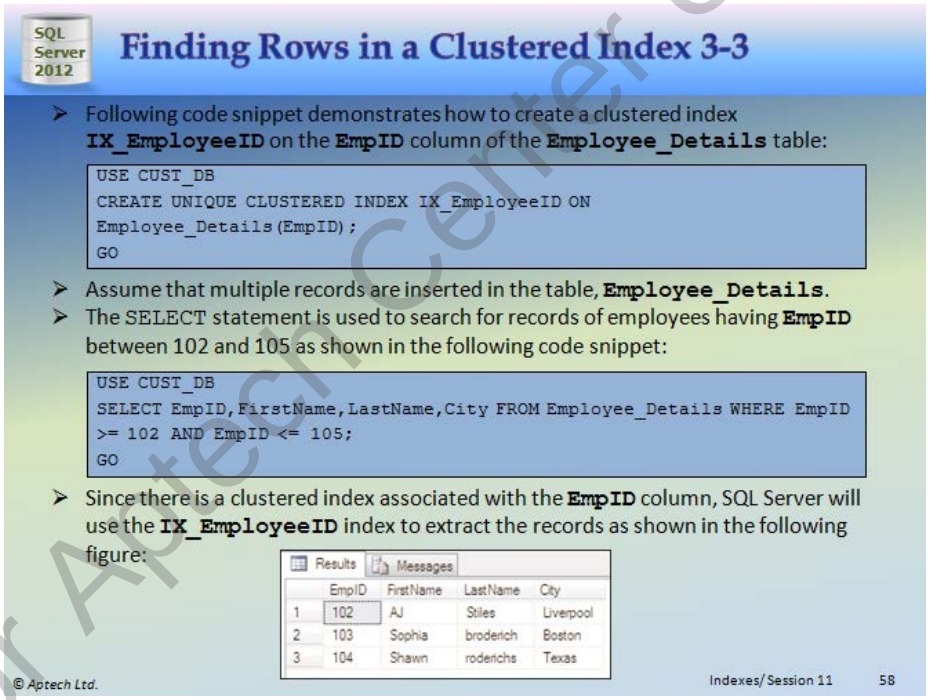
Answer:
Rows can be deleted from the view using the DELETE statement.

Finding Rows in a Clustered Index

Slides 56 to 58







Using slides 56 to 58, explain how to find rows in clustered index.

Clustered indexes store the data rows in the table based on their key values. This data is stored in a sorted manner. If the clustered key value is small, more number of index rows can be placed on an index page. This decreases the number of levels in the index B-Tree that must be traversed to reach the data rows producing faster query results. This minimizes I/O overhead.

For finding rows using clustered indexes, a SELECT statement is used with the clustered index column specified in the WHERE clause.

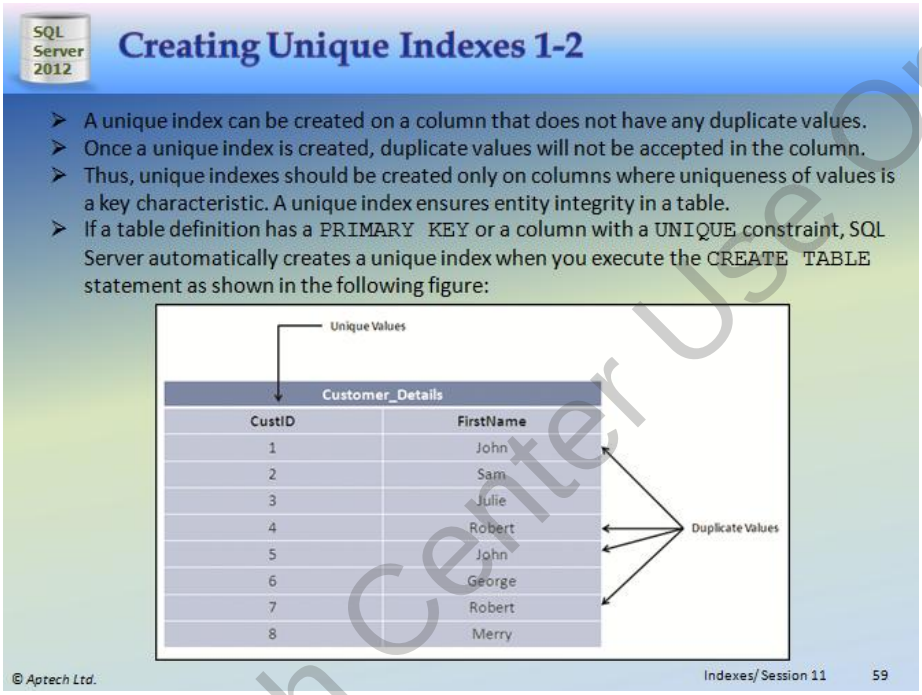
Figure on slide 58 shows the process of finding rows with clustered index.

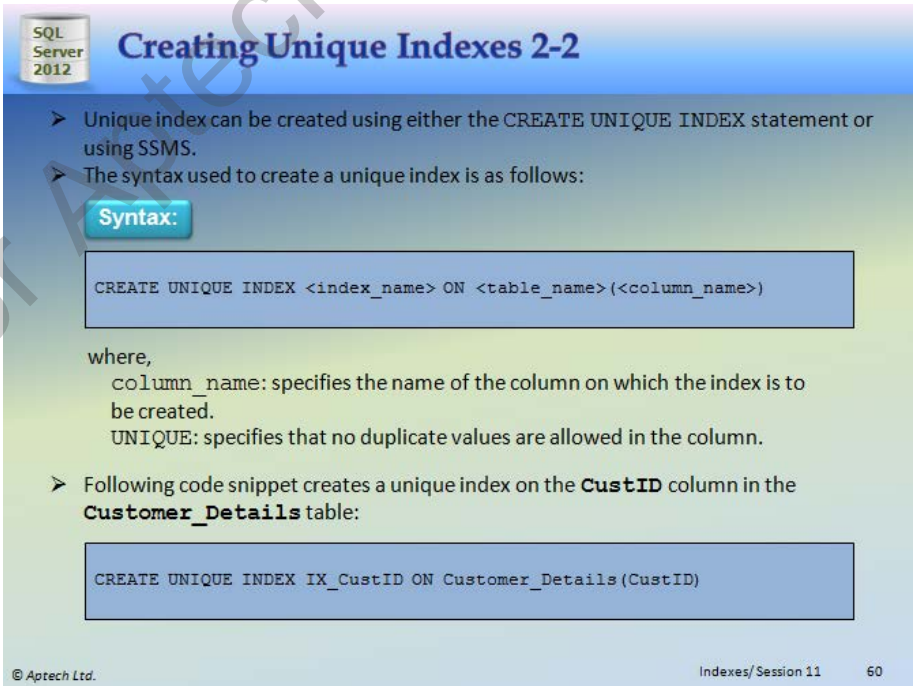
Explain the Code Snippet that demonstrates how to create a clustered index IX_EmployeeID on the EmpID column of the Employee_Details table.

Assume that multiple records are inserted in the table, Employee_Details. The SELECT statement is used to search for records of employees having EmpID between 102 and 105 as shown in the Code Snippet on slide 58.
Since there is a clustered index associated with the EmpID column, SQL Server will use the IX_EmployeeID index to extract the records as shown in the figure on slide 58.

Creating Unique Indexes

Slides 59 and 60



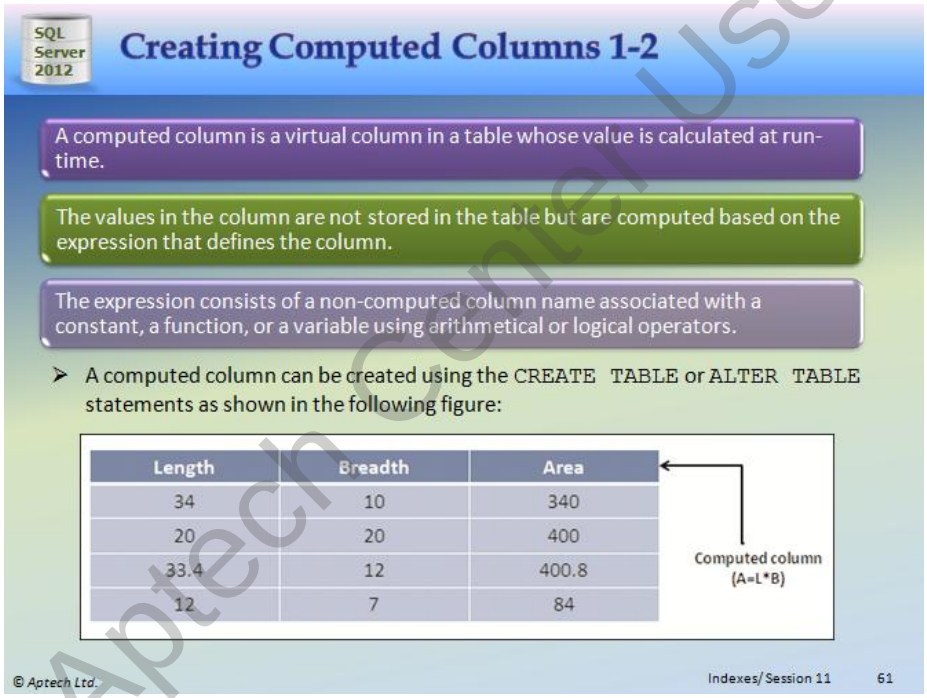


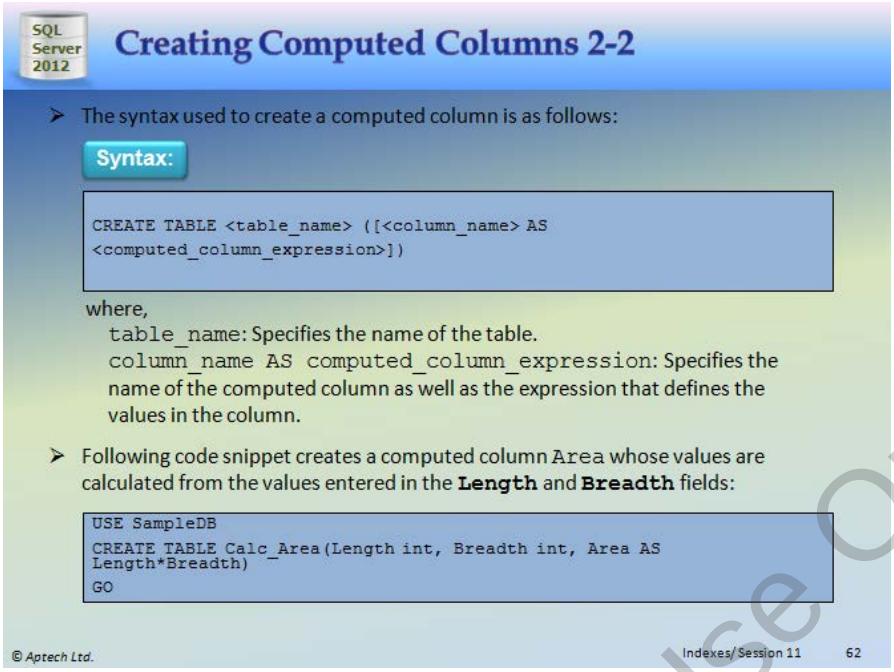
Using slides 59 and 60, explain how to create a unique index.

A unique index can be created on a column that does not have any duplicate values. Also, once a unique index is created, duplicate values will not be accepted in the column. Hence, unique indexes should be created only on columns where uniqueness of values is a key characteristic. A unique index ensures entity integrity in a table.
If a table definition has a PRIMARY KEY or a column with a UNIQUE constraint, SQL Server automatically creates a unique index when you execute the CREATE TABLE statement as shown in the figure on slide 59. Unique index can be created using either the CREATE UNIQUE INDEX statement or using SSMS.
Explain the syntax is used to create a unique index.
Explain the Code Snippet that creates a unique index on the CustID column in the Customer_Details table.

Creating Computed Columns

Slides 61 and 62





Using slides 61 and 62, explain how to create computed columns.

A computed column is a virtual column in a table whose value is calculated at run-time. The values in the column are not stored in the table but are computed based on the expression that defines the column. The expression consists of a non-computed column name associated with a constant, a function, or a variable using arithmetical or logical operators.

A computed column can be created using the CREATE TABLE or ALTER TABLE statements as shown in the figure on slide 61.

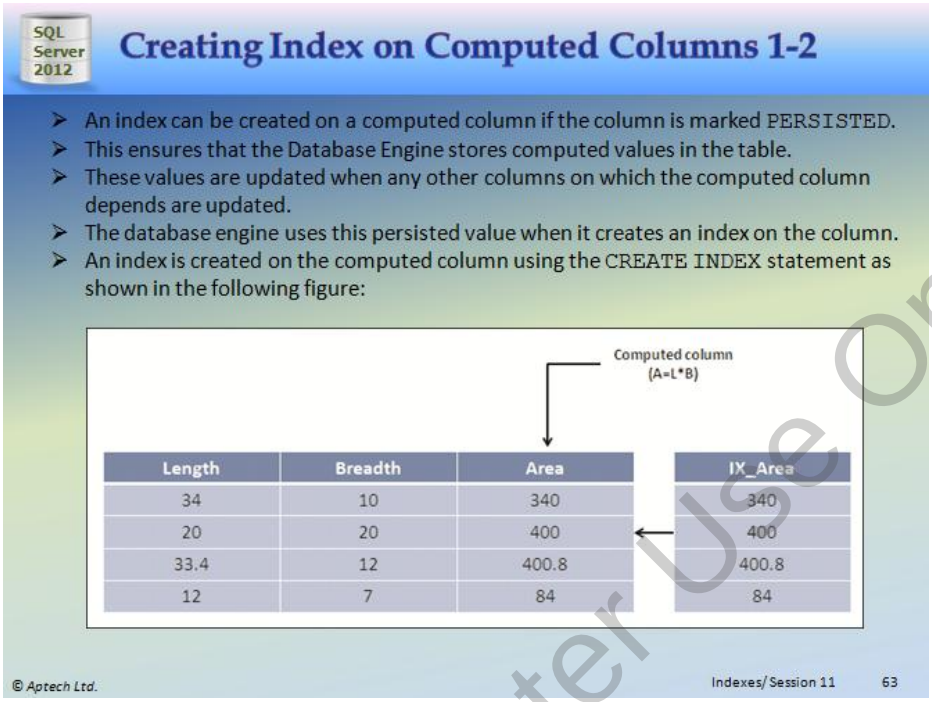
Explain the syntax is used to create a computed column.

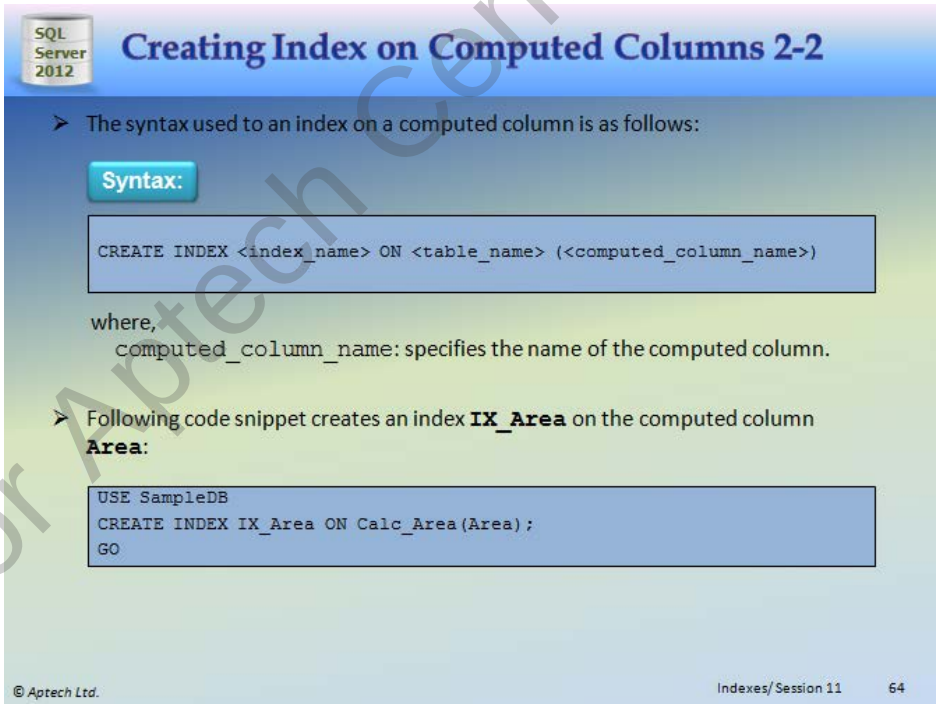
Explain the Code Snippet that creates a computed column Area whose values are calculated from the values entered in the Length and Breadth fields.

Mention a computed column cannot be used as a part of any PRIMARY KEY, UNIQUE KEY, FOREIGN KEY, or CHECK constraint definition.

Creating Index on Computed Columns

Slides 63 and 64



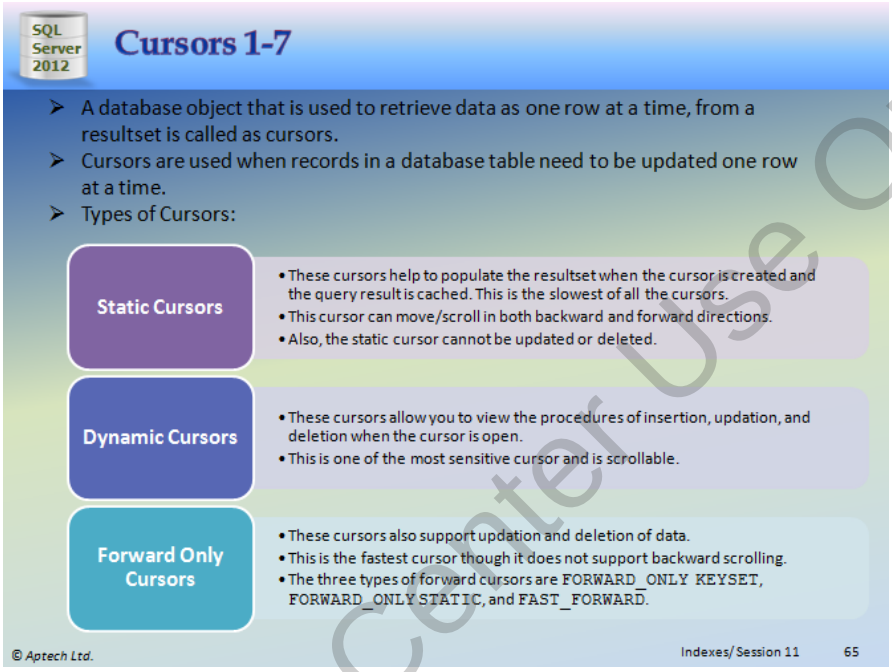


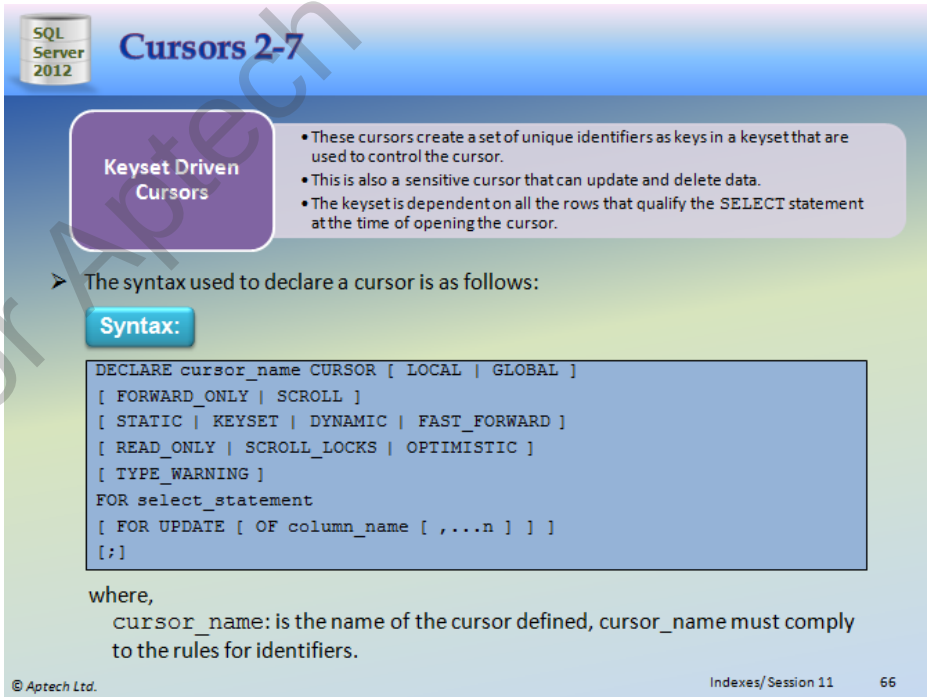
Using slides 63 and 64, explain how to create index on computed columns.
An index can be created on a computed column if the column is marked PERSISTED. This ensures that the Database Engine stores computed values in the table. These values are updated when any other columns on which the computed column depends are updated. The database engine uses this persisted value when it creates an index on the column.

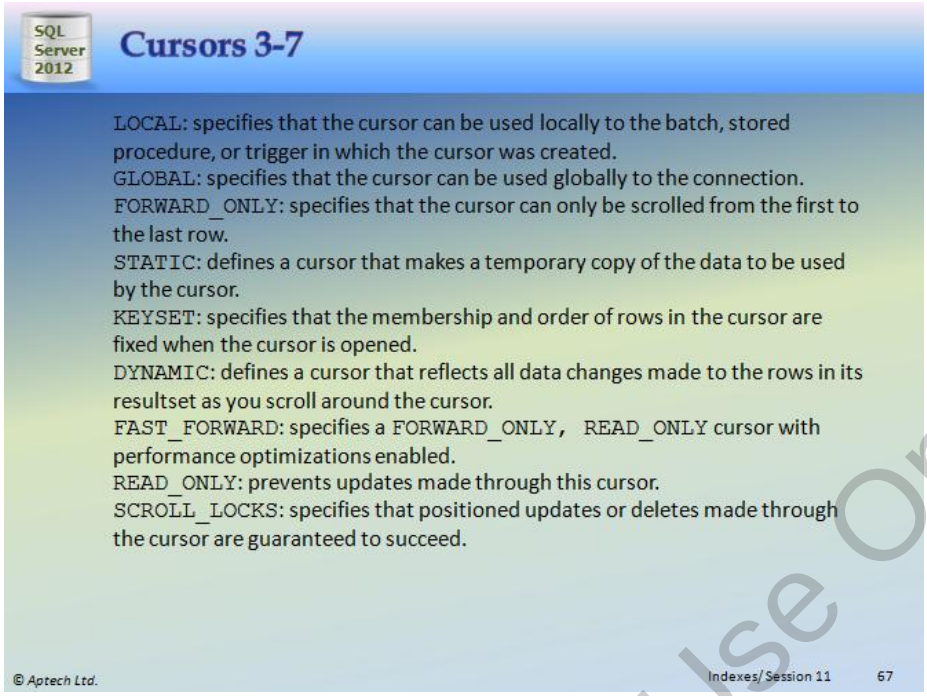
An index is created on the computed column using the CREATE INDEX statement as shown in the figure on slide 63.
Explain the syntax creates an index on a computed column.
Explain the Code Snippet that creates an index IX_Area on the computed column Area.

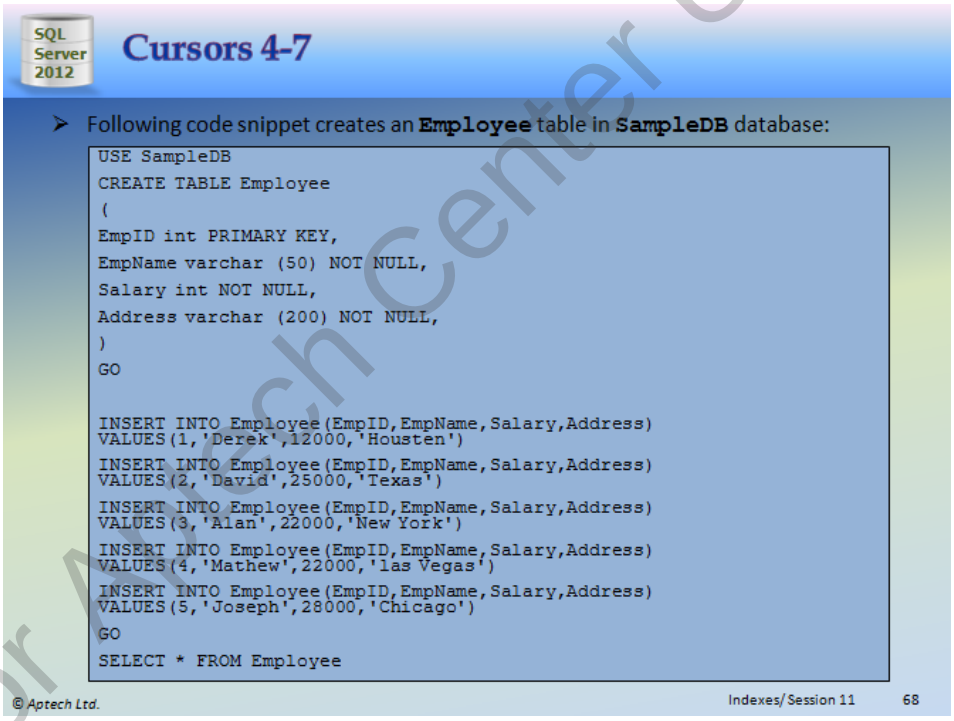
Cursors

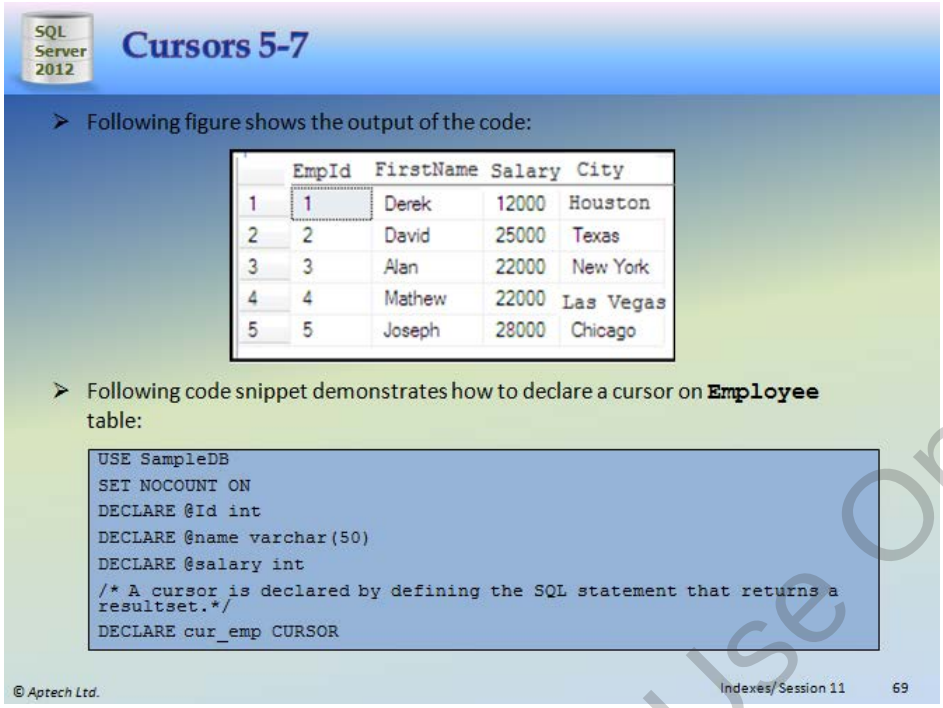
Slides 65 to 71

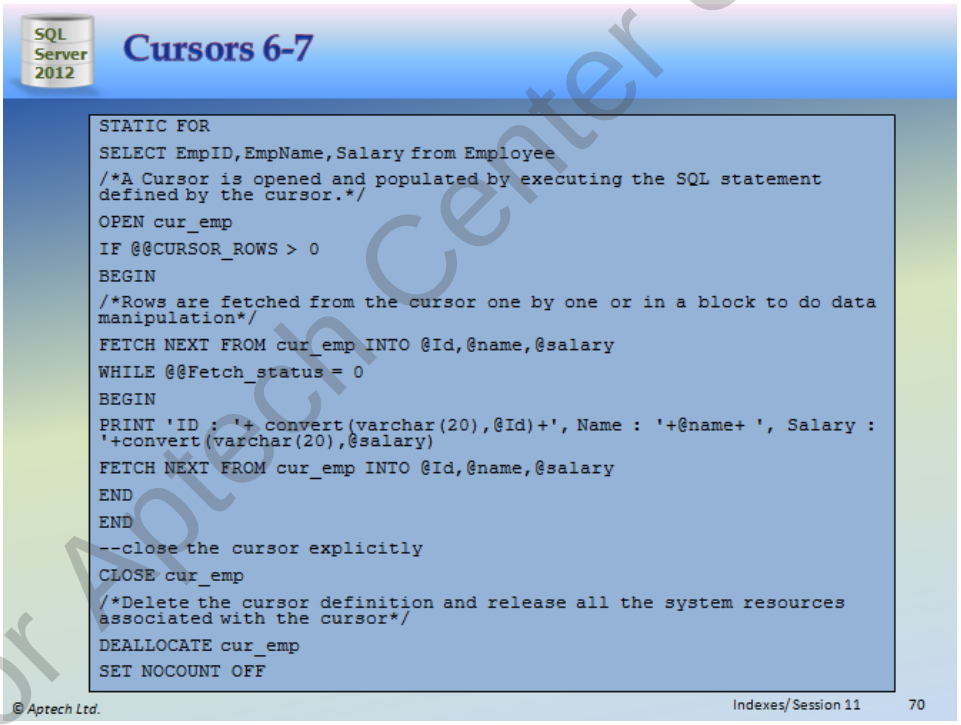


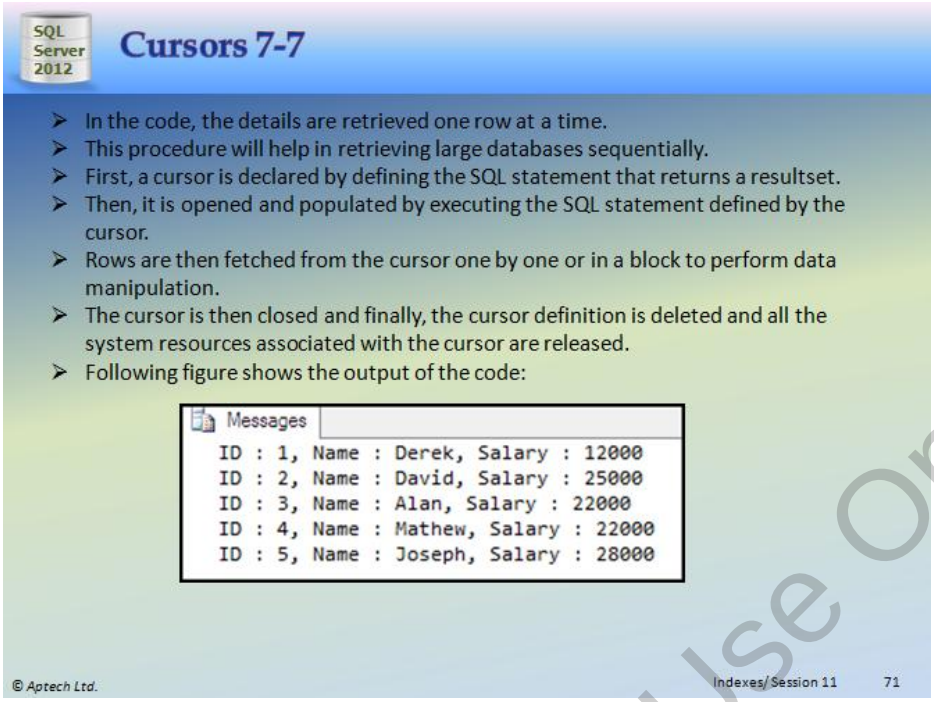












Using slides 65 to 71, explain the cursors.

A database object that is used to retrieve data as one row at a time, from a resultset is called as cursors. Cursors are used instead of the Transact-SQL commands that operate on all the rows at one time in the resultset. Cursors are used when records in a database table need to be updated one row at a time.

Types of Cursors

Static Cursors – These cursors help to populate the resultset when the cursor is created and the query result is cached. This is the slowest of all the cursors. This cursor can move/scroll in both backward and forward directions. Also, the static cursor cannot be updated or deleted.

Dynamic Cursors – These cursors allow you to view the procedures of insertion, updation, and deletion when the cursor is open. This is one of the most sensitive cursor and is scrollable.

Forward Only Cursors - These cursors also support updation and deletion of data. This is the fastest cursor though it does not support backward scrolling. The three types of forward cursors are FORWARD_ONLY KEYSET, FORWARD_ONLY STATIC, and FAST_FORWARD.

Keyset Driven Cursors - These cursors create a set of unique identifiers as keys in a keyset that are used to control the cursor. This is also a sensitive cursor that can update and delete data. The keyset is dependent on all the rows that qualify the SELECT statement at the time of opening the cursor.

Explain the syntax to declare a cursor.

Explain the Code Snippet that 20 creates an Employee table in SampleDB database.

Figure on slide 69 shows the output of the code.

Explain the Code Snippet that demonstrates how to declare a cursor on Employee table.

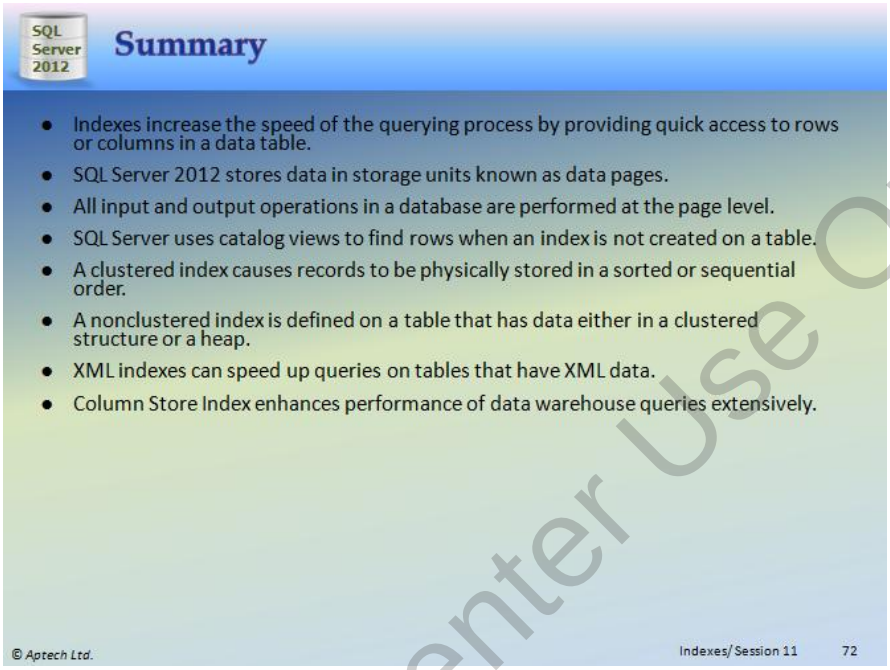
Figure on slide 71 shows the output of the code.

In the code, the details are retrieved one row at a time. This procedure will help in retrieving large databases sequentially. First, a cursor is declared by defining the SQL statement that returns a resultset. Then, it is opened and populated by executing the SQL statement defined by the cursor. Rows are then fetched from the cursor one by one or in a block to

perform data manipulation. The cursor is then closed and finally, the cursor definition is deleted and all the system resources associated with the cursor are released.

Summarize Session

Slide 72



Using slide 72, summarize the session. End the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- Indexes increase the speed of the querying process by providing quick access to rows or columns in a data table.
- SQL Server 2012 stores data in storage units known as data pages.
- All input and output operations in a database are performed at the page level.
- SQL Server uses catalog views to find rows when an index is not created on a table.
- A clustered index causes records to be physically stored in a sorted or sequential order.
- A nonclustered index is defined on a table that has data either in a clustered structure or a heap.
- XML indexes can speed up queries on tables that have XML data.
- Column Store Index enhances performance of data warehouse queries extensively.

11.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Triggers topic that is offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 12 – Triggers

12.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

12.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain triggers
- Explain the different types of triggers
- Explain the procedure to create DML triggers
- Explain the procedure to alter DML triggers
- Describe nested triggers
- Describe update functions
- Explain the handling of multiple rows in a session
- Explain the performance implication of triggers

12.1.2 Teaching Skills

To teach this session, you should be well-versed with the triggers and different types of triggers. Along with this also prepare yourself the procedure to create and alter DML triggers, nested triggers, update functions, handling multiple rows in a session, and performance implication of triggers.
You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

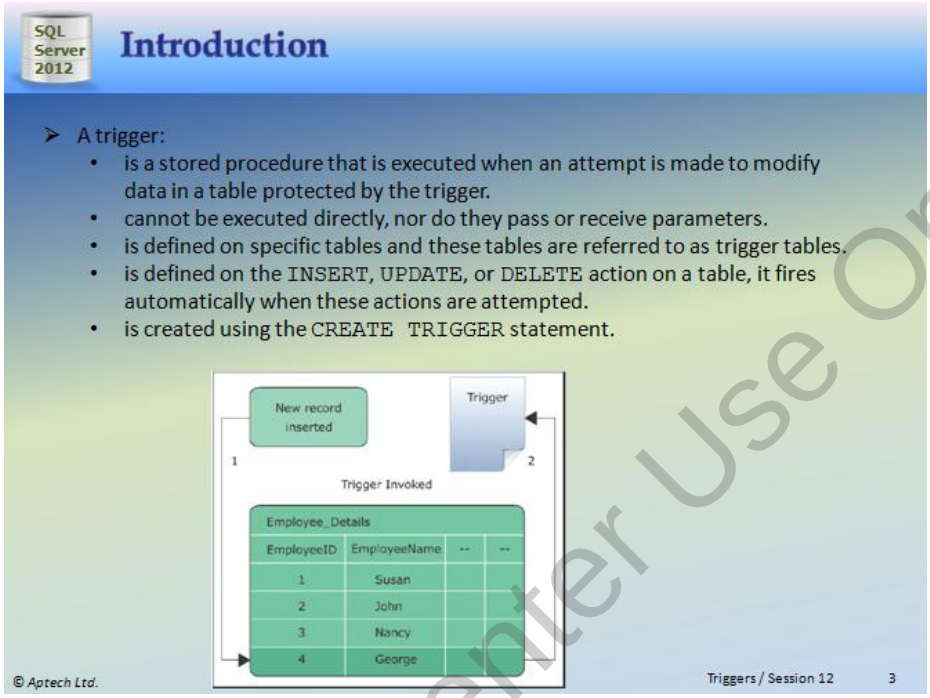
In-Class Activities:
Follow the order given here during In-Class activities.

Overview of the Session:
Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces triggers and different types of triggers. They will also know about the procedure to create and alter DML triggers, nested triggers, update functions, handling multiple rows in a session, and performance implication of triggers.

12.2 In-Class Explanations

Introduction

Slides 3



Using slide 3, explain the introduction of triggers.

A trigger is a stored procedure that is executed when an attempt is made to modify data in a table protected by the trigger. Unlike standard system stored procedures, triggers cannot be executed directly, nor do they pass or receive parameters. Triggers are defined on specific tables and these tables are referred to as trigger tables.

If a trigger is defined on the INSERT, UPDATE, or DELETE action on a table, it fires automatically when these actions are attempted. This automatic execution of the trigger cannot be circumvented. In SQL Server 2012, triggers are created using the CREATE TRIGGER statement. Figure displays an example of triggers.

In-Class Question:

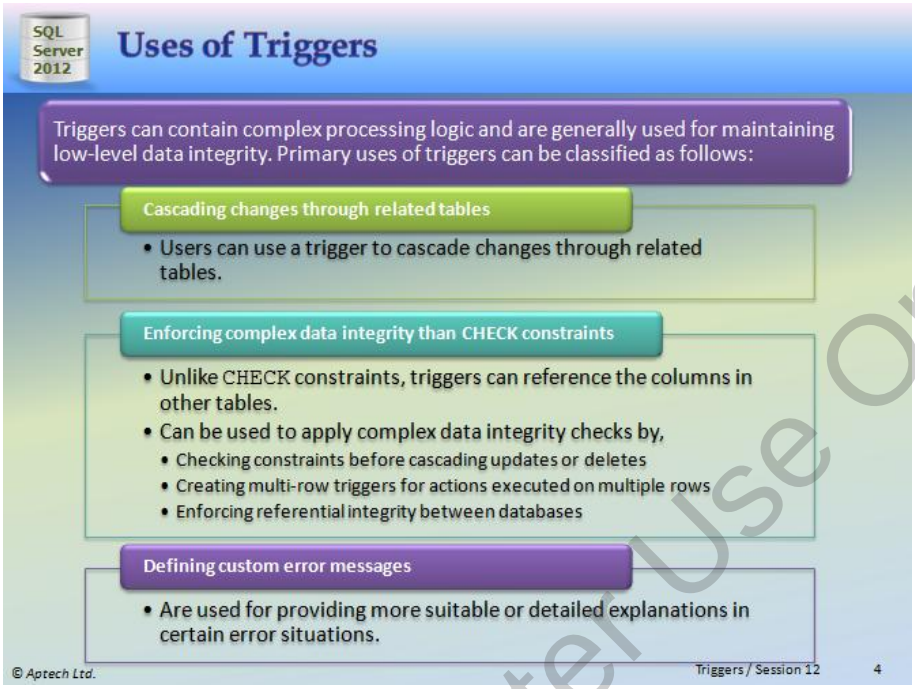
What is a trigger?

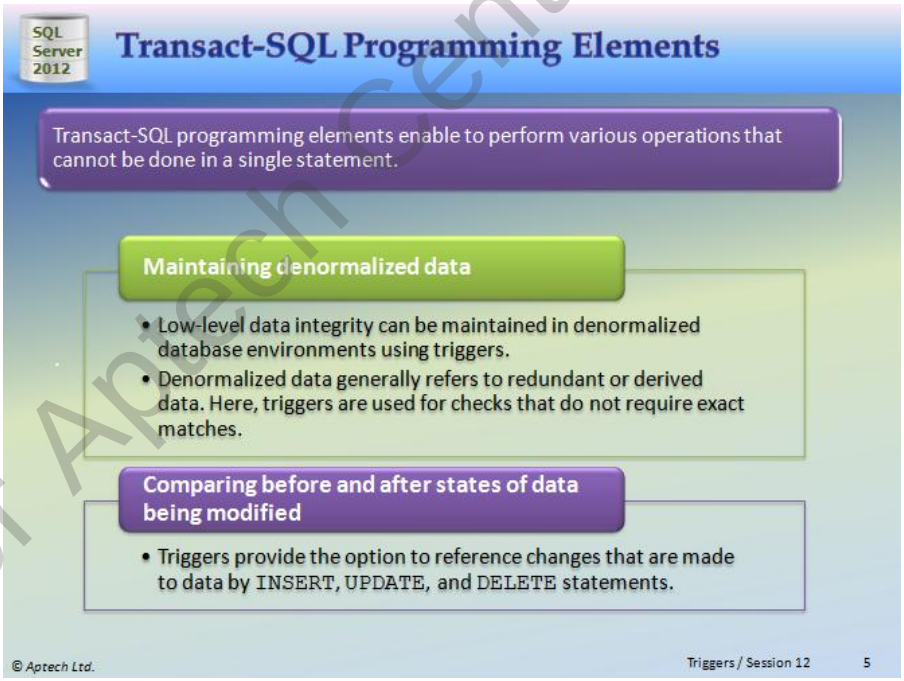
Answer:

A trigger is a stored procedure that is executed when an attempt is made to modify data in a table protected by the trigger.

Uses of Triggers

Slides 4 and 5





Using slides 4 and 5, explain the use of Triggers.

Triggers can contain complex processing logic and are generally used for maintaining low-level data integrity. The primary uses of triggers can be classified as follows:

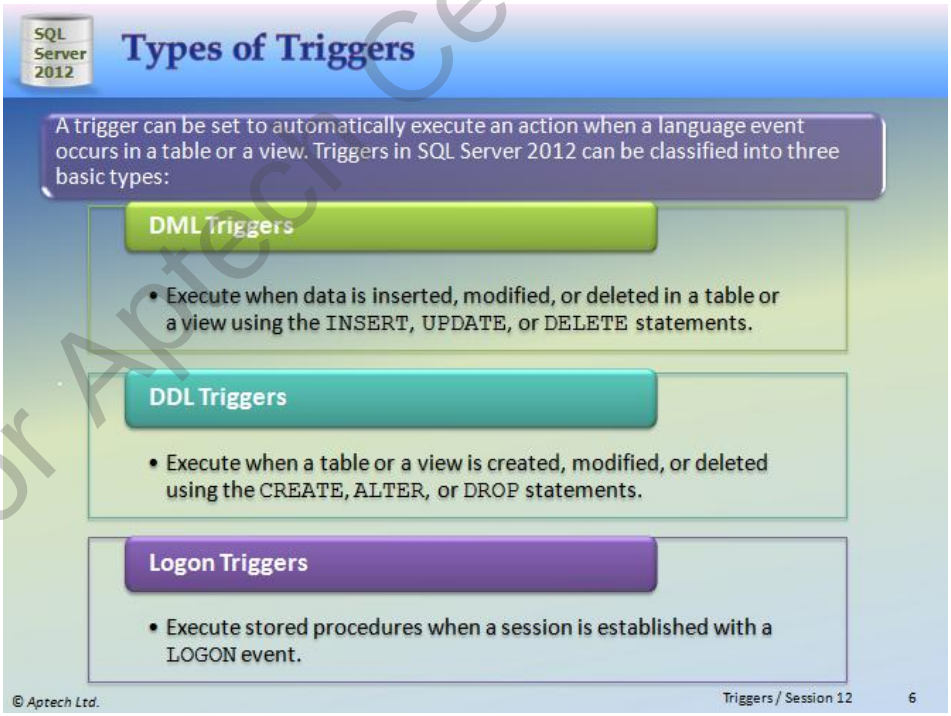
- Cascading changes through related tables: Users can use a trigger to cascade changes through related tables. For example, consider a table Salary_Details having a FOREIGN KEY, EmpID referencing the PRIMARY KEY, and EmpID of the Employee_Details table. If

an update or a delete event occurs in the Employee_Details table, an update or delete trigger can be defined to cascade these changes to the Salary_Details table.

- Enforcing more complex data integrity than CHECK constraints: Unlike CHECK constraints, triggers can reference the columns in other tables. This feature can be used to apply complex data integrity checks. Data integrity can be enforced by: Checking constraints before cascading updates or deletes. Creating multi-row triggers for actions executed on multiple rows. Enforcing referential integrity between databases.
- Defining custom error messages: Custom error messages are used for providing more suitable or detailed explanations in certain error situations. Triggers can be used to invoke such predefined custom error messages when relevant error conditions occur.
- Maintaining denormalized data: Low-level data integrity can be maintained in denormalized database environments using triggers. Denormalized data generally refers to redundant or derived data. Here, triggers are used for checks that do not require exact matches. For example, if the value of the year is to be checked against complete dates, a trigger can be used to perform the check.
- Comparing before and after states of data being modified: Triggers provide the option to reference changes that are made to data by INSERT, UPDATE, and DELETE statements. This allows users to reference the affected rows when modifications are carried out through triggers.

Types of Triggers

Slide 6



Using slide 6, explain the types of triggers.

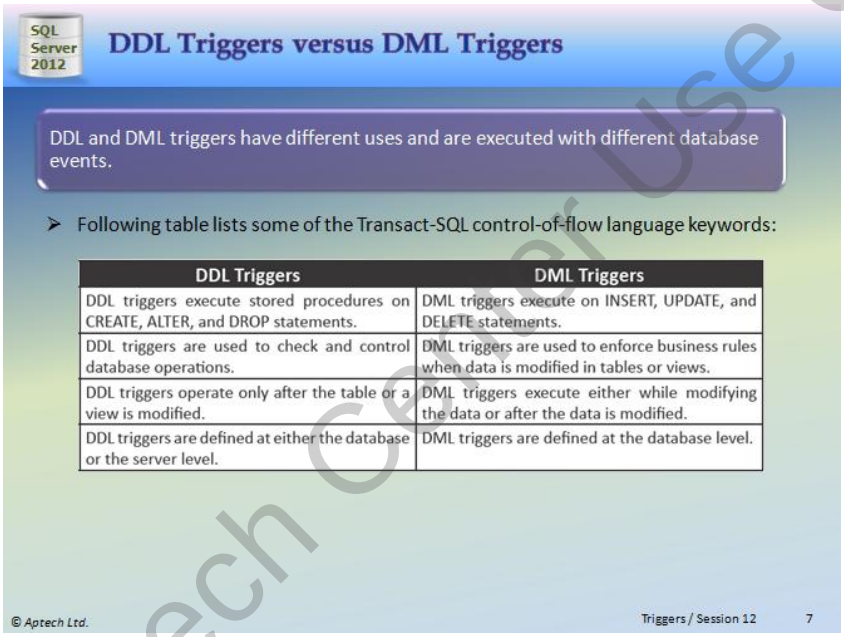
A trigger can be set to automatically execute an action when a language event occurs in a table or a view. Language events can be classified as DML events and DDL events. Triggers associated with DML events are known as DML triggers, whereas triggers associated with DDL events are known as DDL triggers.

Triggers in SQL Server 2012 can be classified into three basic types:

- DML Triggers: DML triggers execute when data is inserted, modified, or deleted in a table or a view using the INSERT, UPDATE, or DELETE statements.
- DDL Triggers: DDL triggers execute when a table or a view is created, modified, or deleted using the CREATE, ALTER, or DROP statements.
- Logon Triggers: Logon triggers execute stored procedures when a session is established with a LOGON event. These triggers are invoked after the login authentication is complete and before the actual session is established. Logon triggers control server sessions by restricting invalid logins or limiting the number of sessions.

DDL Triggers and DML Triggers

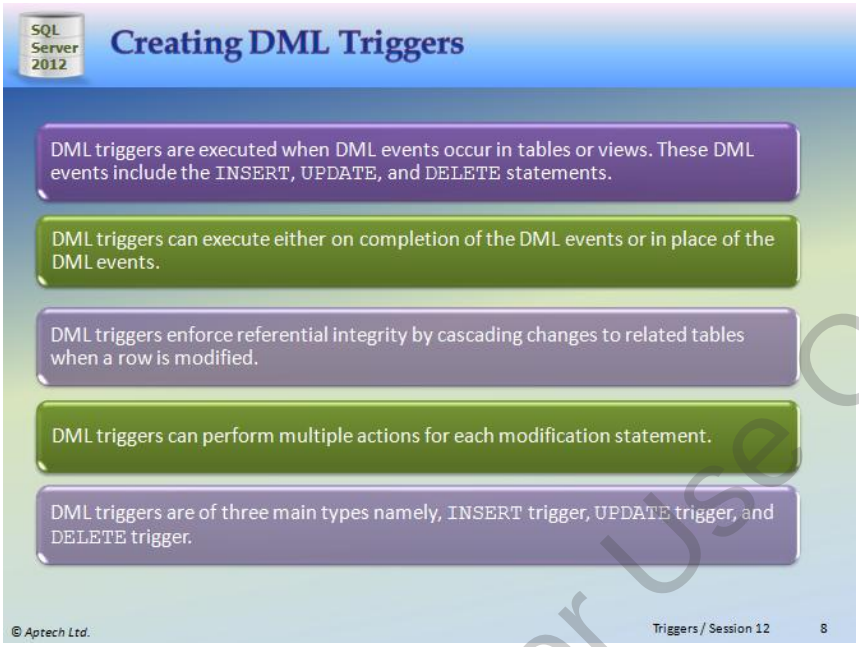
Slide 7



Using slides 6 and 7, explain difference in DDL triggers and DML triggers.
DDL and DML triggers have different uses and are executed with different database events.
Explain the table which displays the differences between DDL and DML triggers.

Creating DML Triggers

Slide 8



Using slide 8, explain how to create DML triggers.

DML triggers are executed when DML events occur in tables or views. These DML events include the INSERT, UPDATE, and DELETE statements. DML triggers can execute either on completion of the DML events or in place of the DML events.

DML triggers enforce referential integrity by cascading changes to related tables when a row is modified. DML triggers can perform multiple actions for each modification statement.

DML triggers are of three main types:

- INSERT trigger
- UPDATE trigger
- DELETE trigger

Figure displays the types of DML triggers.

In-Class Question:

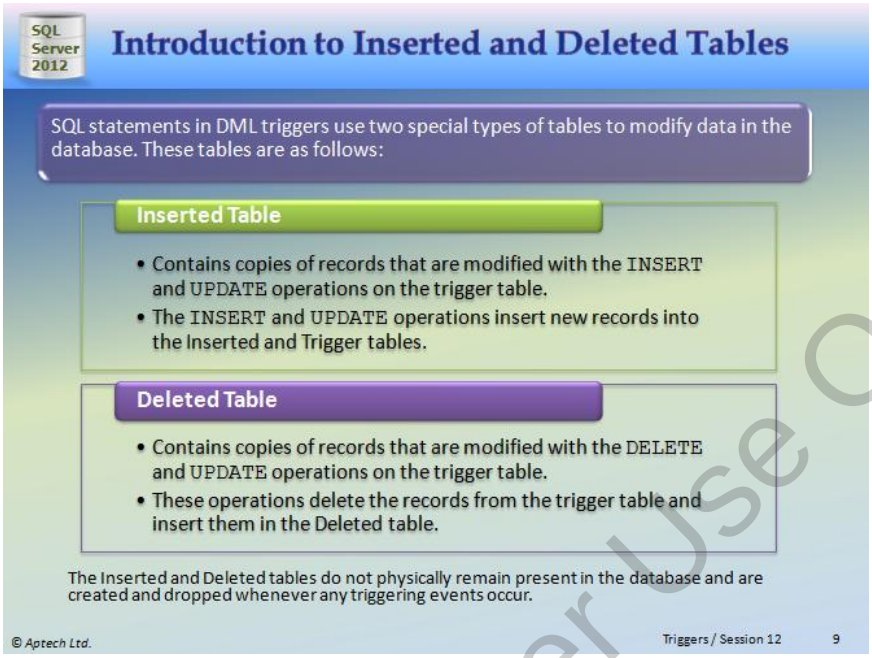
When are DML triggers executed?

Answer:

DML triggers are executed when DML events occur in tables or views.

Introduction to the Inserted and Deleted Tables

Slide 9



Using slide 9, explain the Inserted and Deleted tables.
The SQL statements in DML triggers use two special types of tables to modify data in the database. When the data is inserted, updated, or deleted, SQL Server 2012 creates and manages these tables automatically. The tables temporarily store the original as well as the modified data. These tables are as follows:

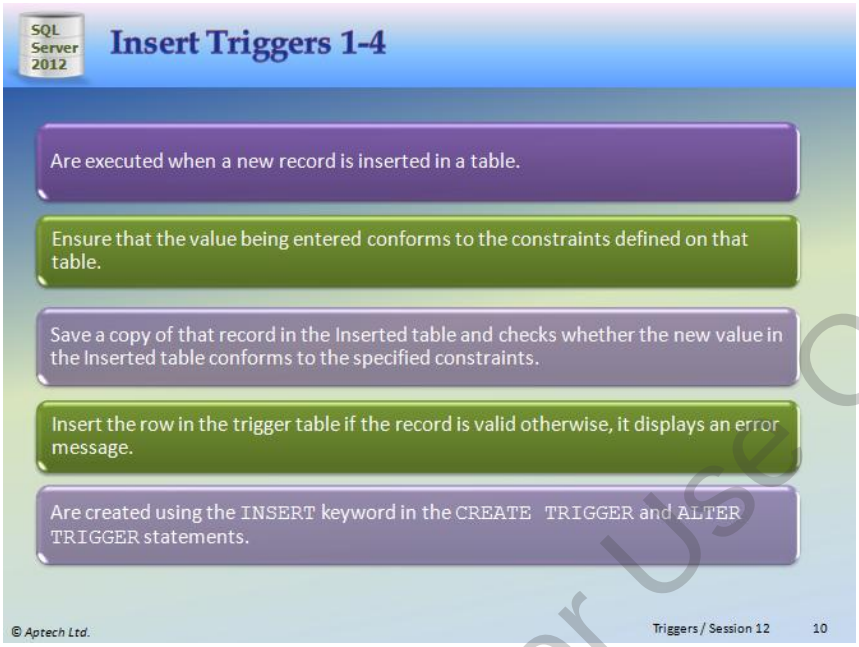
- **Inserted Table**: The Inserted table contains copies of records that are modified with the INSERT and UPDATE operations on the trigger table. Trigger table is the table on which the trigger is defined. The INSERT and UPDATE operations insert new records into the Inserted and Trigger tables.
- **Deleted Table**: The Deleted table contains copies of records that are modified with the DELETE and UPDATE operations on the trigger table. Trigger table is the table on which the trigger is defined. These operations delete the records from the trigger table and insert them in the Deleted table.

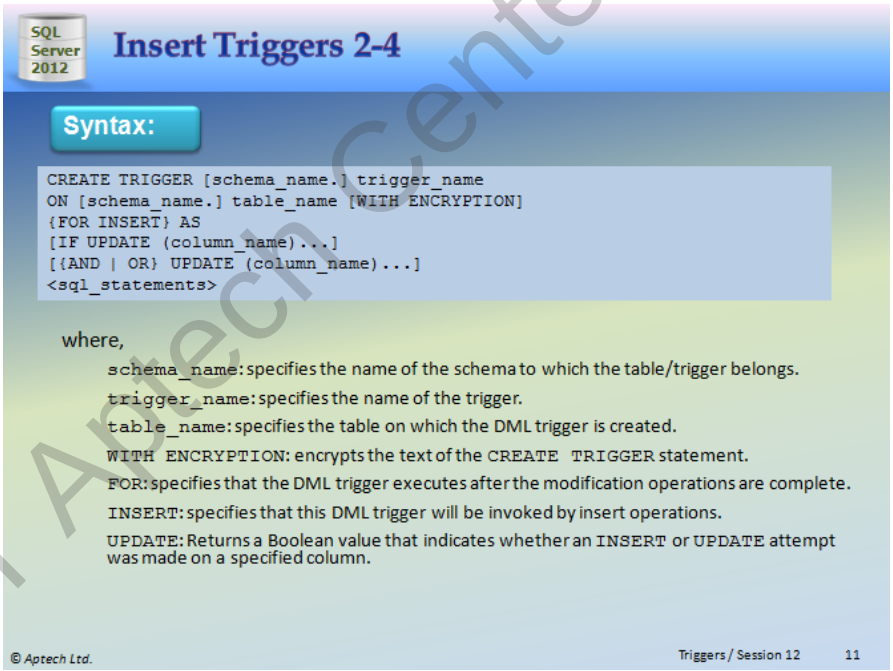
Mention, that the Inserted and Deleted tables do not physically remain present in the database. They are created and dropped whenever any triggering events occur.

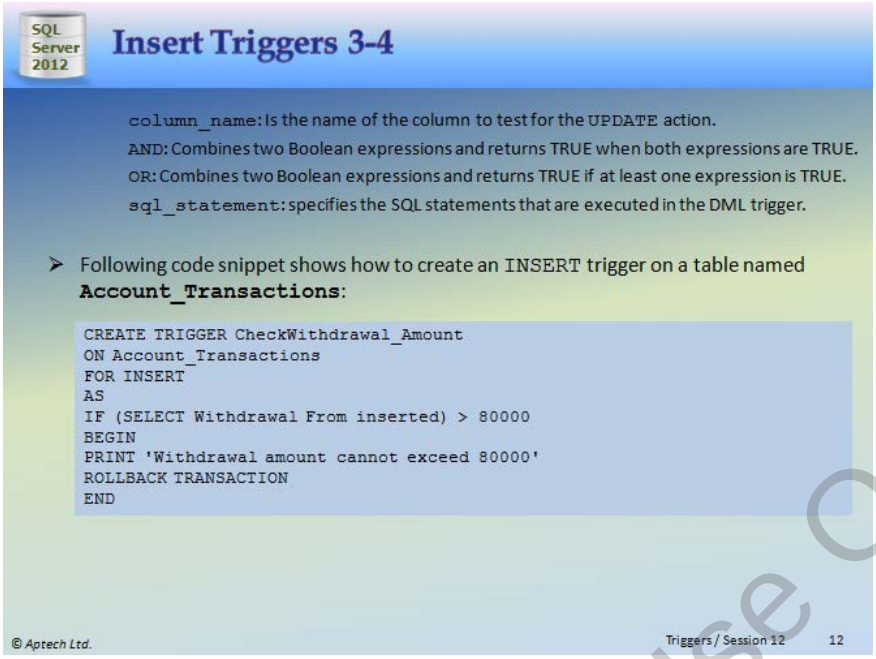
Mention, referencing the deleted table when testing an INSERT or the inserted table when testing a DELETE does not cause any errors or do not trigger in this case.

Insert Triggers

Slides 10 to 13







Using slides 10 to 13, explain the Insert trigger statement.
An INSERT trigger is executed when a new record is inserted in a table. The INSERT trigger ensures that the value being entered conforms to the constraints defined on that table.

When a user inserts a record in the table, the INSERT trigger saves a copy of that record in the Inserted table. It then checks whether the new value in the Inserted table conforms to the specified constraints.

If the record is valid, the INSERT trigger inserts the row in the trigger table otherwise, it displays an error message.

An INSERT trigger is created using the INSERT keyword in the CREATE TRIGGER and ALTER TRIGGER statements.

Explain the syntax for creating an INSERT trigger.

Explain the Code Snippet which creates an INSERT trigger on a table named Account_Transactions. When a new record is inserted, and if the withdrawal amount exceeds 80000, the insert trigger displays an error message and rolls back the transaction using the ROLLBACK TRANSACTION statement.

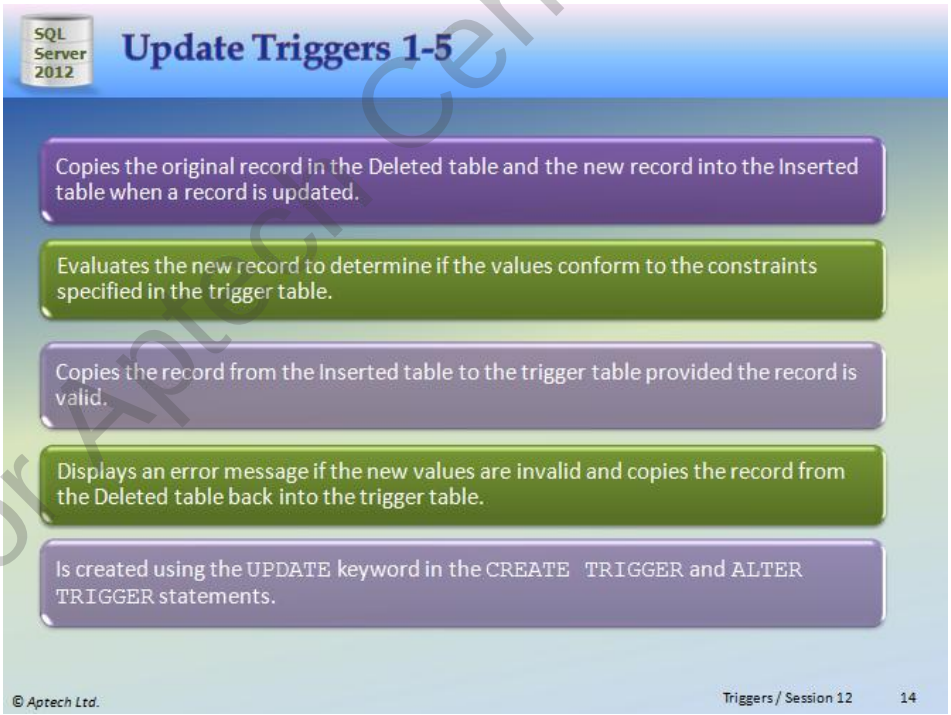
A transaction is a set of one or more statements that are treated as a single unit of work. A transaction can succeed or fail as a whole; therefore, all the statements within it succeed or fail together. The ROLLBACK TRANSACTION statement cancels or rolls back a transaction.

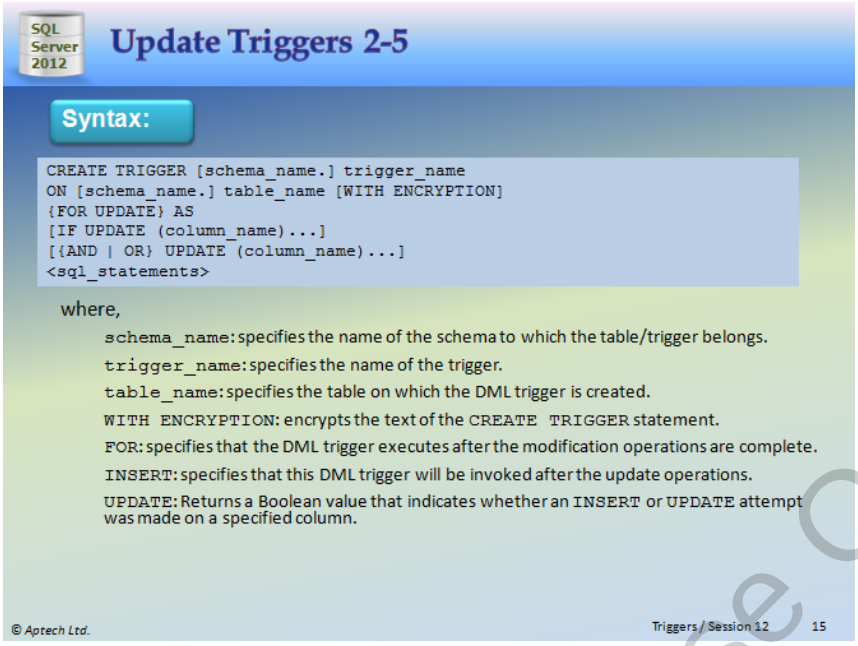
Explain the Code Snippet which inserts a record where the Withdrawal amount exceeds 80000. This causes the INSERT trigger to display an error message and roll back the transaction.

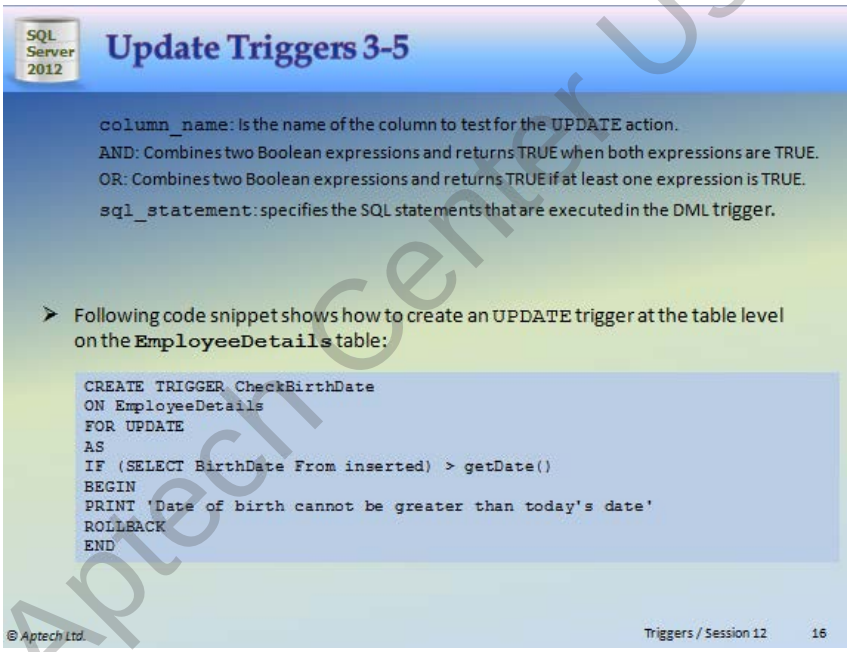
The following error message is displayed as specified by the PRINT statement: Withdrawal amount cannot exceed 80000.

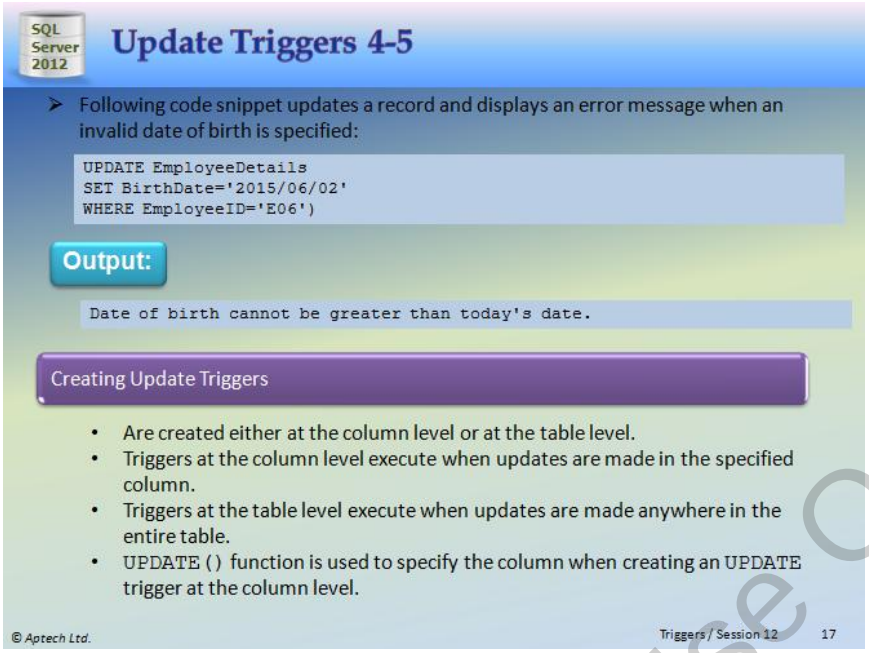
Update Triggers

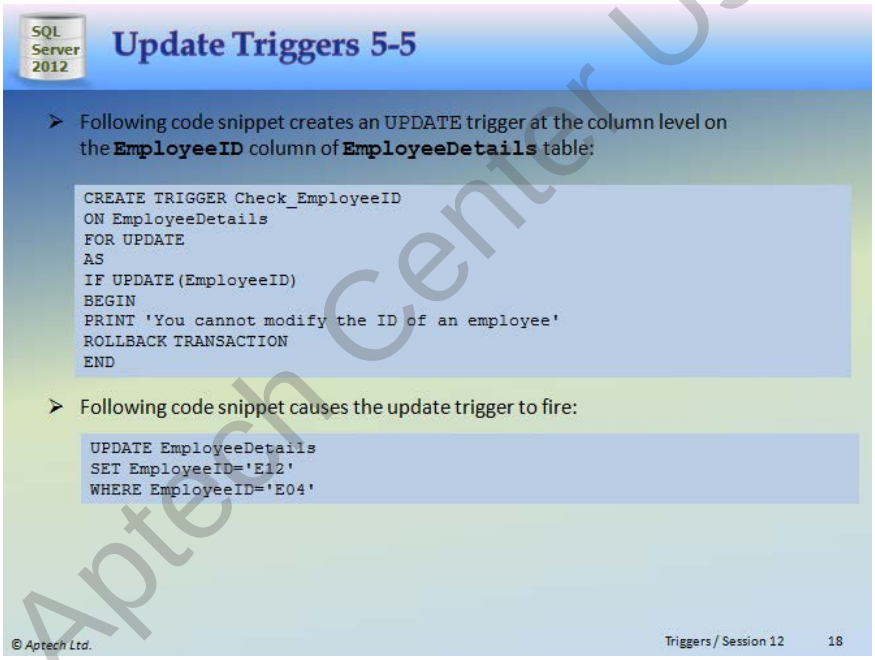
Slides 14 to 18











Using slides 14 to 18, explain the Update trigger statement.

The UPDATE trigger copies the original record in the Deleted table and the new record into the Inserted table when a record is updated. It then evaluates the new record to determine if the values conform to the constraints specified in the trigger table.

If the new values are valid, the record from the Inserted table is copied to the trigger table. However, if the new values are invalid, an error message is displayed. Also, the original record is copied from the Deleted table back into the trigger table.

An UPDATE trigger is created using the UPDATE keyword in the CREATE TRIGGER and ALTER TRIGGER statements. Explain the syntax for creating an UPDATE trigger at the table-level using slide 16.

Explain the Code Snippet which creates an UPDATE trigger at the table level on the EmployeeDetails table. When a record is modified, the UPDATE trigger is activated. It checks whether the date of birth is greater than today's date. It displays an error message for

invalid values and rolls back the modification operation using the ROLLBACK TRANSACTION statement.
Explain the Code Snippet which updates a record where an invalid date of birth is specified. This causes the update trigger to display the error message and roll back the transaction. The following error message is displayed as specified by the PRINT statement: Date of birth cannot be greater than today's date.
Explain creating Update Triggers using slides 17 and 18.
The UPDATE triggers are created either at the column level or at the table level. The triggers at the column level execute when updates are made in the specified column. The triggers at the table level execute when updates are made anywhere in the entire table.
For creating an UPDATE trigger at the column level, the UPDATE() function is used to specify the column name.
Explain the Code Snippet which creates an UPDATE trigger at the column level on the EmployeeID column of EmployeeDetails table. When the EmployeeID is modified, the UPDATE trigger is activated and an error message is displayed. The modification operation is rolled back using the ROLLBACK TRANSACTION statement.
Explain the Code Snippet which updates a record where the value in the EmployeeID column is being modified. This causes the update trigger to fire. The update trigger displays an error message and rolls back the transaction.

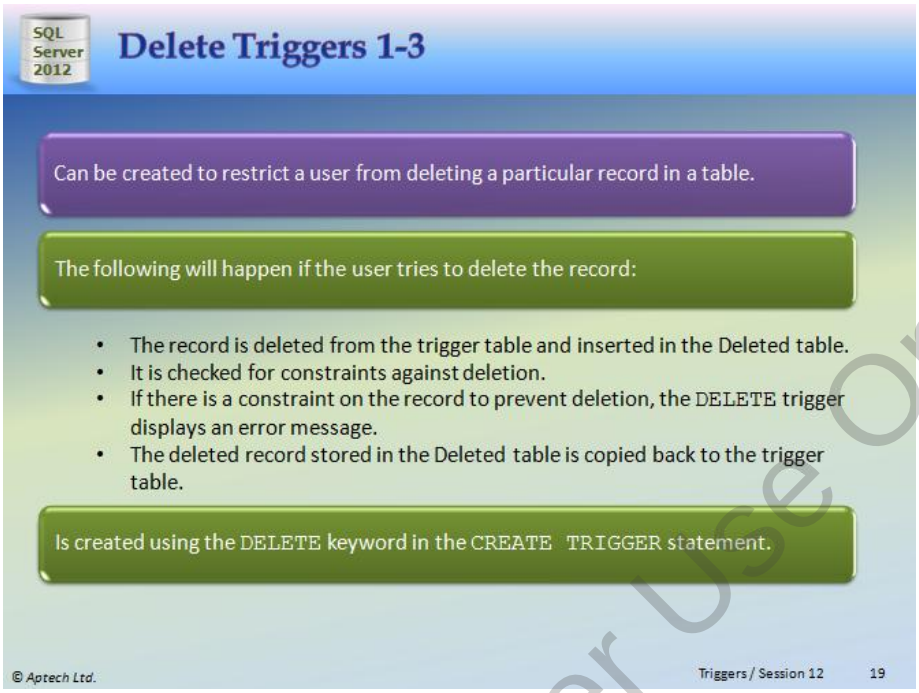
In-Class Question:

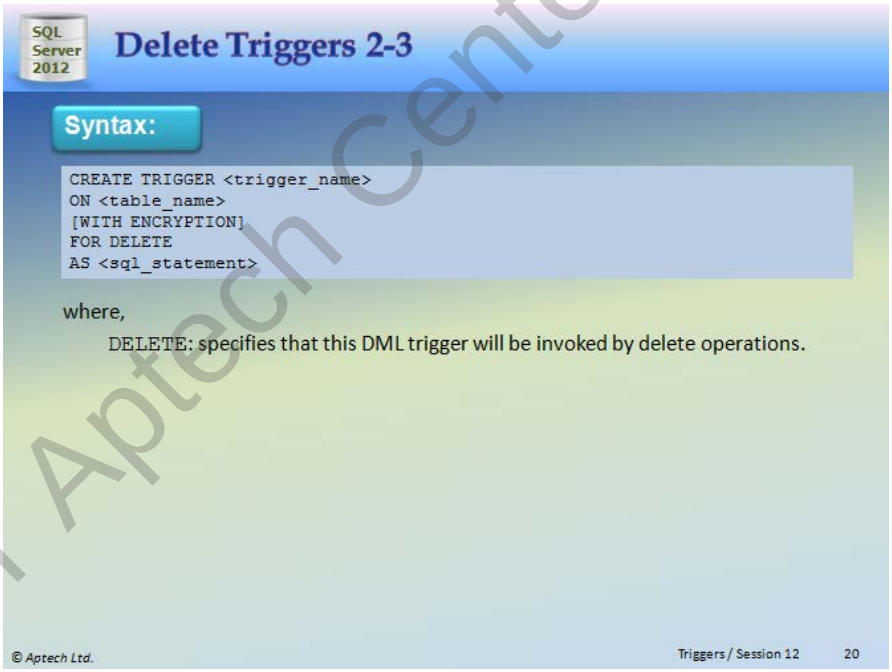
 Which function is used to specify the column for creating an Update trigger at column level?

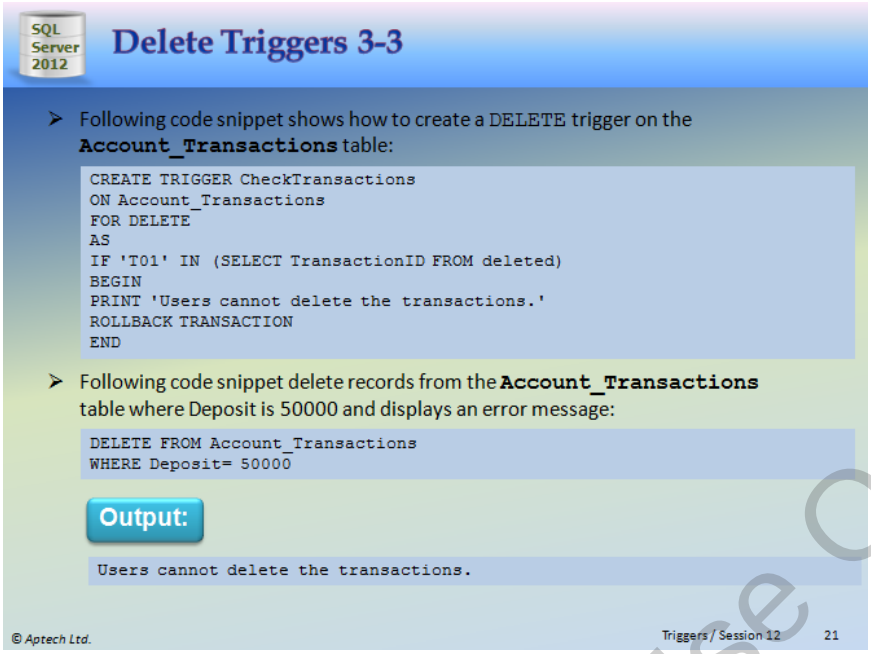
Answer:
For creating an UPDATE trigger at the column level, the UPDATE() function is used to specify the column.

Delete Triggers

Slides 19 to 21







Using slides 19 to 21, explain the Delete trigger statement.

The DELETE trigger can be created to restrict a user from deleting a particular record in a table. The following will happen if the user tries to delete the record:

- The record is deleted from the trigger table and inserted in the Deleted table.
- It is checked for constraints against deletion.
- If there is a constraint on the record to prevent deletion, the DELETE trigger displays an error message.
- The deleted record stored in the Deleted table is copied back to the trigger table.

A DELETE trigger is created using the DELETE keyword in the CREATE TRIGGER statement.

Explain the syntax for creating a DELETE trigger.

Explain the Code Snippet which creates a DELETE trigger on the Account_Transactions table.

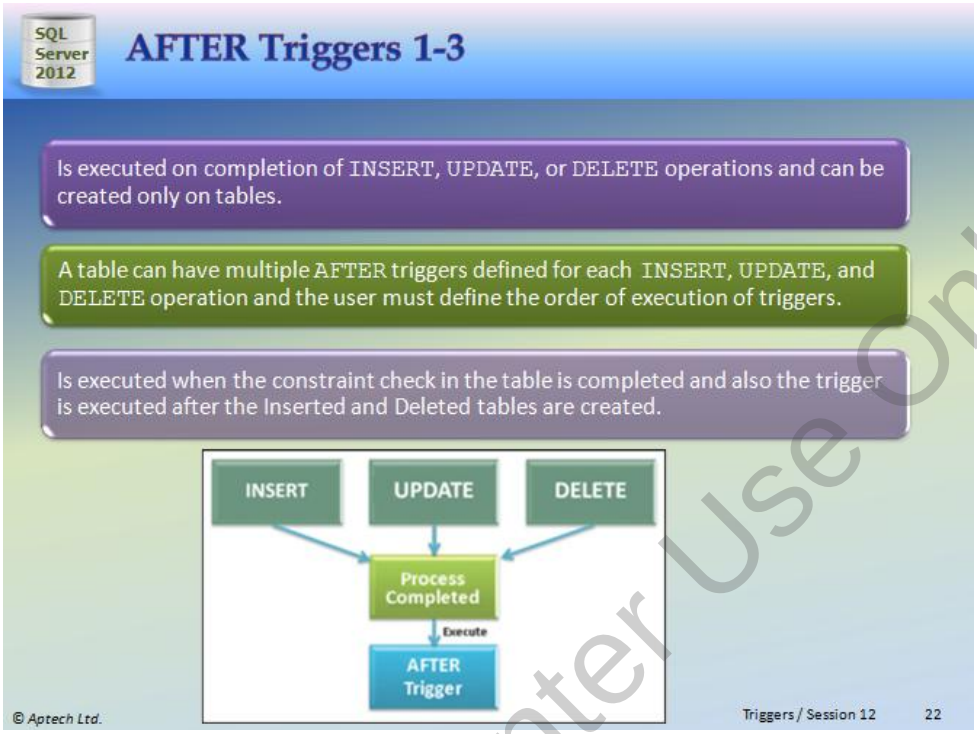
If a record of a TransactionID is deleted, the DELETE trigger is activated and an error message is displayed. The delete operation is rolled back using the ROLLBACK TRANSACTION statement.

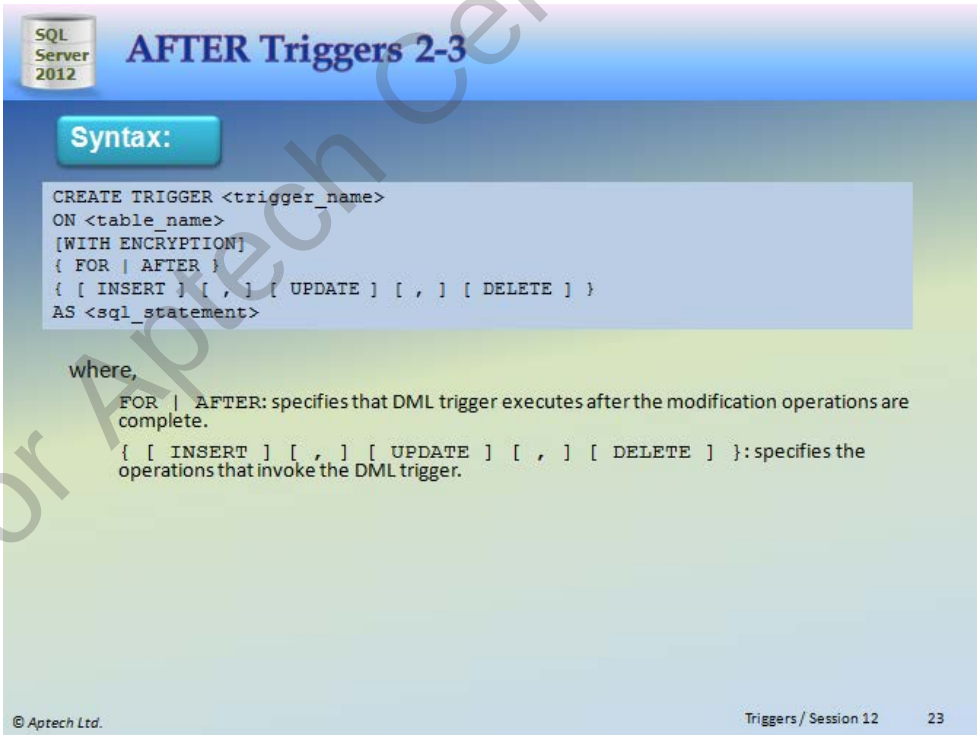
Explain the Code Snippet which attempts to delete records from the Account_Transactions table where Deposit is 50000.

The error message is displayed as specified by the PRINT statement. Users cannot delete the transactions.

After Triggers

Slides 22 to 24





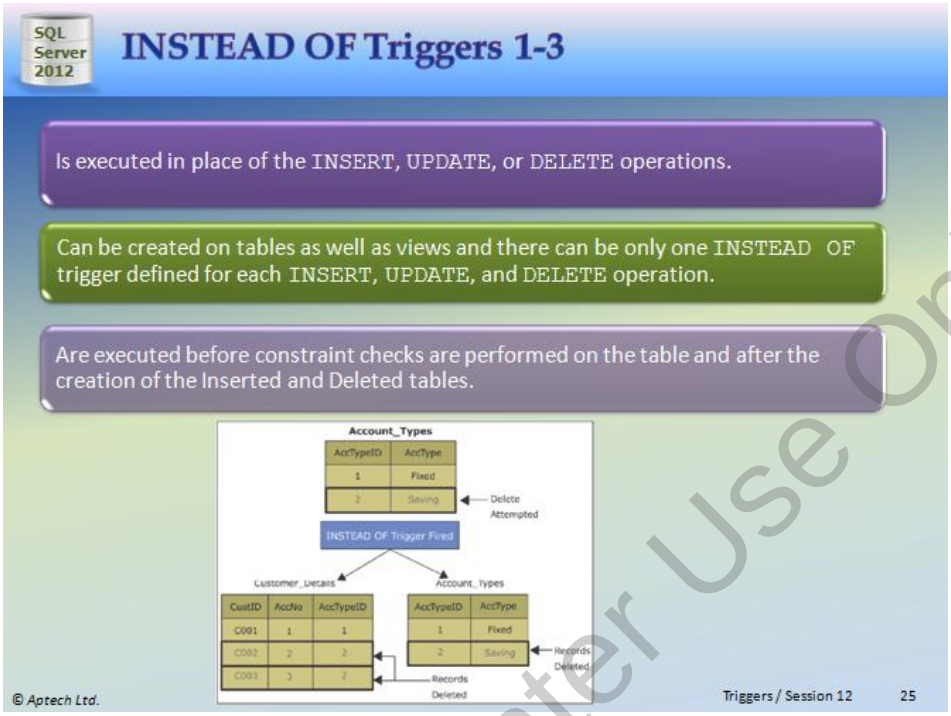
Using slides 22 to 24, explain the After trigger statement.
An AFTER trigger is executed on completion of INSERT, UPDATE, or DELETE operations. AFTER triggers can be created only on tables. A table can have multiple AFTER triggers defined for each INSERT, UPDATE, and DELETE operation. If multiple AFTER triggers are created on the same table, the user must define the order in which the triggers must be executed. An AFTER trigger is executed when the constraint check in the table is completed. Also, the trigger is executed after the Inserted and Deleted tables are created.
Figure on slide 22 displays the types of AFTER triggers.
Explain the syntax for creating an AFTER trigger.
Explain the Code Snippet which creates an AFTER DELETE trigger on the EmployeeDetails table. If any employee record is deleted from the table, the AFTER DELETE trigger activates. The trigger displays the number of employee records deleted from the table.
Explain the Code Snippet which deletes a record from the EmployeeDetails table.
The following error message is displayed:
No. of employees deleted = 0
Mention, to specify the order for an AFTER trigger, use the sp_settriggerorder stored procedure. The following option can be used:

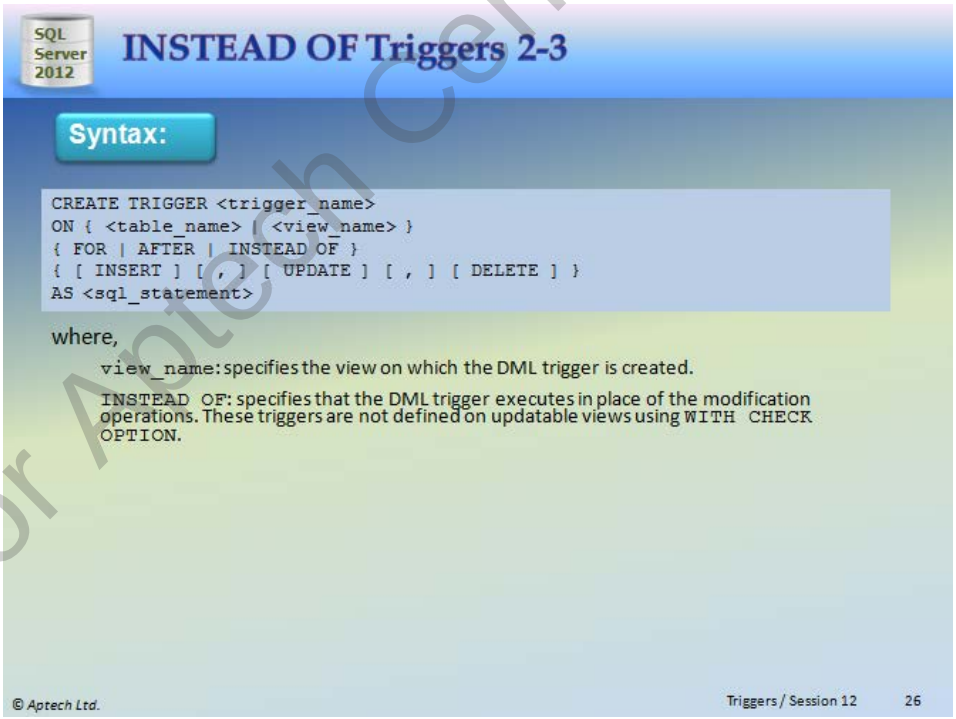
- First
- Last
- None

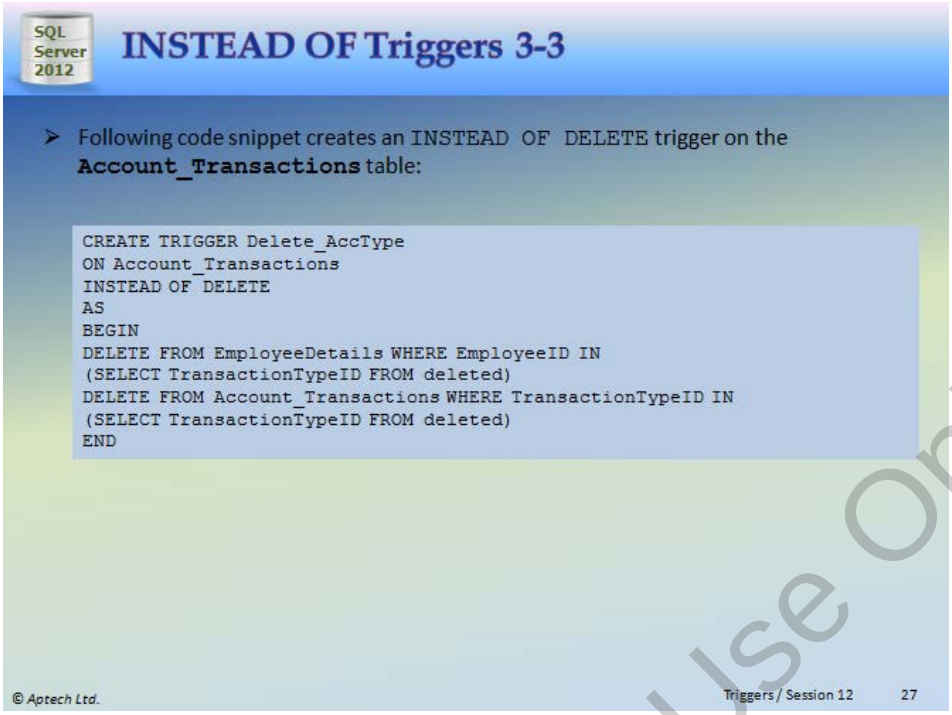
The first and last triggers must be two different DML triggers.

INSTEAD OF Triggers

Slides 25 to 27







Using slides 25 to 27, explain the Instead of triggers.

An INSTEAD OF trigger is executed in place of the INSERT, UPDATE, or DELETE operations. INSTEAD OF triggers can be created on tables as well as views. A table or a view can have only one INSTEAD OF trigger defined for each INSERT, UPDATE, and DELETE operation.

The INSTEAD OF triggers are executed before constraint checks are performed on the table. These triggers are executed after the creation of the Inserted and Deleted tables. The INSTEAD OF triggers increase the variety of types of updates that the user can perform against the view.

Figure on slide 25 displays an example of INSTEAD OF triggers.

In the example shown in the figure, an INSTEAD OF DELETE trigger on the Account_Types table is created. If any record in the Account_Types table is deleted, the corresponding records in the Customer_Details table will also be removed. Thus, instead of working only on one table, here, the trigger ensures that the delete operation is performed on both the tables.

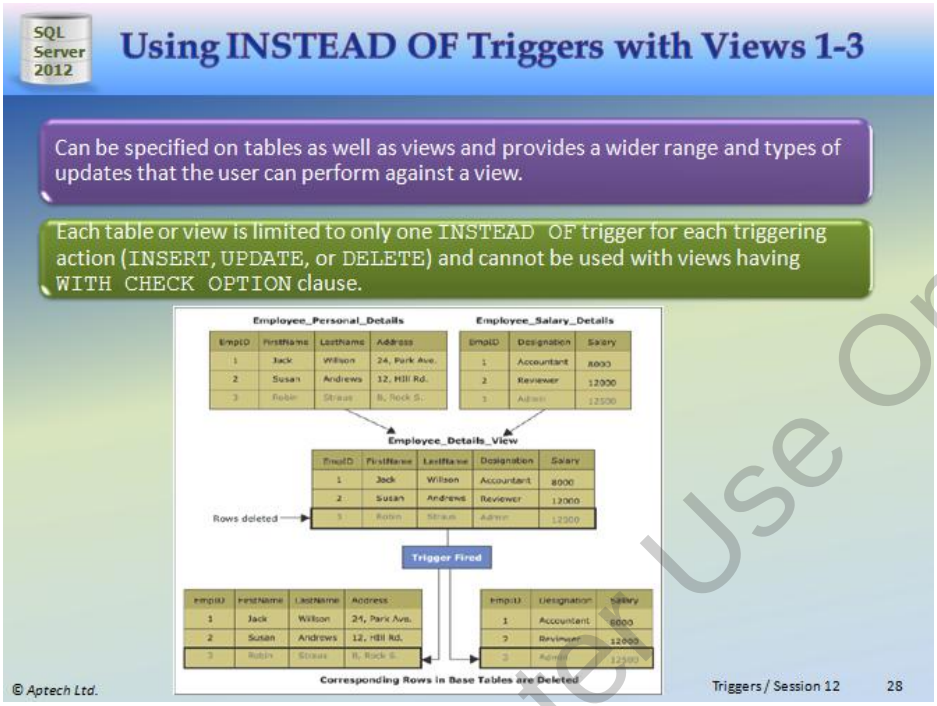
Mention, that users cannot create INSTEAD OF triggers for delete or update operations on tables that have the ON DELETE cascade and ON UPDATE cascade options selected.

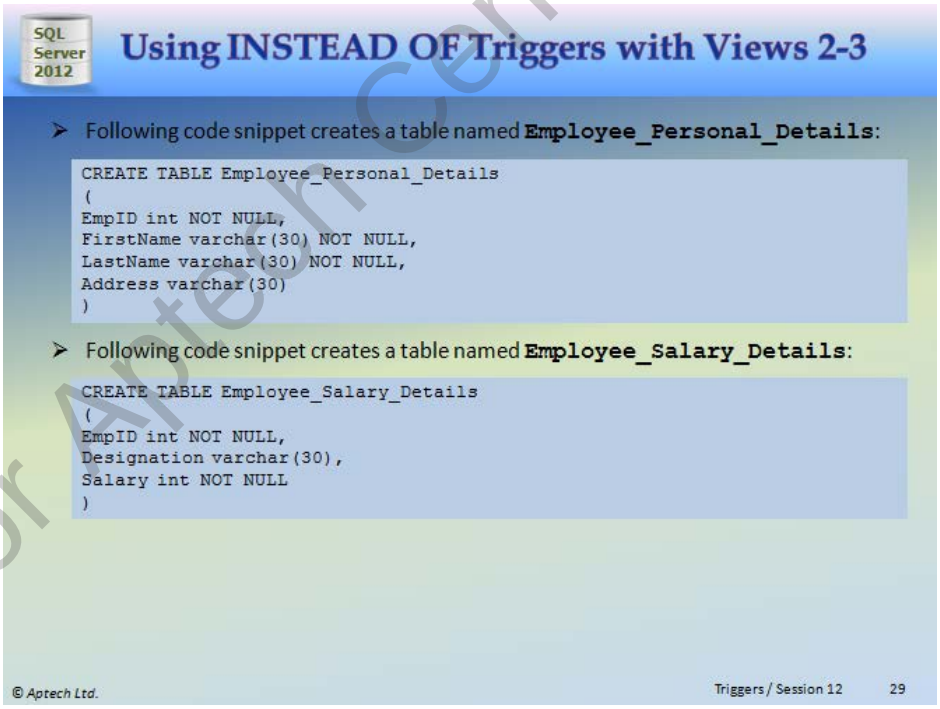
Explain the syntax for creating an INSTEAD OF trigger.

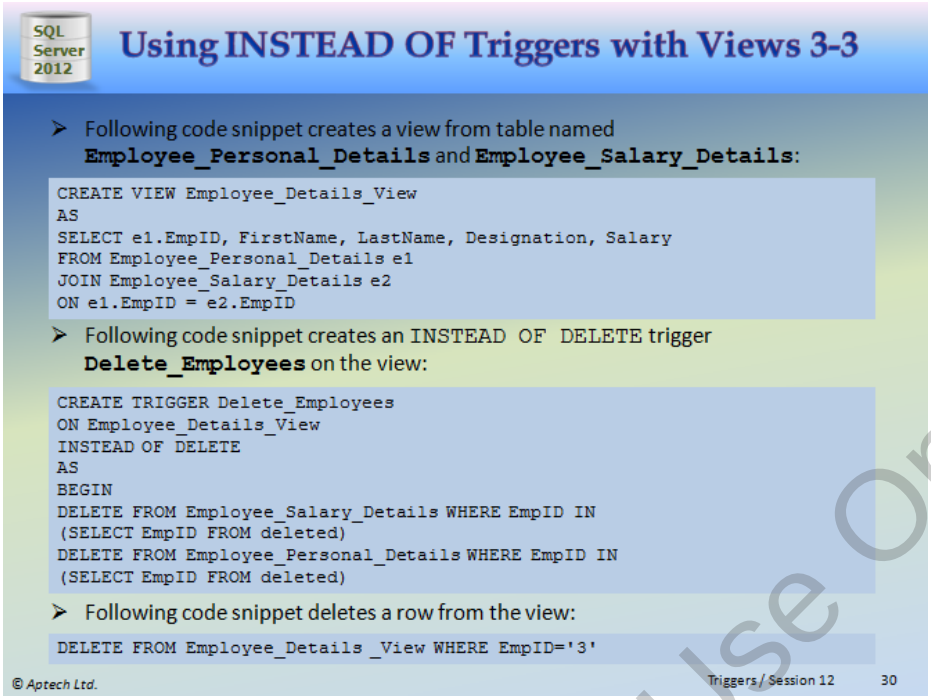
Explain the Code Snippet which creates an INSTEAD OF DELETE trigger on the Account_Transactions table. If any record in the Account_Transactions table is deleted, the corresponding records in the EmployeeDetails table will be removed.

Using the INSTEAD OF Triggers with Views

Slides 28 to 30







Using slides 28 to 30, explain how to use the INSTEAD OF triggers with view.
INSTEAD OF triggers can be specified on tables as well as views. This trigger executes instead of the original triggering action. INSTEAD OF triggers provide a wider range and types of updates that the user can perform against a view. Each table or view is limited to only one INSTEAD OF trigger for each triggering action (INSERT, UPDATE, or DELETE).
Users cannot create an INSTEAD OF trigger on views that have the WITH CHECK OPTION clause defined.
Figure on slide 28 displays an example of using INSTEAD OF triggers with views.
Explain the Code Snippet which creates a table named Employee_Personal_Details and Employee_Salary_Details using slide 29.
Explain the Code Snippet which creates a view Employee_Details_View using columns from the Employee_Personal_Details and Employee_Salary_Details tables by joining the two tables on the EmpID column.
Explain the Code Snippet which creates an INSTEAD OF DELETE trigger Delete_Employees on the view Employee_Details_View. When a row is deleted from the view, the trigger is activated. It deletes the corresponding records from the base tables of the view, namely, Employee_Personal_Details and Employee_Salary_Details.
Explain the Code Snippet which deletes a row from the view Employee_Details_View where EmpID='2'.

In-Class Question:

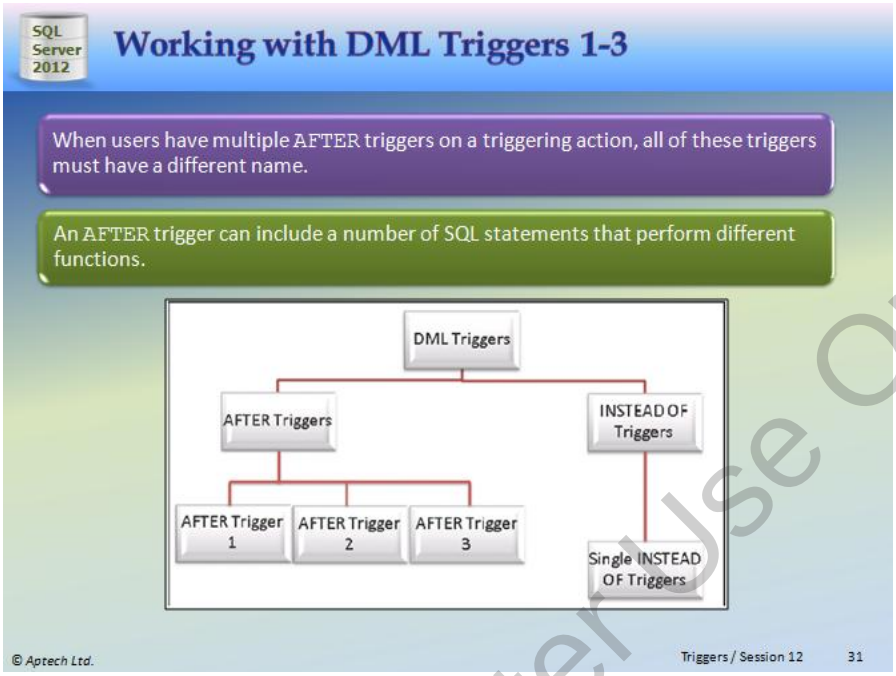
What is condition to have a INSTEAD OF trigger on a view?

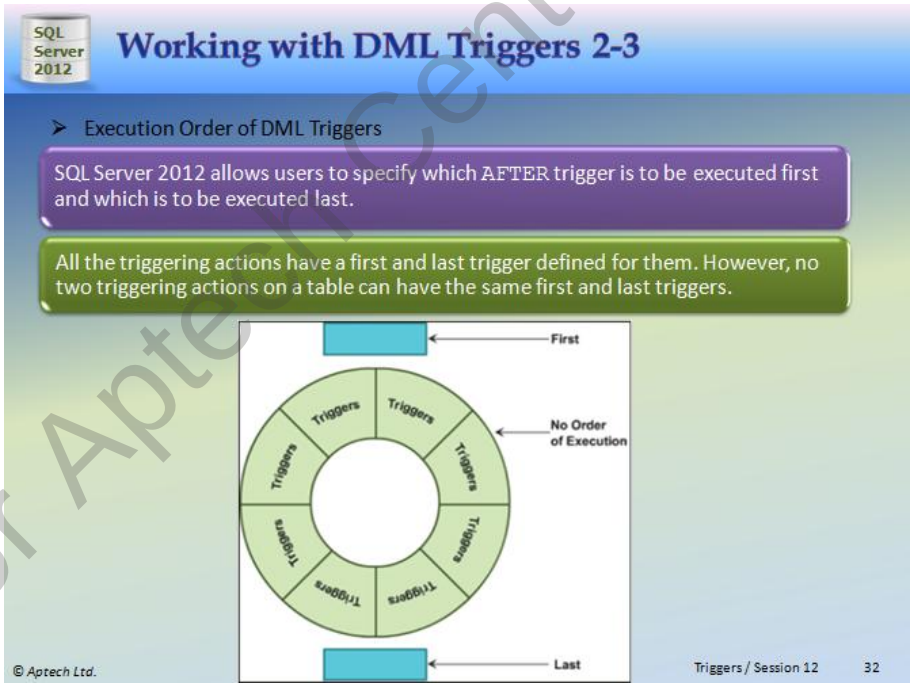
Answer:

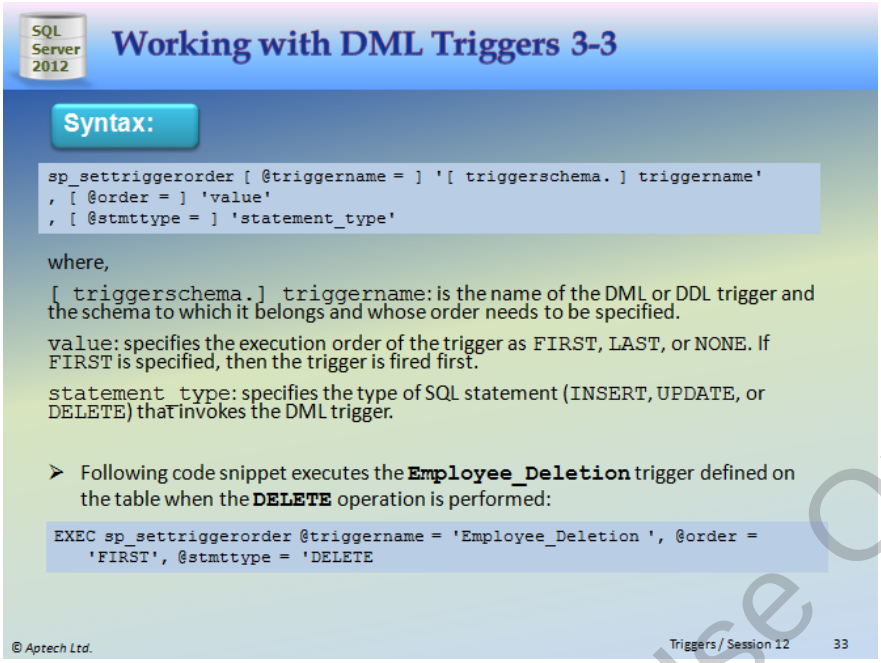
The WITH CHECK OPTION clause must not be defined for a table to have a INSTEAD OF trigger.

Working with DML Triggers

Slides 31 to 33



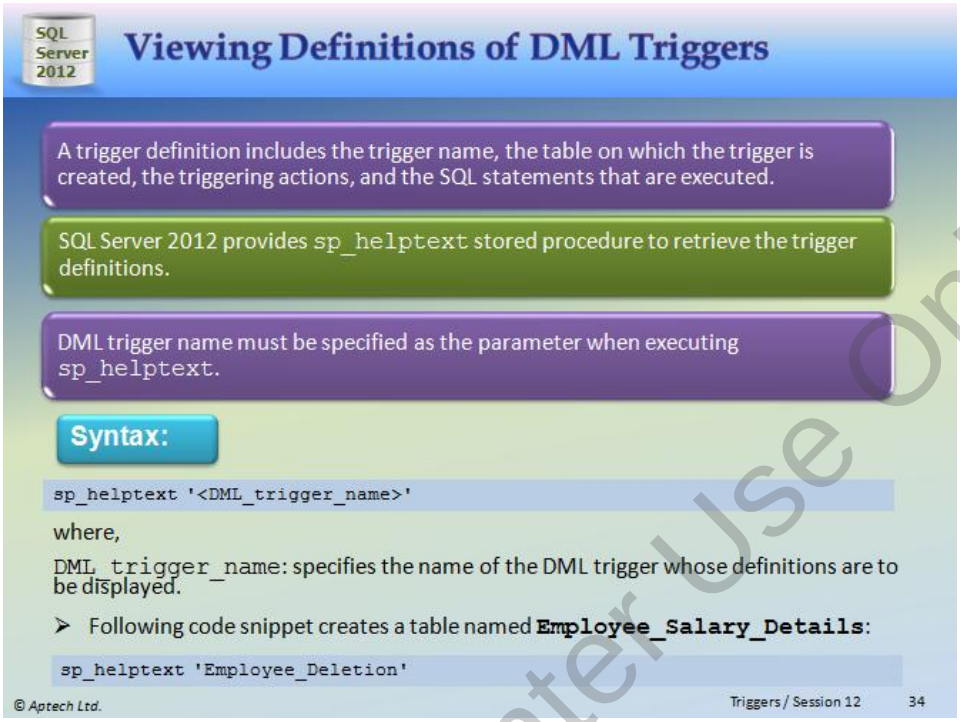




Using slides 31 to 33, explain how to work with DML triggers.
SQL Server 2012 allows users to create multiple AFTER triggers for each triggering action (such as UPDATE, INSERT, and DELETE) on a table. However, the user can create only one INSTEAD OF trigger for each triggering action on a table.
When users have multiple AFTER triggers on a triggering action, all of these triggers must have a different name. An AFTER trigger can include a number of SQL statements that perform different functions.
Mention, a single AFTER trigger can be invoked by more than one triggering action.
Explain the execution order of DML triggers using slide 32.
Explain the syntax for specifying execution order of multiple AFTER DML triggers on slide 33.
Explain the Code Snippet which first executes the Employee_Deletion trigger defined on the table when the DELETE operation is performed on the Withdrawal column of the table.

View Definition of DML Triggers

Slide 34



Using slide 34, explain how to view definition of DML triggers.
A trigger definition includes the trigger name, the table on which the trigger is created, the triggering actions, and the SQL statements that are executed. SQL Server 2012 provides sp_helptext stored procedure to retrieve the trigger definitions.

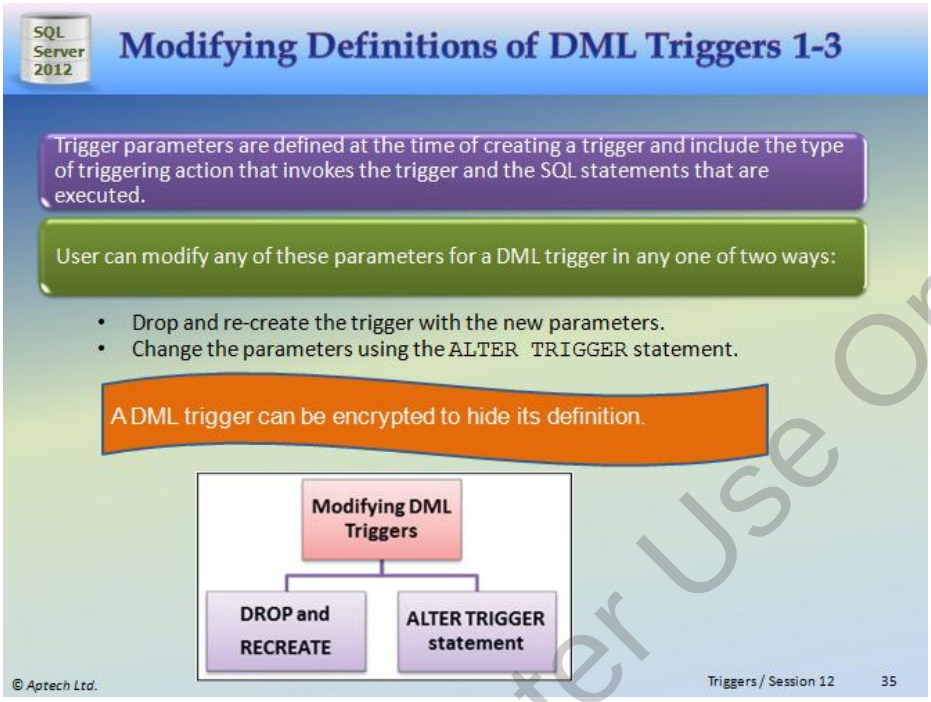
The DML trigger name must be specified as the parameter when executing sp_helptext.

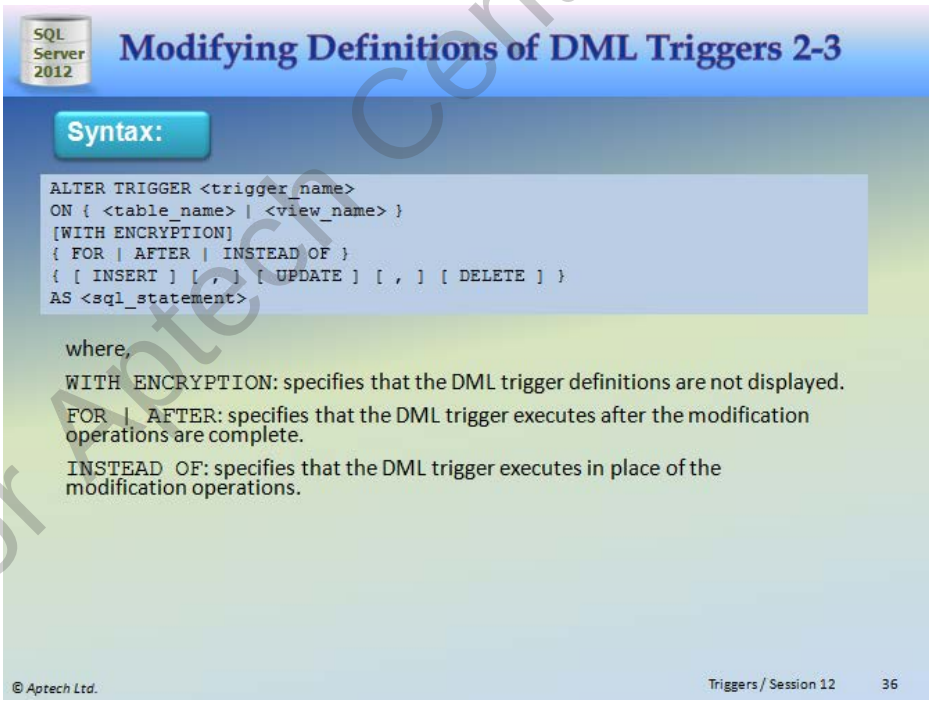
Mention that trigger definition cannot be viewed if the definition is encrypted.

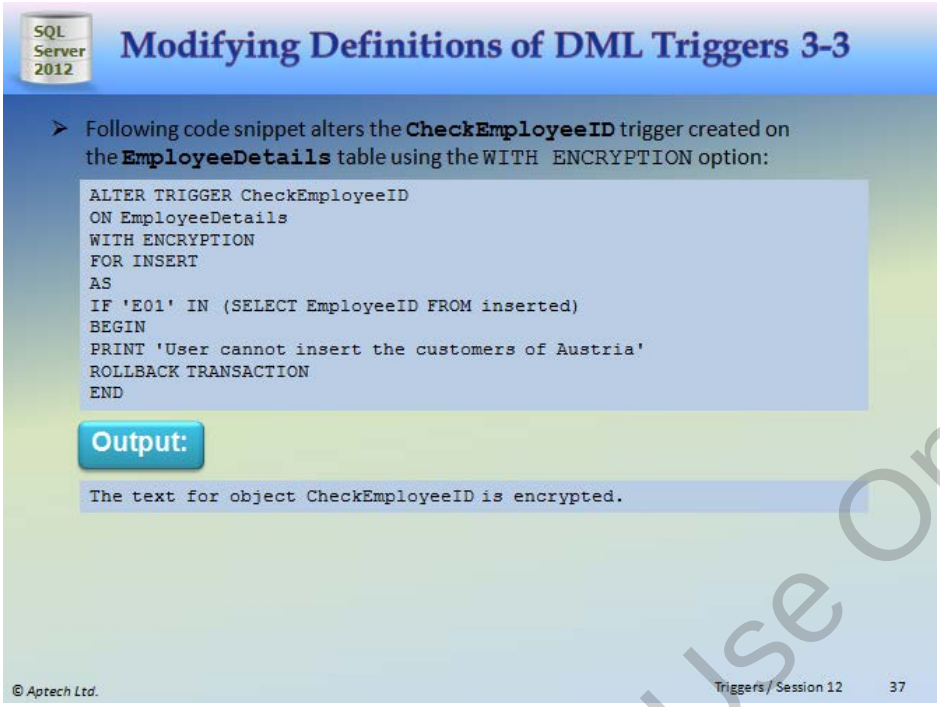
Explain the syntax for viewing a DML trigger.
Explain the Code Snippet which displays the definitions of the trigger, Employee_Deletion, created on the table.

Modifying Definition of DML Triggers

Slides 35 to 37







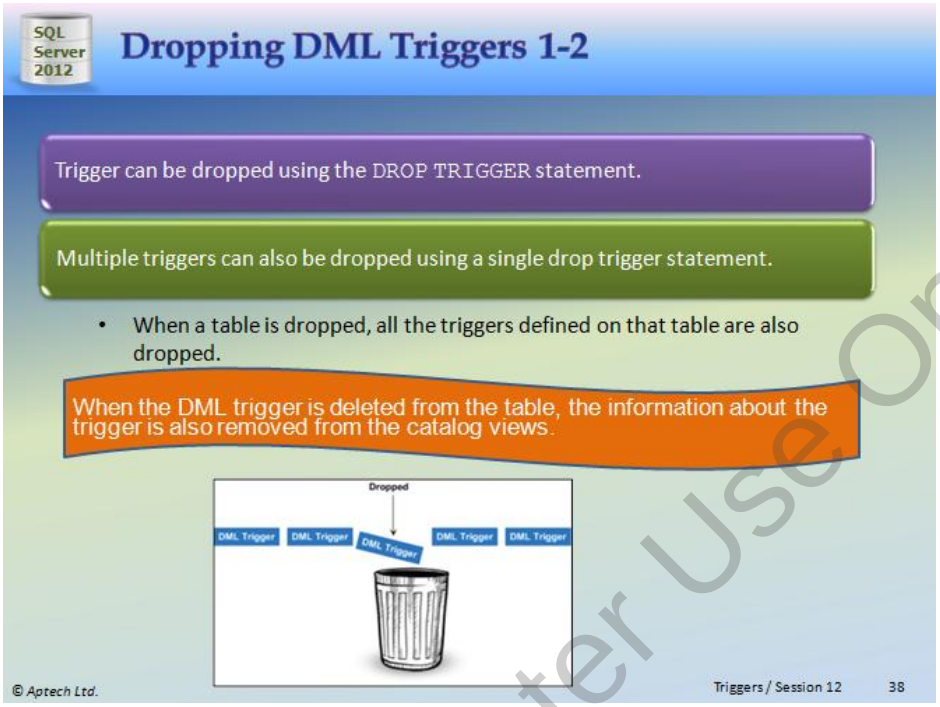
Using slides 35 to 37, explain how to modify definition of DML triggers.
Trigger parameters are defined at the time of creating a trigger. These parameters include the type of triggering action that invokes the trigger and the SQL statements that are executed.
If the user wants to modify any of these parameters for a DML trigger, a user can do so in any one of the two ways:

- Drop and re-create the trigger with the new parameters.
- Change the parameters using the ALTER TRIGGER statement.

If the object referencing a DML trigger is renamed, the trigger must be modified to reflect the change in object name.
Mention, a DML trigger can be encrypted to hide its definition.
Explain the syntax for modifying a DML trigger.
Explain the Code Snippet which alters the CheckEmployeeID trigger created on the EmployeeDetails table using the WITH ENCRYPTION option.
Now, if the user tries to view the definition of the CheckEmployeeID trigger using the sp_helptext stored procedure, the following error message is displayed:
The text for object CheckEmployeeID is encrypted.

Dropping DML Triggers

Slides 38 and 39



Using slides 38 and 39, explain how to drop DML triggers.

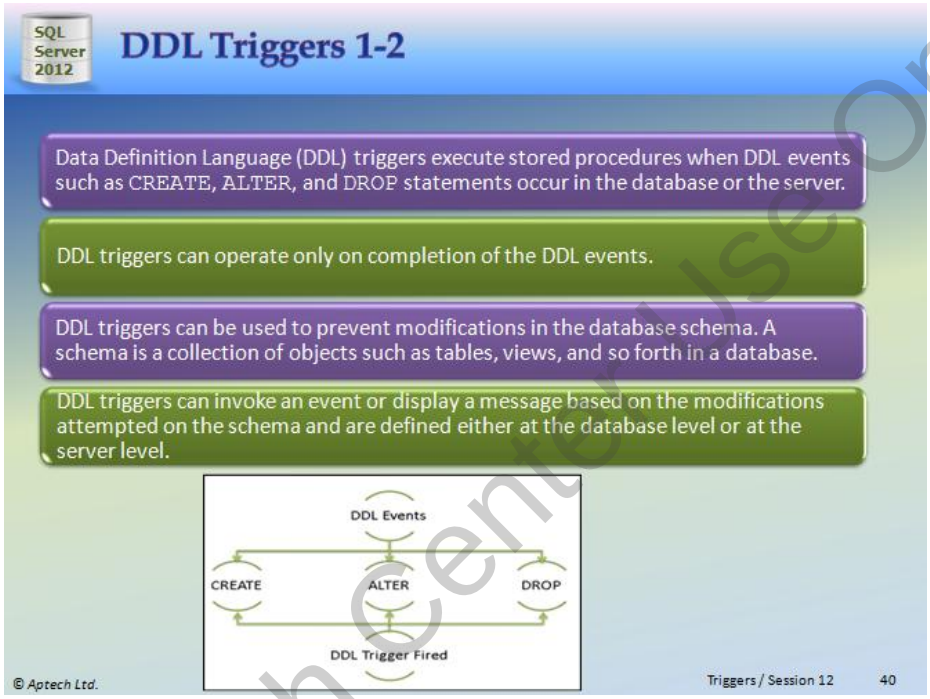
SQL Server 2012 provides the option of dropping a DML trigger created on a table if the trigger is no longer required. The trigger can be dropped using the DROP TRIGGER statement. Multiple triggers can also be dropped using a single drop trigger statement. When a table is dropped, all the triggers defined on that table are also dropped.

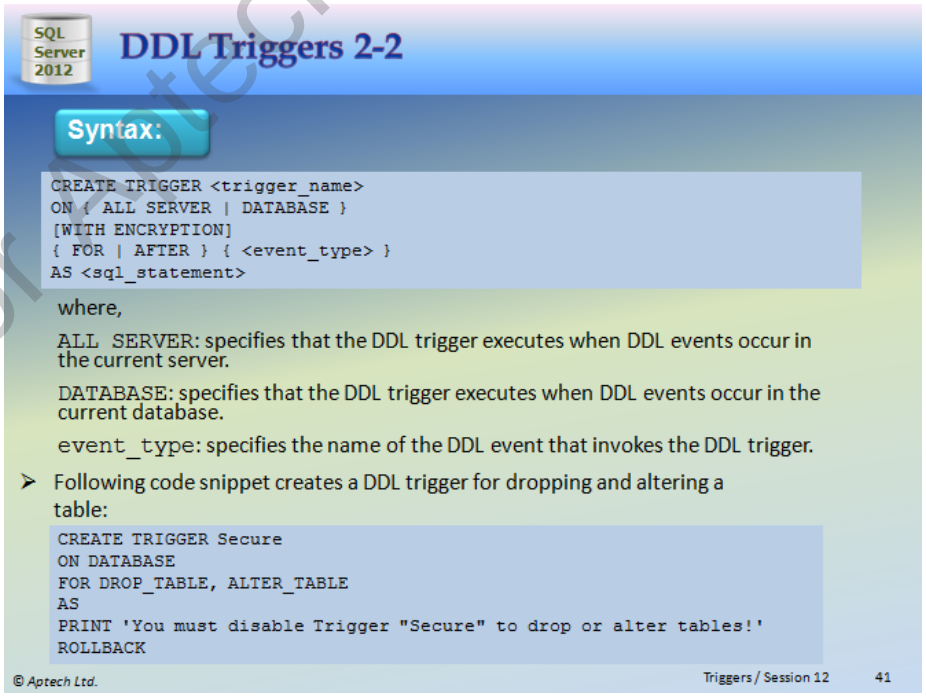
Figure on slide 38 depicts the concept of dropped DML triggers.

When the DML trigger is deleted from the table, the information about the trigger is also removed from the catalog views.
Explain the syntax for dropping DML triggers.
Explain the Code Snippet which drops the CheckEmployeeID trigger created on the EmployeeDetails table.

DDL Triggers

Slides 40 and 41



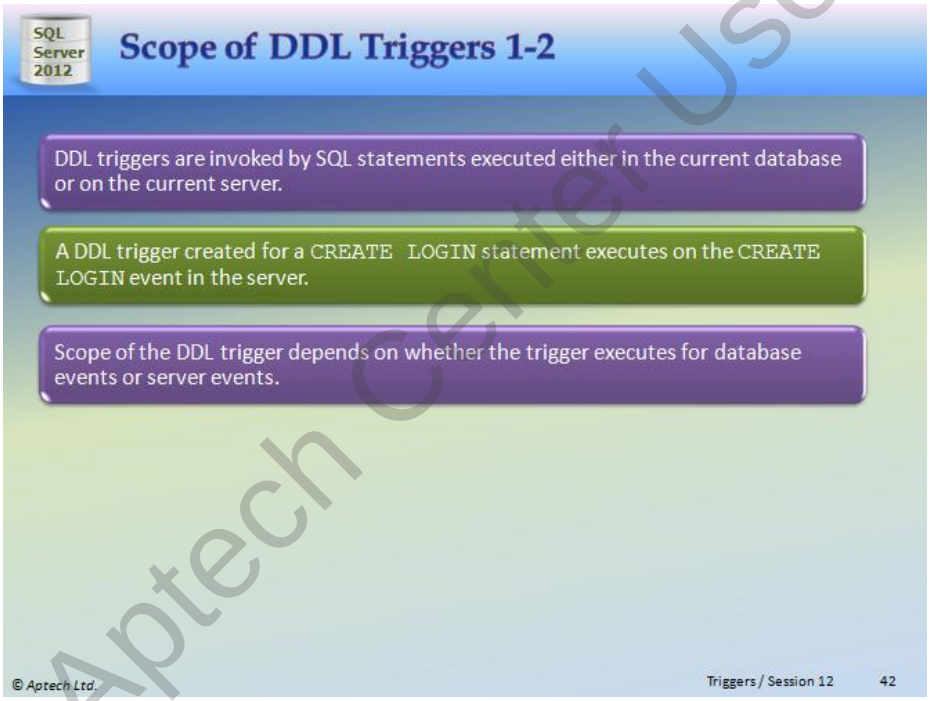


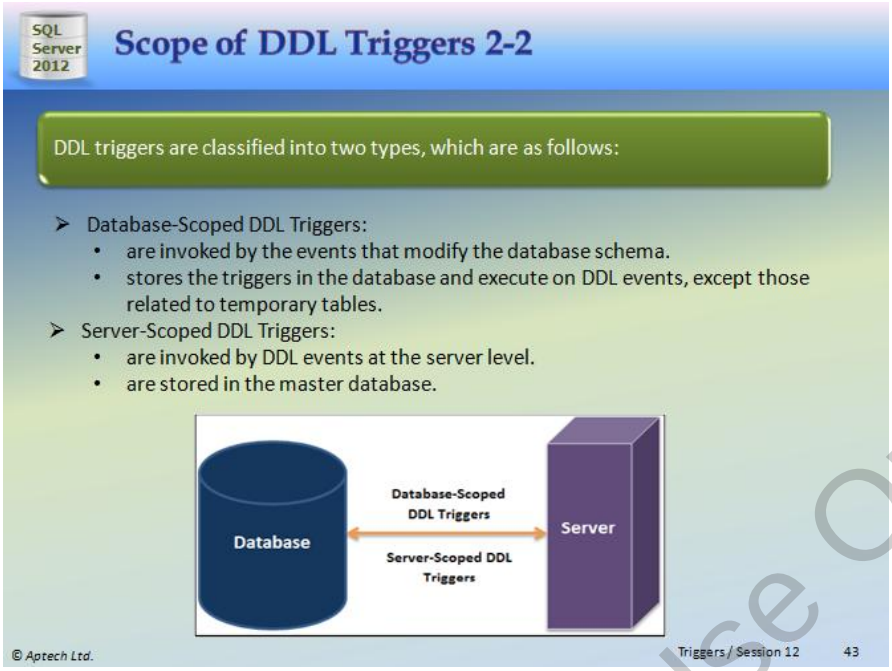
Using slides 40 and 41, explain the DDL triggers.

A Data Definition Language (DDL) triggers execute stored procedures when DDL events such as CREATE, ALTER, and DROP statements occur in the database or the server. DDL triggers can operate only on completion of the DDL events.
DDL triggers can be used to prevent modifications in the database schema. A schema is a collection of objects such as tables, views, and so forth in a database.
DDL triggers can invoke an event or display a message based on the modifications attempted on the schema. DDL triggers are defined either at the database level or at the server level. Figure on slide 40 displays the types of DDL triggers.
Explain the syntax for creating DDL triggers.
Explain the Code Snippet which creates a DDL trigger for dropping and altering a table.
In this code, the DDL trigger is created for DROP TABLE and ALTER TABLE statements.

Scope of DDL Triggers

Slides 42 and 43





Using slides 42 and 43, explain the scope of DDL triggers.

DDL triggers are invoked by SQL statements executed either in the current database or on the current server. For example, a DDL trigger created for a CREATE TABLE statement executes on the CREATE TABLE event in the database. A DDL trigger created for a CREATE LOGIN statement executes on the CREATE LOGIN event in the server.

The scope of the DDL trigger depends on whether the trigger executes for database events or server events. Accordingly, the DDL triggers are classified into two types, which are as follows:

- Database-Scoped DDL: Triggers Database-scoped DDL triggers that are invoked by the events that modify the database schema. These triggers are stored in the database and execute on DDL events, except those related to temporary tables.
- Server-Scoped DDL: Triggers Server-scoped DDL triggers are invoked by DDL events at the server level. These triggers are stored in the master database.

Figure on slide 43 displays the scope of DDL triggers.

In-Class Question:

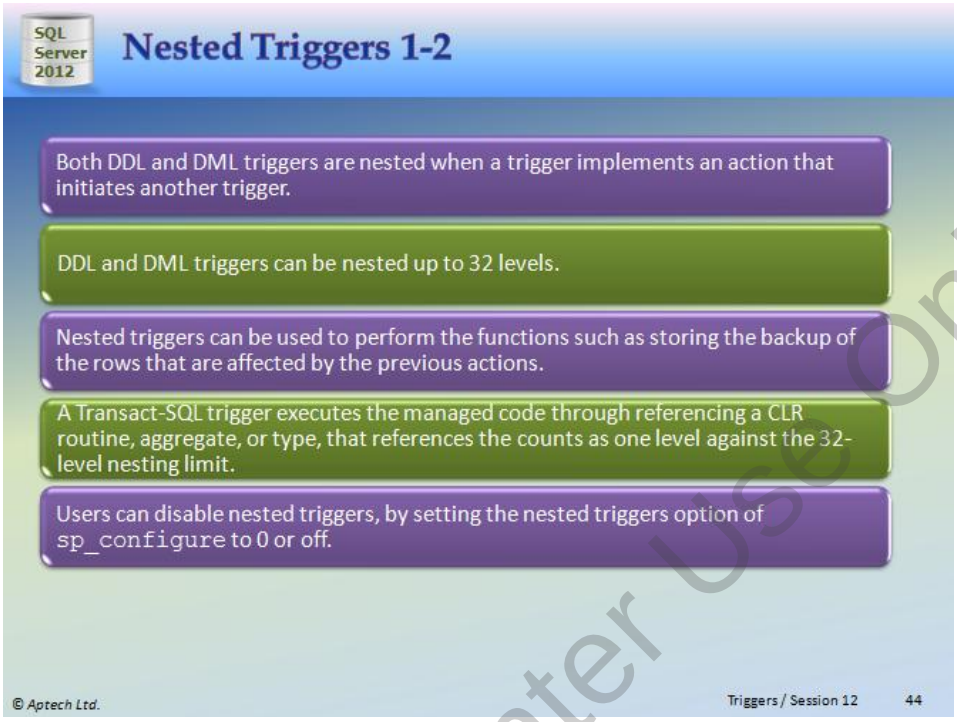
What are two scopes of DDL triggers?

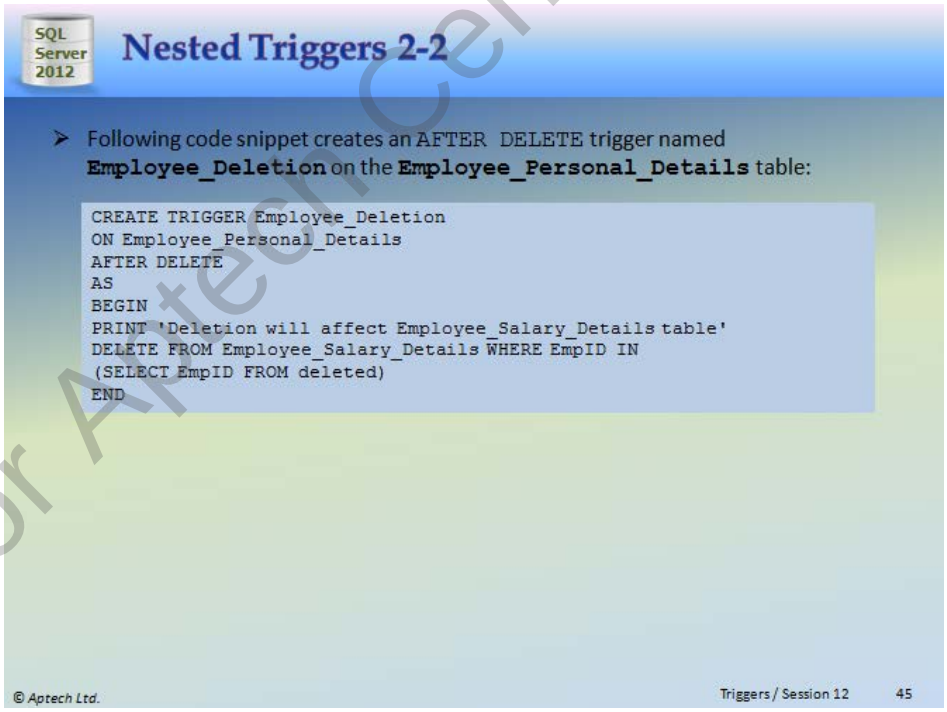
Answer:

Database-Scoped DDL and Server-Scoped DDL are two scopes of DDL triggers.

Nested Triggers

Slides 44 and 45





Using slides 44 and 45, explain the nested triggers.

Both DDL and DML triggers are nested when a trigger implements an action that initiates another trigger. DDL and DML triggers can be nested up to 32 levels. Suppose if a trigger modifies a table on which there is another trigger, the second trigger is initiated, which then calls a third trigger, and so on.

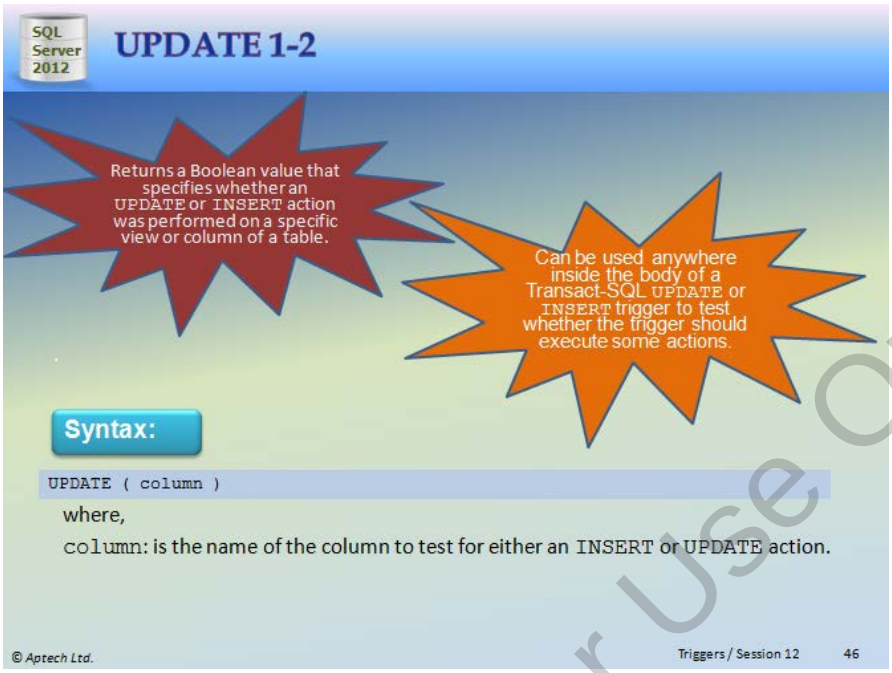
If the nested triggers are allowed, then the triggers in the sequence start an infinite loop. This will exceed the nesting level and the trigger will terminate. Nested triggers can be used to perform the functions such as storing the backup of the rows that are affected by previous actions.
A Transact-SQL trigger executes the managed code through referencing a CLR routine, aggregate, or type, that references the counts as one level against the 32-level nesting limit. Methods which are invoked from within managed code are not counted against this limit.
Users can disable nested triggers, by setting the nested triggers option of sp_configure to On or Off. The default configuration is allowed for nested triggers. If nested trigger option is Off, then the recursive trigger is disabled, irrespective of the recursive triggers setting that is set by using the ALTER DATABASE.
Mention that there are two types of recursion:

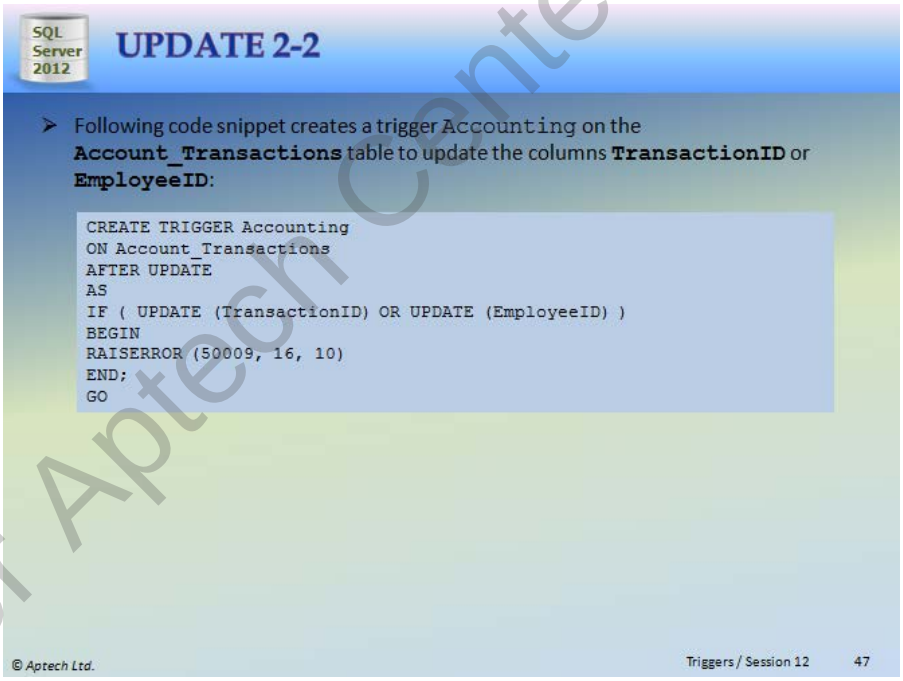
- Direct recursion: when a trigger fires and performs an action that causes the same trigger to fire again.
- Indirect recursion: when a trigger fires and performs an action that causes another trigger of the same type to fire.

Explain the Code Snippet which creates an AFTER DELETE trigger named Employee_Deletion on the Employee_Personal_Details table.
When a record is deleted from the Employee_Personal_Details table, the Employee_Deletion trigger is activated and a message is displayed. Also, the record of the employee is deleted from the Employee_Salary_Details table.
Explain the Code Snippet which creates an AFTER DELETE trigger Deletion_Confirmation on the Employee_Salary_Details table.
When a record is deleted from the Employee_Salary_Details table, the Deletion_Confirmation trigger is activated. This trigger prints the confirmation message of the record being deleted.
Thus, the Employee_Deletion and the Deletion_Confirmation triggers are seen to be nested.

UPDATE

Slides 46 and 47





Using slides 46 and 47, explain the Update function.

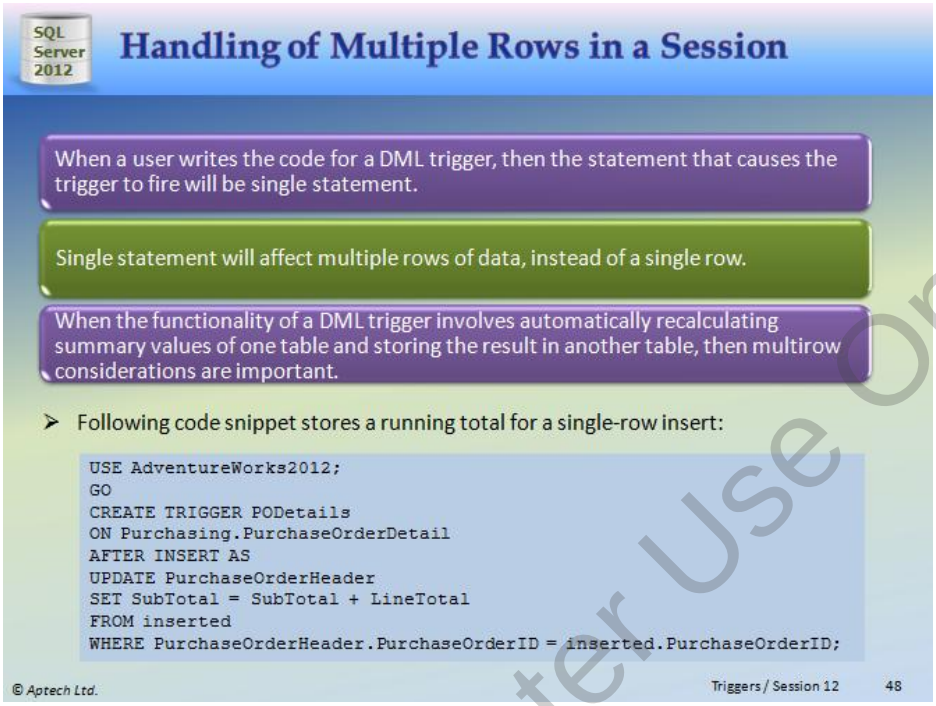
UPDATE () function returns a Boolean value that specifies whether an UPDATE or INSERT action was performed on a specific view or column of a table. UPDATE () function can be used anywhere inside the body of a Transact-SQL UPDATE or INSERT trigger to test whether the trigger should execute some actions.

Explain the syntax for UPDATE ().

Explain the Code Snippet which creates a trigger Accounting on the Account_Transactions table to update the columns TransactionID or EmployeeID.

Handling Multiple Rows in a Session

Slide 48



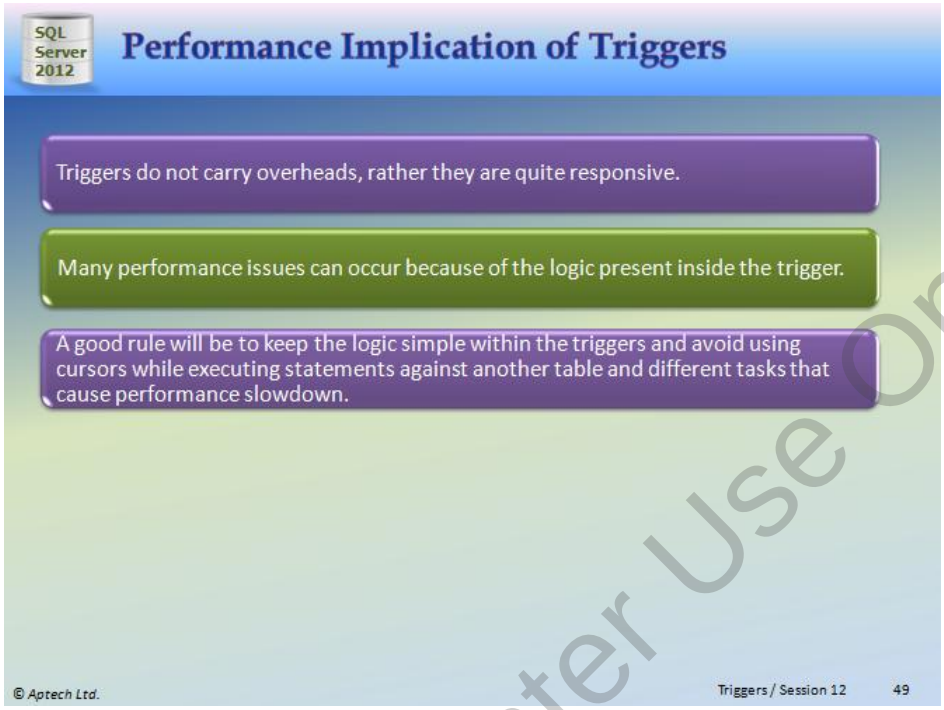
Using slide 48, explain how to handle multiple rows in a session.
When a user writes the code for a DML trigger, then the statement that causes the trigger to fire will be a single statement. This single statement will affect multiple rows of data, instead of a single row. This is a common behavior for DELETE and UPDATE triggers as these statements often affect multiple rows. The behavior for INSERT triggers is less common as the basic INSERT statement adds only one row.
When the functionality of a DML trigger involves automatically recalculating summary values of one table and storing the result in another table, then multirow considerations are important.
Explain the Code Snippet which stores a running total for a single-row insert.

In this code, the subtotal is calculated and stored for a single-row insert operation.

Tips:
Do not use cursors in triggers because they could potentially reduce performance instead use the rowset logic.

Performance Implications of Triggers

Slide 49



Using slide 49, explain the performance implications of triggers.

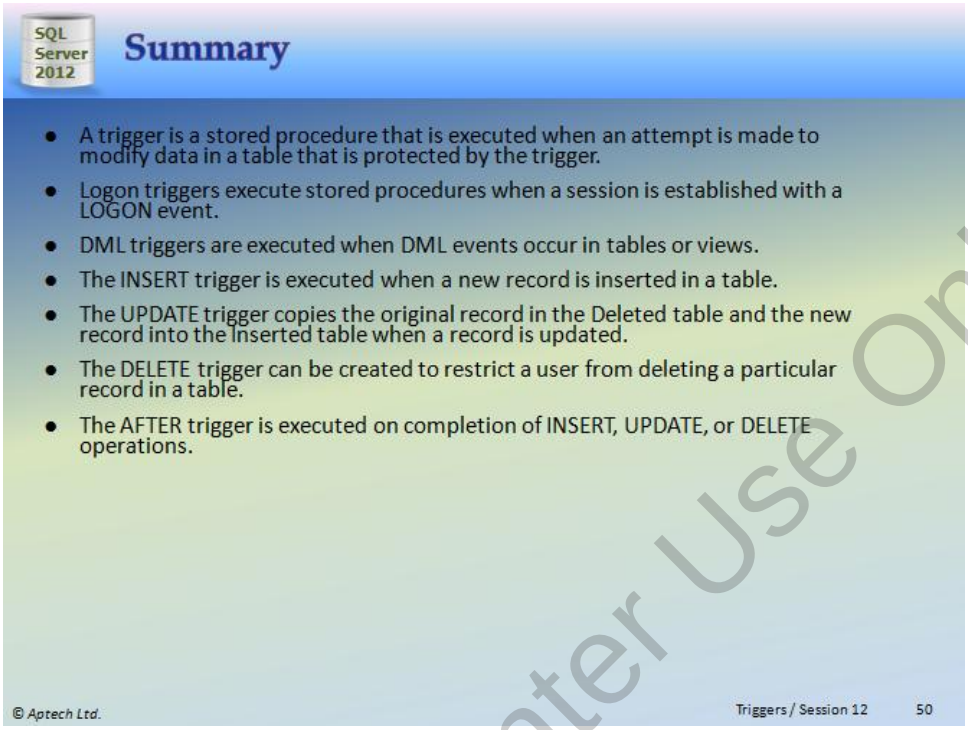
In reality, triggers do not carry overheads, rather they are quite responsive. However, many performance issues can occur because of the logic present inside the trigger. Suppose a trigger creates a cursor and loops through many rows, then there will be a slowdown in the process.

Similarly, consider that the trigger executes various SQL statements against other tables separate from the Inserted and Deleted tables. This will again result in the slowdown of speed of SQL statements that are within the trigger.

A good rule will be to keep the logic simple within the triggers and avoid using cursors while executing statements against another table and different tasks that cause performance slowdown.

Summarize Session

Slide 50



Using slide 50, summarize the session. End the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- A trigger is a stored procedure that is executed when an attempt is made to modify data in a table that is protected by the trigger.
- Logon triggers execute stored procedures when a session is established with a LOGON event.
- DML triggers are executed when DML events occur in tables or views.
- The INSERT trigger is executed when a new record is inserted in a table.
- The UPDATE trigger copies the original record in the Deleted table and the new record into the Inserted table when a record is updated.
- The DELETE trigger can be created to restrict a user from deleting a particular record in a table.
- The AFTER trigger is executed on completion of INSERT, UPDATE, or DELETE operations.

12.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Programming Transact-SQL topic that is offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 13 – Programming Transact-SQL

13.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

13.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe an overview of Transact-SQL programming
- Describe the Transact-SQL programming elements
- Describe program flow statements
- Describe various Transact-SQL functions
- Explain the procedure to create and alter User-defined Functions (UDFs)
- Explain creation of windows with OVER
- Describe window functions

13.1.2 Teaching Skills

To teach this session, you should be well-versed with the programming with Transact-SQL and describes various Transact-SQL programming elements. Along with this also prepare yourself with program flow statements, Transact-SQL functions, and so on.
The session also covers the procedure to create and alter user-defined functions, and create windows using the OVER and window functions.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

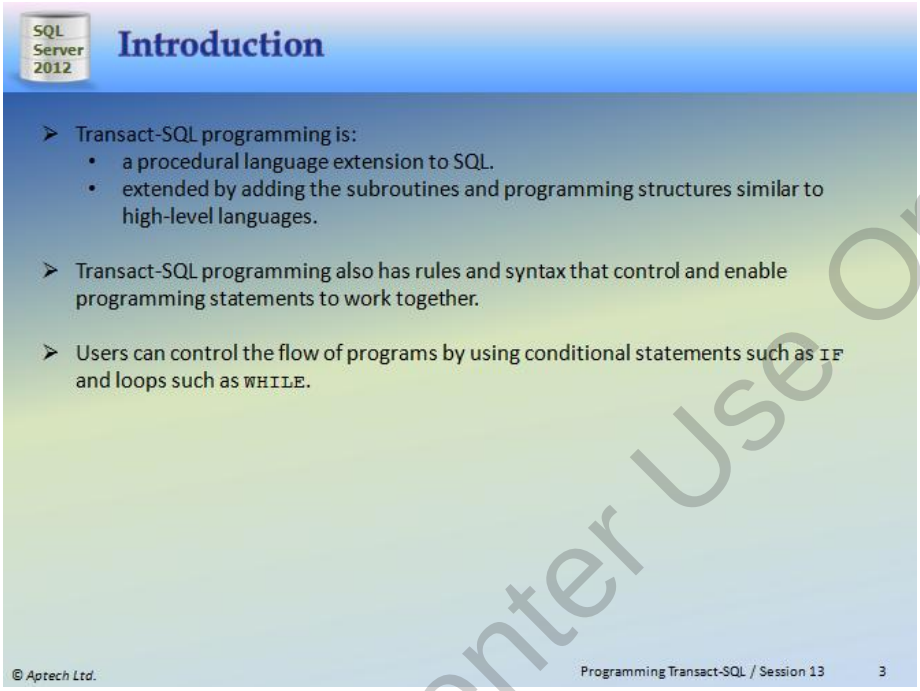
In-Class Activities:
Follow the order given here during In-Class activities.

Overview of the Session:
Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces programming with Transact-SQL and describes various Transact-SQL programming elements. They will also learn about program flow statements, Transact-SQL functions, and so on. They will also know about the procedure to create and alter user-defined functions, create windows using the OVER and window functions.

13.2 In-Class Explanations

Introduction

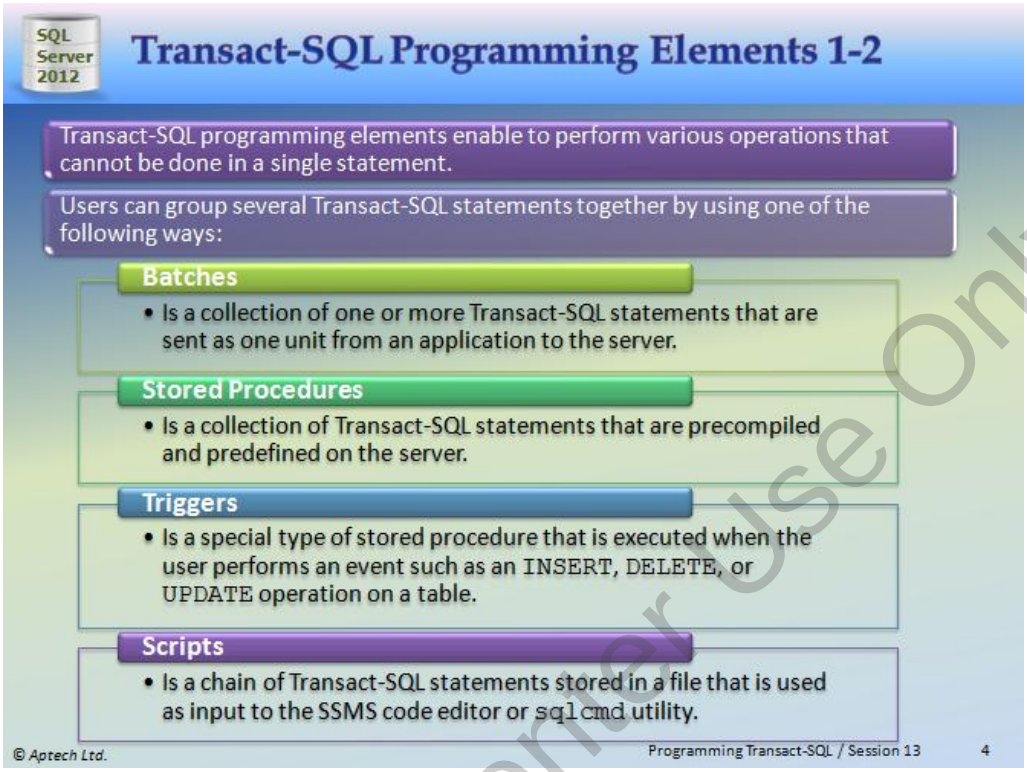
Slide 3

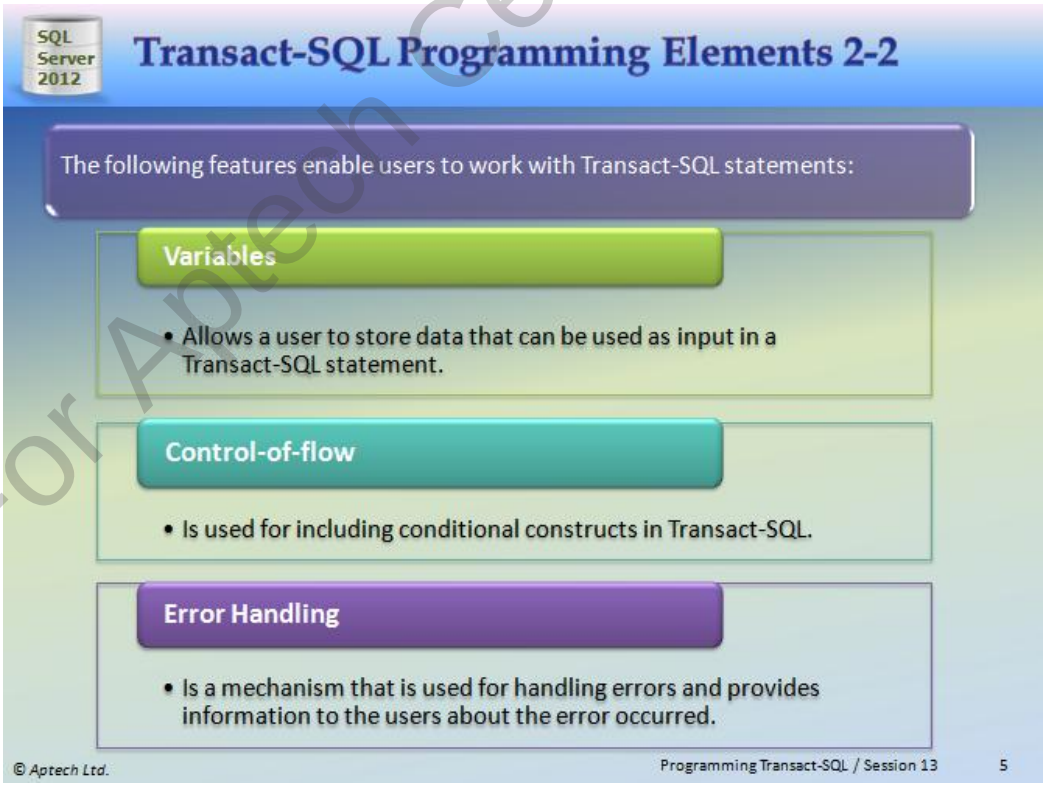


Using slide 3, explain the introduction.
Transact-SQL programming is a procedural language extension to SQL. Transact-SQL programming is extended by adding the subroutines and programming structures similar to high-level languages. Like high-level languages, Transact-SQL programming also has rules and syntax that control and enable programming statements to work together. Users can control the flow of programs by using conditional statements such as IF and loops such as WHILE.

Transact-SQL Programming Elements

Slides 4 and 5





Using slides 4 and 5, explain Transact-SQL programming elements.

Transact-SQL programming elements enable to perform various operations that cannot be done in a single statement. Users can group several Transact-SQL statements together.

Explain the ways where the Transact-SQL statements can be grouped together.
Also, explain the features which enable the users to work with Transact-SQL statement.

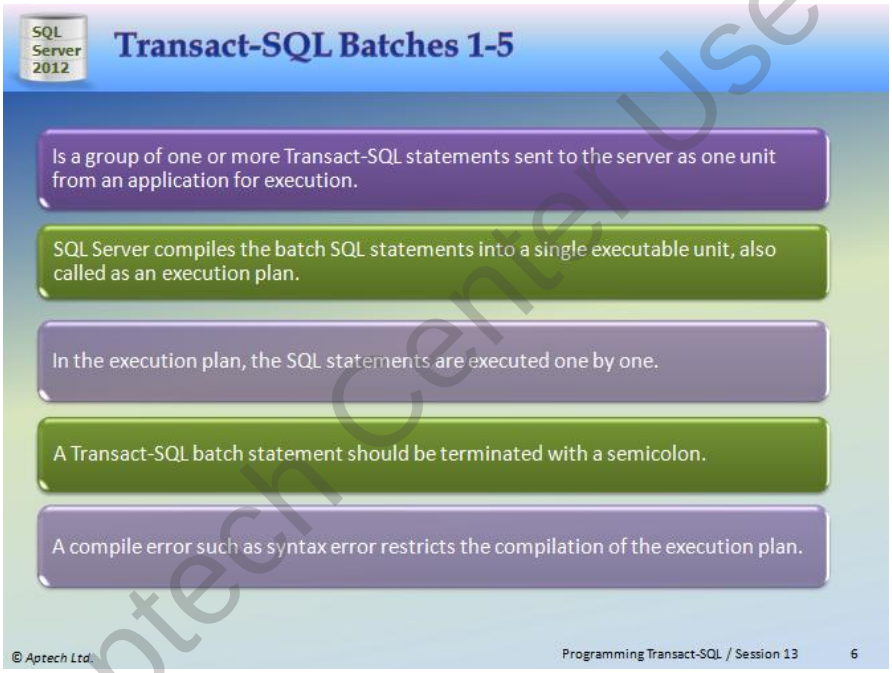
In-Class Question:

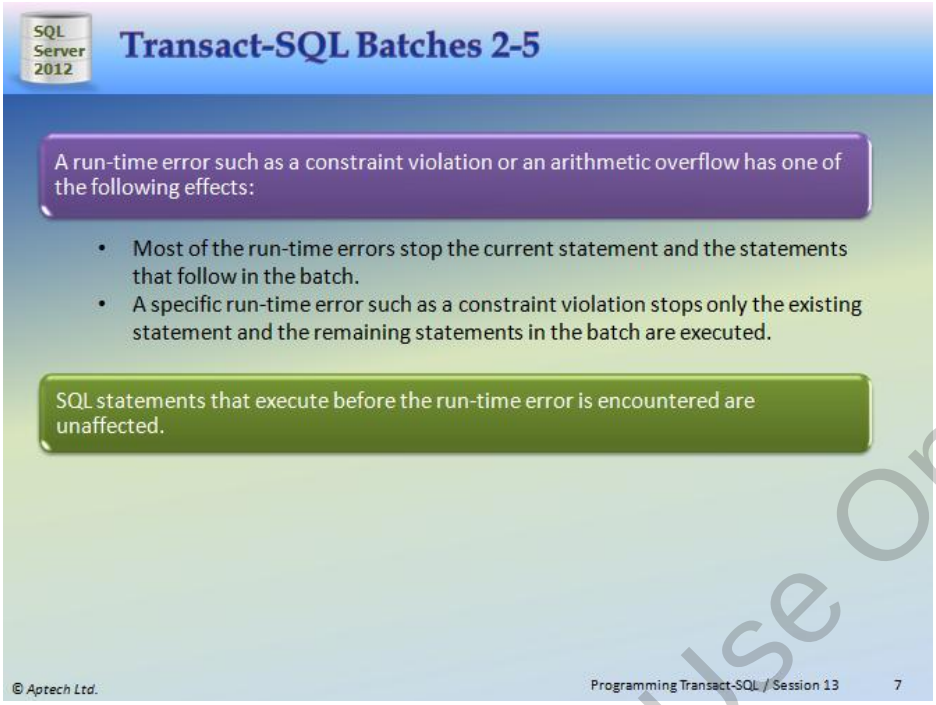
What is a batch?

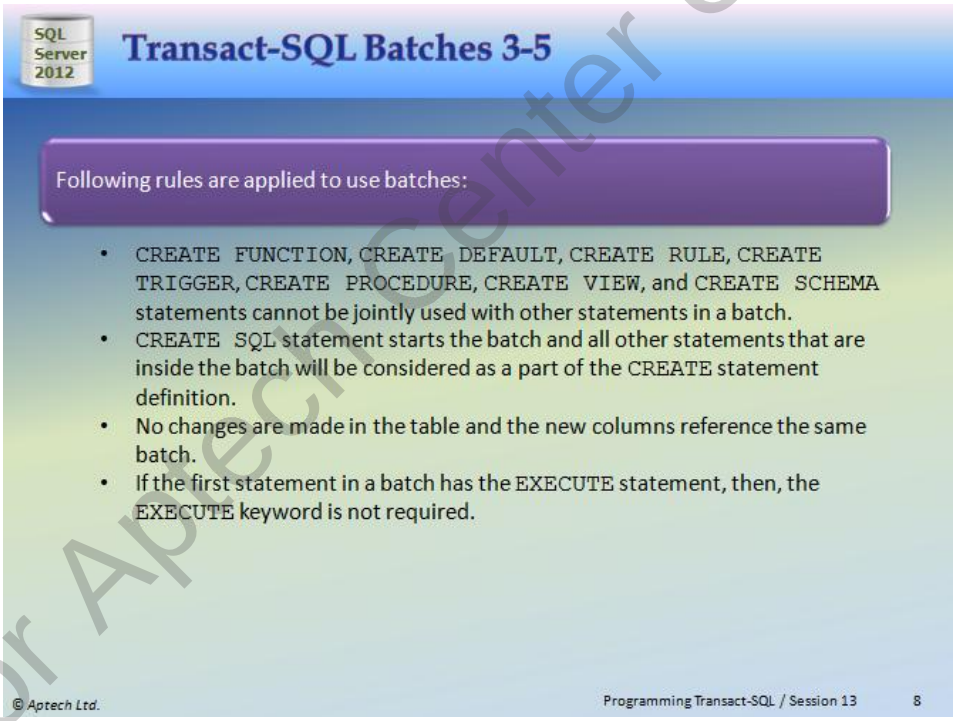
Answer:
A batch is a collection of one or more Transact-SQL statements that are sent as one unit from an application to the server.

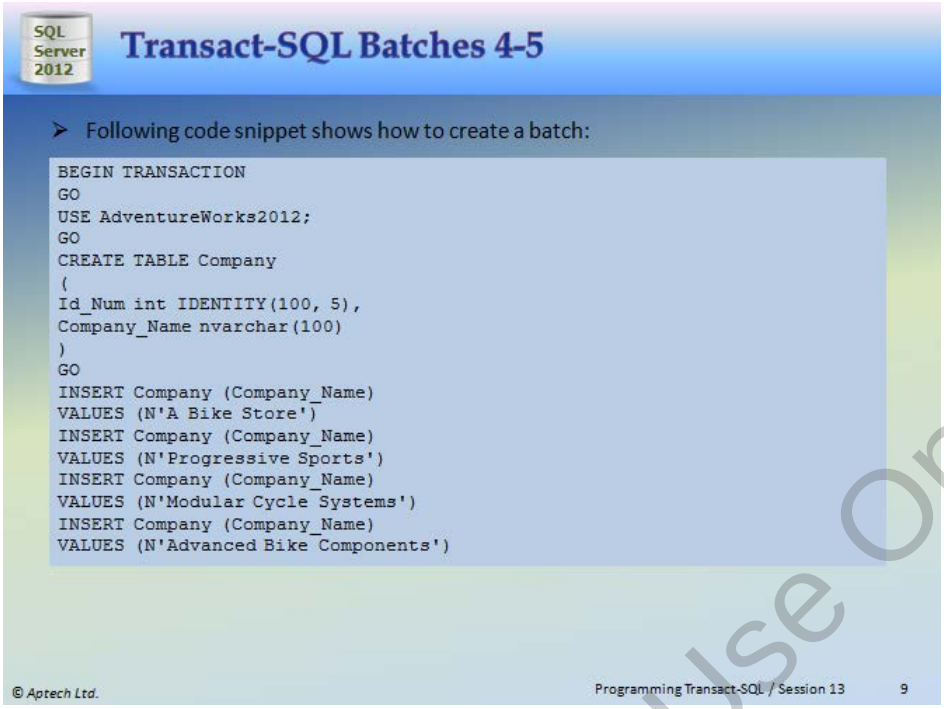
Transact-SQL Batches

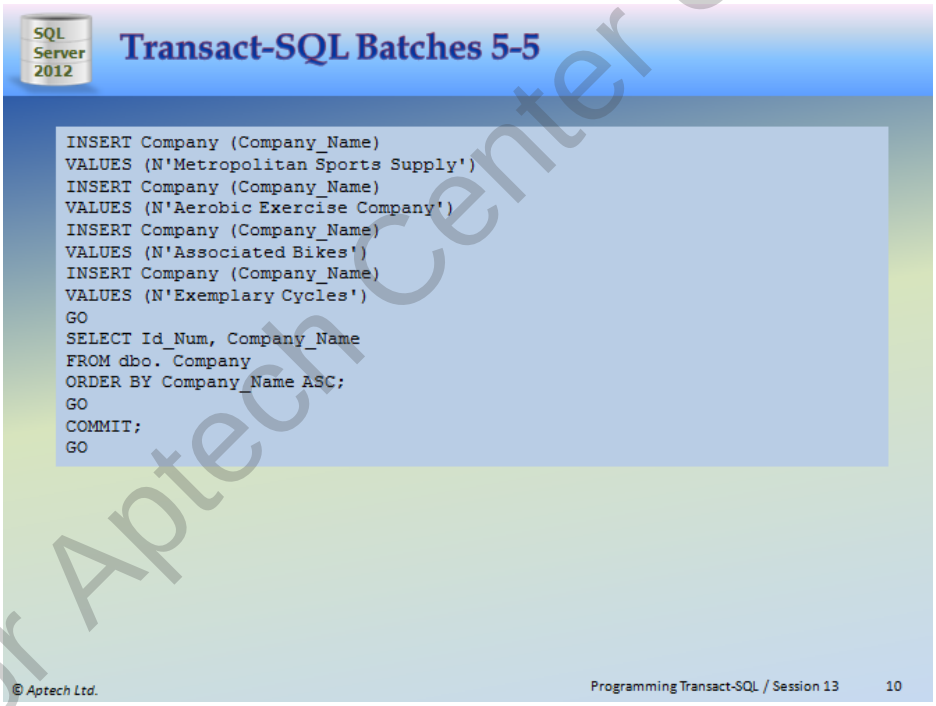
Slides 6 to 10











Using slides 6 to 10, explain the Transact-SQL batches.

A Transact-SQL batch is a group of one or more Transact-SQL statements sent to the server as one unit from an application for execution. SQL Server compiles the batch SQL statements into a single executable unit, also called as an execution plan. In the execution plan, the SQL statements are executed one by one. A Transact-SQL batch statement should be terminated with a semicolon. This condition is not mandatory, but the facility to end a statement without a semicolon is deprecated and may be removed in the new versions of SQL Server in the future. Hence, it is recommended to use semicolons to terminate batches. A compile error such as syntax error restricts the compilation of the execution plan. So, if a compile-time error occurs, no statements in the batch are executed.

A run-time error such as a constraint violation or an arithmetic overflow has one of the following effects:

- Most of the run-time errors stop the current statement and the statements that follow in the batch.
- A specific run-time error such as a constraint violation stops only the existing statement and the remaining statements in the batch are executed.

The SQL statements that execute before the run-time error is encountered are unaffected. The only exception is when the batch is in a transaction and the error results in the transaction being rolled back.

For example, suppose there are 10 statements in a batch and the sixth statement has a syntax error, then the remaining statements in the batch will not execute. If the batch is compiled and the third statement fails to run, then the results of the first two statements remains unaffected as it is already executed.

The following rules are applied to use batches:

- CREATE FUNCTION, CREATE DEFAULT, CREATE RULE, CREATE TRIGGER, CREATE PROCEDURE, CREATE VIEW, and CREATE SCHEMA statements cannot be jointly used with other statements in a batch. The CREATE SQL statement starts the batch and all other statements that are inside the batch will be considered as a part of the CREATE statement definition.
- No changes are made in the table and the new columns reference the same batch.
- If the first statement in a batch has the EXECUTE statement, then the EXECUTE keyword is not required. It is required only when the EXECUTE statement does not exist in the first statement in the batch.

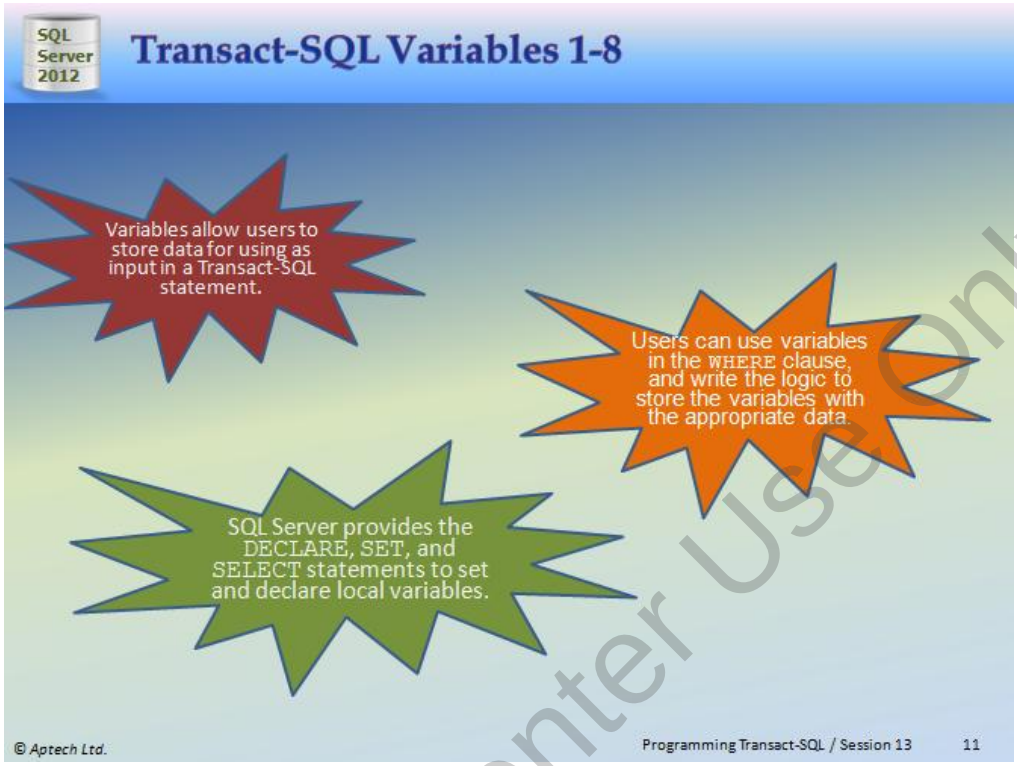
Explain the code snippet that creates a view in a batch on slides 9 and 10.

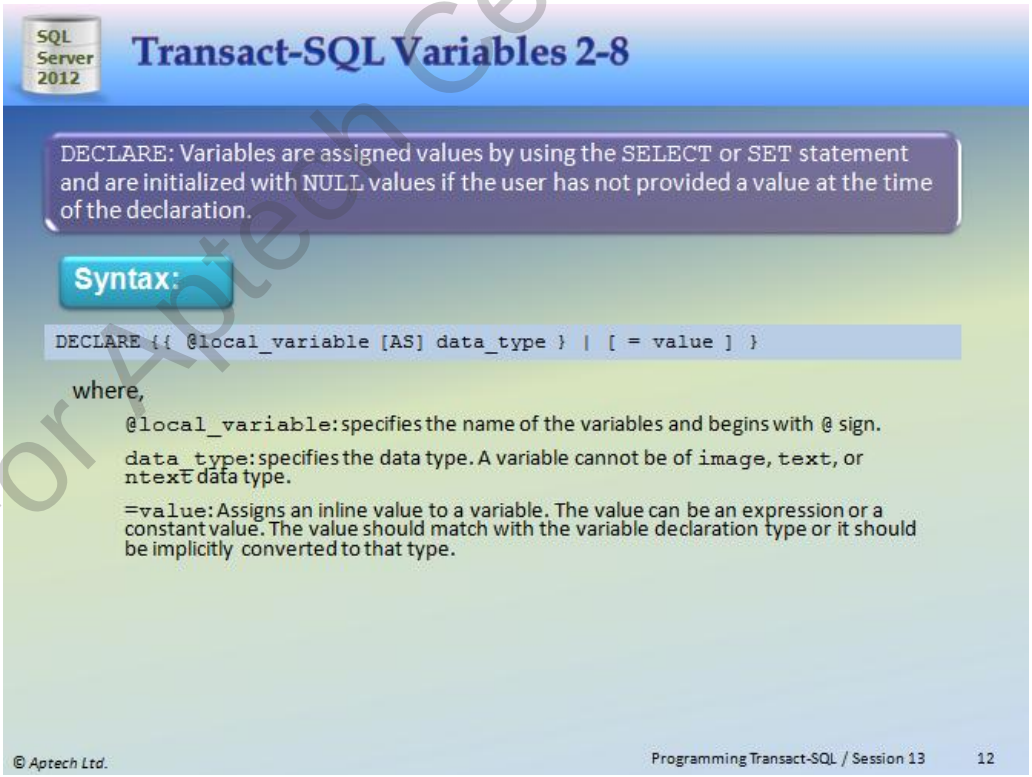
In this code snippet, several batches are combined into one transaction. The BEGIN TRANSACTION and COMMIT statements enclose the transaction statements. The CREATE TABLE, BEGIN TRANSACTION, SELECT, COMMIT, and USE statements are in single-statement batches. The INSERT statements are all included in one batch.

Go statement signals the end of a batch of Transact-SQL statements to the SQL Server utilities. The scope of local variables is limited to a batch so cannot be referenced after a GO command.

Transact-SQL Variables

Slides 11 to 18





Transact-SQL Variables 3-8

- Following code snippet shows the use of a local variable to retrieve contact information for the last names starting with **Man**:

```
USE AdventureWorks2012;
GO
DECLARE @find varchar(30) = 'Man%';
SELECT p.LastName, p.FirstName, ph.PhoneNumber
FROM Person.Person AS p
JOIN Person.PersonPhone AS ph ON p.BusinessEntityID =
   ph.BusinessEntityID
WHERE LastName LIKE @find;
```

Output:

Results | Messages

	LastName	FirstName	PhoneNumber
1	Manchepalli	Ajay	1 (11) 500 555-0174
2	Manek	Parul	1 (11) 500 555-0146
3	Manzanares	Tomas	1 (11) 500 555-0178

Transact-SQL Variables 4-8

SET: statement sets the local variable created by the DECLARE statement to the specified value.

Syntax:

```
SET
{ @local_variable = { expression}
}
|
{ @local_variable
{+= | -= | *= | /= | %= | &= | ^= | |= } expression
}
```

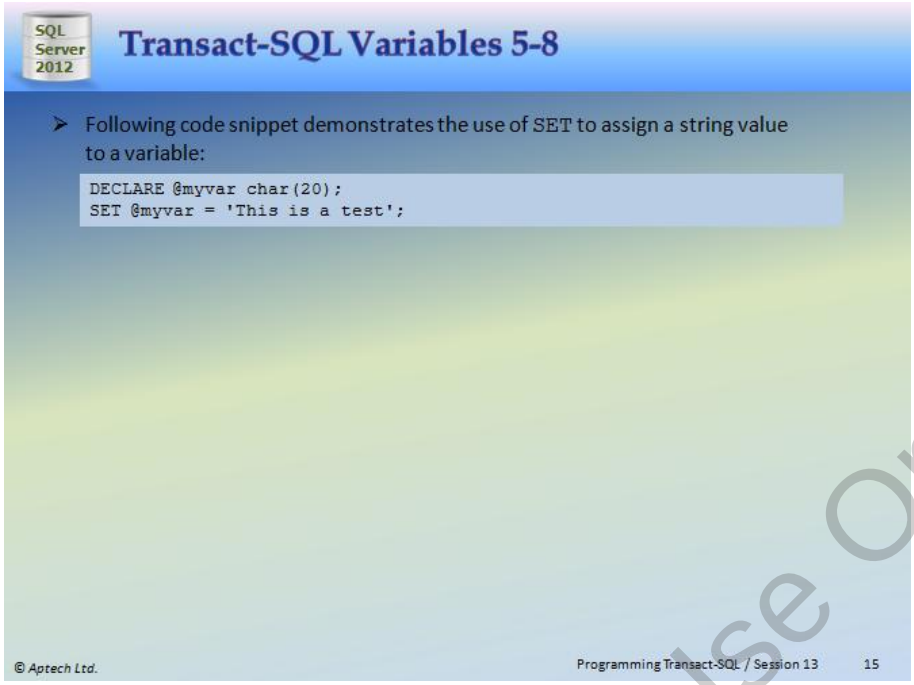
where,

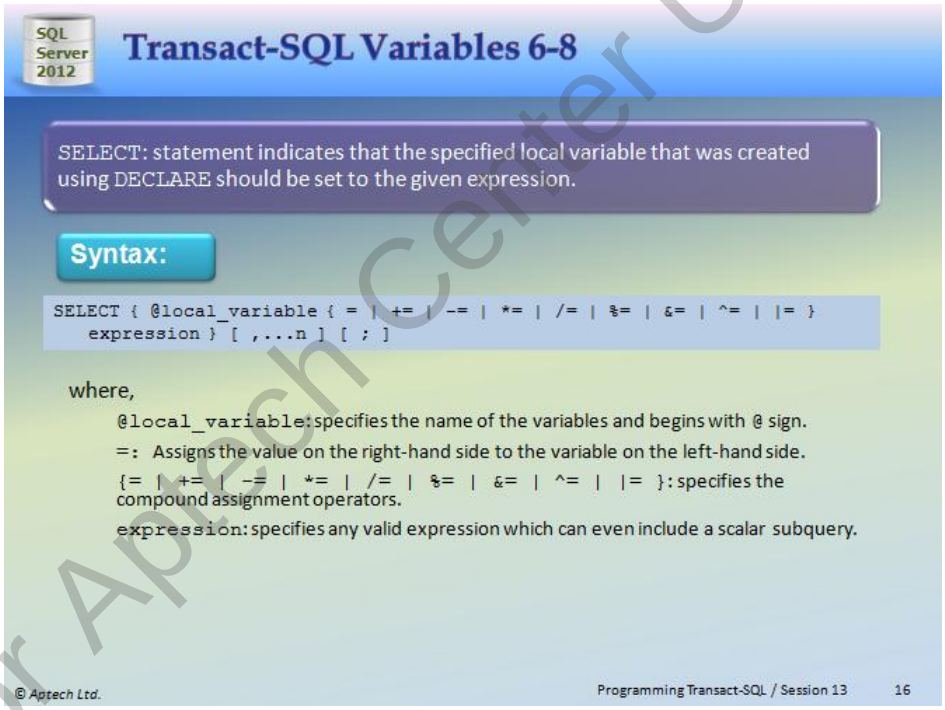
`@local_variable`: specifies the name of the variables and begins with @ sign.

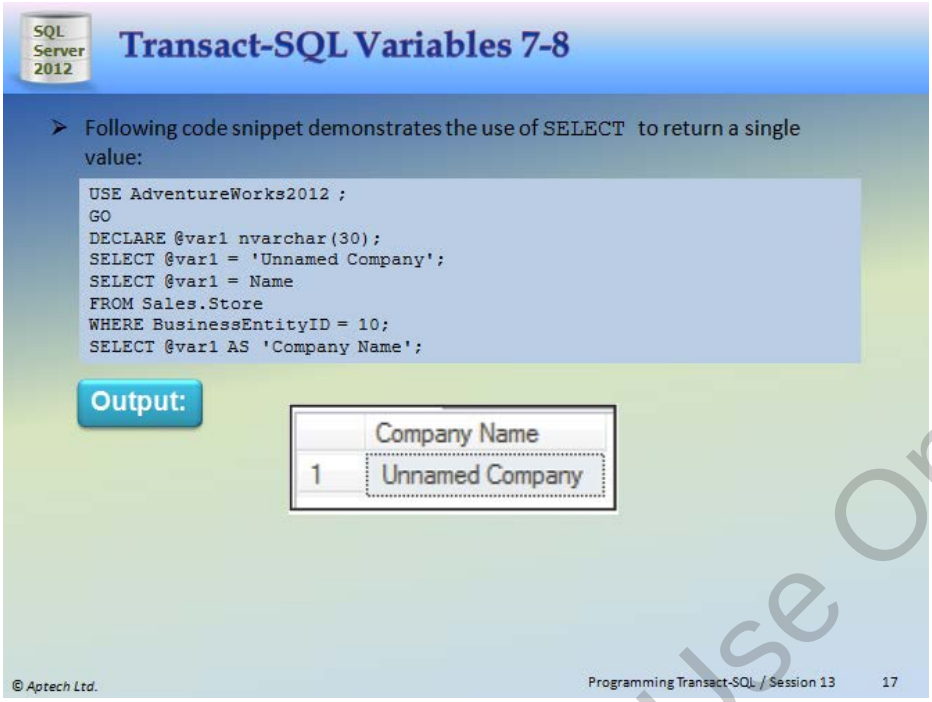
`=`: Assigns the value on the right-hand side to the variable on the left-hand side.

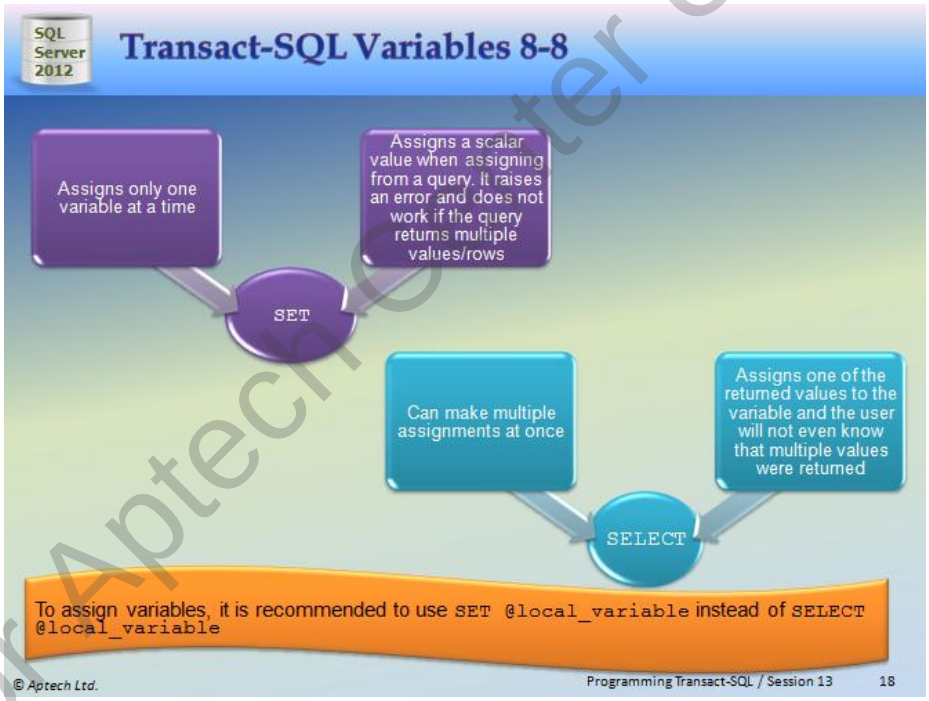
`{= | += | -= | *= | /= | %= | &= | ^= | |= }`: specifies the compound assignment operators.

`expression`: specifies any valid expression which can even include a scalar subquery.









Using slides 11 to 18, explain the Transact-SQL variables.
Variables allow users to store data to be used as input in a Transact-SQL statement. For example, users can create a query that requires various types of data values specified in the WHERE clause each time the query is executed. Here, the users can use variables in the WHERE clause, and write the logic to store the variables with the appropriate data.

SQL Server provides the following statements to set and declare local variables:

- **DECLARE**:

Variables are declared with the DECLARE statement in the body of a batch. These variables are assigned values by using the SELECT or SET statement. The variables are initialized with

NULL values if the user has not provided a value at the time of the declaration.

Explain the basic syntax to declare a local variable on slide 12.

Explain the code snippet using slide 13 which uses a local variable to retrieve contact information for the last names starting with Man.
In this code snippet, a local variable named, @find is used to store the search criteria, which will be then used to retrieve the contact information. Here, the criteria include all last names beginning with Man. Figure on slide 13 displays the output.

Mention that, the SET statement sets the local variable created by the DECLARE statement to the specified value.
Explain the basic syntax to set a local variable using slide 14.

Explain the code snippet which demonstrates the use of SET to assign a string value to a variable using slide 15.
In this code snippet, the @myvar variable is assigned a string value.

Mention that, the SELECT statement indicates that the specified local variable that was created using DECLARE should be set to the given expression.

Explain the syntax of the SELECT statement using slide 16.

Explain the code snippet using slide 17 which shows how to use SELECT to return a single value.
In this code snippet, the variable, @var1 is assigned Unnamed Company as its value. The query against the Store table will return zero rows as the value specified for the BusinessEntityID does not exist in the table. The variable will then, retain the Unnamed Company value and will be displayed with the heading, Company Name. Figure on slide 17 displays the output.

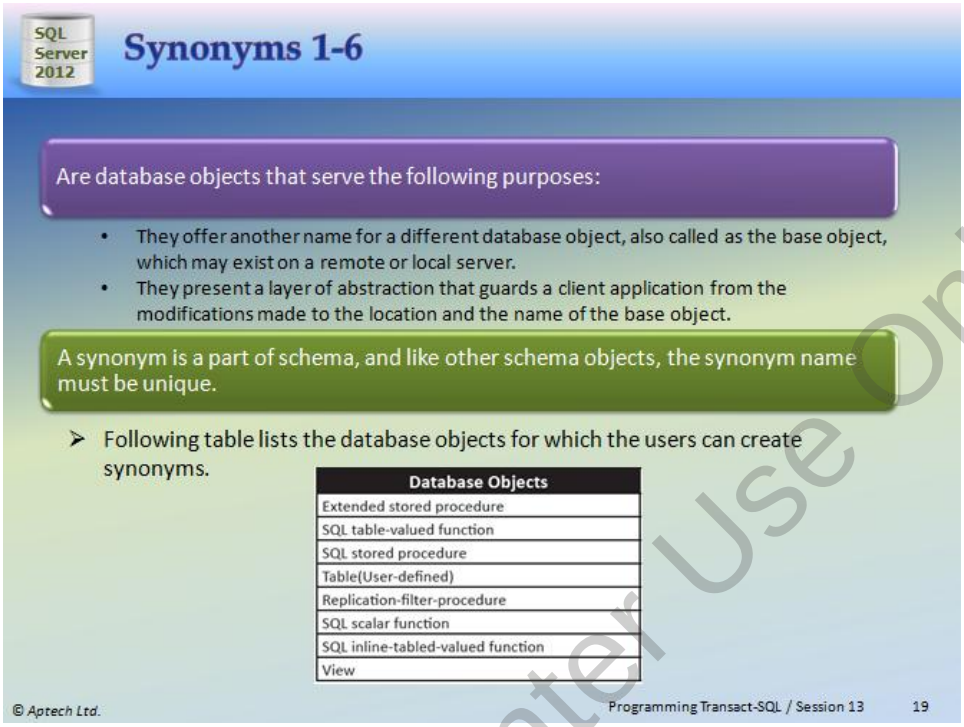
Explain the differences between the SET and SELECT statement.

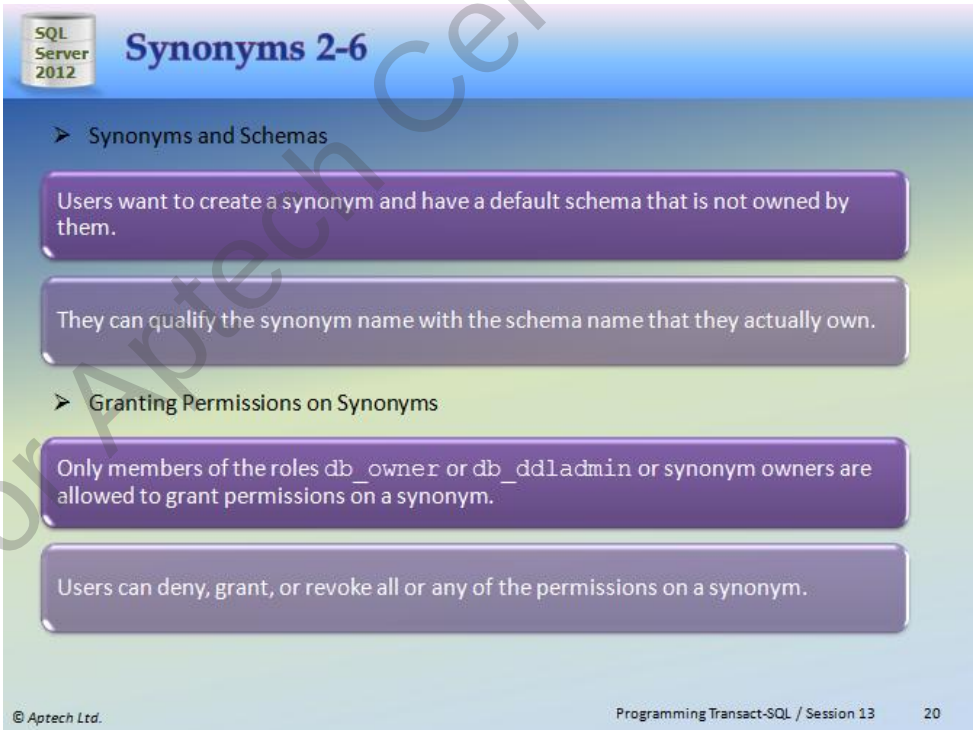
- It is possible to assign only one variable at a time using SET. However, using SELECT, you can make multiple assignments at once.
- SET can only assign a scalar value while assigning from a query. It raises an error and does not work if the query returns multiple values/rows. However, SELECT assigns one of the returned values to the variable and the user will not even know that multiple values were returned.

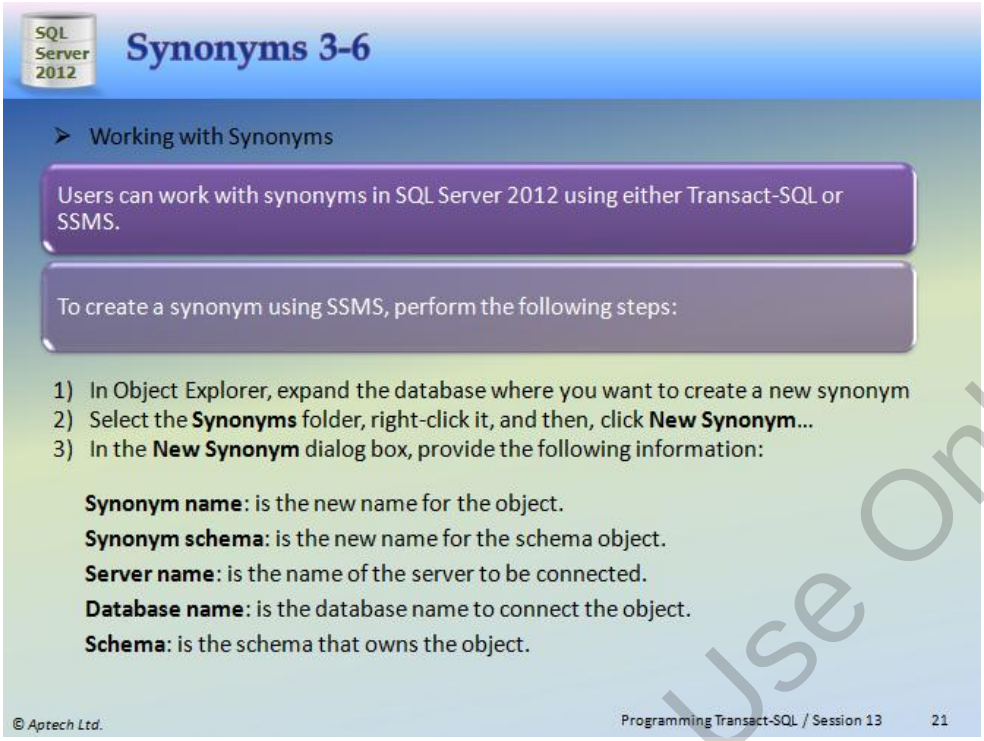
Mention that, to assign variables, it is recommended to use SET @local_variable instead of SELECT @ local_variable.

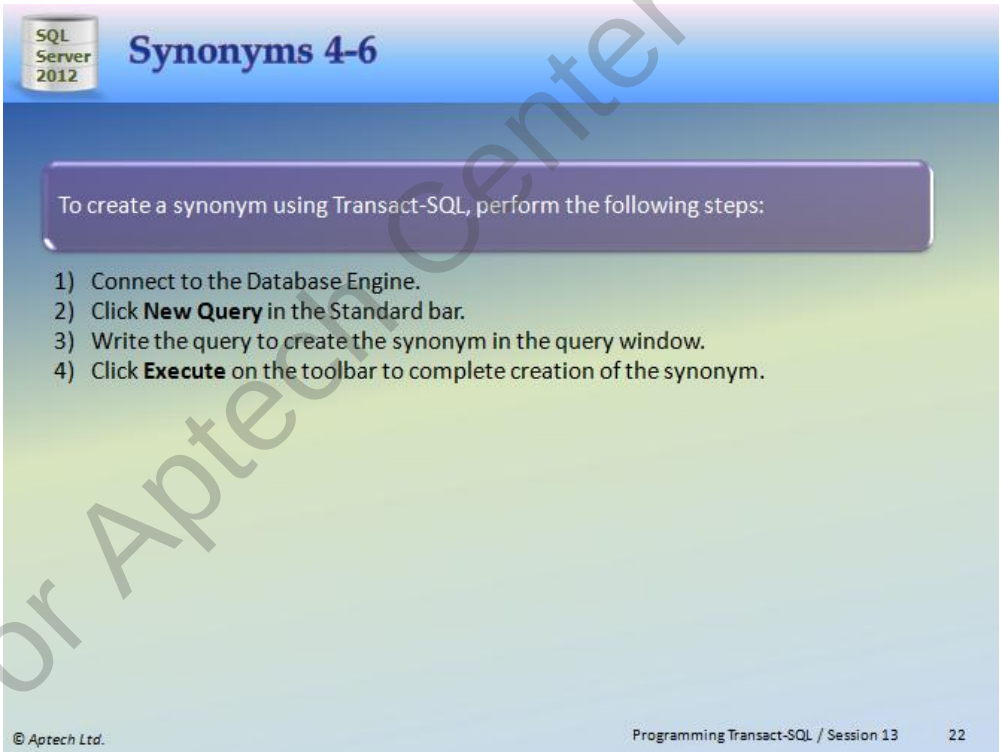
Synonyms

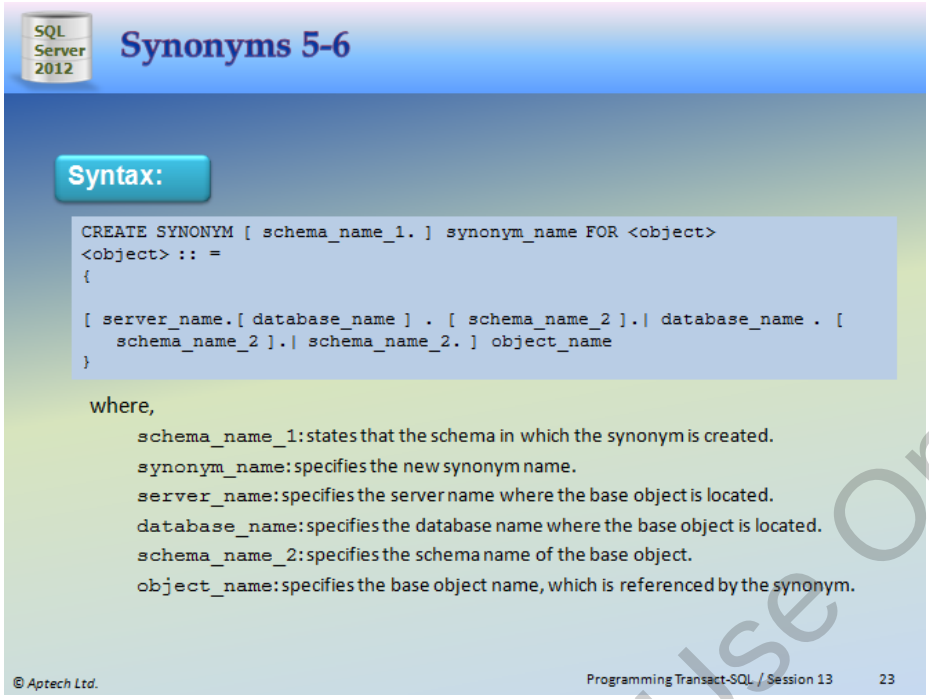
Slides 19 to 24

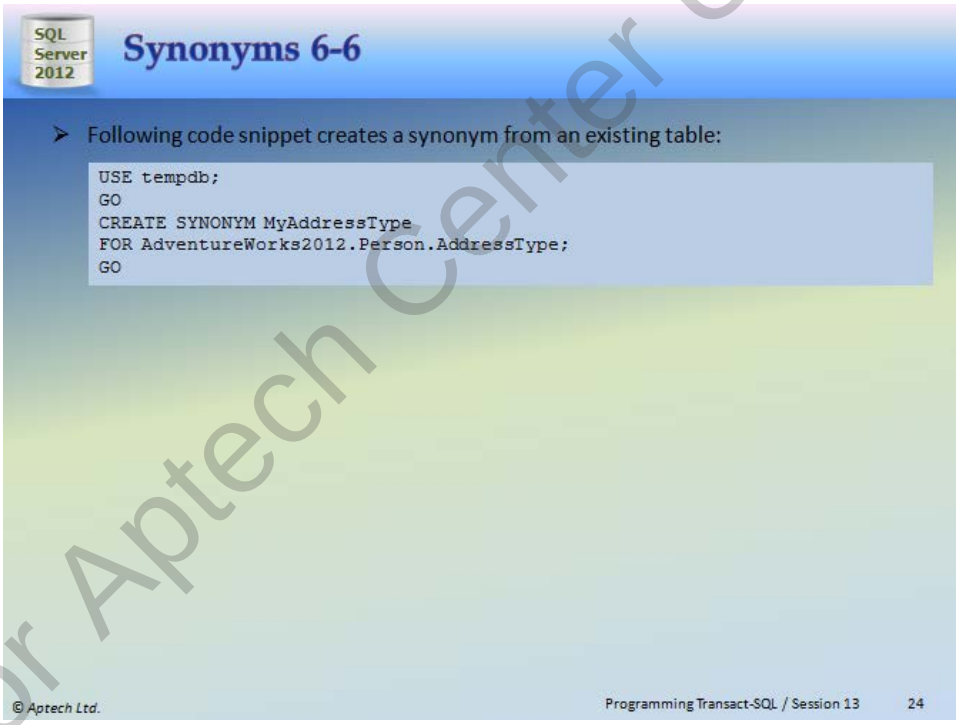












Using slides 19 to 24, explain the synonyms.
Synonyms are database objects that serve the following purposes:

- They offer another name for a different database object, also called as the base object, which may exist on a remote or local server.
- They present a layer of abstraction that guards a client application from the modifications made to the location and the name of the base object.

For example, consider that the Department table of AdventureWorks2012 is located on the first server named, Server1. To reference this table from the second server, Server2, a client application would have to use the four-part name

Server1.AdventureWorks2012.Person.Department. If the location of the table was modified, for example, to another server, the client application would have to be rectified to reflect that change. To address both these issues, users can create a synonym, DeptEmpTable, on Server2 for the Department table on Server1. Now, the client application only has to use the single name, DeptEmpTable, to refer to the Employee table.

Similarly, if the location of the Department table changes, users have to modify the synonym, DeptEmpTable, to point to the new location of the Department table. Since there is no ALTER SYNONYM statement, you first have to drop the synonym, DeptEmpTable, and then, re-create the synonym with the same name, but point the synonym to the new location of Department.

Mention that, a synonym is a part of schema, and similar to other schema objects, the synonym name must be unique.

Explain the list of the database objects for which the users can create synonyms.

- **Synonyms and Schemas**: Suppose users want to create a synonym and have a default schema that is not owned by them. In such a case, they can qualify the synonym name with the schema name that they actually own. Consider that a user owns a schema Resources, but Materials is the user's default schema. If this user wants to create a synonym, he/she must prefix the name of the synonym with the schema Resources.
- **Granting Permissions on Synonyms**: Only members of the roles db_owner or db_ddladmin or synonym owners are allowed to grant permissions on a synonym. Users can deny, grant, or revoke all or any of the permissions on a synonym. Table displays the list of permissions that are applied on a synonym.

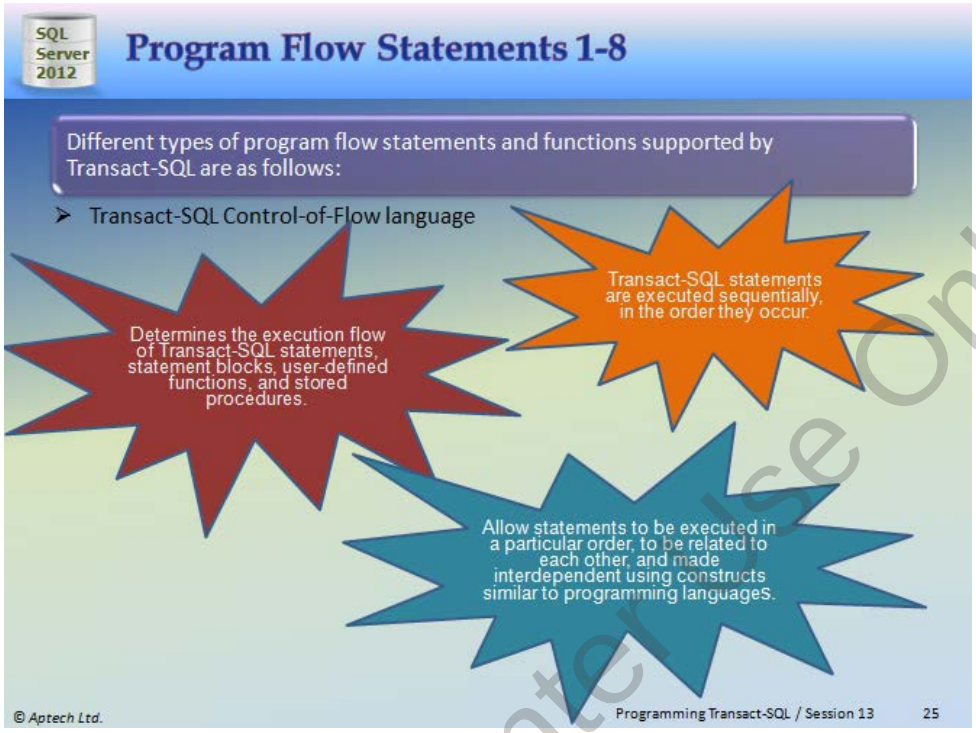
Also explain how to work with the synonyms using SSMS using slide 21.
Also explain the procedure to create synonyms using Transact-SQL.

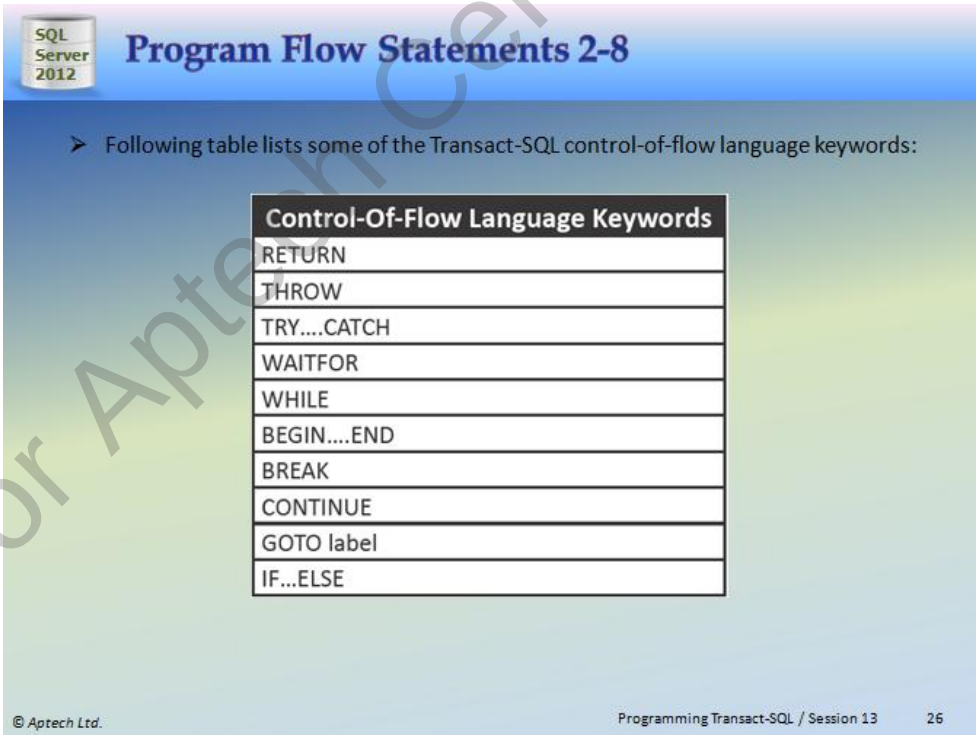
Explain the syntax to create a synonym using slide 23.

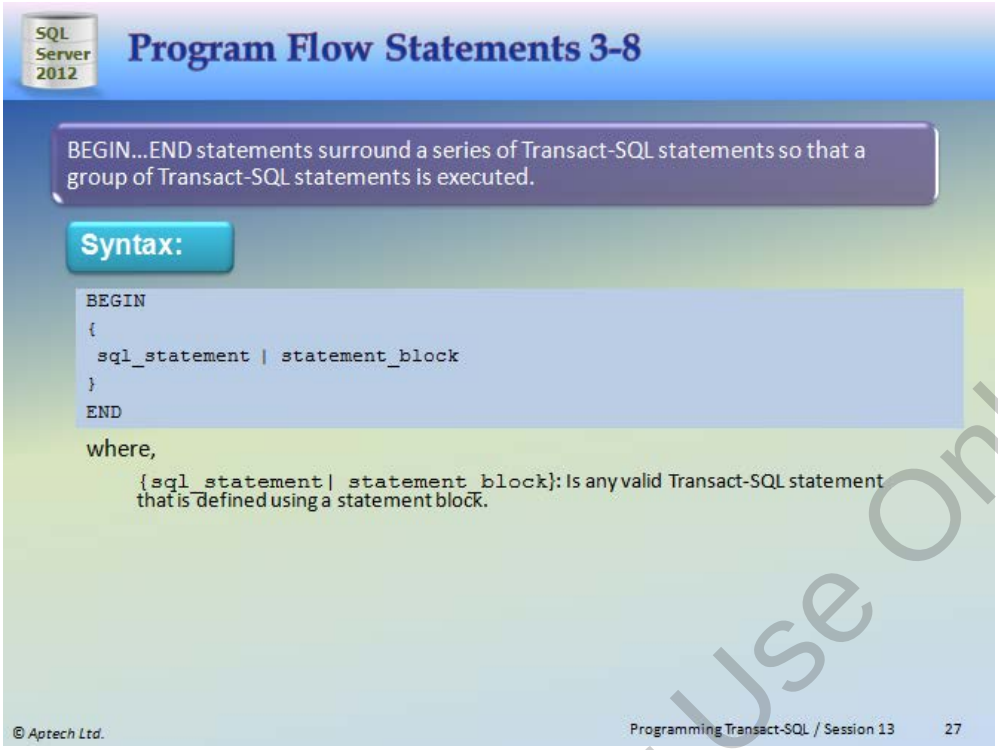
Explain the code snippet using slide 24 which creates a synonym from an existing table.
In this code snippet, a synonym is created from an existing table present in the AdventureWorks2012 database.

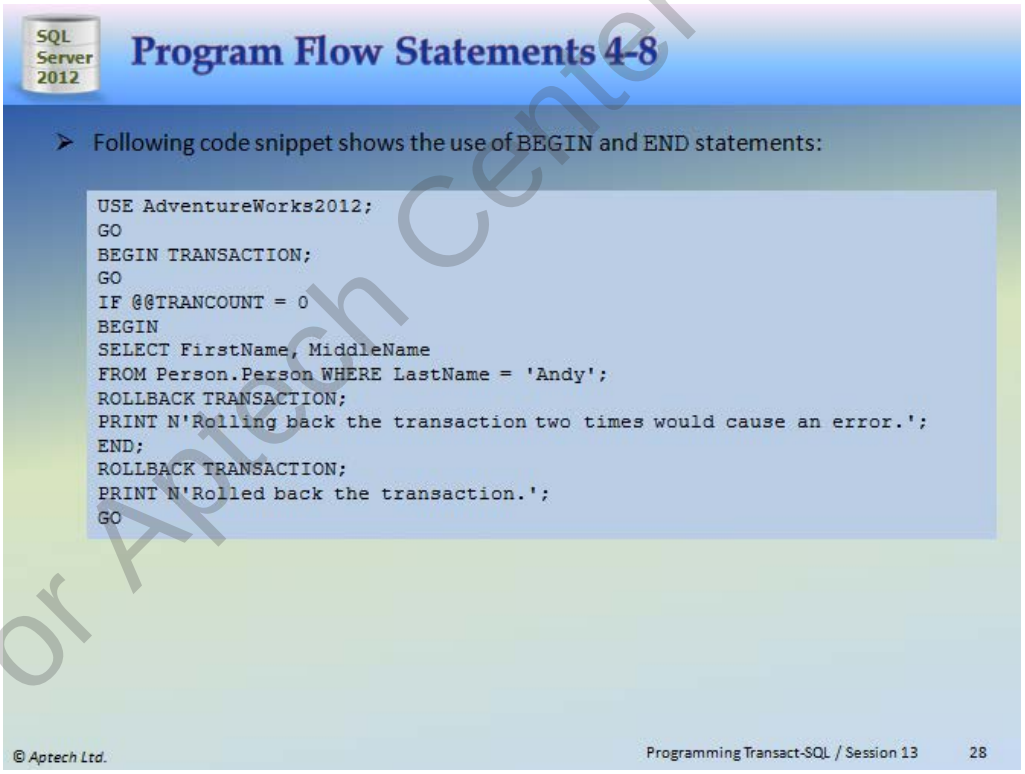
Program Flow Statements

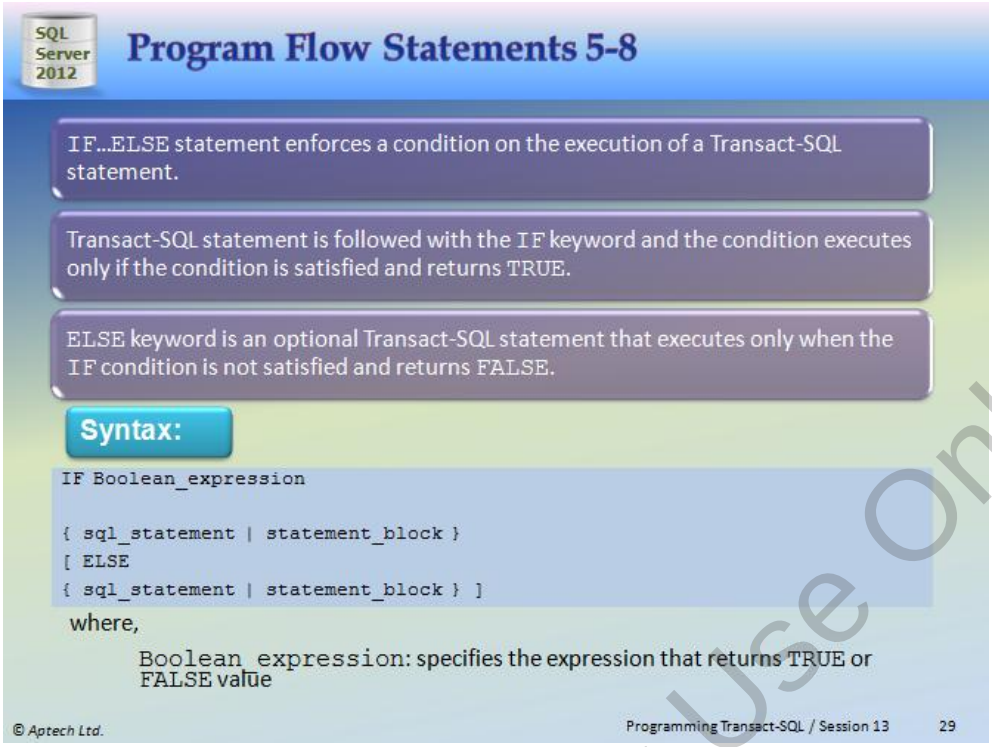
Slides 25 to 32

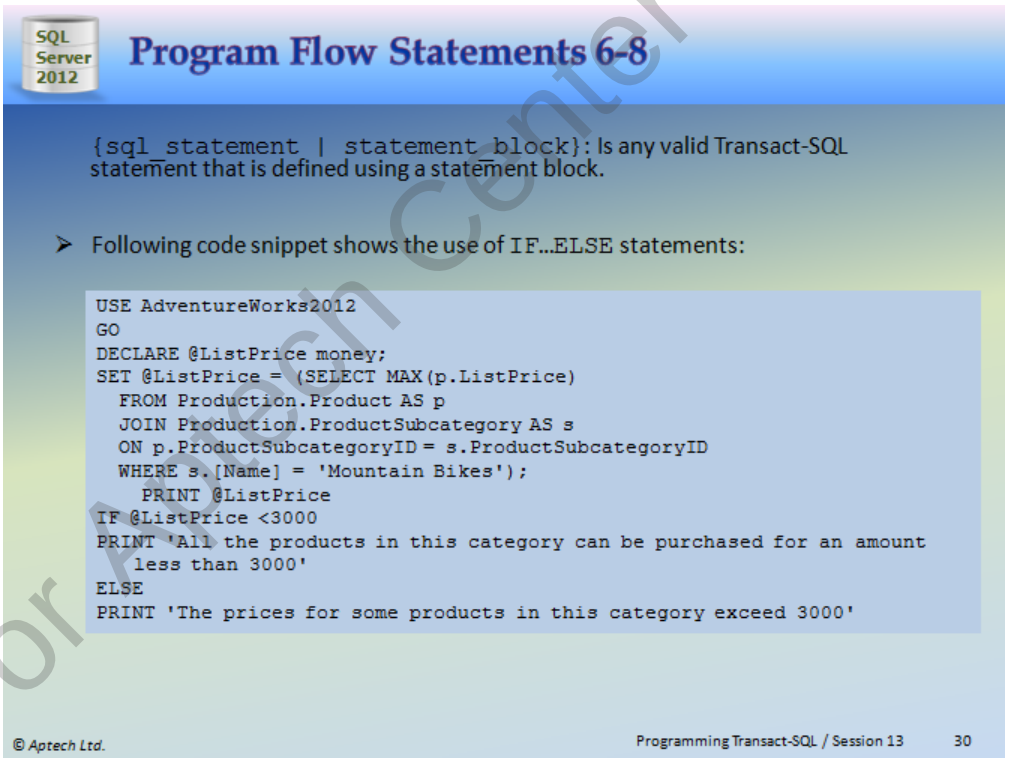


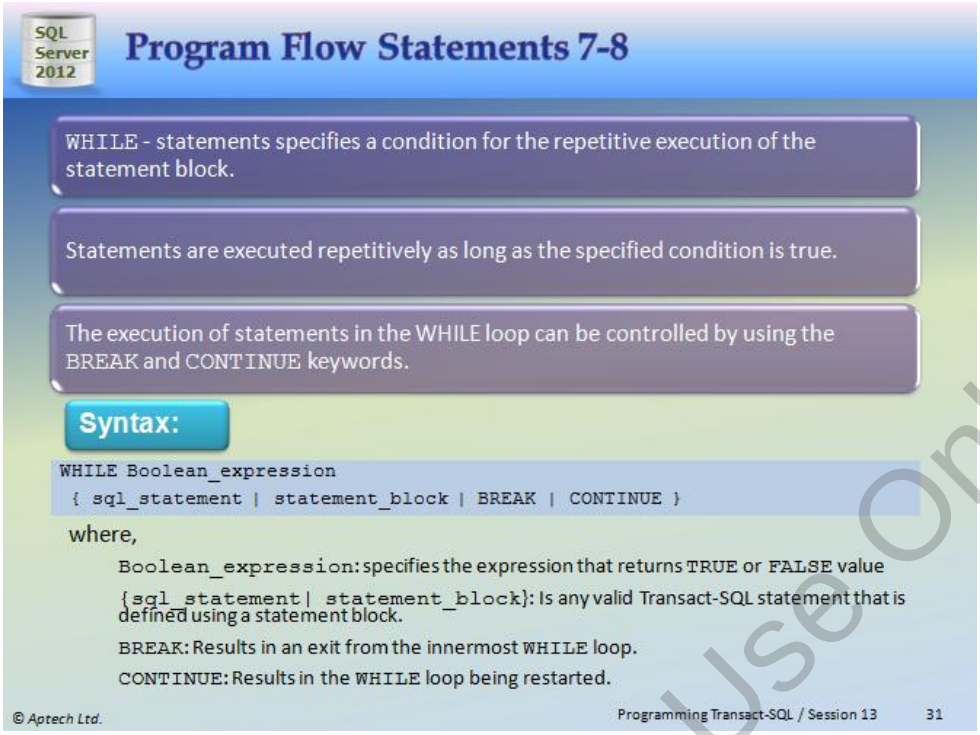


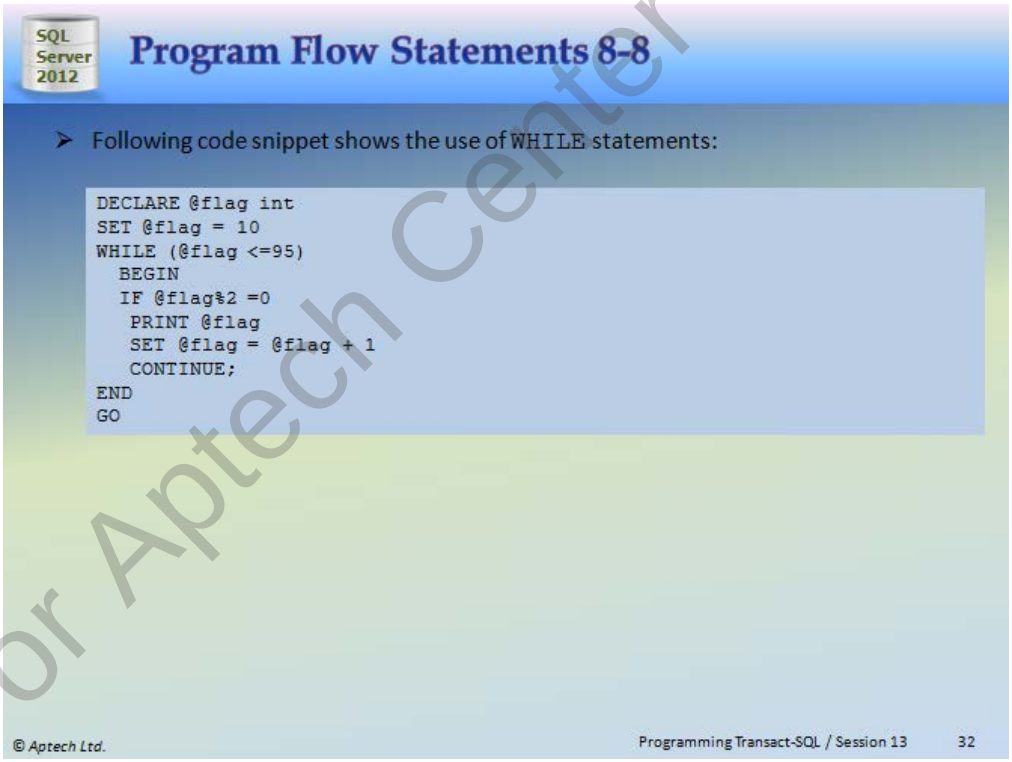












Using slides 25 to 32, explain the program flow statements.

There are different types of program flow statements and functions supported by Transact-SQL. Some of these are as follows:

- **Transact-SQL Control-of-Flow language**
  Control-of-flow language determines the execution flow of Transact-SQL statements, statement blocks, user-defined functions, and stored procedures. By default, Transact-SQL statements are executed sequentially, in the order they occur. Control-of-flow

language elements allow statements to be executed in a particular order, to be related to each other, and made interdependent using constructs similar to programming languages.

Explain the list of some of the Transact-SQL control-of-flow language keywords.

- **BEGIN….END**
  The BEGIN…END statements surround a series of Transact-SQL statements so that a group of Transact-SQL statements is executed.

  Explain the basic syntax for the BEGIN and END statement and also the code snippet which shows the use of BEGIN and END statements on slides 27 and 28.

  In this code snippet, BEGIN and END statements describe a sequence of Transact-SQL statements that are executed together. Suppose the BEGIN and END are not included, then the ROLLBACK TRANSACTION statements will execute and both the PRINT messages will be displayed.

  Mention ROLLBACK TRANSACTION is used to erase all data modifications made from the start of the transaction or to a savepoint. It also frees resources held by the transaction. COMMIT TRANSACTION makes all data modifications performed since the start of the transaction a permanent part of the database, frees the resources held by the transaction, and decrements @@TRANCOUNT to 0.
- **IF…ELSE**
  The IF…ELSE statement enforces a condition on the execution of a Transact-SQL statement. The Transact-SQL statement is followed with the IF keyword and the condition executes only if the condition is satisfied and returns TRUE. The ELSE keyword is an optional Transact-SQL statement that executes only when the IF condition is not satisfied and returns FALSE.

  Explain the syntax for the IF…ELSE statement and the code snippet which shows the use of IF…ELSE statements using slides 29 and 30.

  In this code snippet, the IF…ELSE statement is used to form a conditional statement. First, a variable, @ListPrice is defined and a query is created to return the maximum list price of the product category, Mountain Bikes. Then, this price is compared with a value of 3000 to determine if products can be purchased for an amount less than 3000. If yes, an appropriate message is printed using the first PRINT statement. If not, then the second PRINT statement executes.
- **WHILE**
  The WHILE statement specifies a condition for the repetitive execution of the statement block. The statements are executed repetitively as long as the specified condition is true. The execution of statements in the WHILE loop can be controlled by using the BREAK and CONTINUE keywords.

  Explain the syntax for the WHILE statement and the code snippet which shows the use of WHILE statement on slides 31 and 32.

Using this code snippet, all the even numbers beginning from 10 until 95 are displayed. This is achieved using a WHILE loop along with an IF statement. Similarly, a WHILE loop can also be used with queries and other Transact-SQL statements.

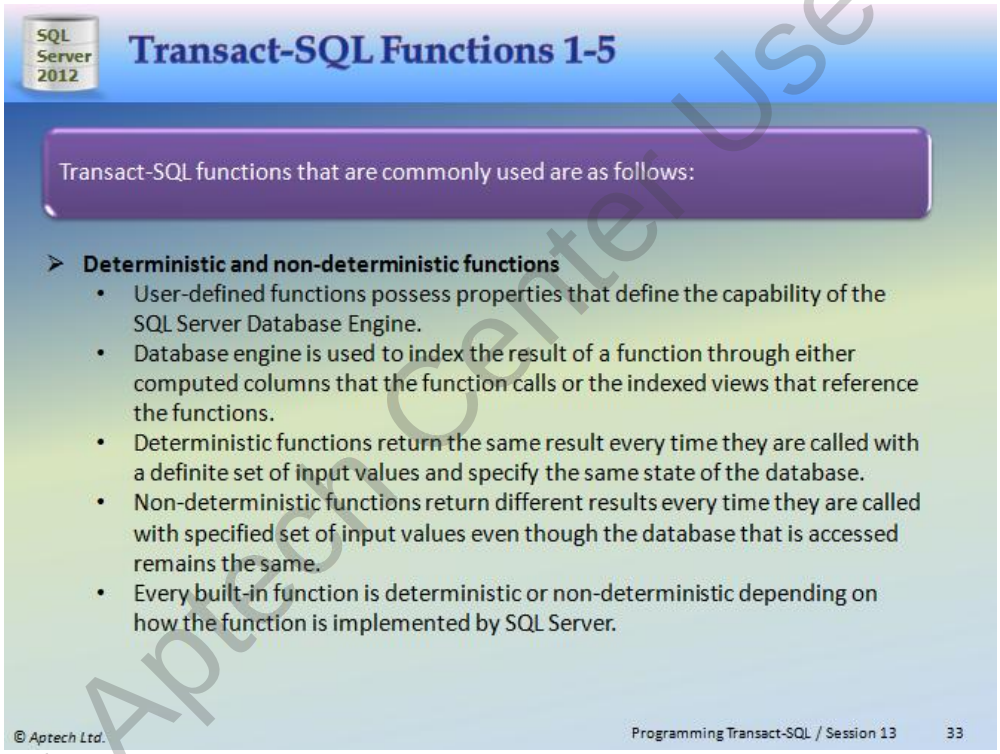
In-Class Question:

What is used to come out of loop without its full execution?

Answer:
BREAK keyword is used to come out of loop without its full execution.

Transact-SQL Functions

Slides 33 to 37



Transact-SQL Functions 2-5

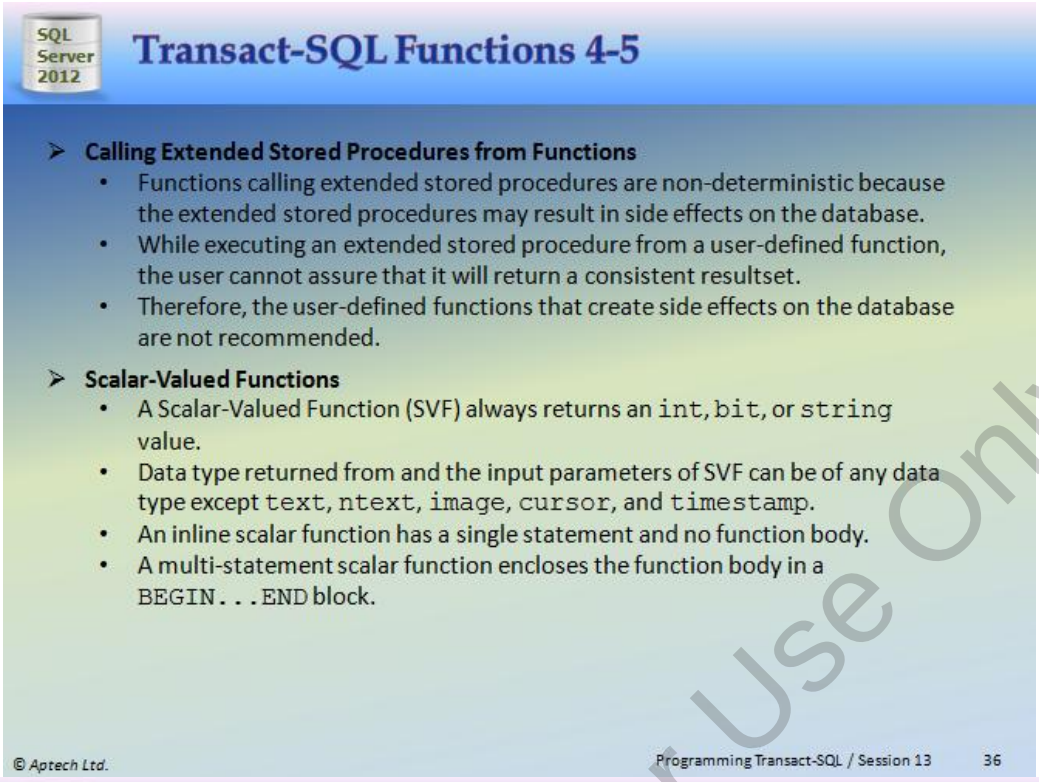
- Following table lists some deterministic and non-deterministic built-in functions:

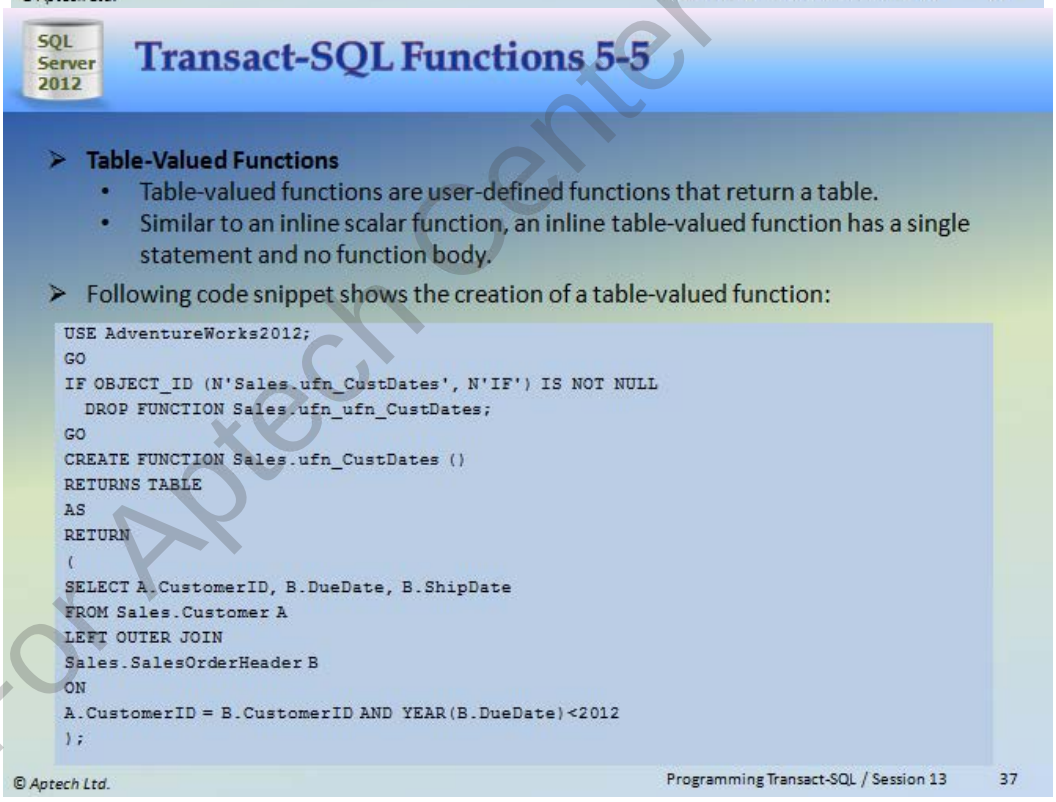
Deterministic Built-in Functions	Non-Deterministic Built-in Functions
POWER	@@TOTAL_WRITE
ROUND	CURRENT_TIMESTAMP
RADIANS	GETDATE
EXP	GETUTCDATE
FLOOR	GET_TRANSMISSION_STATUS
SQUARE	NEWID
SQRT	NEWSEQUENTIALID
LOG	@@CONNECTIONS
YEAR	@@CPU_BUSY
ABS	@@DBTS
ASIN	@@IDLE
ACOS	@@IOBUSY
SIGN	@@PACK_RECEIVED

Transact-SQL Functions 3-5

- Following table lists some functions that are not always deterministic but you can use them in indexed views if they are given in a deterministic manner:

Function	Description
CONVERT	Is deterministic only if one of these conditions exists: → Has an sql_variant source type. → Has an sql_variant target type and source type is non-deterministic. → Has its source or target type as smalldatetime or datetime, has the other source or target type as a character string, and has a non-deterministic style specified. The style parameter must be a constant to be deterministic.
CAST	Is deterministic only if it is used with smalldatetime, sql_variant, or datetime.
ISDATE	Is deterministic unless used with the CONVERT function, the CONVERT style parameter is specified, and style is not equal to 0, 100, 9, or 109.
CHECKSUM	Is deterministic, with the exception of CHECKSUM(*).





Using slides 33 to 37, explain the Transact-SQL functions.

The Transact-SQL functions that are commonly used are as follows:

Deterministic and non-deterministic functions

User-defined functions possess properties that define the capability of the SQL Server Database Engine. The Database engine is used to index the result of a function through

either computed columns that the function calls or the indexed views that reference the functions. One such property is the determinism of a function.

Deterministic functions return the same result every time they are called with a definite set of input values and specify the same state of the database. Non-deterministic functions return different results every time they are called with specified set of input values even though the database that is accessed remains the same.

For example, if a user calls the DAY() function on a particular column, it always returns the numerical day for the date parameter passed in. However, if the user calls the DATENAME() function, the output cannot be predicted since it may be different each time, depending on what part of the date is passed as input. Thus, here, DAY() is a deterministic function, while DATENAME() is a non-deterministic function. Similarly, RAND (without any seed), @@TIMETICKS, @@CONNECTIONS, and GETDATE() are non-deterministic.

Users cannot influence the determinism of built-in functions. Every built-in function is deterministic or non-deterministic depending on how the function is implemented by SQL Server.

Explain the list of some of the deterministic and non-deterministic built-in functions shown on slide 34.

There are also some functions that are not always deterministic but you can use them in indexed views if they are given in a deterministic manner. Table lists some of these functions shown on slide 35.

Explain the calling extended stored procedures from functions using slide 36.
Functions calling extended stored procedures are non-deterministic because the extended stored procedures may result in side effects on the database. Changes made to the global state of a database such as a change to an external resource, or updates to a table, file, or a network are called side effects. For example, sending an e-mail, or deleting a file can cause side effects. While executing an extended stored procedure from a user-defined function, the user cannot assure that it will return a consistent resultset.
Therefore, the user-defined functions that create side effects on the database are not recommended.

Explain the Scalar-Valued functions.
A Scalar-Valued Function (SVF) always returns an int, bit, or string value. The data type returned from and the input parameters of SVF can be of any data type except text, ntext, image, cursor, and timestamp. An inline scalar function has a single statement and no function body. A multi-statement scalar function encloses the function body in a BEGIN...END block.

Explain the Table-Valued functions using slide 37.
Table-valued functions are user-defined functions that return a table. Similar to an inline scalar function, an inline table-valued function has a single statement and no function body.
Explain the code snippet that shows the creation of a table-valued function.

Here, an inline table-valued function defines a left outer join between the tables Sales.Customer and Sales.SalesOrderHeader.
The tables are joined based on customer ids. In this case, all records from the left table and only matching records from the right table are returned. The resultant table is then returned from the table-valued function.

The function is invoked as shown in code snippet on slide 37.

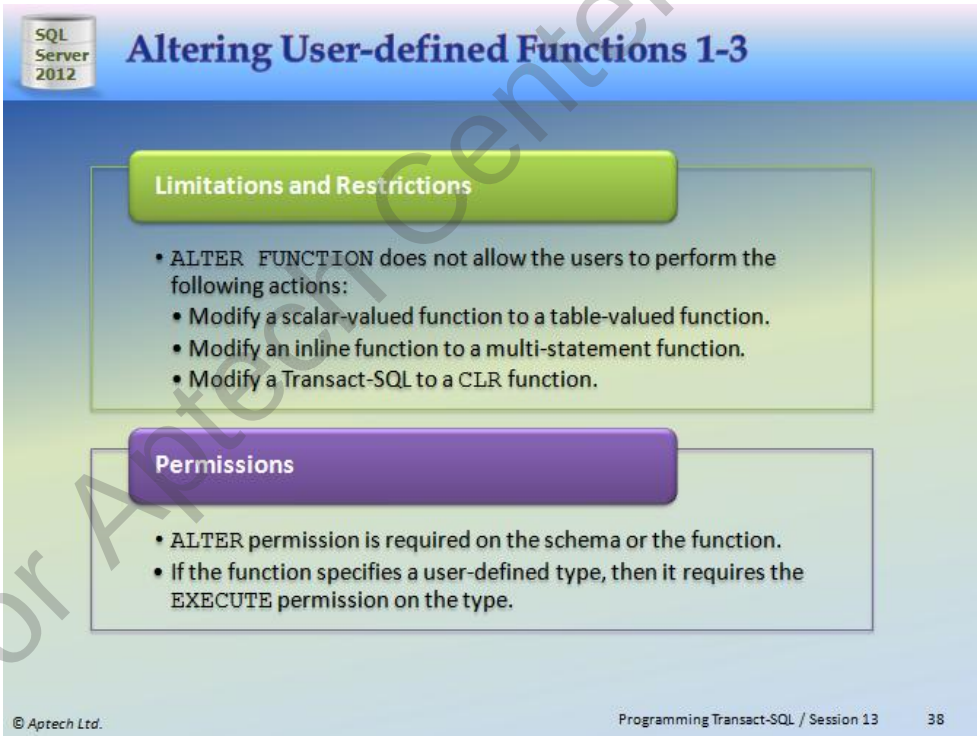
In-Class Question:

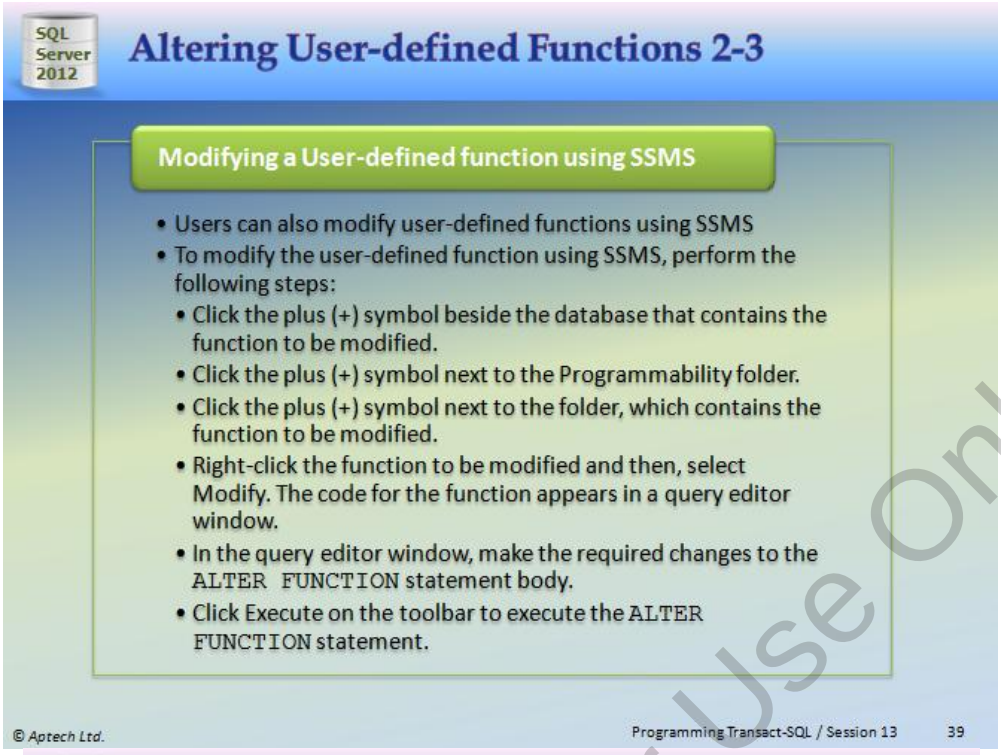
What type of functions return a table?

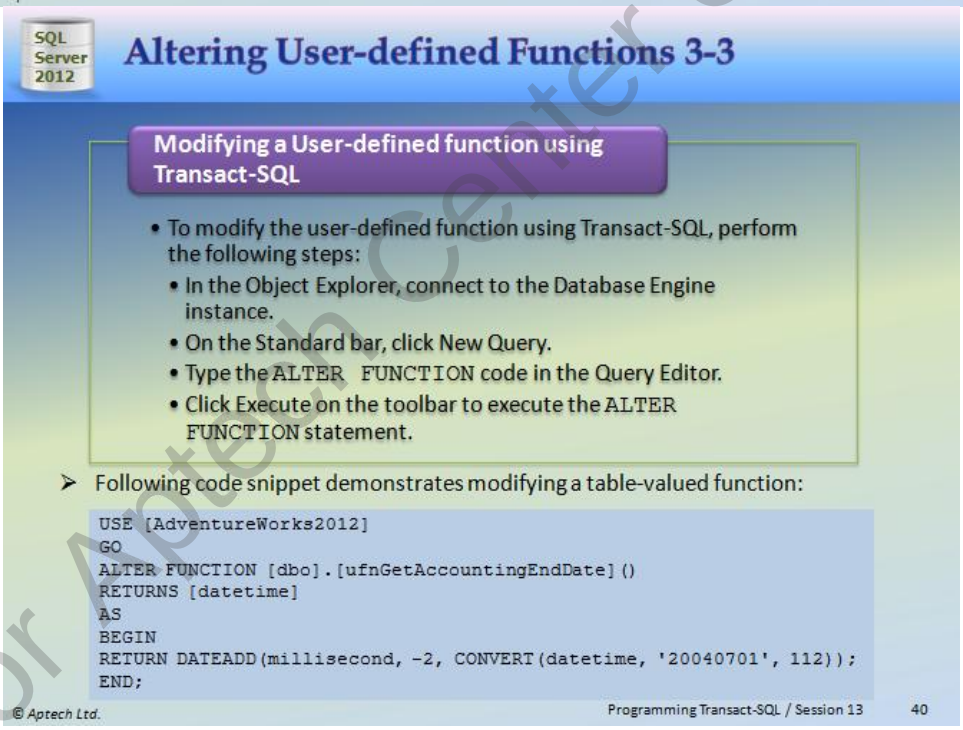
Answer:
Table-valued functions return a table.

Altering User-defined Functions

Slides 38 to 40







Using slides 38 to 40, explain how to alter user-defined functions.
Users can modify the user-defined functions in SQL Server 2012 by using the Transact-SQL or SSMS. Changing the user-defined functions does not modify the functions permissions, nor will it affect any stored procedures, triggers, or functions.

The ALTER FUNCTION does not allow the users to perform the following actions:

- Modify a scalar-valued function to a table-valued function.
- Modify an inline function to a multi-statement function.
- Modify a Transact-SQL to a CLR function.

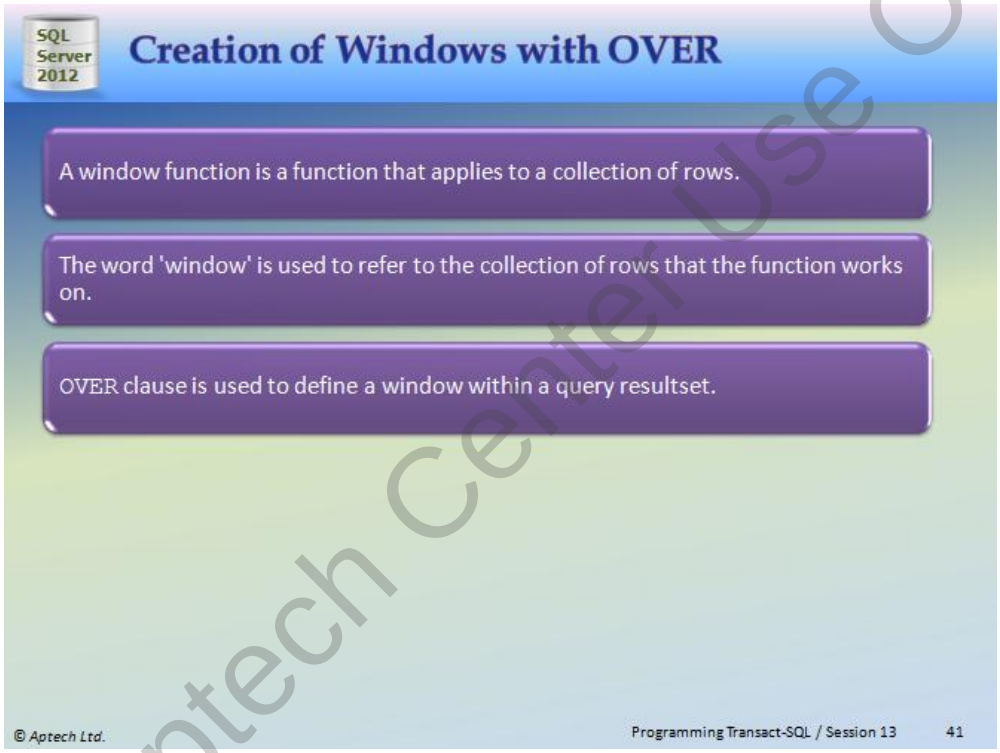
The ALTER permission is required on the schema or the function. If the function specifies a user-defined type, then it requires the EXECUTE permission on the type.

Users can also modify user-defined functions using SSMS. Explain the steps to modify user-defined function using SSMS using slide 39.

Also explain how to modify the user-defined function using Transact-SQL using slide 40.

Creation of Windows with OVER

Slide 41



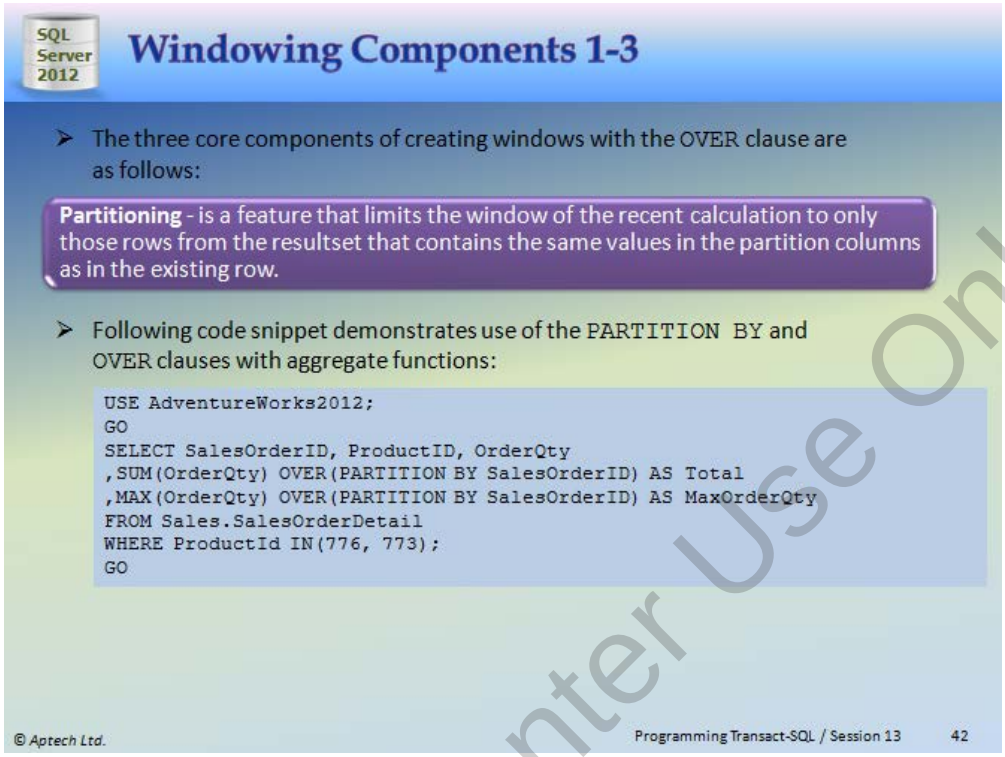
Using slide 41, explain how to create Windows with OVER.

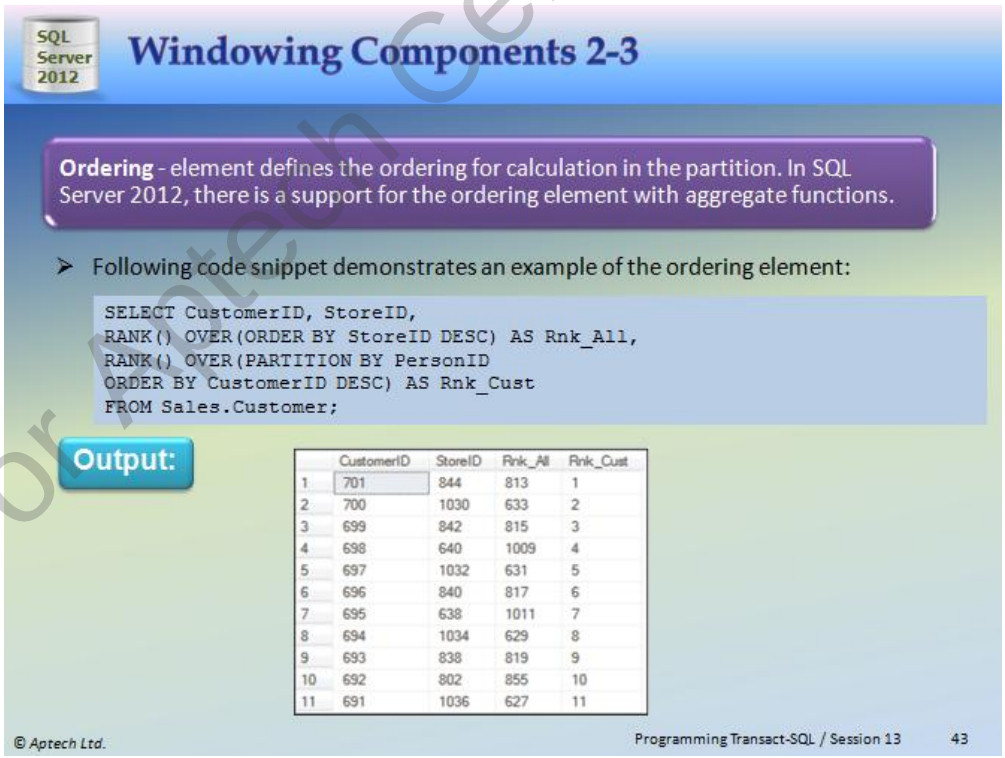
A window function is a function that applies to a collection of rows. The word 'window' is used to refer to the collection of rows that the function works on.

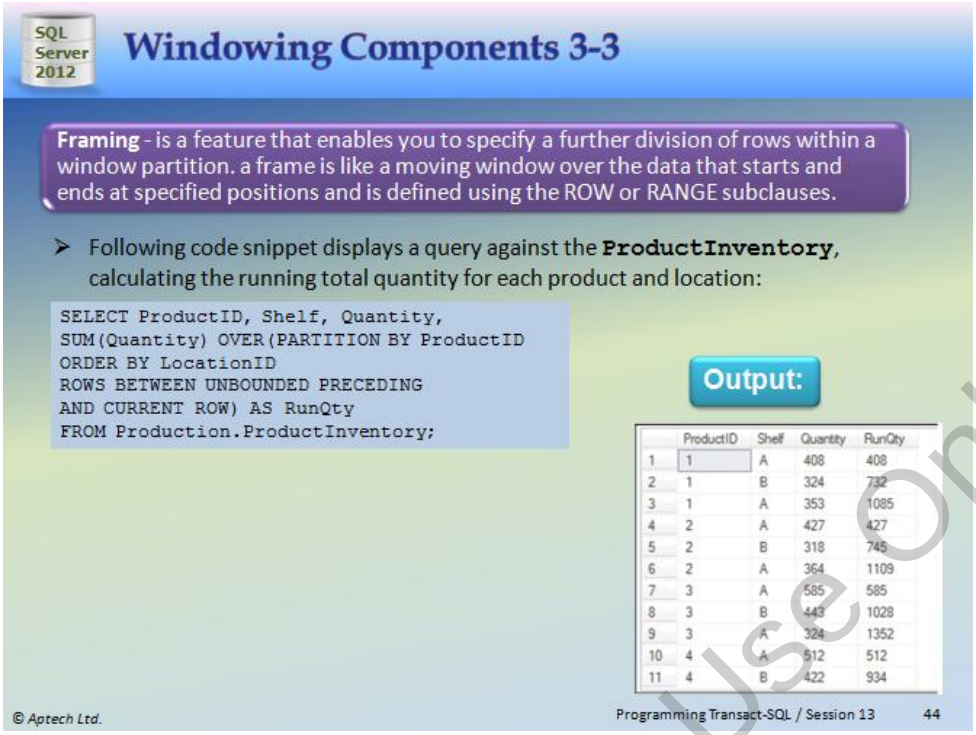
In Transact-SQL, the OVER clause is used to define a window within a query resultset. Using windows and the OVER clause with functions provides several advantages. For instance, they help to calculate aggregated values. They also enable row numbers in a resultset to be generated easily.

Windowing Components

Slides 42 to 44







Using slides 42 to 44, explain the Windowing components.

The three core components of creating windows with the OVER clause, are as follows:

- Partitioning
- Ordering
- Framing

Mention that, partitioning is a feature that limits the window of the recent calculation to only those rows from the resultset that contains the same values in the partition columns as in the existing row. It uses the PARTITION BY clause. Code snippet shown on slide 42 demonstrates use of the PARTITION BY and OVER clauses with aggregate functions. Here, using the OVER clause proves to be better efficient than using subqueries to calculate the aggregate values.

Using slide 43, explain the component ordering.
The ordering element defines the ordering for calculation in the partition. In a standard SQL ordering element, all functions are supported. Earlier, SQL Server had no support for the ordering elements with aggregate functions as it only supported partitioning. In SQL Server 2012, there is a support for the ordering element with aggregate functions. The ordering element has different meaning to some extent for different function categories. With ranking functions, ordering is spontaneous.

Explain the code snippet which demonstrates an example of the ordering element shown on slide 43.

This code snippet makes use of the RANK() function which returns the rank of each row in the partition of a resultset. The rank of a row is determined by adding 1 to the number of ranks that come before the specified row. For example, while using descending ordering,

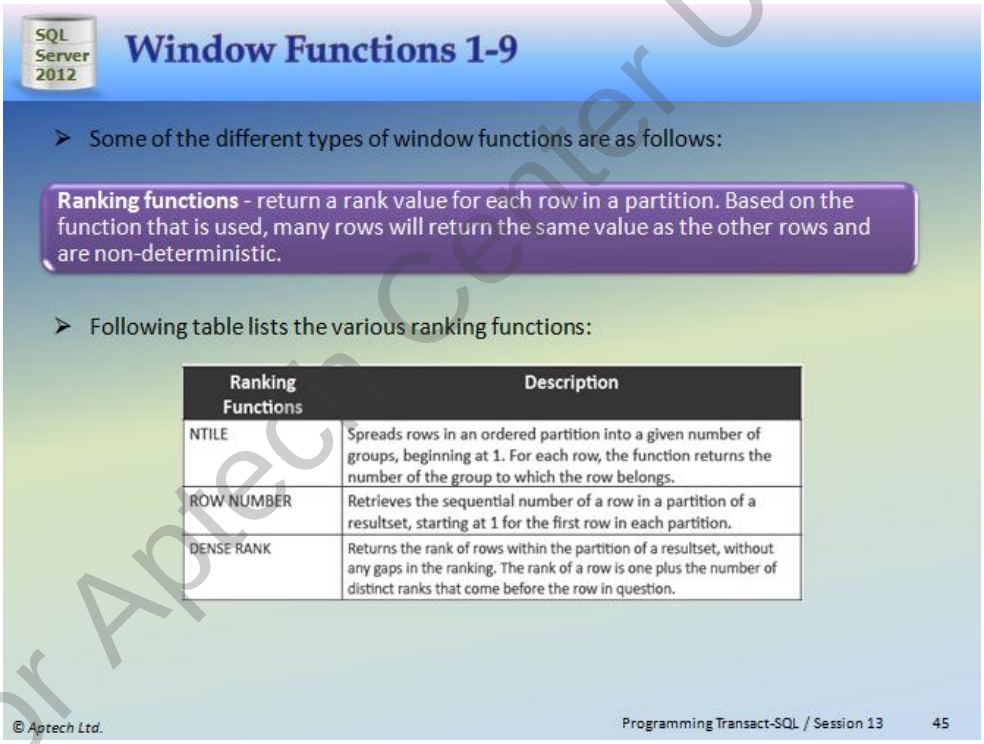
the RANK() function returns one more than the number of rows in the respective partition that has a greater ordering value than the specified one.

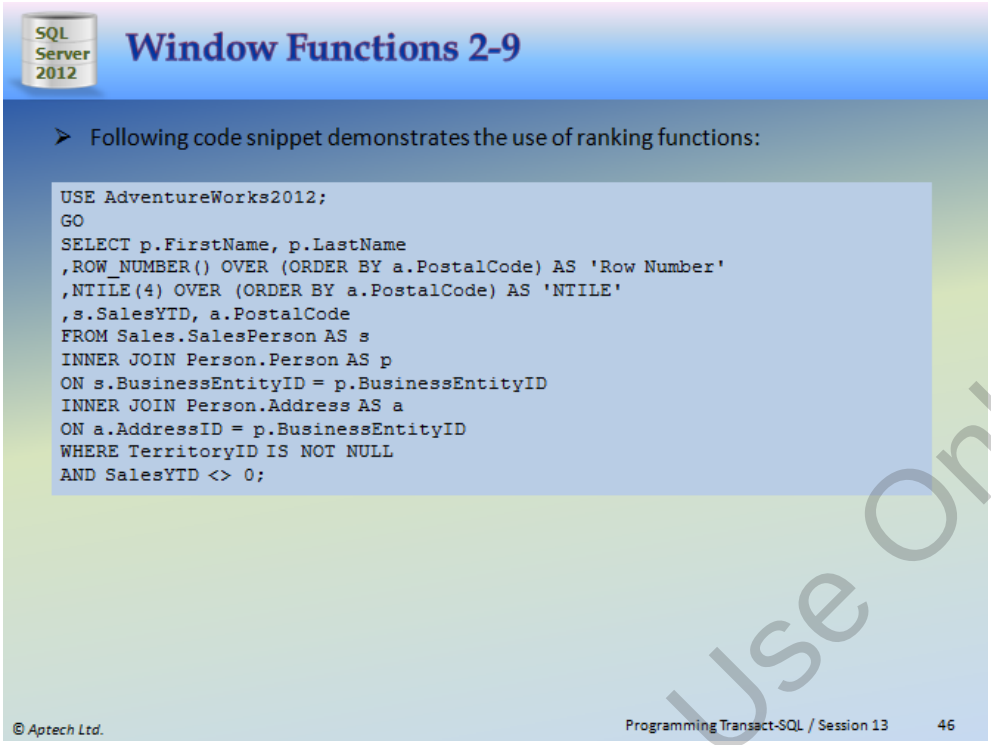
Figure on slide 43 displays the output of code snippet.

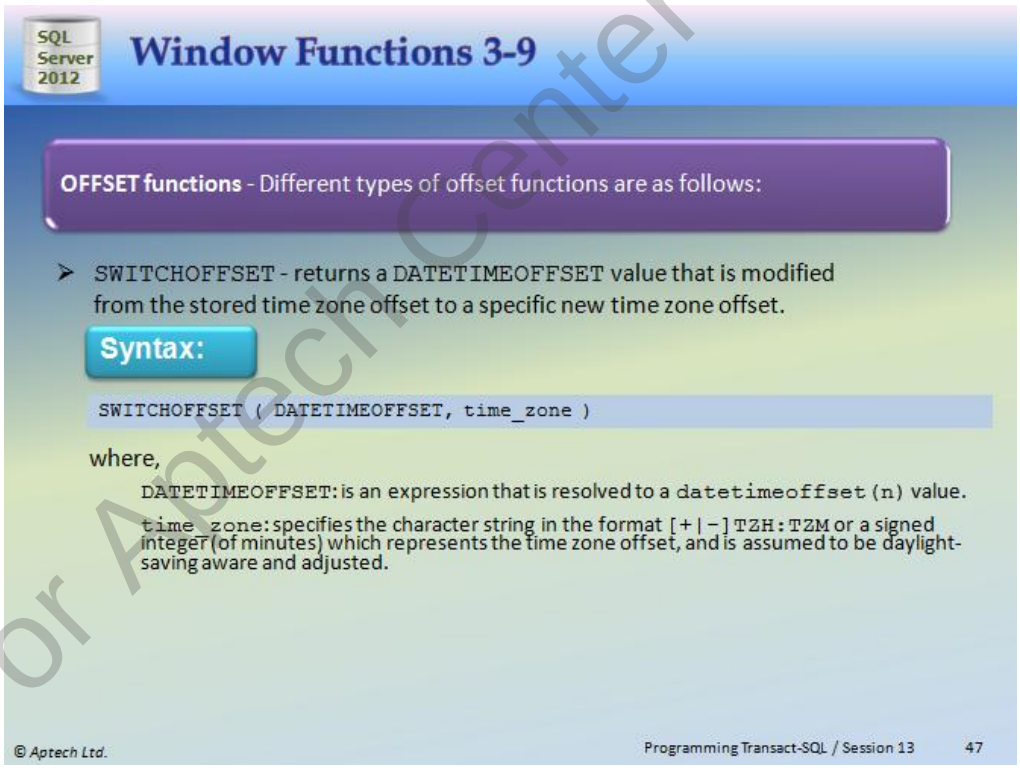
Using slide 44, explain framing component.
Framing is a feature that enables you to specify a further division of rows within a window partition. This is done by assigning upper and lower boundaries for the window frame that presents rows to the window function. In simple terms, a frame is similar to a moving window over the data that starts and ends at specified positions. Window frames can be defined using the ROW or RANGE subclauses and providing starting and ending boundaries. Explain the code snippet that displays a query against the ProductInventory, calculating the running total quantity for each product and location on slide 44.

Window Functions

Slides 45 to 53







Window Functions 4-9

- Following code snippet demonstrates the use of `SWITCHOFFSET` function:

```
CREATE TABLE Test
(
ColDatetimeoffset datetimeoffset
);
GO
INSERT INTO Test
VALUES ('1998-09-20 7:45:50.71345 -5:00');
GO
SELECT SWITCHOFFSET (ColDatetimeoffset, '-08:00')
FROM Test;
GO
--Returns: 1998-09-20 04:45:50.7134500 -08:00
SELECT ColDatetimeoffset
FROM Test;
```

Output:

Results | Messages

	(No column name)
1	1998-09-20 04:45:50.7134500 -08:00

	ColDatetimeoffset
1	1998-09-20 07:45:50.7134500 -05:00

Window Functions 5-9

- `DATETIMEOFFSETFROMPARTS` – returns a `datetimeoffset` value for the specified date and time with specified precision and offset.

Syntax:

```
DATETIMEOFFSETFROMPARTS ( year, month, day, hour,
minute, seconds, fractions, hour_offset,
minute_offset, precision )
```

where,

- `year`: specifies the integer expression for a year.
- `month`: specifies the integer expression for a month.
- `day`: specifies the integer expression for a day.
- `hour`: specifies the integer expression for an hour.
- `minute`: specifies the integer expression for a minute.
- `seconds`: specifies the integer expression for a day.
- `fractions`: specifies the integer expression for fractions.
- `hour_offset`: specifies the integer expression for the hour portion of the time zone offset.
- `minute_offset`: specifies the integer expression for the minute portion of the time zone offset.
- `precision`: specifies the integer literal precision of the `datetimeoffset` value to be returned.

Window Functions 6-9

- Following code snippet demonstrates the use of DATETIMEOFFSETFROMPARTS function:

```
SELECT DATETIMEOFFSETFROMPARTS ( 2010, 12, 31, 14, 23, 23, 0, 12, 0,
7 )
AS Result;
```

Output:

Results | Messages

	Result
1	2010-12-31 14:23:23.0000000 +12:00

Window Functions 7-9

- SYSDATETIMEOFFSET – returns datetimeoffset(7) value which contains the date and time of the computer on which the instance of SQL Server is running.

Syntax:

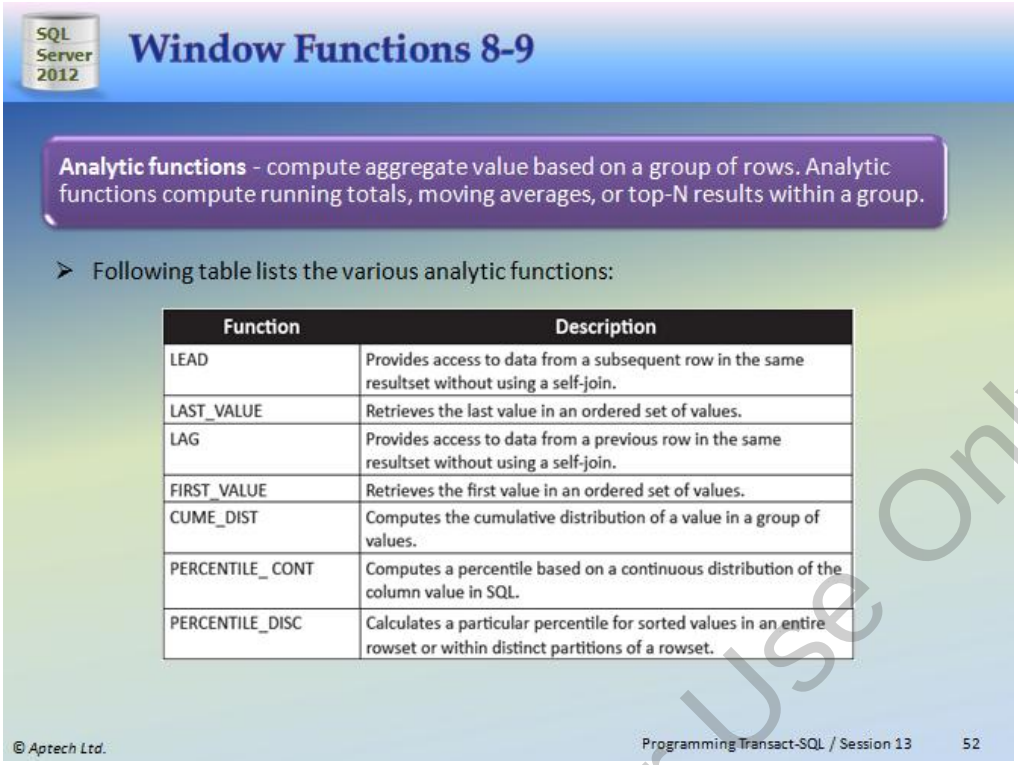
```
SYSDATETIMEOFFSET ()
```

- Following code snippet demonstrates the use of different formats used by the date and time functions:

```
SELECT SYSDATETIME() AS SYSDATETIME
,SYSDATETIMEOFFSET() AS SYSDATETIMEOFFSET
,SYSUTCDATETIME() AS SYSUTCDATETIME
```

Output:

	SYSDATETIME	SYSDATETIMEOFFSET	SYSUTCDATETIME
1	2013-02-08 16:08:16.6565247	2013-02-08 16:08:16.6565247 +05:30	2013-02-08 10:38:16.6565247

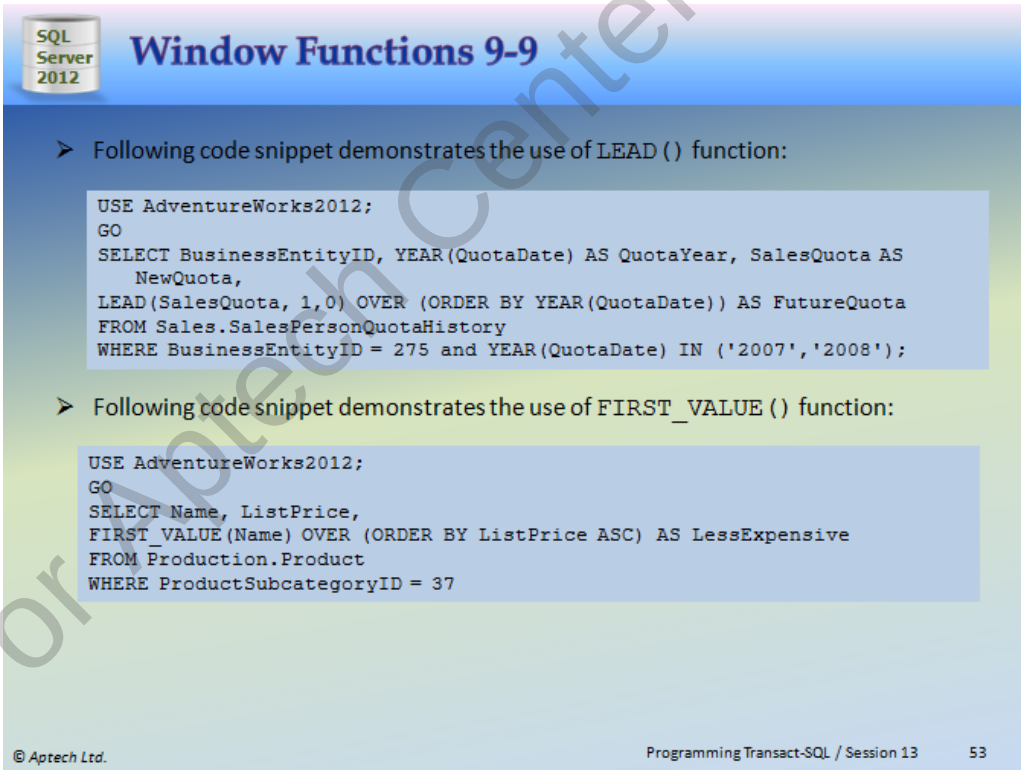


Window Functions 8-9

Analytic functions - compute aggregate value based on a group of rows. Analytic functions compute running totals, moving averages, or top-N results within a group.

- Following table lists the various analytic functions:

Function	Description
LEAD	Provides access to data from a subsequent row in the same resultset without using a self-join.
LAST_VALUE	Retrieves the last value in an ordered set of values.
LAG	Provides access to data from a previous row in the same resultset without using a self-join.
FIRST_VALUE	Retrieves the first value in an ordered set of values.
CUME_DIST	Computes the cumulative distribution of a value in a group of values.
PERCENTILE_ CONT	Computes a percentile based on a continuous distribution of the column value in SQL.
PERCENTILE_DISC	Calculates a particular percentile for sorted values in an entire rowset or within distinct partitions of a rowset.



Window Functions 9-9

- Following code snippet demonstrates the use of LEAD () function:

```
USE AdventureWorks2012;
GO
SELECT BusinessEntityID, YEAR(QuotaDate) AS QuotaYear, SalesQuota AS
   NewQuota,
LEAD(SalesQuota, 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS FutureQuota
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275 and YEAR(QuotaDate) IN ('2007','2008');
```

- Following code snippet demonstrates the use of FIRST_VALUE () function:

```
USE AdventureWorks2012;
GO
SELECT Name, ListPrice,
FIRST_VALUE(Name) OVER (ORDER BY ListPrice ASC) AS LessExpensive
FROM Production.Product
WHERE ProductSubcategoryID = 37
```


Using slides 45 to 53, explain the Window functions.

These functions return a rank value for each row in a partition. Based on the function that is used, many rows will return the same value as the other rows. Ranking functions are non-deterministic.

Explain the list of various ranking functions in table.

Explain the code snippet which demonstrates the use of ranking functions shown on slide 46.
The NTILE() function breaks a given input collection into N equal sized logical groups. To determine how many rows belong in each group, SQL Server has to determine the total number of rows in the input collection. The OVER clause decides the order of the rows when they have been divided into groups. It is possible to perform the grouping in one order and return the resultset in another order.

Explain the offset function using slides 46 and 47.
SWITCHOFFSET function returns a DATETIMEOFFSET value that is modified from the stored time zone offset to a specific new time zone offset.
Explain the syntax for the SWITCHOFFSET function given on slide 46.
Explain the code snippet on slide 47 which displays the use of SWITCHOFFSET function.
Figure on slide 47 displays the output of code snippet.

DATETIMEOFFSETFROMPARTS function returns a datetimeoffset value for the specified date and time with specified precision and offset. Explain the syntax for the same shown on slide 48.
Explain the code snippet that displays the use of DATETIMEOFFSETFROMPARTS function shown on slide 49.

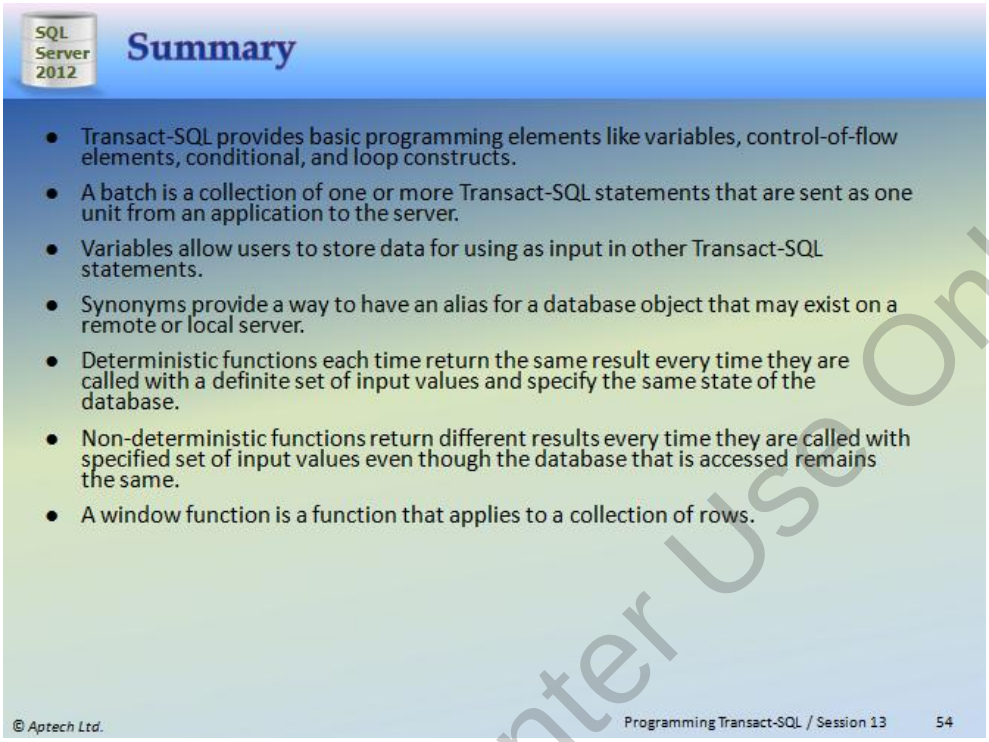
The code displays a datetimeoffset value for the given date and time with the specified precision and offset. Figure on slide 49 displays the output of code snippet.
SYSDATETIMEOFFSET functions returns datetimeoffset(7) value which contains the date and time of the computer on which the instance of SQL Server is running.
Explain the syntax using slide 50 for SYSDATETIMEOFFSET.

Explain the code snippet on slide 51 that displays the different formats used by the date and time functions.
SQL Server 2012 supports several analytic functions. These functions compute aggregate value based on a group of rows. Analytic functions compute running totals, moving averages, or top-N results within a group.

Explain the list of some of the analytic functions mentioned on slide 52.
In this code snippet, the LEAD() function is used to return the difference in the sales quotas for a particular employee over the subsequent years.
Explain the code snippet on slide 53 which demonstrates the use of FIRST_VALUE() function.
In this code snippet, the FIRST_VALUE() function compares products in the product category 37 and returns the product name that is less expensive.

Summarize Session

Slide 54



Using slide 54, summarize the session. End the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- Transact-SQL provides basic programming elements such as variables, control-of-flow elements, conditional, and loop constructs.
- A batch is a collection of one or more Transact-SQL statements that are sent as one unit from an application to the server.
- Variables allow users to store data for using as input in other Transact-SQL statements.
- Synonyms provide a way to have an alias for a database object that may exist on a remote or local server.
- Deterministic functions each time return the same result every time they are called with a definite set of input values and specify the same state of the database.
- Non-deterministic functions return different results every time they are called with specified set of input values even though the database that is accessed remains the same.
- A window function is a function that applies to a collection of rows.

13.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Transactions topic offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 14 – Transactions

14.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

14.1.1 Objectives

By the end of this session, the learners will be able to:

- Define and describe transactions
- Explain the procedure to implement transactions
- Explain the process of controlling transactions
- Explain the steps to mark a transaction
- Distinguish between implicit and explicit transactions
- Explain isolation levels
- Explain the scope and different types of locks
- Explain transaction management

14.1.2 Teaching Skills

To teach this session, you should be well-versed with the programming with the types of transactions and the procedure to implement these transactions. Along with this also prepare yourself with the process to control and mark a transaction, and lists the differences between the implicit and explicit transactions.
The session also covers the isolation levels, scope, different types of locks, and transaction management.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

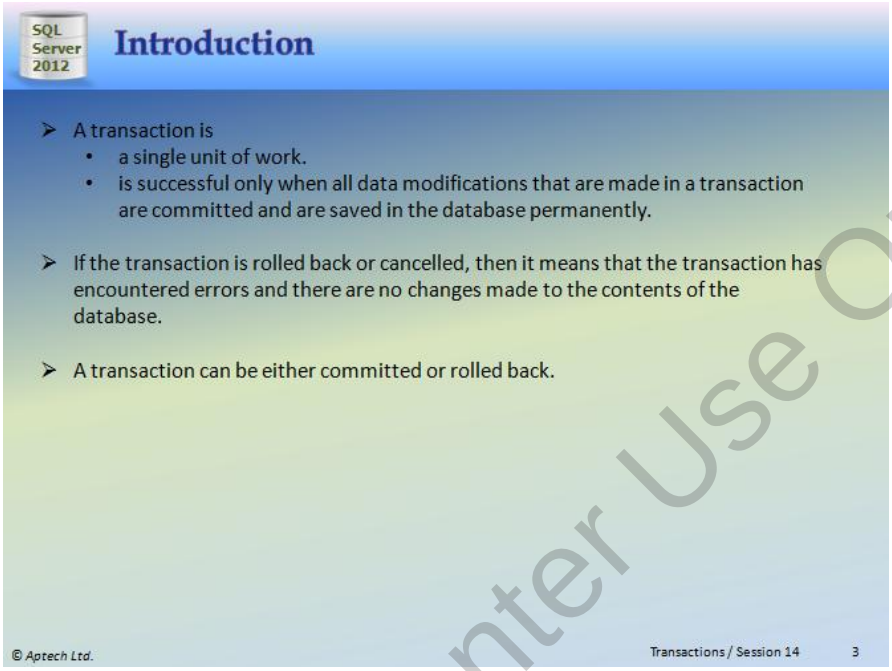
In-Class Activities:
Follow the order given here during In-Class activities.

Overview of the Session:
Then give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces the types of transactions and the procedure to implement these transactions. They will also know about the process to control and mark a transaction, and lists the differences between the implicit and explicit transactions. They will also know about the isolation levels, scope, different types of locks, and transaction management.

14.2 In-Class Explanations

Introduction

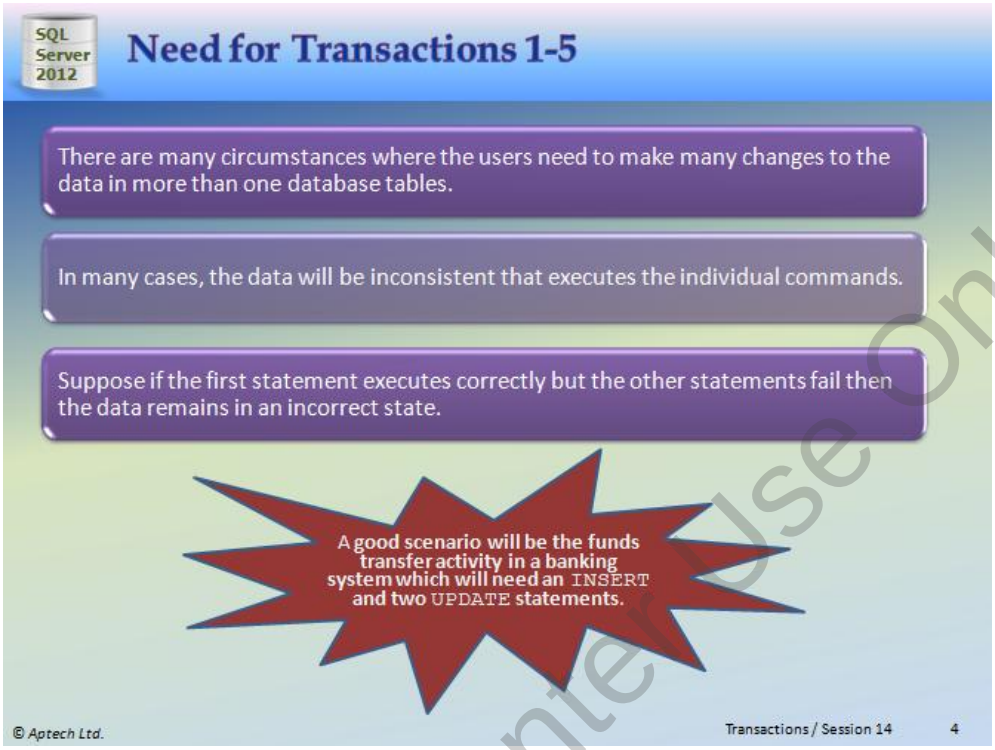
Slide 3

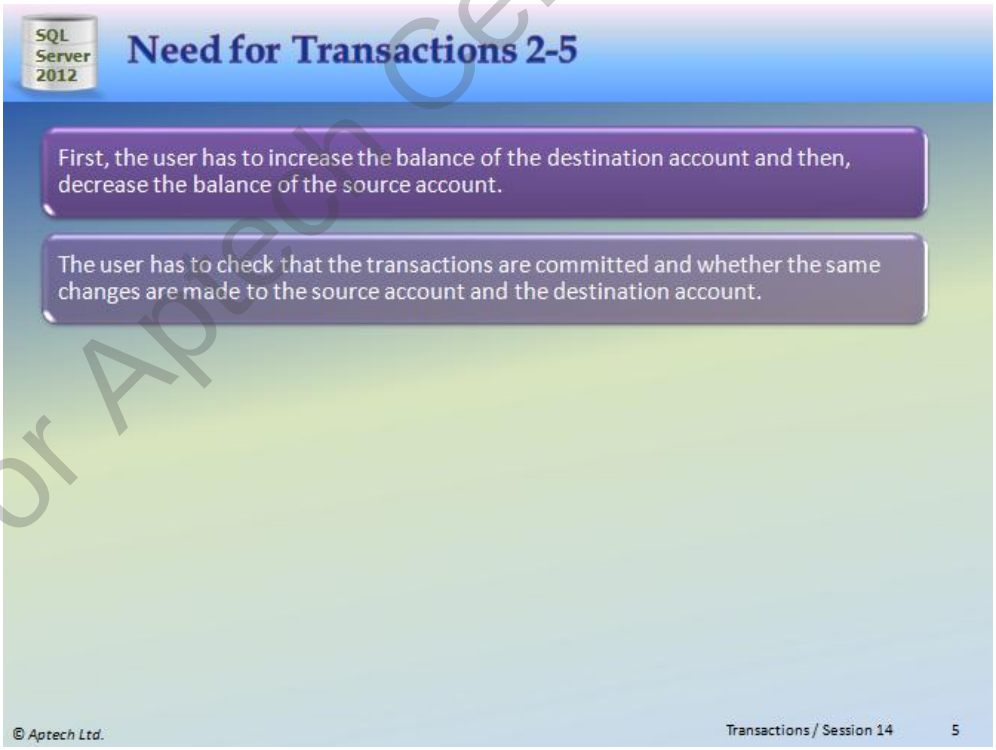


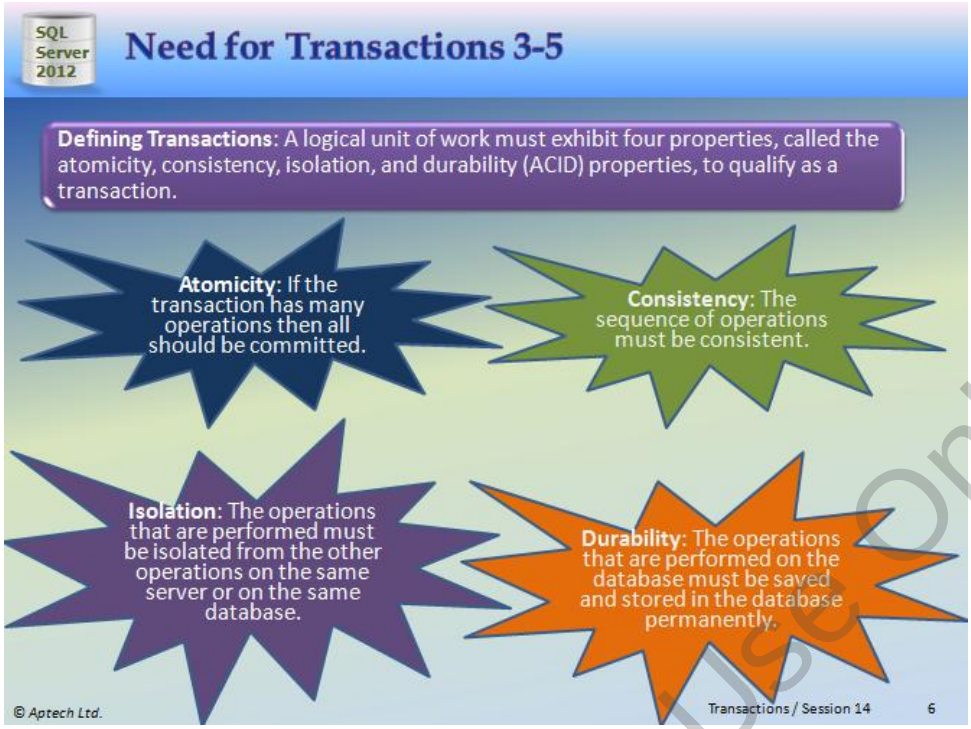
Using slide 3, explain the introduction of transaction.
A transaction is a single unit of work. A transaction is successful only when all data modifications that are made in a transaction are committed and are saved in the database permanently. If the transaction is rolled back or cancelled, then it means that the transaction has encountered errors and there are no changes made to the contents of the database. Hence, a transaction can be either committed or rolled back.

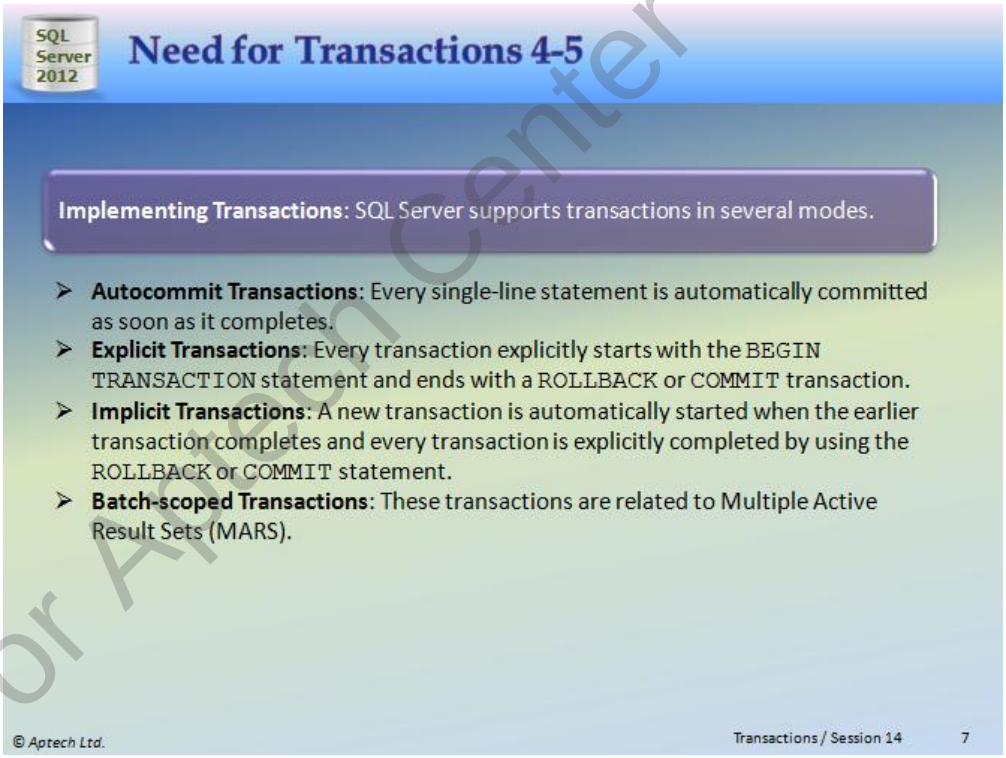
Need for Transactions

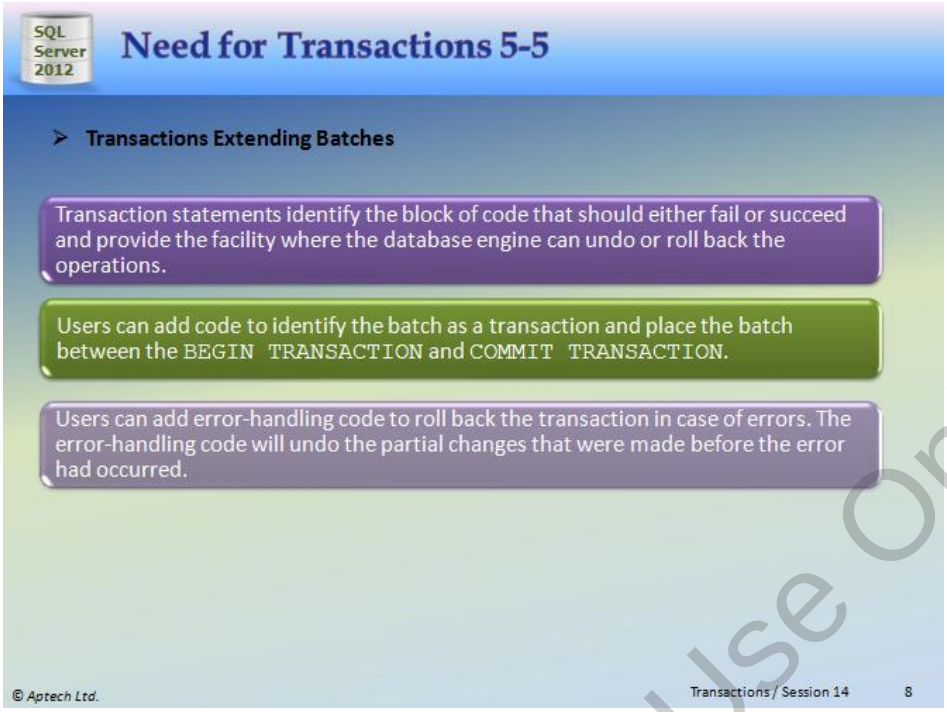
Slides 4 to 8











Using slides 4 to 8, explain the need for transactions.
There are many circumstances where the users need to make many changes to the data in more than one database tables. In many cases, the data will be inconsistent that executes the individual commands. Suppose if the first statement executes correctly but the other statements fail, then the data remains in an incorrect state.

For example, a good scenario will be the funds transfer activity in a banking system. The transfer of funds will need an INSERT and two UPDATE statements. First, the user has to increase the balance of the destination account and then, decrease the balance of the source account. The user has to check that the transactions are committed and whether the same changes are made to the source account and the destination account.

A logical unit of work must exhibit four properties, called the atomicity, consistency, isolation, and durability (ACID) properties, to qualify as a transaction.

- **Atomicity**: If the transaction has many operations, then all should be committed. If any of the operation in the group fails, then it should be rolled back.
- **Consistency**: The sequence of operations must be consistent.
- **Isolation**: The operations that are performed must be isolated from the other operations on the same server or on the same database.
- **Durability**: The operations that are performed on the database must be saved and stored in the database permanently.

SQL Server supports transactions in several modes are shown on slide 7. Some of these modes are as follows:

- **Autocommit Transactions**: Every single-line statement is automatically committed as soon as it completes. In this mode, one does not need to write any specific statements to start and end the transactions. It is the default mode for SQL Server Database Engine.
- **Transactions**: Every transaction explicitly starts with the BEGIN TRANSACTION statement and ends with a ROLLBACK or COMMIT transaction.

- **Implicit Transactions**: A new transaction is automatically started when the earlier transaction completes and every transaction is explicitly completed by using the ROLLBACK or COMMIT statement.
- **Batch-scoped Transactions**: These transactions are related to Multiple Active Result Sets (MARS). Any implicit or explicit transaction that starts in a MARS session is a batch-scoped transaction. A batch-scoped transaction that is rolled back when a batch completes automatically is rolled back by SQL Server.

Mention the transaction statements identify the block of code that should either fail or succeed and provide the facility where the database engine can undo or roll back the operations. The errors that encounter during the execution of simple batch have the possibility of partial success, which is not a desired result. This also led to inconsistencies in the tables and databases. To overcome this, users can add code to identify the batch as a transaction and place the batch between the BEGIN TRANSACTION and COMMIT TRANSACTION. Users can add error-handling code to roll back the transaction in case of errors. The error-handling code will undo the partial changes that were made before the error had occurred. This way, inconsistencies in the tables and databases can be prevented.

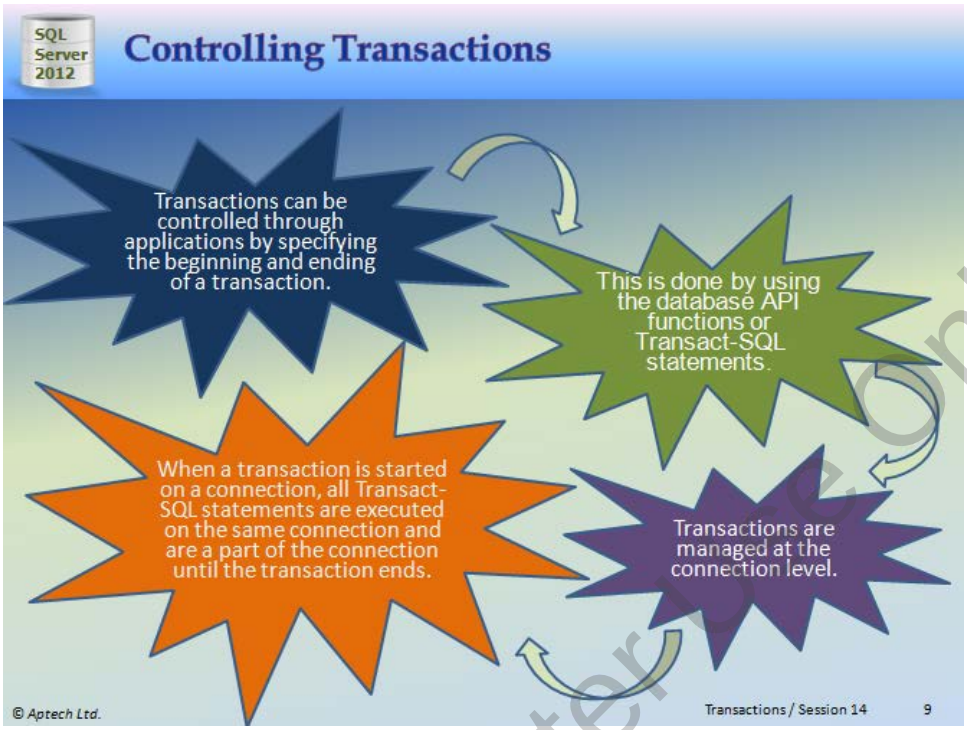
In-Class Question:

 What does isolation indicates in ACID properties?

Answer:
The operations that are performed must be isolated from the other operations on the same server or on the same database is indicated by isolation in ACID properties

Controlling Transactions

Slide 9

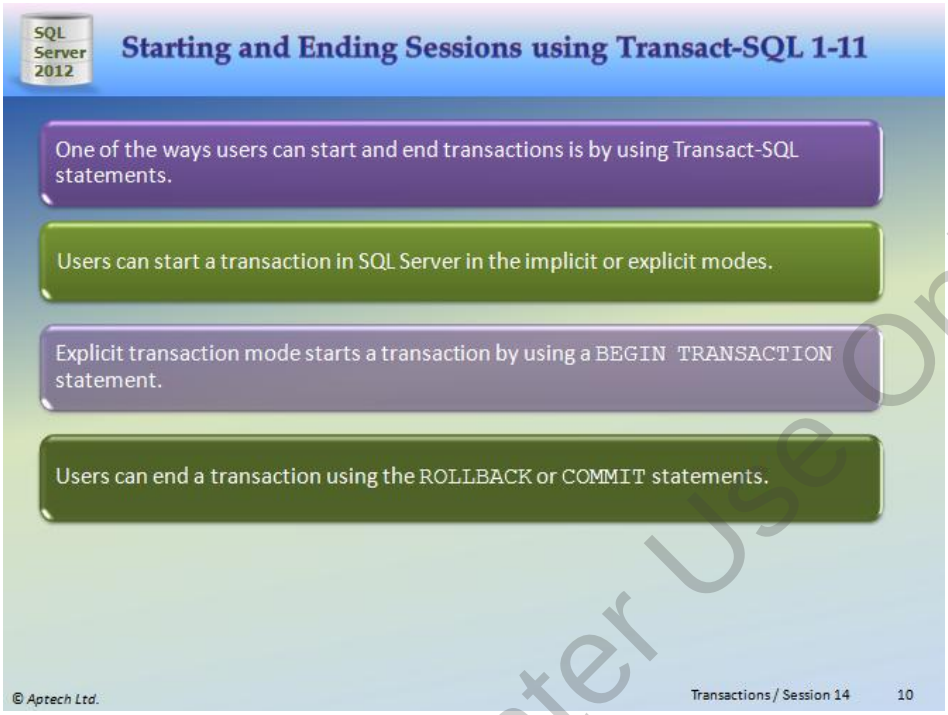


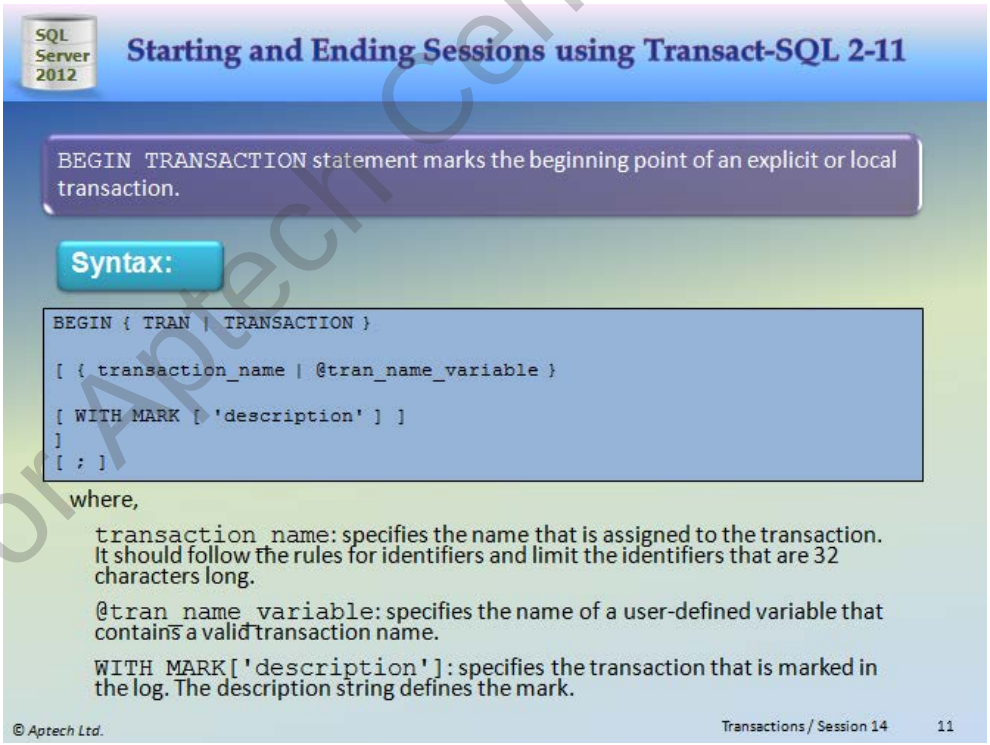
Using slide 9, explain how to control a transaction.
Transactions can be controlled through applications by specifying the beginning and ending of a transaction. This is done by using the database API functions or Transact-SQL statements.

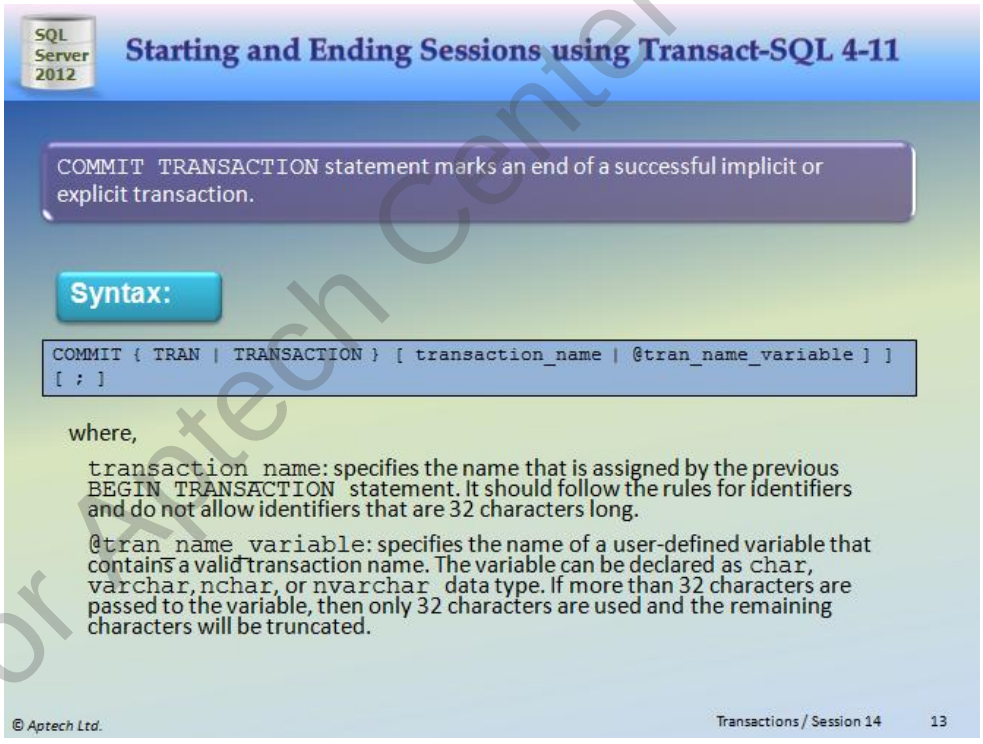
Transactions are managed at the connection level, by default. When a transaction is started on a connection, all Transact-SQL statements are executed on the same connection and are a part of the connection until the transaction ends.

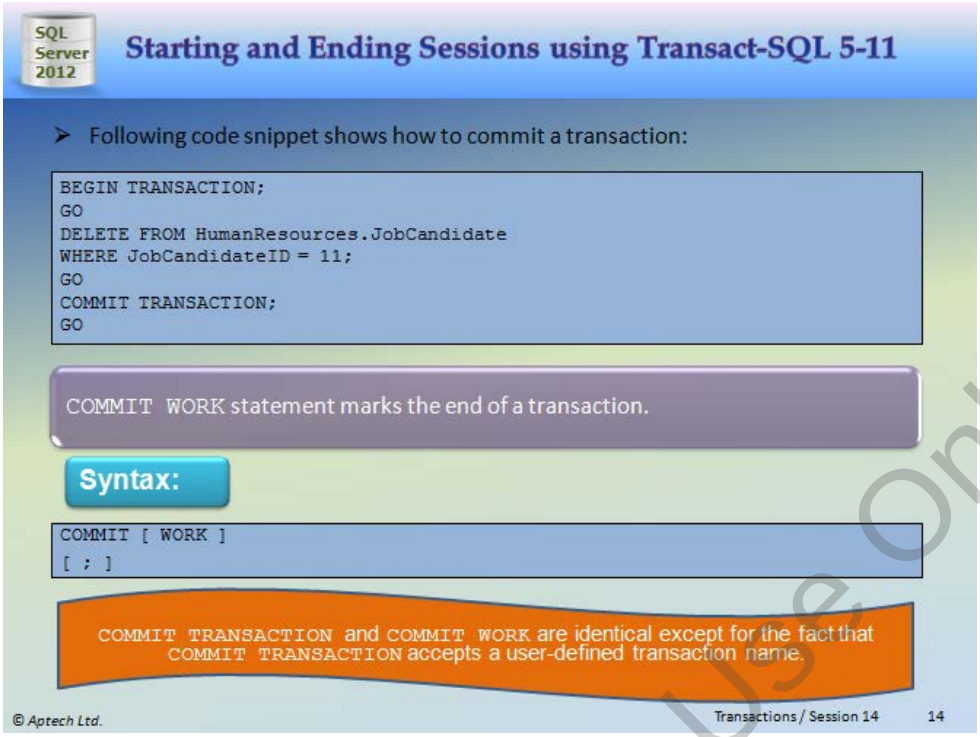
Starting and Ending a Session Using Transact-SQL

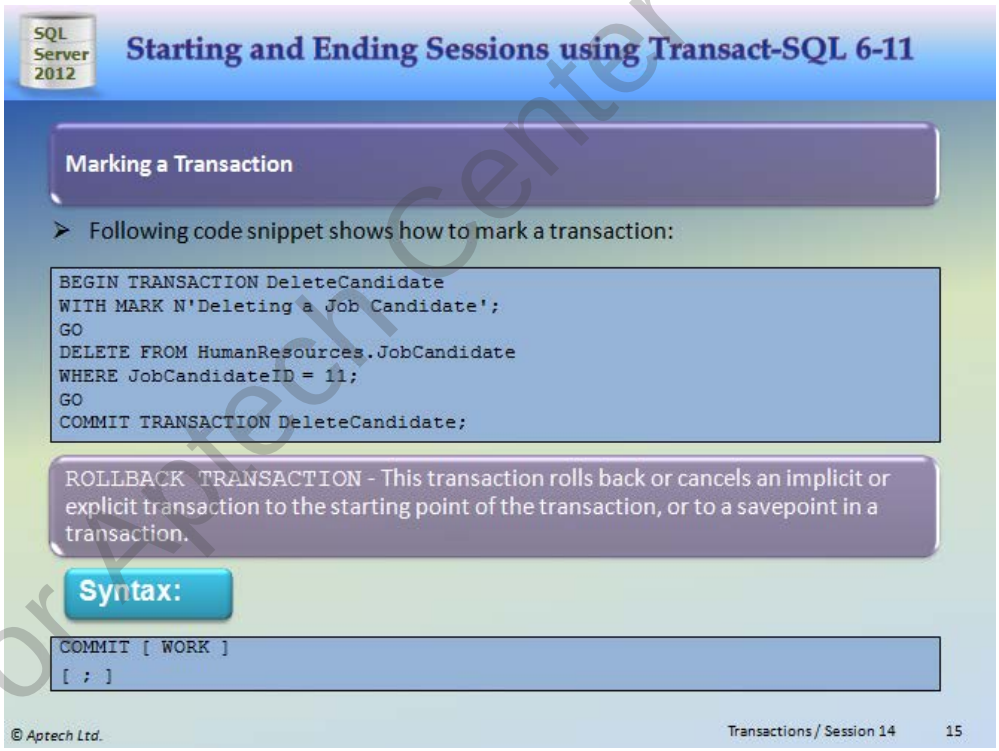
Slides 10 to 20

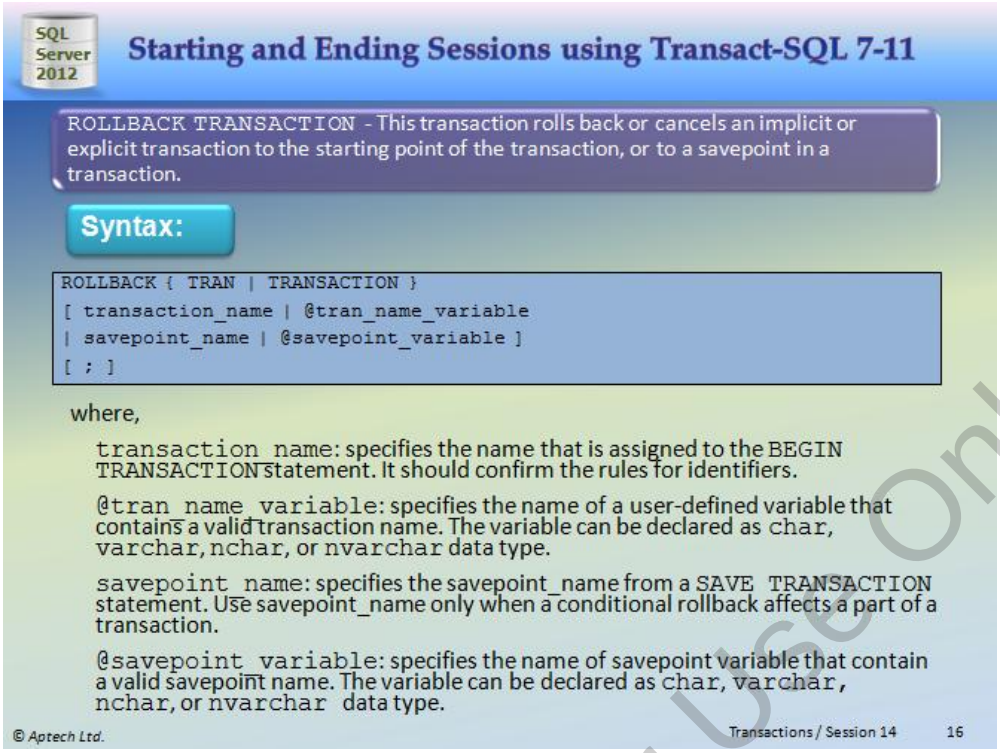


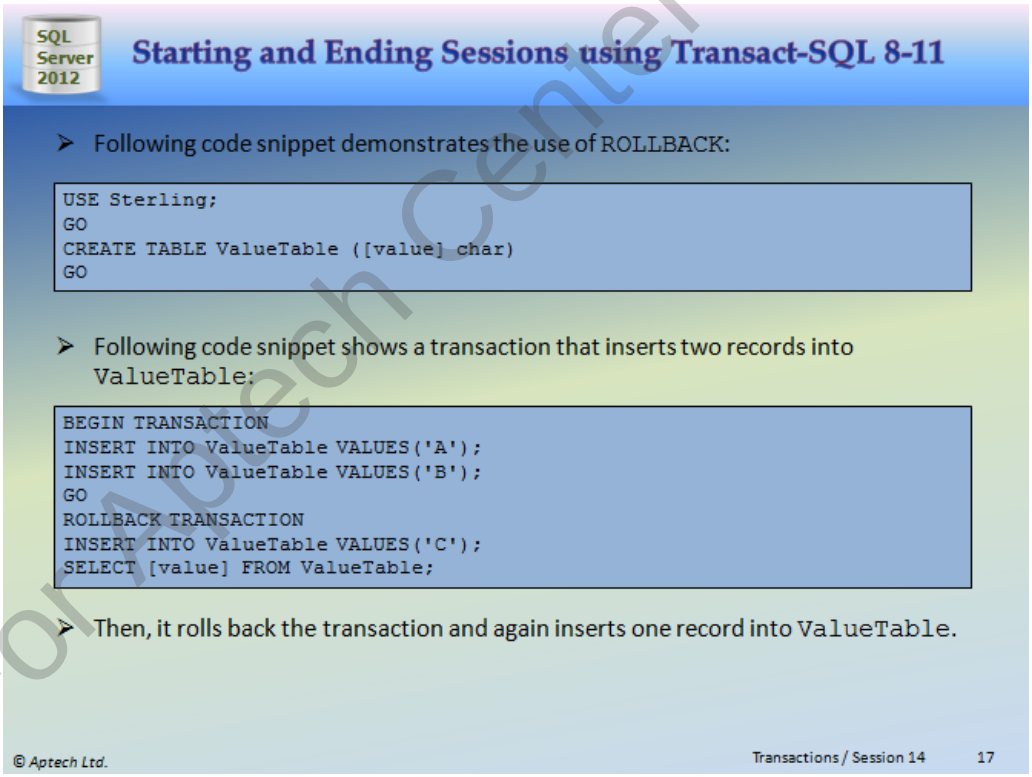










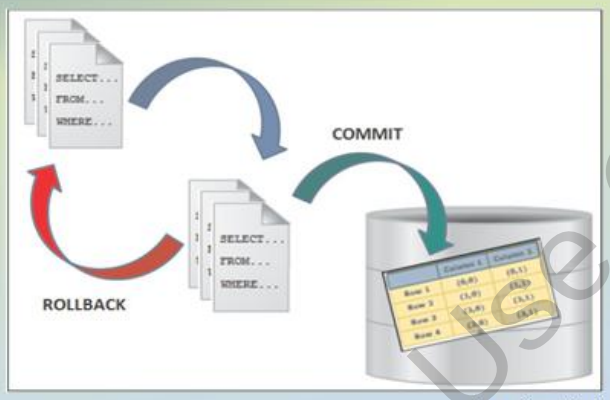


Starting and Ending Sessions using Transact-SQL 9-11

`ROLLBACK WORK` statement rolls back a user-specified transaction to the beginning of the transaction.

Syntax:

```
ROLLBACK [ WORK ]
[ ; ]
```



Starting and Ending Sessions using Transact-SQL 10-11

`SAVE TRANSACTION` statement sets a savepoint within a transaction.

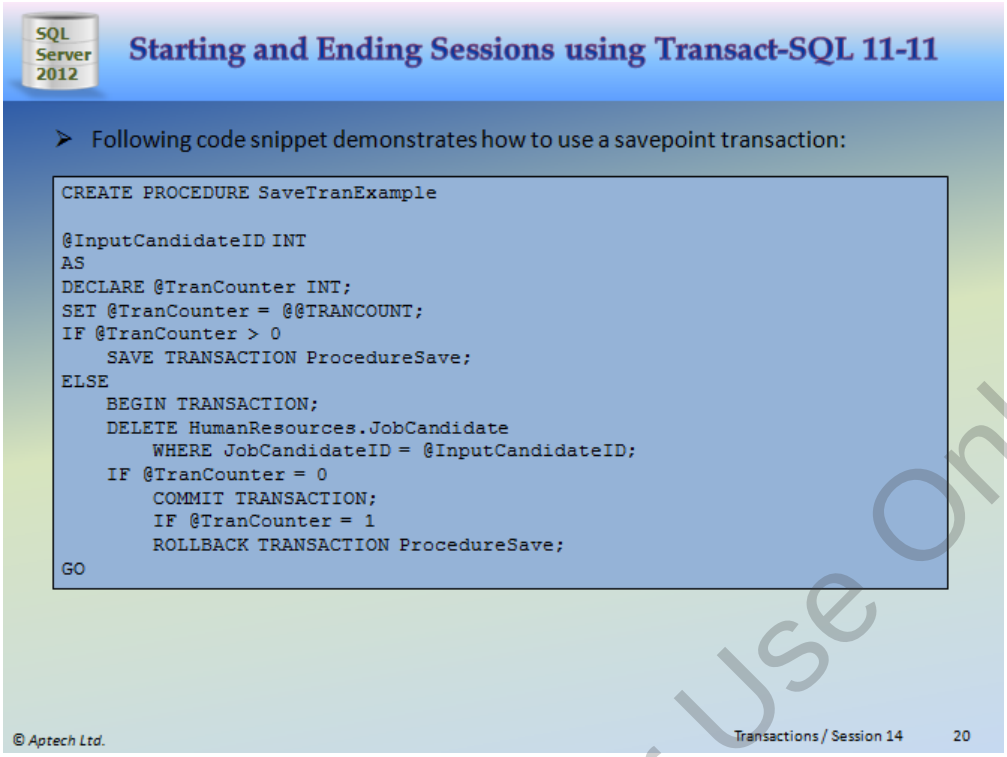
Syntax:

```
SAVE { TRAN | TRANSACTION } { savepoint_name | @savepoint_variable }
[ ; ]
```

where,

`savepoint_name`: specifies the `savepoint_name` assigned. These names conform to the rules of identifiers and are restricted to 32 characters.

`@savepoint_variable`: specifies the name of a user-defined variable that contain a valid savepoint name. The variable can be declared as `char`, `varchar`, `nchar`, or `nvarchar` data type. More than 32 characters are allowed to pass to the variables but only the first 32 characters are used.



Using slides 10 to 20, explain how to start and end a session using Transact-SQL.
One of the ways, users can start and end transactions by using Transact-SQL statements. Users can start a transaction in SQL Server in the implicit or explicit modes. Explicit transaction mode starts a transaction by using a BEGIN TRANSACTION statement. Users can end a transaction using the ROLLBACK or COMMIT statements.

Explain some of the types of transaction statements.

The BEGIN TRANSACTION statement marks the beginning point of an explicit or local transaction.

Explain the syntax for the BEGIN TRANSACTION statement shown on slide 11.

Explain the code snippet shown on slide 12 which shows how to create and begin a transaction.

In this code snippet, a transaction name is declared using a variable with value FirstTransaction. A new transaction with this name is then created having a DELETE statement. As the transaction comprises a single-line statement, it is implicitly committed.

Explain the COMMIT TRANSACTION statement.

The COMMIT TRANSACTION statement marks an end of a successful implicit or explicit transaction. If the @@TRANCOUNT is 1, then the COMMIT TRANSACTION performs all data modifications performed on the database and becomes a permanent part of the database. Further, it releases the resources held by the transaction and decrements @@TRANCOUNT by 0. If @@TRANCOUNT is greater than 1, then the COMMIT TRANSACTION decrements the @@TRANCOUNT by 1 and keeps the transaction in active state.

Explain the syntax for the COMMIT TRANSACTION statement on slide 13.

Explain the code snippet on slide 14 which shows how to commit a transaction in the HumanResources.JobCandidate table of AdventureWorks2012 database.

This code snippet defines a transaction that will delete a job candidate record having JobCandidateID as 11.

The COMMIT WORK statement marks the end of a transaction. Explain the syntax for the COMMIT WORK statement shown on slide 14.

COMMIT TRANSACTION and COMMIT WORK are identical except for the fact that COMMIT TRANSACTION accepts a user-defined transaction name.

Explain the step for marking a transaction. Explain the code snippet on that slide 15 shows how to mark a transaction in the HumanResources.JobCandidate table of AdventureWorks2012 database.

In this code snippet, a transaction named DeleteCandidate is created and marked in the log.

This transaction rolls back or cancels an implicit or explicit transaction to the starting point of the transaction, or to a savepoint in a transaction. A savepoint is a mechanism to roll back some parts of transactions. The ROLLBACK TRANSACTION is used to delete all data modifications made from the beginning of the transaction or to a savepoint. It also releases the resources held by the transaction.

Explain the syntax for the ROLLBACK TRANSACTION statement shown slide 16.

Consider an example that demonstrates the use of ROLLBACK. Assume that a database named, Sterling has been created. A table named, ValueTable is created in this database as shown in code snippet on slide 17.

Explain the code snippet on slide 17 which creates a transaction that inserts two records into ValueTable. Then, it rolls back the transaction and again inserts one record into ValueTable. When a SELECT statement is used to query the table, you will see that only a single record with value C is displayed. This is because the earlier INSERT operations have been rolled back or cancelled.

This statement rolls back a user-specified transaction to the beginning of the transaction. Explain the syntax for the ROLLBACK WORK statement shown slide 18.

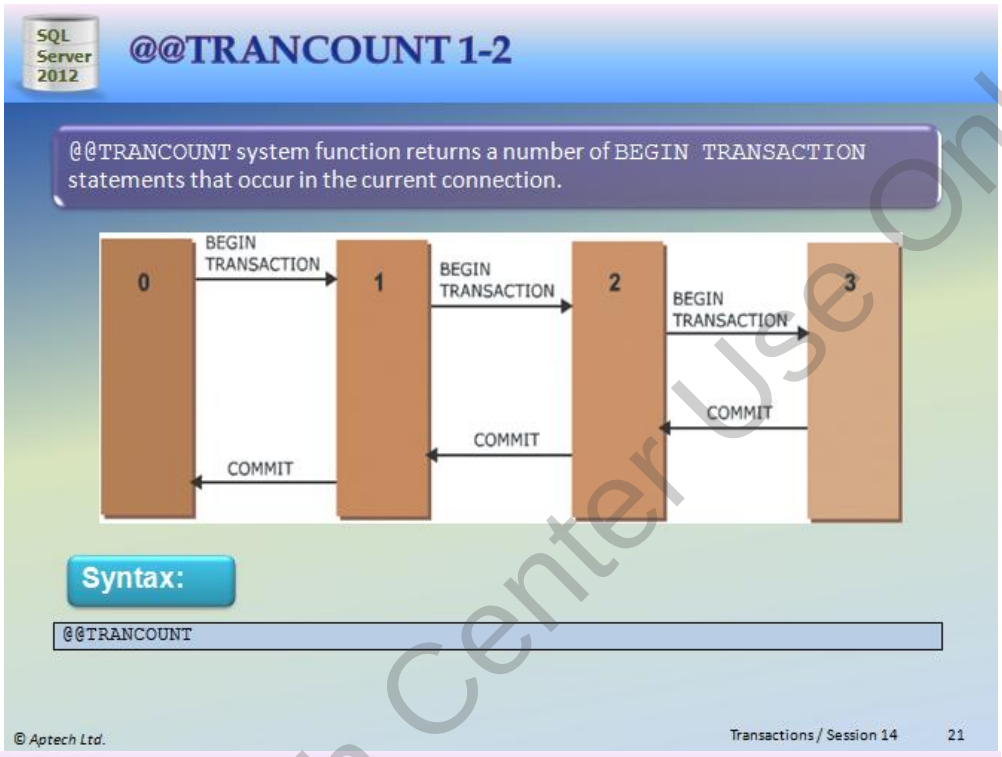
The keyword WORK is optional and is rarely used. Figure shown on slide 18 displays the working of transactions. The SAVE TRANSACTION statement sets a savepoint within a transaction. Explain the syntax for the SAVE TRANSACTION statement shown on slide 19.

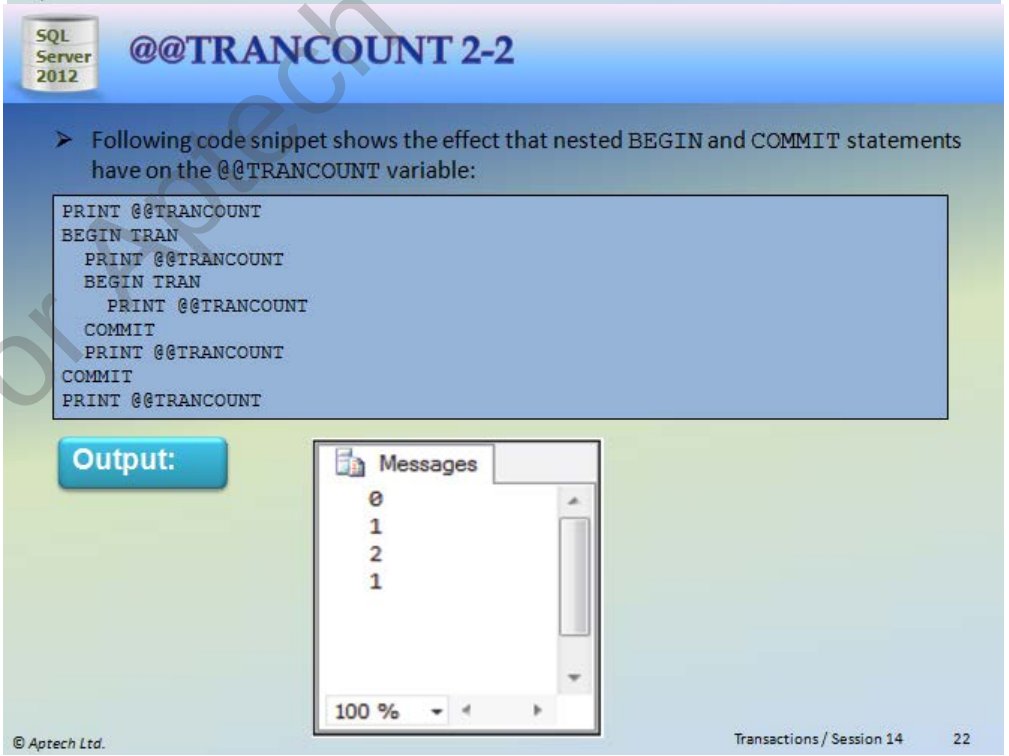
Explain the code snippet on slide 20 which shows how to use a savepoint transaction.

In this code snippet, a savepoint transaction is created within a stored procedure. This will then be used to roll back only the changes made by the stored procedure if an active transaction has started before the stored procedure executes.

@@TRANCOUNT

Slides 21 and 22





Using slides 21 and 22, explain the @@TRANCOUNT.

The @@TRANCOUNT system function returns a number of BEGIN TRANSACTION statements that occur in the current connection. Figure on slide 21 displays an example of using @@TRANCOUNT.

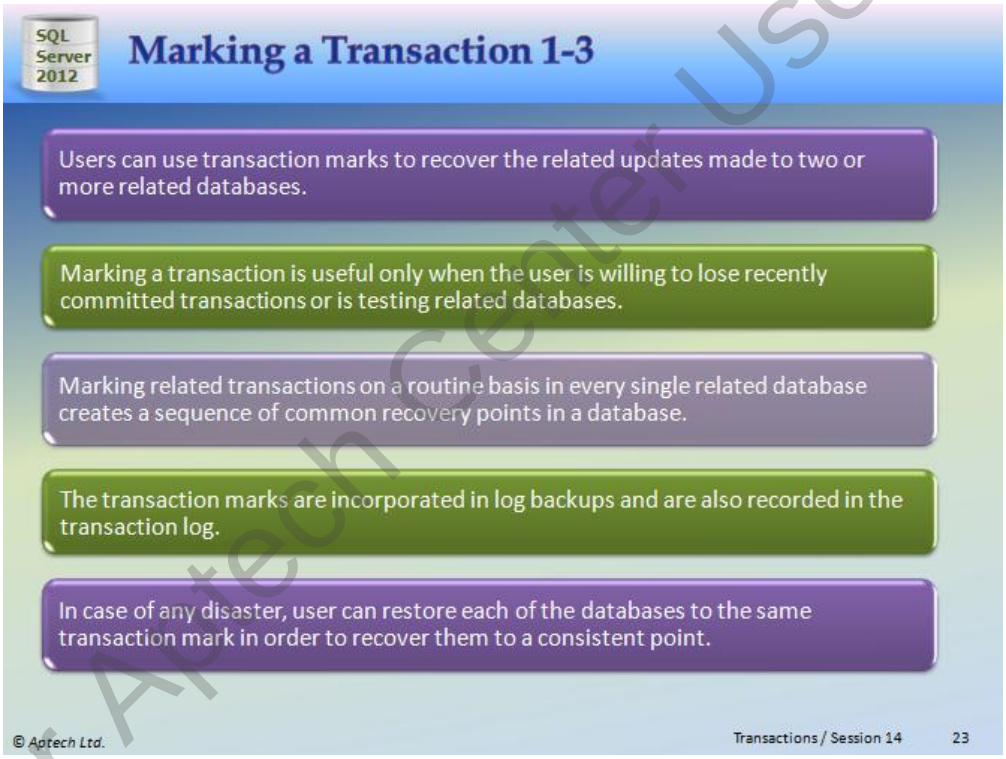
Explain the syntax for the @@TRANCOUNT statement shown on slide 21.

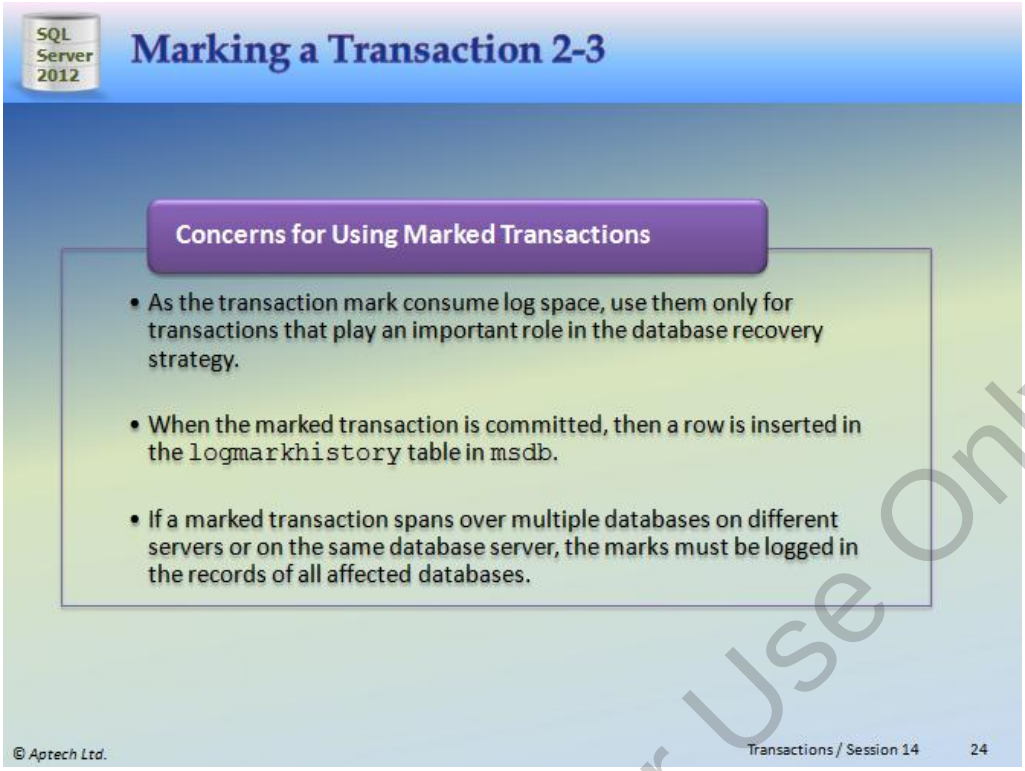
Explain the code snippet on slide 22 which shows the effect that nested BEGIN and COMMIT statements have on the @@TRANCOUNT variable.

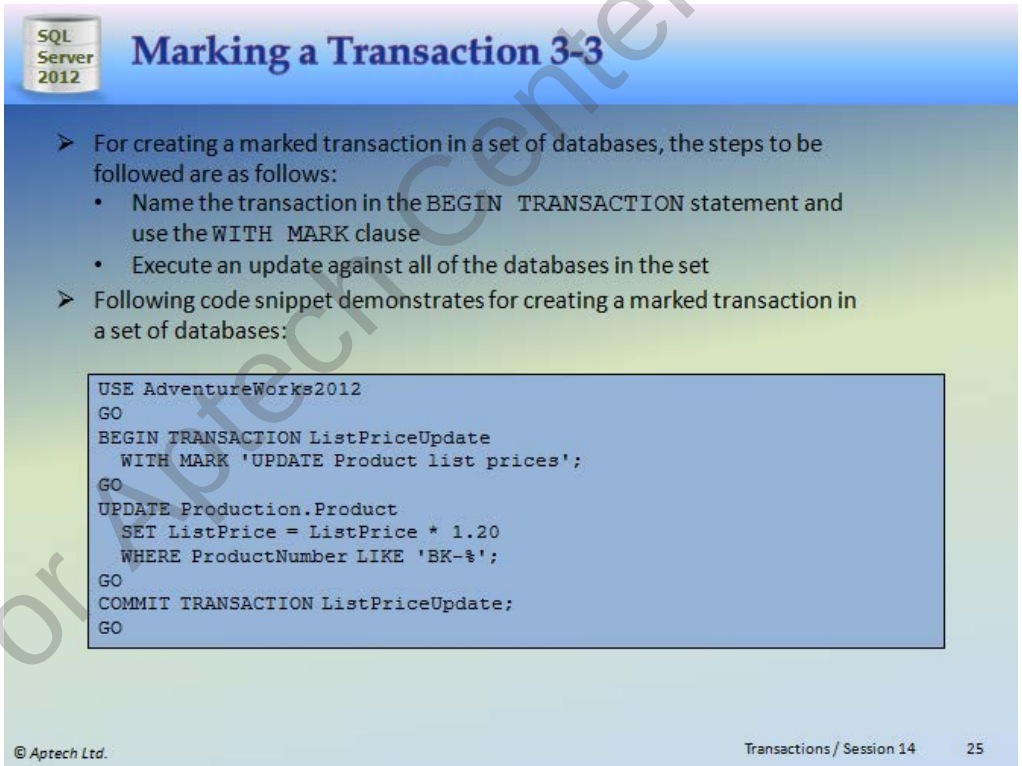
This code snippet displays the number of times the BEGIN TRAN and COMMIT statement execute in the current connection. Figure displays the output of code snippet on slide 22.

Marking a Transaction

Slides 23 to 25







Using slides 23 to 25, explain how to mark a transaction.

Users can use transaction marks to recover the related updates made to two or more related databases. Though this recovery loses every transaction that is committed after the mark was used as the recovery point. Marking a transaction is useful only when the user is willing to lose recently committed transactions or is testing related databases. Marking related transactions on a routine basis in every single related database creates a sequence of common recovery points in a database. The transaction marks are incorporated in log

backups and are also recorded in the transaction log. In case of any disaster, user can restore each of the databases to the same transaction mark in order to recover them to a consistent point.

Consider the following situations before inserting the named marks in the transaction:

- As the transaction mark consume log space, use them only for transactions that play an important role in the database recovery strategy.
- When the marked transaction is committed, then a row is inserted in the logmarkhistory table in msdb.
- If a marked transaction spans over multiple databases on different servers or on the same database server, the marks must be logged in the records of all affected databases.

Explain the steps to be followed to create marked transaction using slide 25.
Explain the code snippet which shows updation of the ListPrice in the Product table of the AdventureWorks2012 database shown on slide 25.
Mention that, the mark is placed in the transaction log only if the database is updated by the marked transaction. Transactions that do not modify data are not marked.

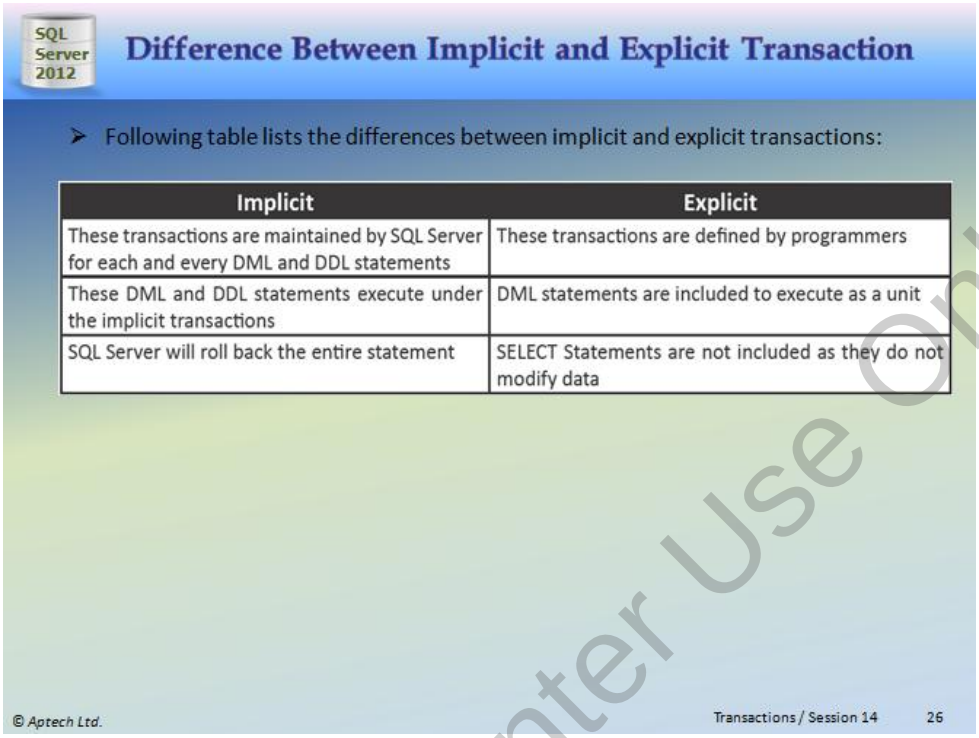
In-Class Question:

 What is table used to insert record for a marked transaction when transaction is committed?

Answer:
The logmarkhistory table in msdb is used to insert record for a marked transaction when transaction is committed.

Difference between Implicit and Explicit Transactions

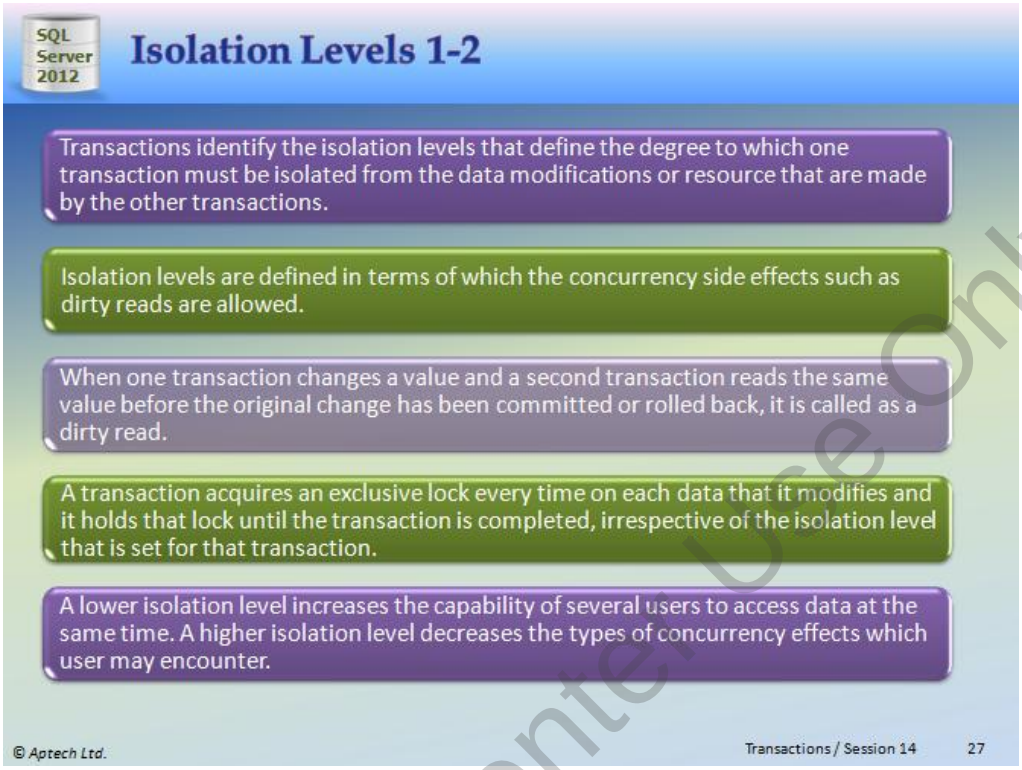
Slide 26

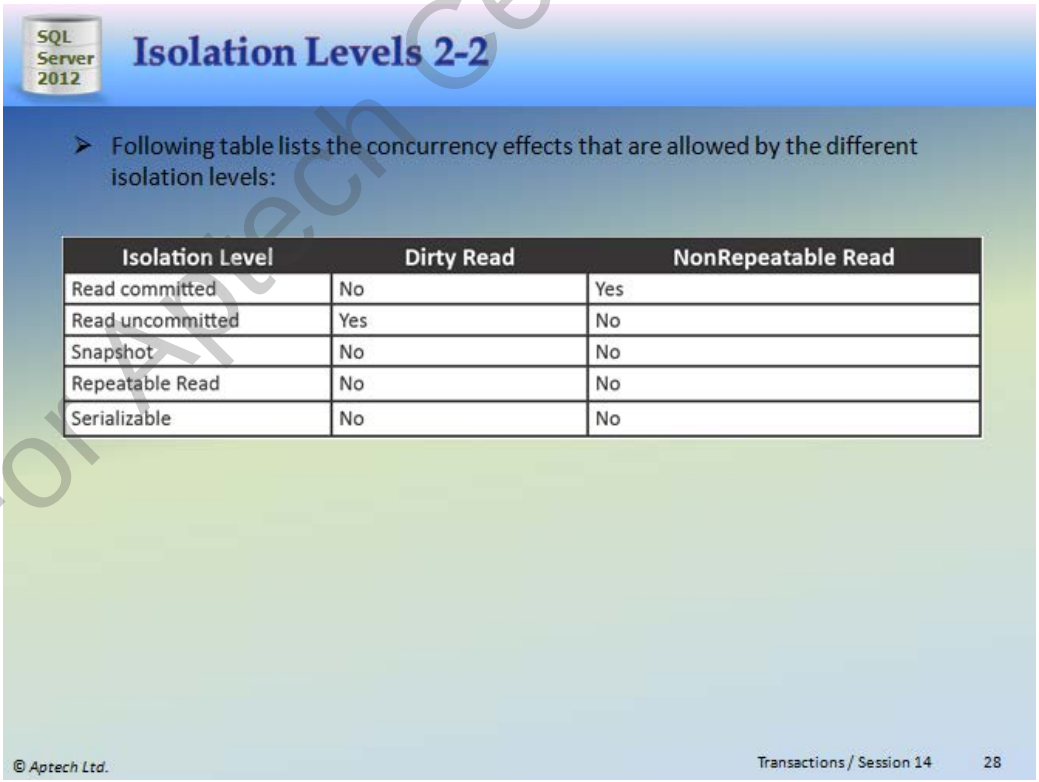


Using slide 26, explain the difference between implicit and explicit transactions. Mention that, the SET IMPLICIT_TRANSACTIONS ON statement to turn implicit transaction mode. The other statements related to the transactions such as COMMIT TRANSACTION, COMMIT WORK, ROLLBACK TRANSACTION, or ROLLBACK WORK statements to end each transaction can be used.

Isolation Levels

Slides 27 and 28





Using slides 27 and 28, explain the isolation levels.

Transactions identify the isolation levels that define the degree to which one transaction must be isolated from the data modifications or resource that are made by the other

transactions. Isolation levels are defined in terms of which the concurrency side effects such as dirty reads are allowed. When one transaction changes a value and a second transaction reads the same value before the original change has been committed or rolled back, it is called as a dirty read.

Transaction isolation levels control the following:

- When data is read, are there any locks taken and what types of locks are requested?
- How much amount of time are the read locks held?
- If a read operation that is referencing a row modified by some other transaction is:
    - Blocking until the exclusive lock on the row is free
    - Retrieving the committed version of the row that exists at the time when the transaction or statement started.
    - Reading the uncommitted data modification.

While choosing a transaction isolation level, those locks that prevent data modification are not affected. A transaction acquires an exclusive lock every time on each data that it modifies. Then, it holds that lock until the transaction is completed, irrespective of the isolation level that is set for that transaction.

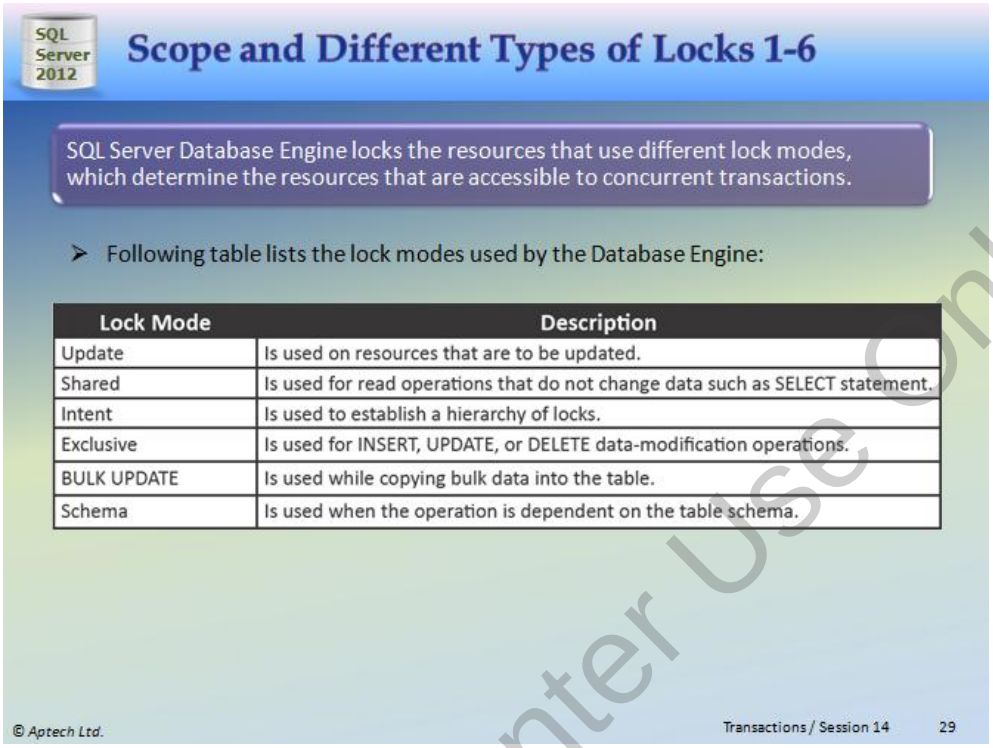
Transaction isolation levels mainly describe the protection levels from the special effects of changes made by other transactions for read operations. A lower isolation level increases the capability of several users to access data at the same time. However, it increases the number of concurrency effects such as dirty reads or lost updates that users might come across. On the other hand, a higher isolation level decreases the types of concurrency effects which user may encounter. This requires additional system resources and increases the chance of one transaction blocking another transaction.

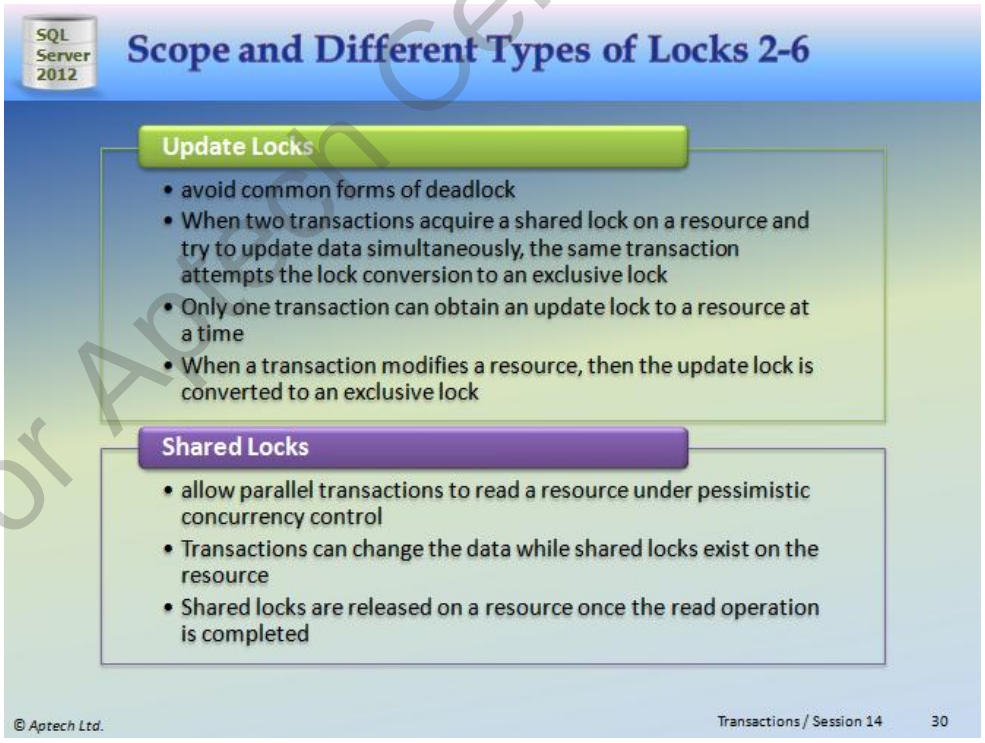
Selecting a suitable isolation level is based on the data integrity requirements of the application as compared to the overheads of each isolation level. The higher isolation level, serializable, assures that a transaction will recover the same data each time it repeats the read operation. Then, it does this by performing a level of locking that is expected to influence other users in a multi-user system. The lower isolation level, read uncommitted, retrieves data that is modified, and is not committed by other transactions. All concurrency side effects occur in read uncommitted, however, there is no read versioning or locking, hence, the overhead is minimized.

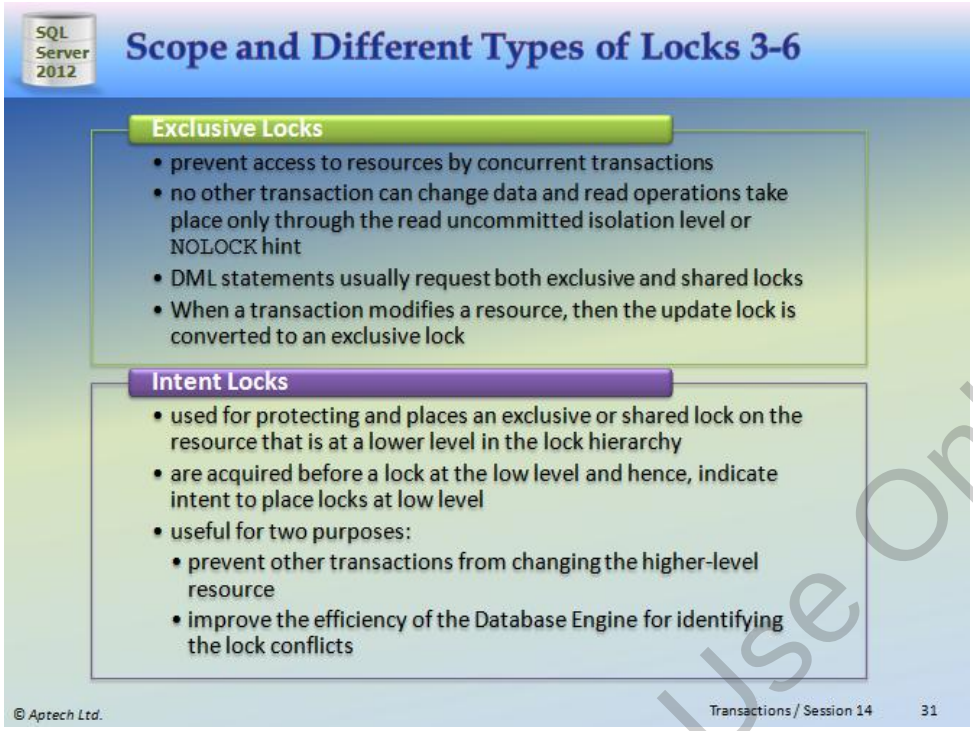
Table shown on slide 28 lists the concurrency effects that are allowed by the different isolation levels.
Transactions need to execute at an isolation level of at least repeatable read that prevents lost updates occurring when two transactions each retrieve the same row, and then updates the row that is dependent on the originally retrieved rows.

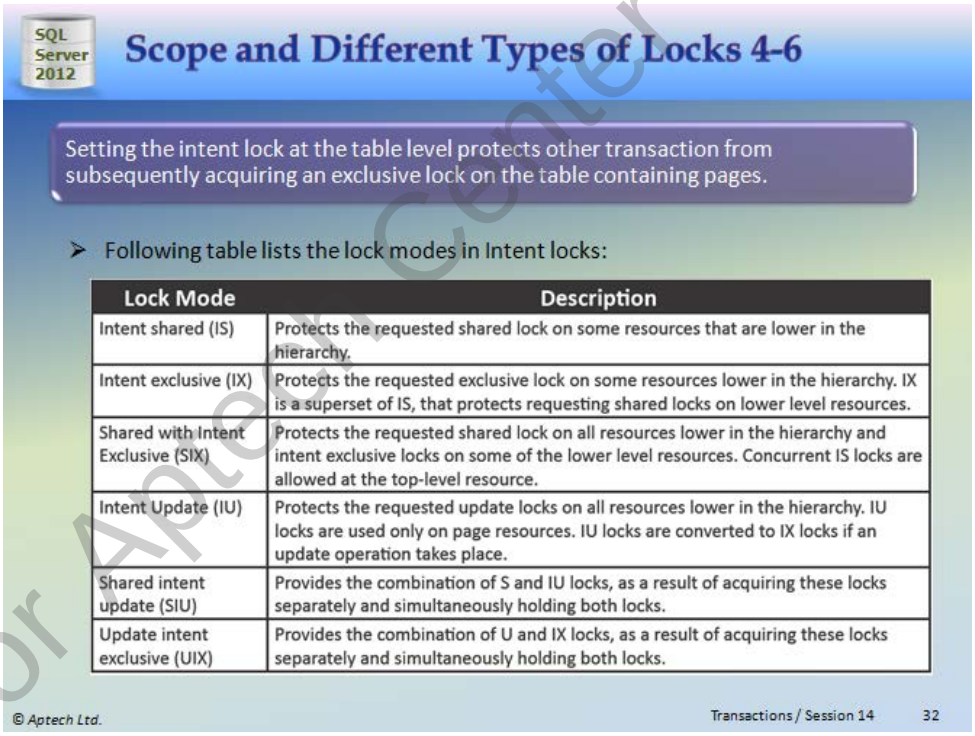
Scope and Different Types of Locks

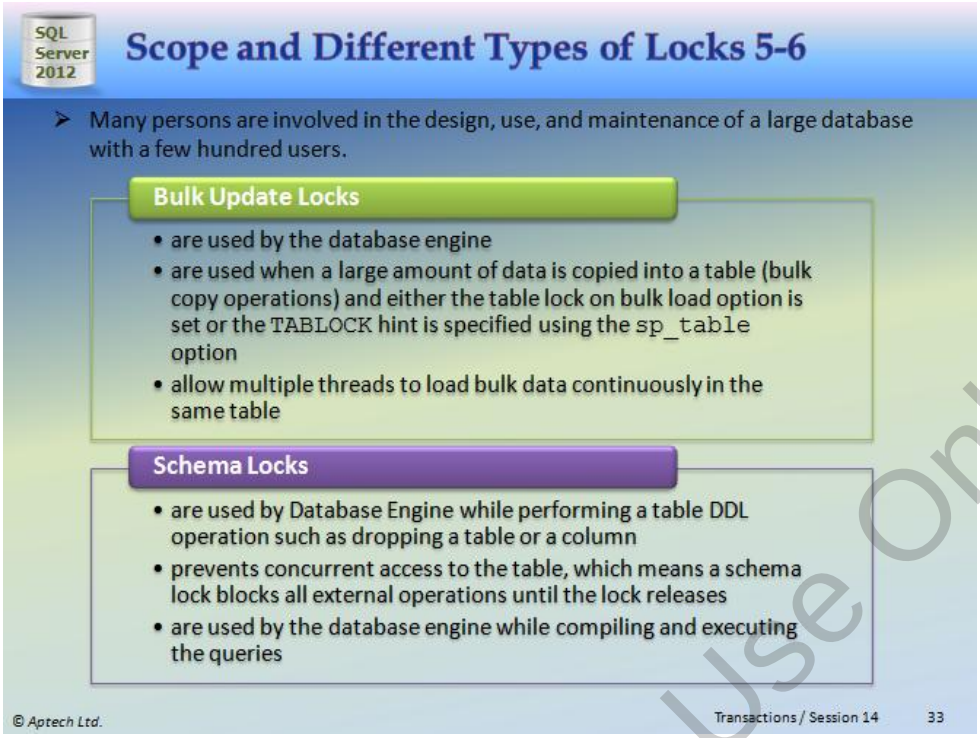
Slides 29 to 34

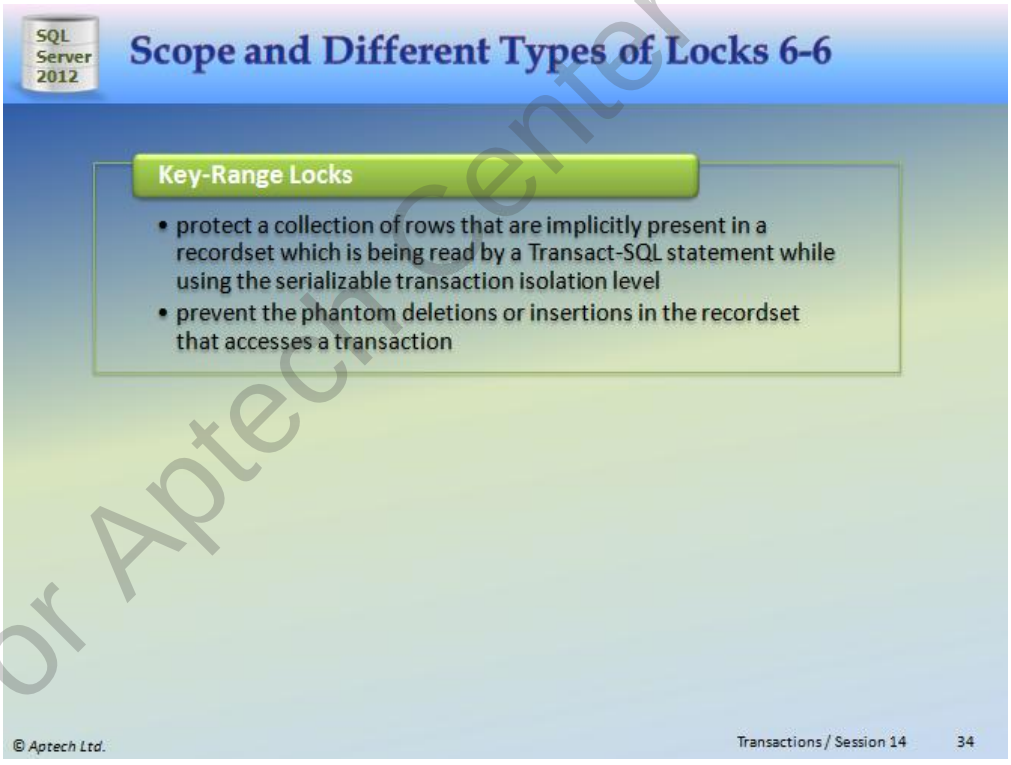












Using slides 29 to 34, explain the scope and different types of locks.
The SQL Server Database Engine locks the resources that use different lock modes, which determine the resources that are accessible to concurrent transactions. Table on slide 29 lists the resource lock modes used by the database engine.

Following are the types of locks:

- **Update Locks**: These locks avoid common forms of deadlock. In a serializable transaction, the transaction will read data, acquire a shared lock on the row or a page,

and modify the data that requires lock conversion to an exclusive lock. When two transactions acquire a shared lock on a resource and try to update data simultaneously, the same transaction attempts the lock conversion to an exclusive lock. The shared mode to exclusive lock conversion should wait as the exclusive lock for one transaction is not compatible with shared mode lock of the other transaction a lock wait occurs. Similarly, the second transaction tries to acquire an exclusive lock for update. As both the transactions are converting to exclusive locks and each waits for the other transaction to release its shared lock mode, a deadlock occurs.

- **Shared Locks**: These locks allow parallel transactions to read a resource under pessimistic concurrency control. Transactions can change the data while shared locks exist on the resource. Shared locks are released on a resource once the read operation is completed, except the isolation level is set to repeatable read or higher.
- **Exclusive Locks**: These locks prevent access to resources by concurrent transactions. By using an exclusive lock, no other transaction can change data and read operations take place only through the read uncommitted isolation level or NOLOCK hint. DML statements such as INSERT, DELETE, and UPDATE combine modification and read operations. These statements first perform a read operation to get data before modifying the statements. DML statements usually request both exclusive and shared locks. For example, if the user wants to use an UPDATE statement that modifies the row in one table that is dependent on a join with other table. Therefore, the update statements request shared lock on the rows that reads from the join table and request exclusive locks on the modified rows.
- **Intent Locks**: The database engine uses intent locks for protecting and places an exclusive or shared lock on the resource that is at a lower level in the lock hierarchy. The name intent locks is given because they are acquired before a lock at the low level and hence, indicate intent to place locks at low level. An intent lock is useful for two purposes:
  - To prevent other transactions from changing the higher-level resource in a way that will invalidate the lock at the lower-level.
  - To improve the efficiency of the database engine for identifying the lock conflicts those are at the higher level of granularity.

  For example, a shared intent locks are requested at the table level before requesting the shared locks on rows or pages within the table. Setting the intent lock at the table level protects other transaction from subsequently acquiring an exclusive lock on the table containing pages. Intent locks also contain Intent Exclusive (IX), Intent Shared (IS), and Shared with Intent Exclusive (SIX). Table shown on slide 32 lists the lock modes in Intent locks.
- **Bulk Update locks:** Bulk update locks are used by the database engine. These locks are used when a large amount of data is copied into a table (bulk copy operations) and either the table lock on bulk load option is set or the TABLOCK hint is specified using the sp_table option. These locks allow multiple threads to load bulk data continuously in the same table however, preventing other processes that are not bulk loading data from accessing the table.
- **Schema Locks**: Schema modification locks are used by database engine while performing a table DDL operation such as dropping a table or a column. Schema locks prevent concurrent access to the table, which means a schema lock blocks all external

operations until the lock releases. Some DML operations such as truncating a table use the Schema lock to prevent access to affected tables by concurrent operations. Schema stability locks are used by the database engine while compiling and executing the queries. These stability locks do not block any of the transaction locks including the exclusive locks. Hence, the transactions that include X locks on the table continue to execute during the query compilation. Though, the concurrent DML and DDL operations that acquire the Schema modification locks do not perform on the tables.

- **Key-Range Locks**: These types of locks protect a collection of rows that are implicitly present in a recordset which is being read by a Transact-SQL statement while using the serializable transaction isolation level. Key-range locks prevent phantom reads. By protecting the range of keys between rows, they also prevent the phantom deletions or insertions in the recordset that accesses a transaction.

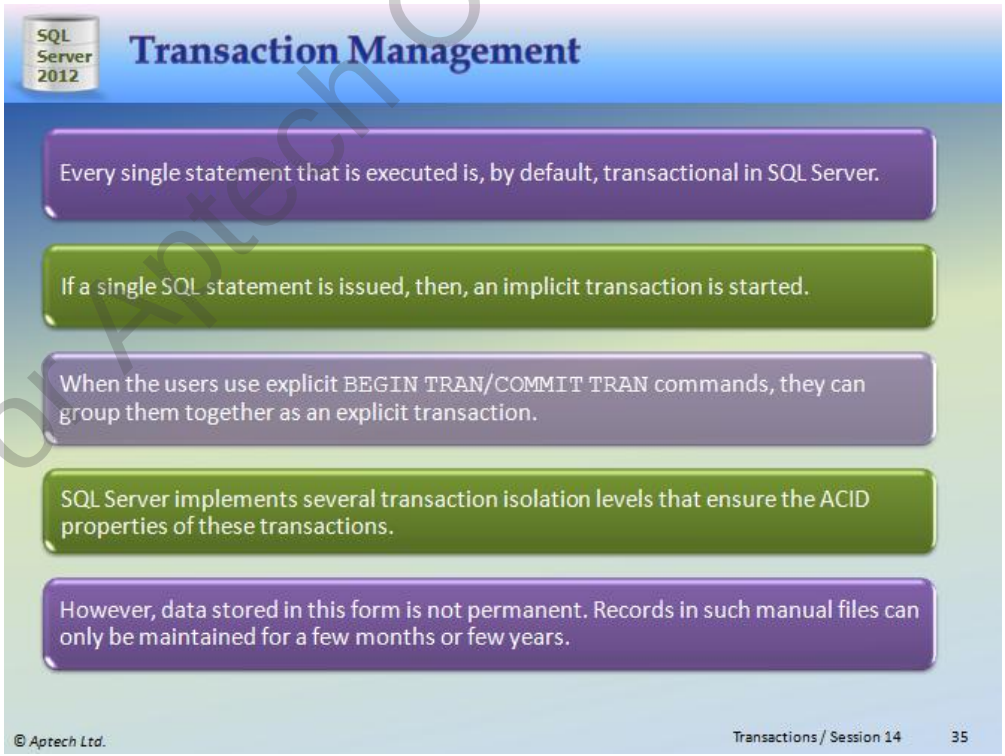
In-Class Question:

 When a bulk update lock is used?

Answer:
A bulk update lock is used when a large amount of data is copied into a table (bulk copy operations).

Transaction Management

Slide 35

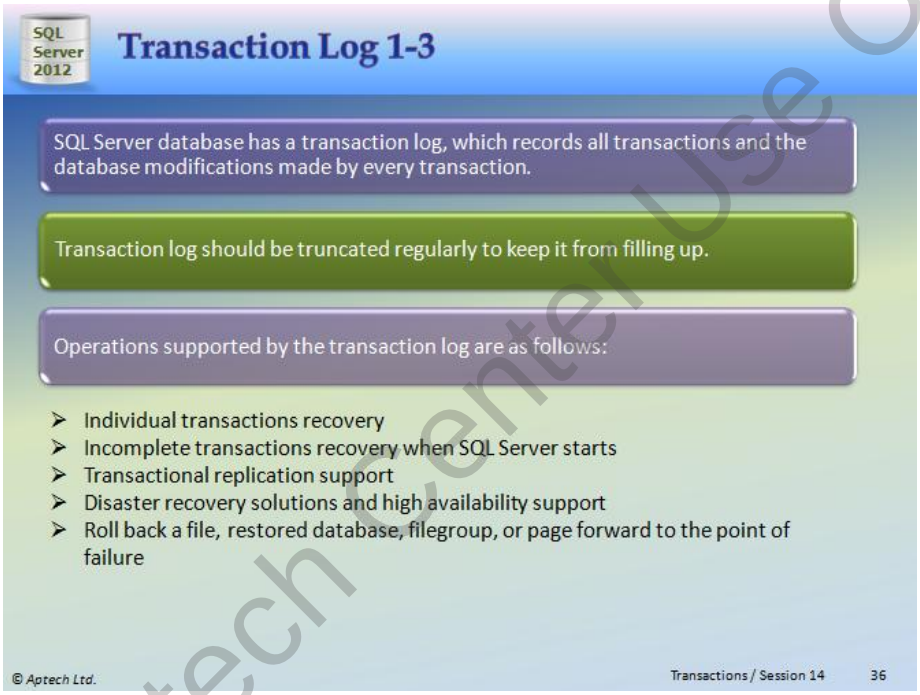


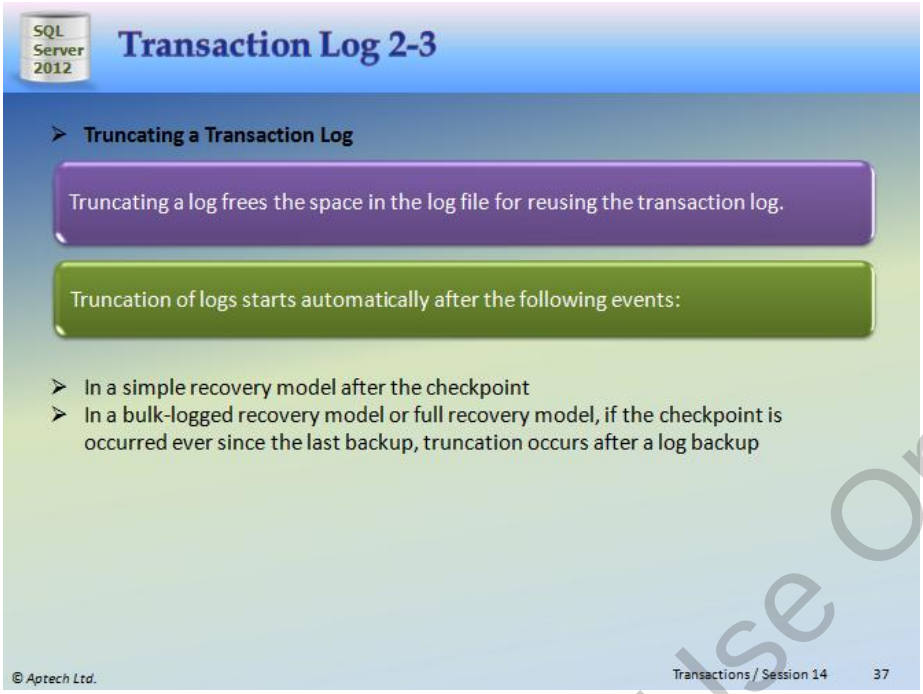
Using slide 35, explain the transaction management.

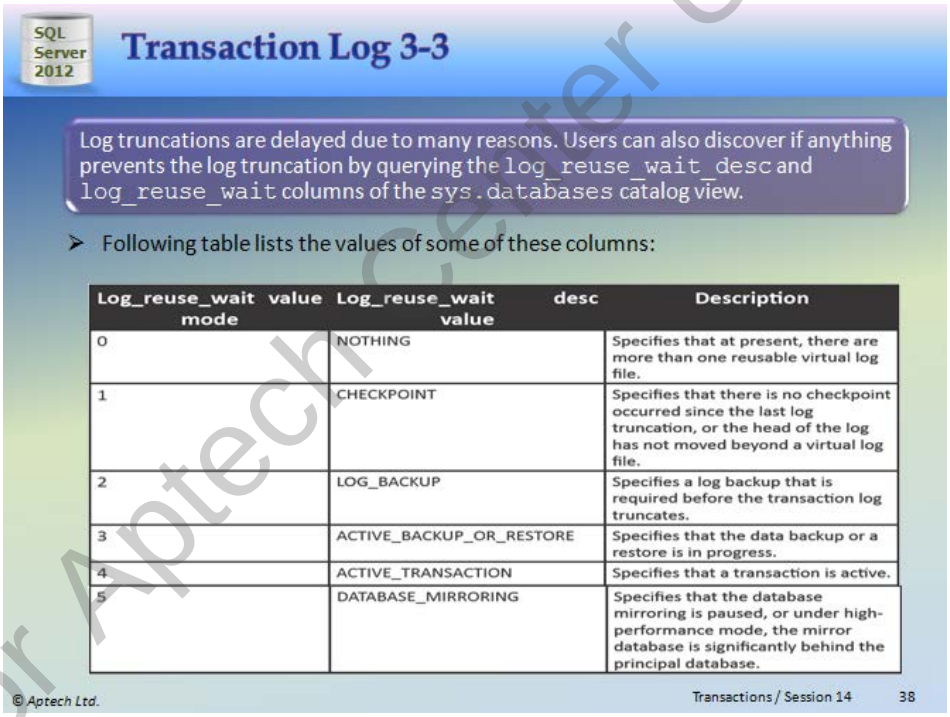
Every single statement that is executed is, by default, transactional in SQL Server. If a single SQL statement is issued, then an implicit transaction is started. It means the statement will start and complete implicitly. When the users use explicit BEGIN TRAN/COMMIT TRAN commands, they can group them together as an explicit transaction. These statements will either succeed or fail. SQL Server implements several transaction isolation levels that ensure the ACID properties of these transactions. In reality, it means that it uses locks to facilitate transactional access to shared database resources and also, prevent the interference between the transactions.

Transaction Log

Slides 36 to 38







Using slides 36 and 37, explain the transaction log.

Each SQL Server database has a transaction log, which records all transactions and the database modifications made by every transaction. The transaction log should be truncated regularly to keep it from filling up. Monitoring log size is important as there may be some factors that delay the log truncation.

Transaction log is a critical component of the database and if a system failure occurs, the transaction log will be required to bring the database to a consistent data. The transaction log should not be moved or deleted until users understand the consequences of doing it.

The operations supported by the transaction log are as follows:

- Individual transactions recovery
- Incomplete transactions recovery when SQL Server starts
- Transactional replication support
- Disaster recovery solutions and high availability support
- Roll back a file, restored database, filegroup, or page forward to the point of failure

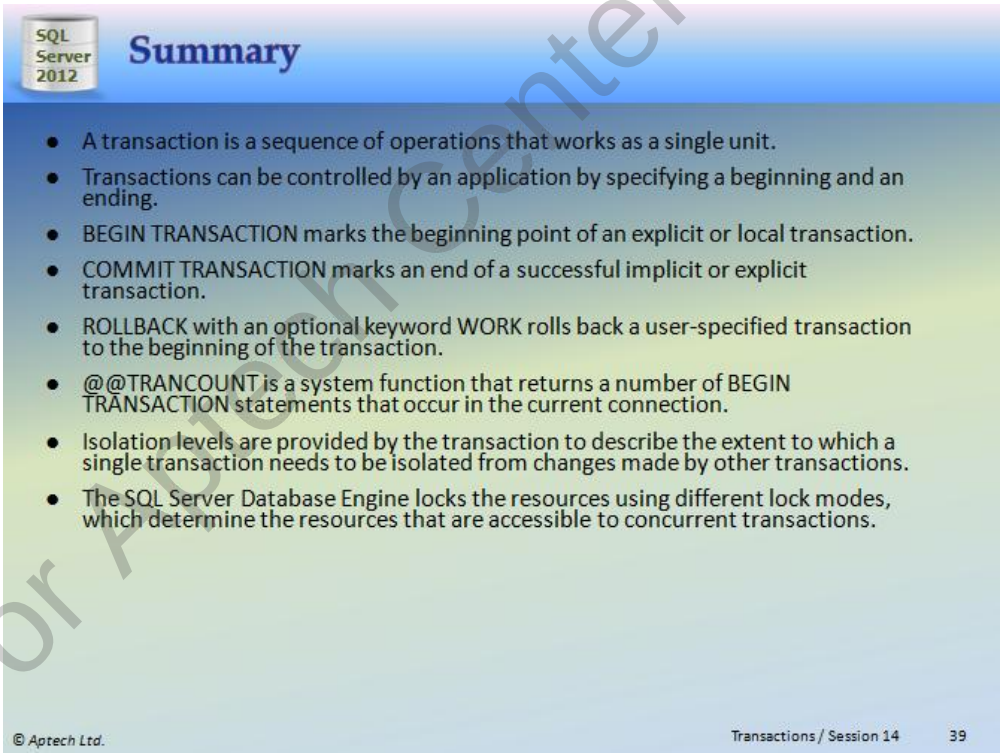
Truncating a log frees the space in the log file for reusing the transaction log. Truncation of logs starts automatically after the following events:

- In a simple recovery model, after the checkpoint.
- In a bulk-logged recovery model or full recovery model, if the checkpoint is occurred ever since the last backup, truncation occurs after a log backup.

Users can also discover if anything prevents the log truncation by querying the log_reuse_wait_desc and log_reuse_wait columns of the sys.databases catalog view. Table shown on slide 38 lists the values of some of these columns.

Summarize Session

Slide 39



Using slide 39, summarize the session. End the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- A transaction is a sequence of operations that works as a single unit.
- Transactions can be controlled by an application by specifying a beginning and an ending.

- BEGIN TRANSACTION marks the beginning point of an explicit or local transaction.
- COMMIT TRANSACTION marks an end of a successful implicit or explicit transaction.
- ROLLBACK with an optional keyword WORK rolls back a user-specified transaction to the beginning of the transaction.
- @@TRANCOUNT is a system function that returns a number of BEGIN TRANSACTION statements that occur in the current connection.
- Isolation levels are provided by the transaction to describe the extent to which a single transaction needs to be isolated from changes made by other transactions.
- The SQL Server Database Engine locks the resources using different lock modes, which determine the resources that are accessible to concurrent transactions.

14.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Error Handling topic offered with the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 15 – Error Handling

15.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

15.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the various types of errors
- Explain error handling and the implementation
- Describe the TRY-CATCH block
- Explain the procedure to display error information
- Describe the @@ERROR and RAISERROR statements
- Explain the use of ERROR_STATE, ERROR_SEVERITY, and ERROR_PROCEDURE
- Explain the use of ERROR_NUMBER, ERROR_MESSAGE, and ERROR_LINE
- Describe the THROW statement

15.1.2 Teaching Skills

To teach this session, you should be well-versed with error-handling techniques in SQL Server and describes the use of TRY-CATCH blocks. Along with this also prepare yourself with various system functions and statements that can help display error information.
You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

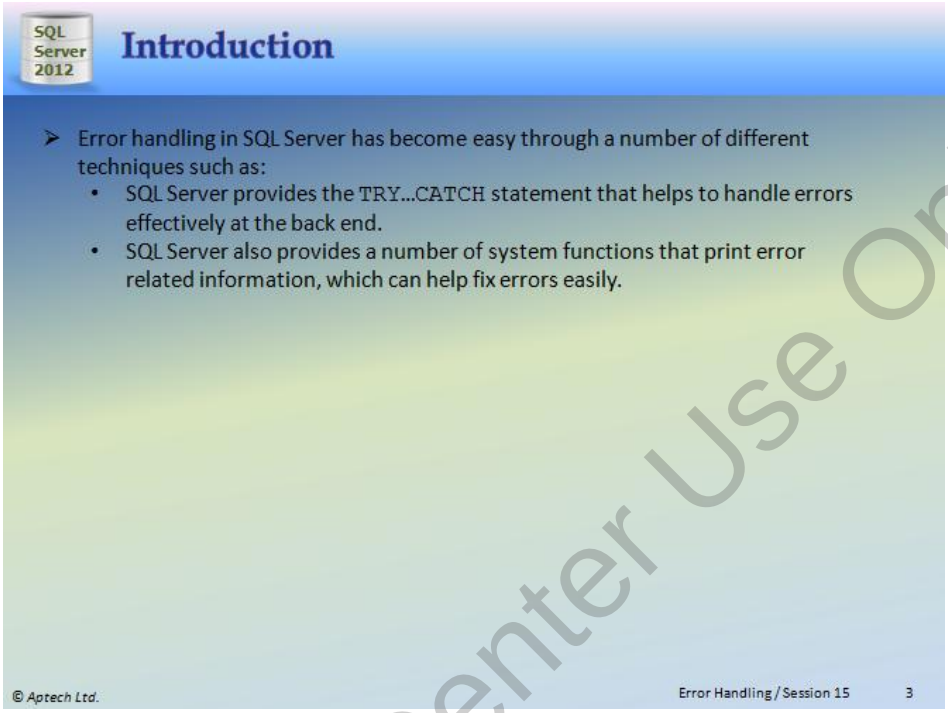
Overview of the Session:

Then, give the students the overview of the current session in the form of session objectives. Show the students slide 2 of the presentation. Tell the students that this session introduces programming with error-handling techniques in SQL Server and describes the use of TRY-CATCH blocks. They will also know about various system functions and statements that can help display error information.

15.2 In-Class Explanations

Introduction

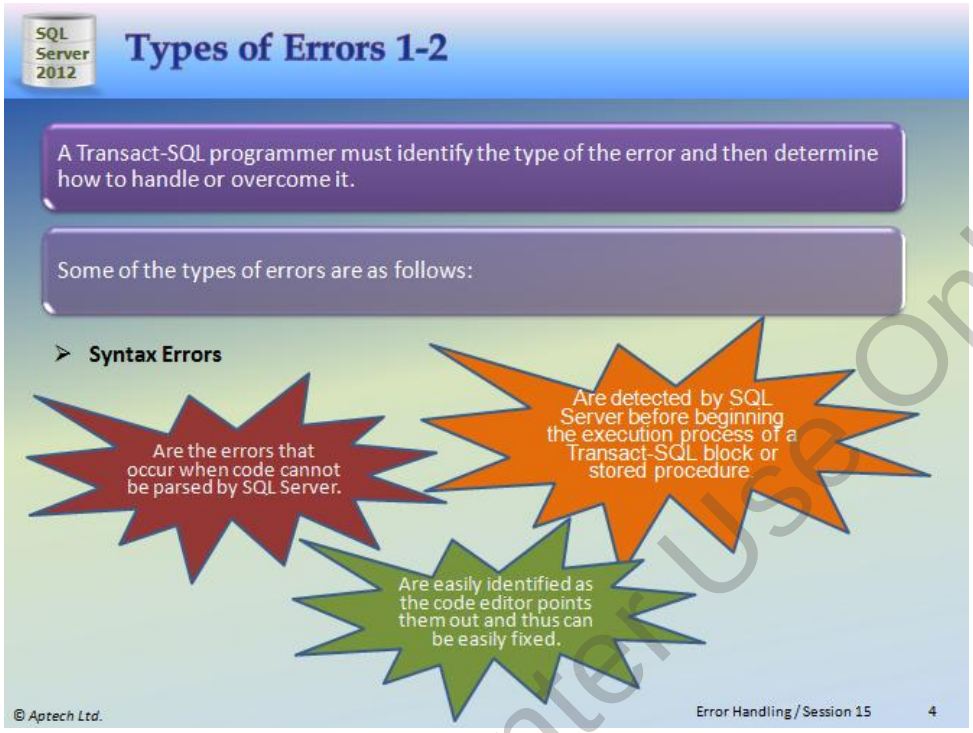
Slide 3

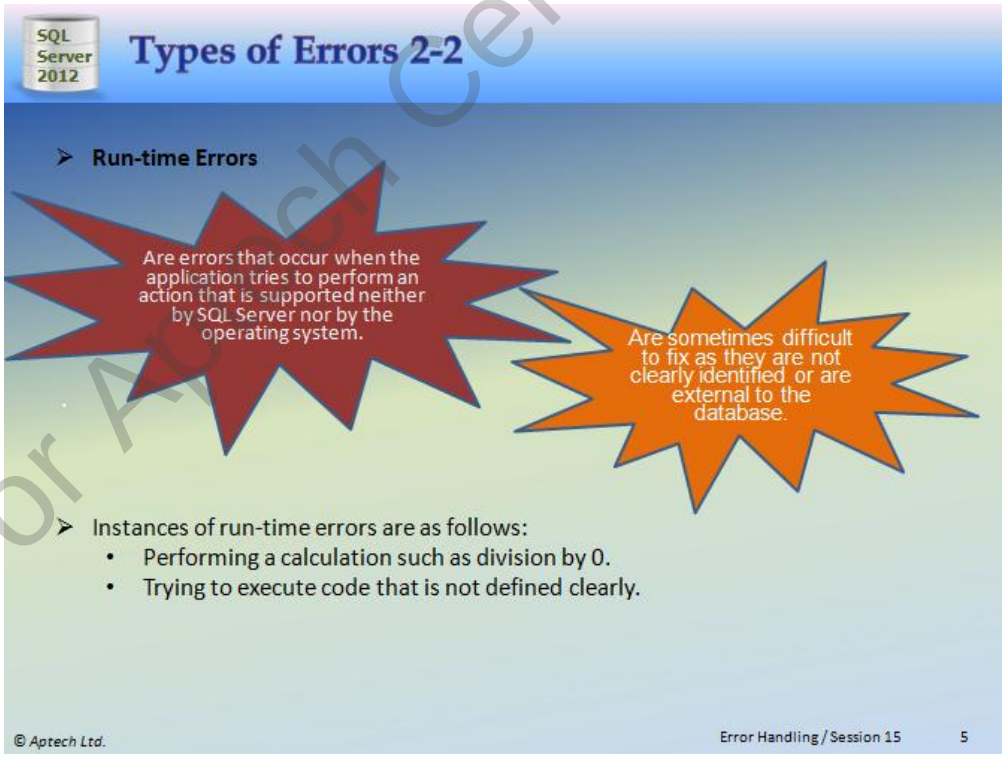


Using slide 3, explain the introduction of error handling in SQL Server.
Error handling in SQL Server has become easy through a number of different techniques. SQL Server has introduced options that can help you to handle errors efficiently. Often, it is not possible to capture errors that occur at the user's end. SQL Server provides the TRY…CATCH statement that helps to handle errors effectively at the back end. There are also a number of system functions that print error related information, which can help fix errors easily.

Types of Errors

Slides 4 and 5





Using slides 4 and 5, explain the types of errors.

As a Transact-SQL programmer, one must be aware of various types of errors that can occur while working with SQL Server statements. The first step one can perform is to identify the type of error and then determine how to handle or overcome it.

Some types of errors are as follows:

- Syntax Errors
- Run-time Errors

Syntax errors are the errors that occur when code cannot be parsed by SQL Server. Such errors are detected by SQL Server before beginning the execution process of a Transact-SQL block or stored procedure.

Run-time errors are the errors that occur when the application tries to perform an action that is supported neither by SQL Server nor by the operating system. Run-time errors are sometimes difficult to fix as they are not clearly identified or are external to the database.

In-Class Question:

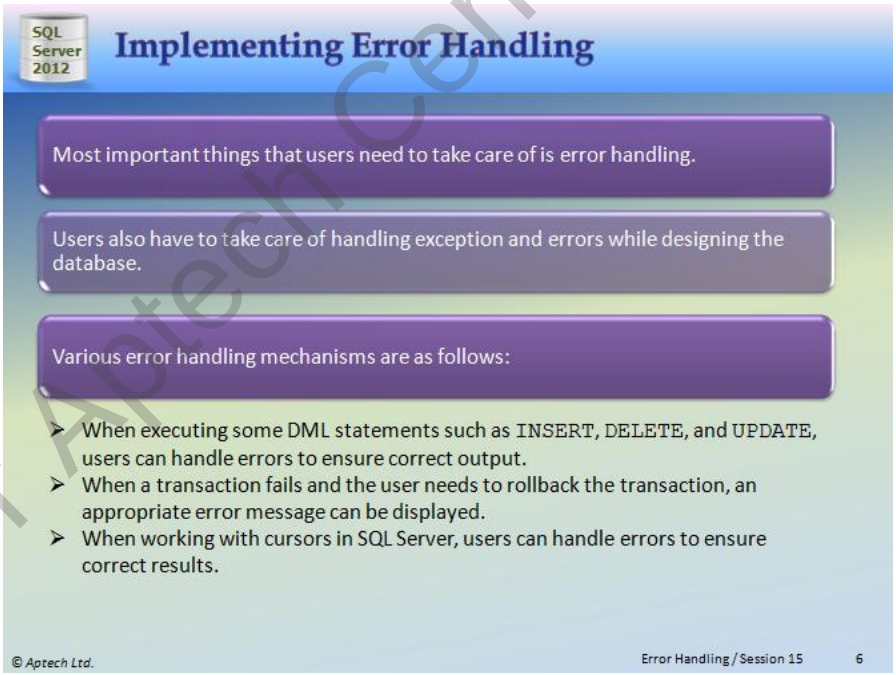
 What is a runtime error?

Answer:

Runtime errors are errors that occur when the application tries to perform an action that is supported neither by SQL Server nor by the operating system.

Implementing Error Handling

Slide 6



Using slide 6, explain how to implement error handling.

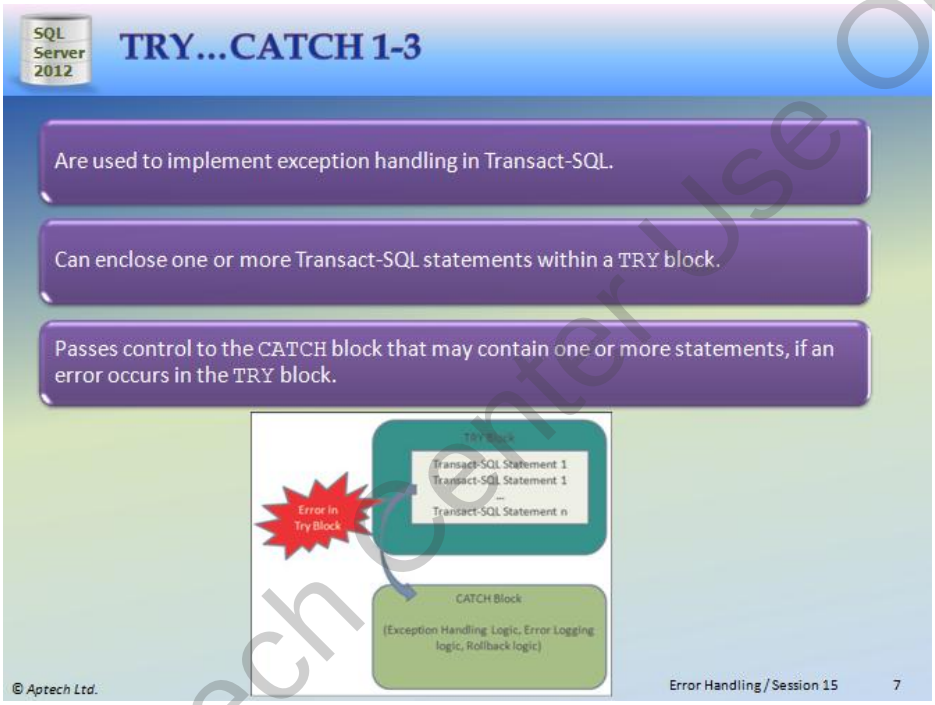
While developing any application, one of the most important things that users need to take care of is error handling. In the same way, users also have to take care of handling exception and errors while designing the database. Various error handling mechanisms can be used.

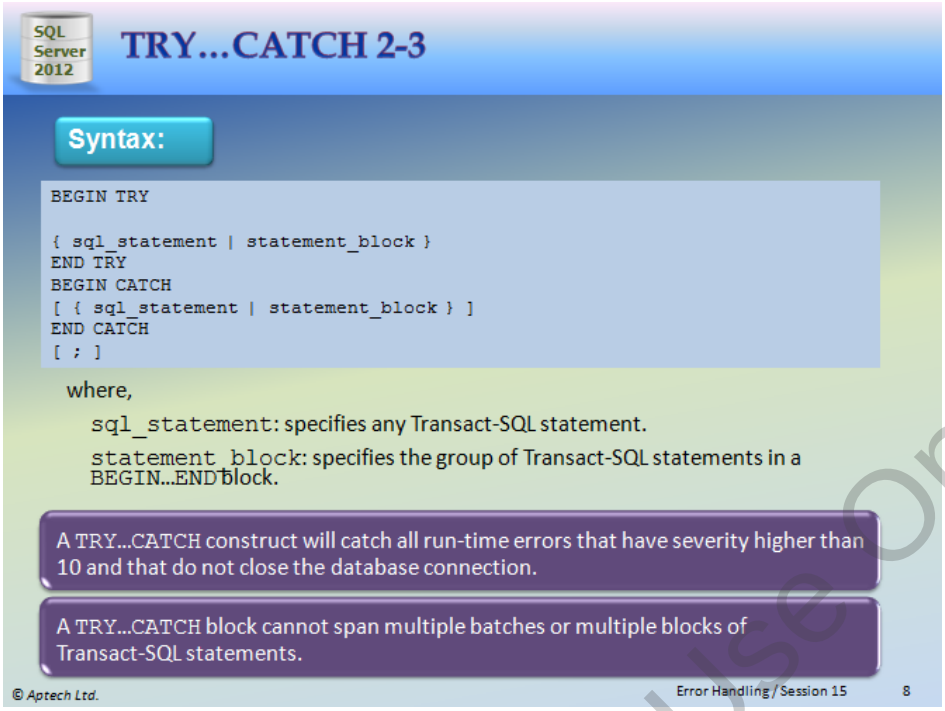
Some of them are as follows:

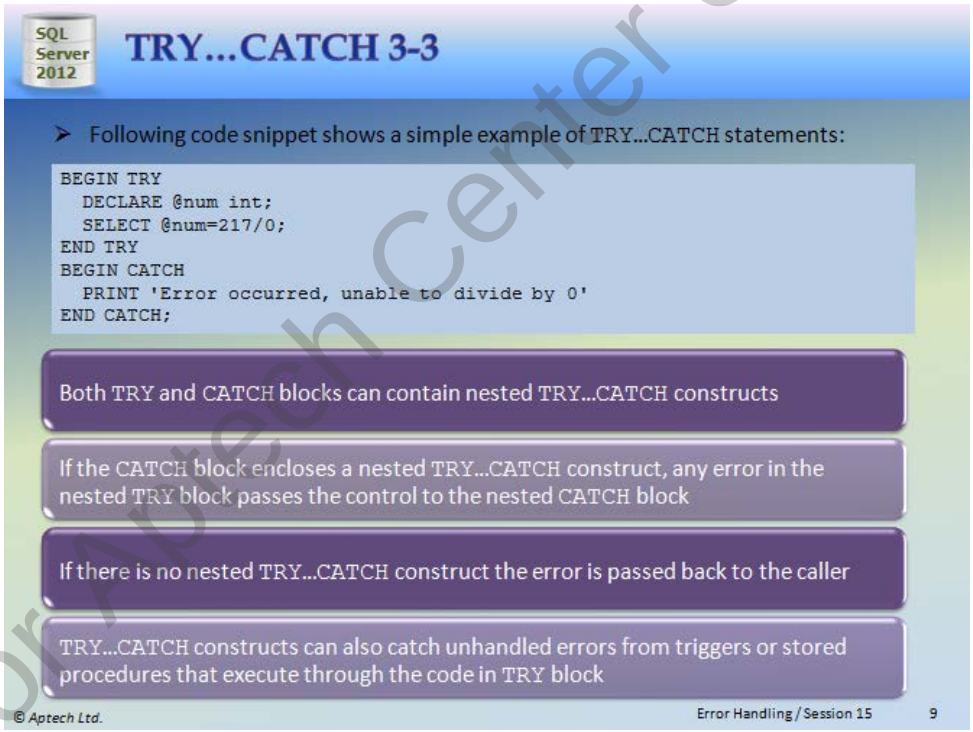
- When executing some DML statements such as INSERT, DELETE, and UPDATE, users can handle errors to ensure correct output.
- When a transaction fails and the user needs to roll back the transaction, an appropriate error message can be displayed.
- When working with cursors in SQL Server, users can handle errors to ensure correct results.

TRY..CATCH Statements

Slides 7 to 9







Using slides 7 and 8, explain the TRY..CATCH statements.

TRY…CATCH statements are used to implement exception handling in Transact-SQL. One or more Transact-SQL statements can be enclosed within a TRY block. If an error occurs in the TRY block, the control is passed to the CATCH block that may contain one or more statements.

Figure on slide 7 illustrates the TRY…CATCH logic. Explain the syntax for the TRY…CATCH statements shown on slide 8.

A TRY…CATCH construct will catch all run-time errors that have severity higher than 10 and that do not close the database connection. A TRY block is followed by a related CATCH block. A TRY…CATCH block cannot span multiple batches or multiple blocks of Transact-SQL statements.

If there are no errors in the TRY block, after the last statement in the TRY block has executed, control is passed to the next statement following the END CATCH statement. If there is an error in the TRY block, control is passed to the first statement inside the CATCH block. If END CATCH is the last statement in a trigger or a stored procedure, control is passed back to the calling block.

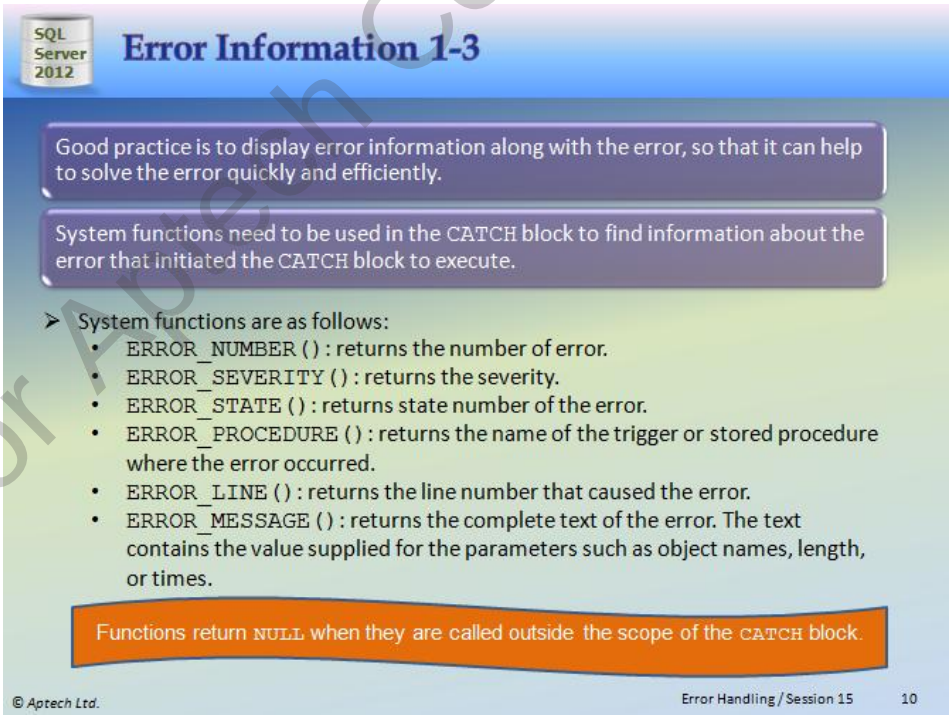
Explain the Code Snippet on slide 9 which demonstrates a simple example of TRY…CATCH statements.

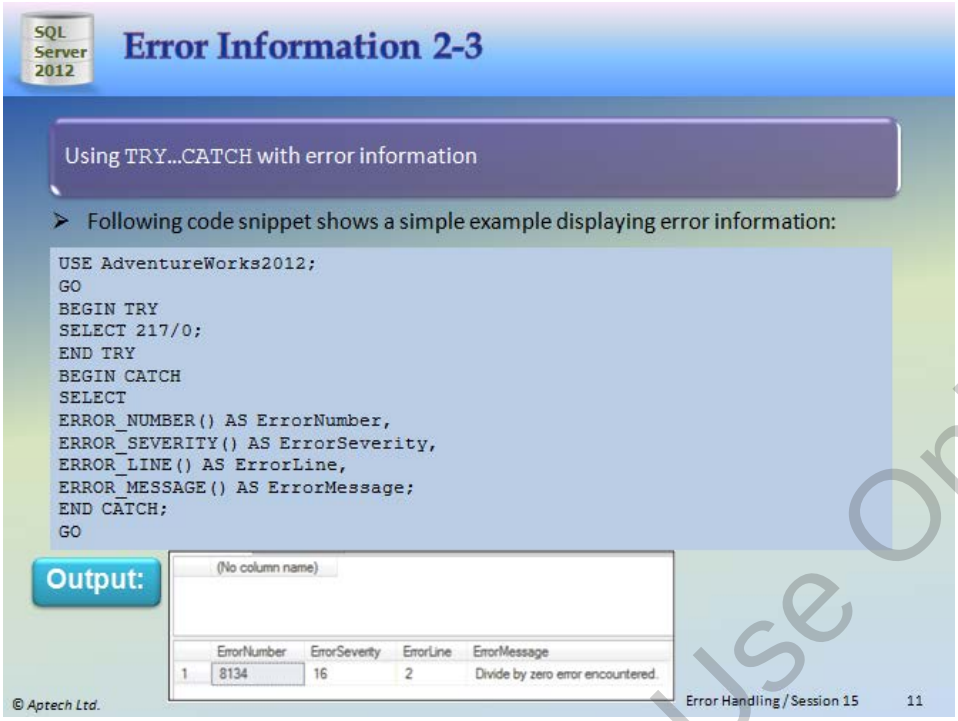
In this code, an attempt is made to divide a number by zero. This will cause an error hence, the TRY…CATCH statement is used here to handle the error.

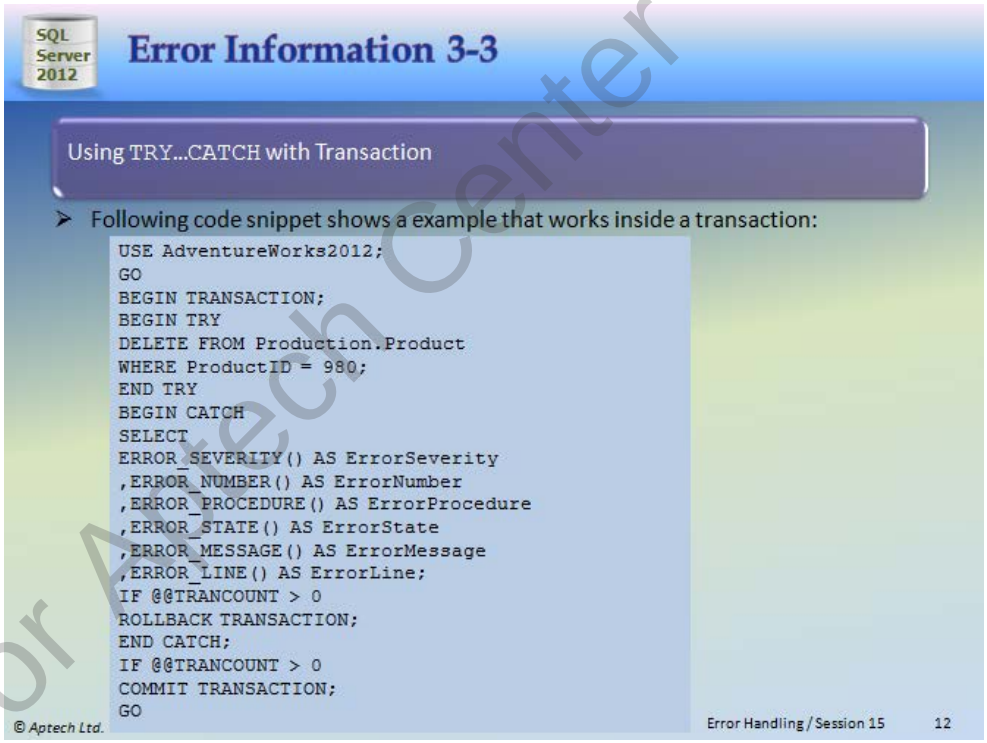
Mention, GOTO statements can be used to jump to a label inside the same TRY…CATCH block or to leave the TRY…CATCH block. The TRY…CATCH construct should not be used in a user-defined function.

Error Information

Slides 10 to 12







Using slides 10 to 12, explain the error information.

It is a good practice to display error information along with the error, so that it can help to solve the error quickly and efficiently.

To achieve this, system functions need to be used in the CATCH block to find information about the error that initiated the CATCH block to execute.

Explain the system functions mentioned on slide 10.

The functions return NULL when they are called outside the scope of the CATCH block.

Explain the Code Snippet on slide 11 which demonstrates a simple example displaying error information.

In this code, the SELECT statement will cause a divide-by-zero error that is handled using the TRY…CATCH statement. The error causes the execution to jump to the associated CATCH block within which the error information will be displayed.
Figure on slide 11 displays the result of the error information. The first resultset is blank because the statement fails.
Explain the Code Snippet on slide 12 demonstrates a TRY…CATCH block that works inside a transaction.
In this code, the TRY…CATCH block works within the transaction. The statement inside the TRY block generates a constraint violation error as follows:
The DELETE statement is conflicted with the REFERENCE constraint "FK_BillOfMaterials_Product_ProductAssemblyID". The conflict occurred in database "AdventureWorks2012", table "Production.BillOfMaterials", and column 'ProductAssemblyID'.

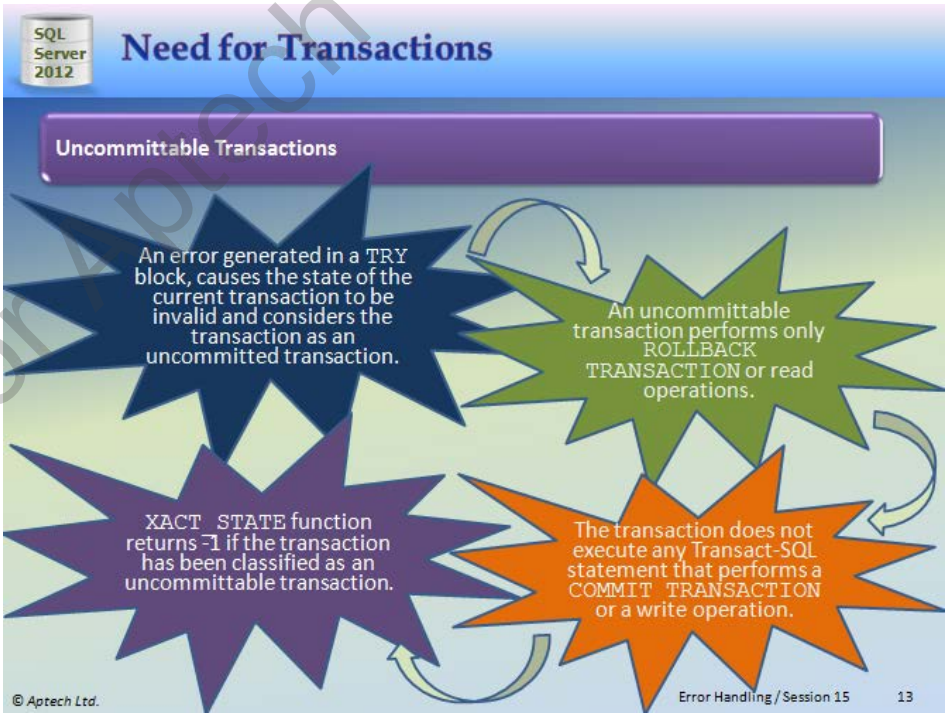
In-Class Question:

What is the system function name to get the stored procedure name in which the error is encountered?

Answer:
ERROR_PROCEDURE is the system function name to get the stored procedure name in which the error is encountered.

Need of Transactions

Slide 13



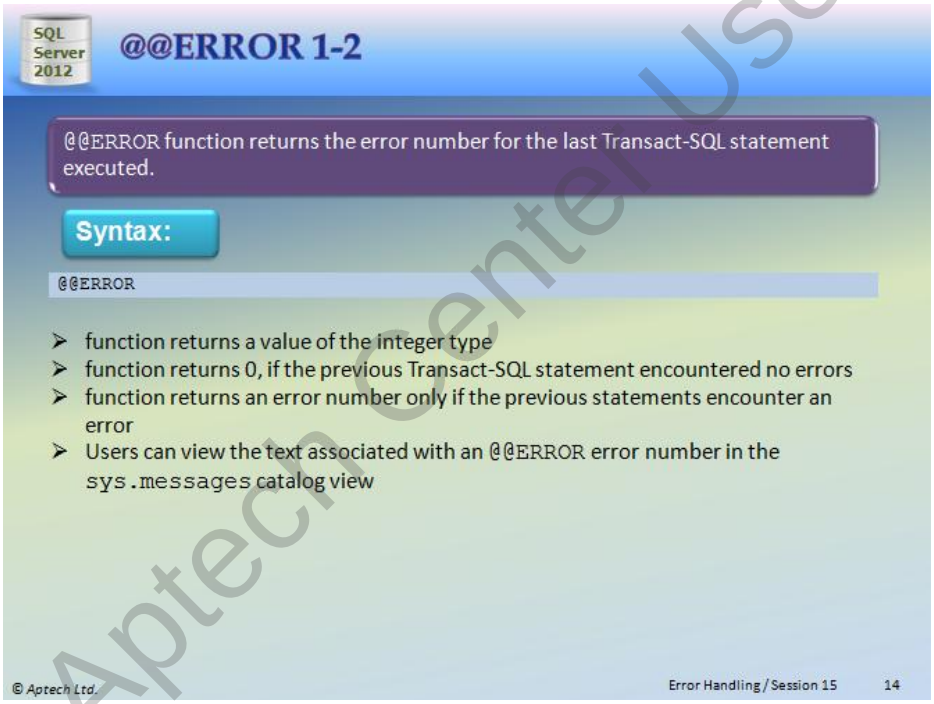
Using slide 13, explain the need of transactions.

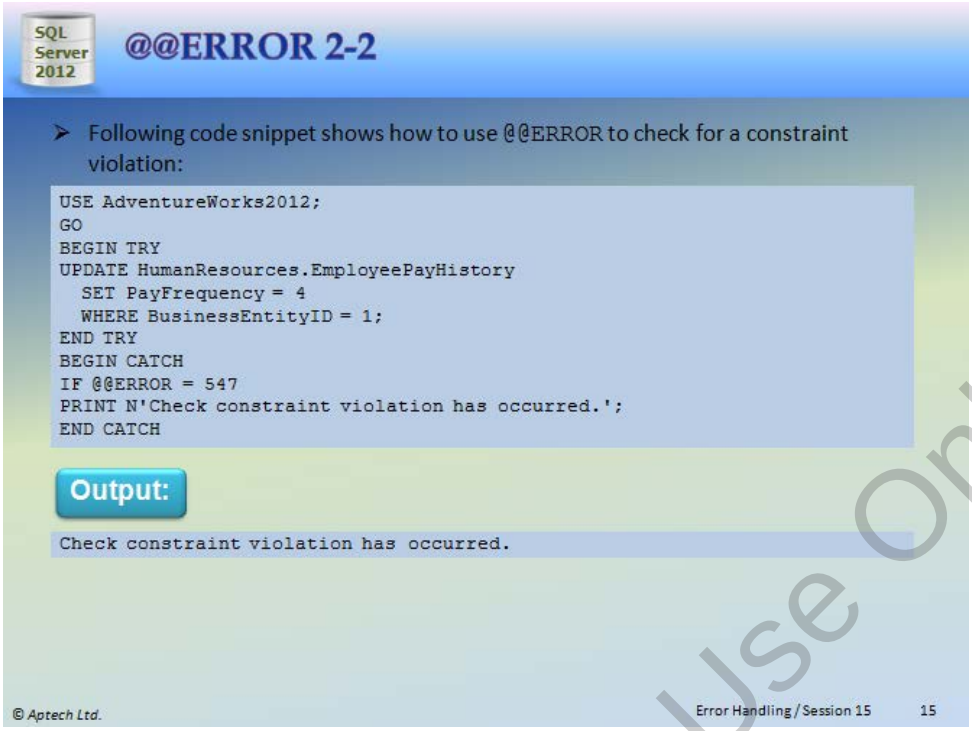
Mention, if an error is generated in a TRY block, it causes the state of the current transaction to be invalid and the transaction is considered as an uncommitted transaction. An uncommittable transaction performs only ROLLBACK TRANSACTION or read operations. The transaction does not execute any Transact-SQL statement that performs a COMMIT TRANSACTION or a write operation.

The XACT_STATE function returns -1 if the transaction has been classified as an uncommittable transaction. When a batch is completed, the Database Engine rolls back any uncommittable transactions. If no error messages are sent when the transaction enters the uncommittable state on completing the batch, then the error messages are sent to the client. This specifies that an uncommittable transaction is detected and rolled back.

@@ERROR

Slides 14 and 15





Using slides 14 and 15, explain the Transact-SQL functions.

The @@ERROR function returns the error number for the last Transact-SQL statement executed.

Explain the syntax for the @@ERROR function on slide 14.

The @@ERROR system function returns a value of the integer type. This function returns 0, if the previous Transact-SQL statement encountered no errors. It also returns an error number only if previous statements encounter an error. If an error is among the list of errors in the sys.messages catalog view including the value from the sys.messages.messages_id column for that error. Users can view the text associated with an @@ERROR error number in the sys.messages catalog view.

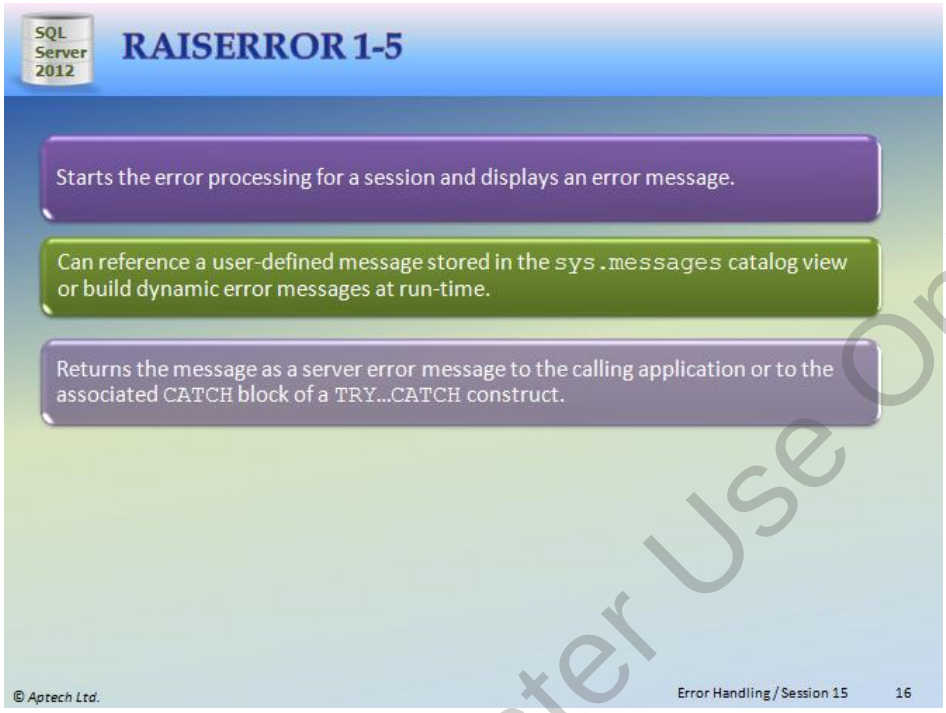
Explain the Code Snippet on slide 15 which demonstrates how to use @@ERROR to check for a constraint violation.

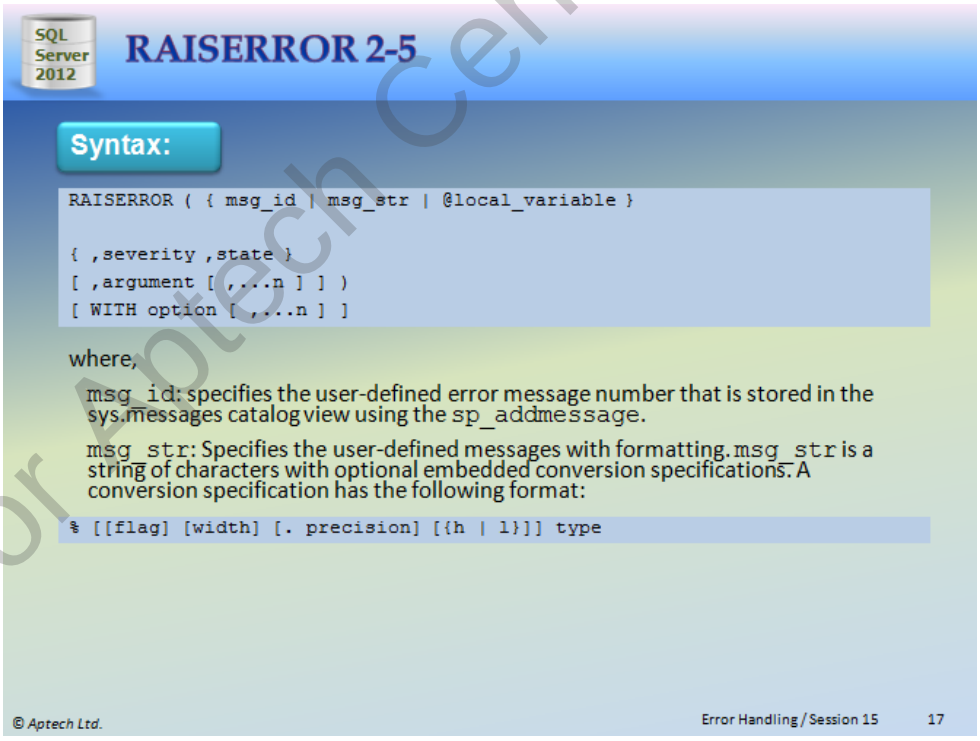
In this code, @@ERROR is used to check for a check constraint violation (which has error number 547) in an UPDATE statement.

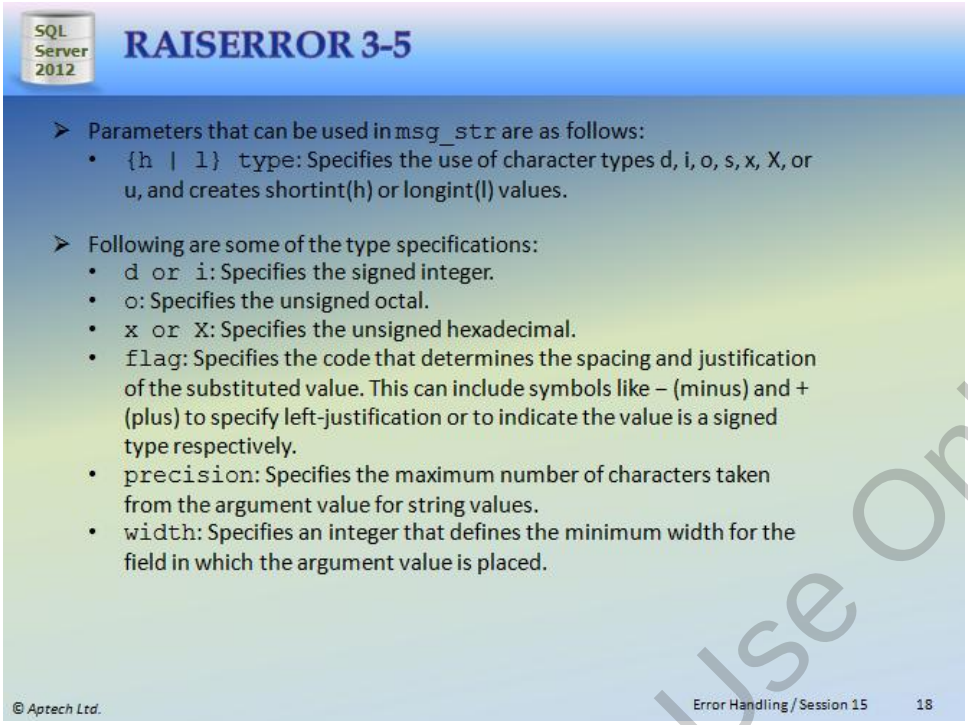
Also, explain the error message.

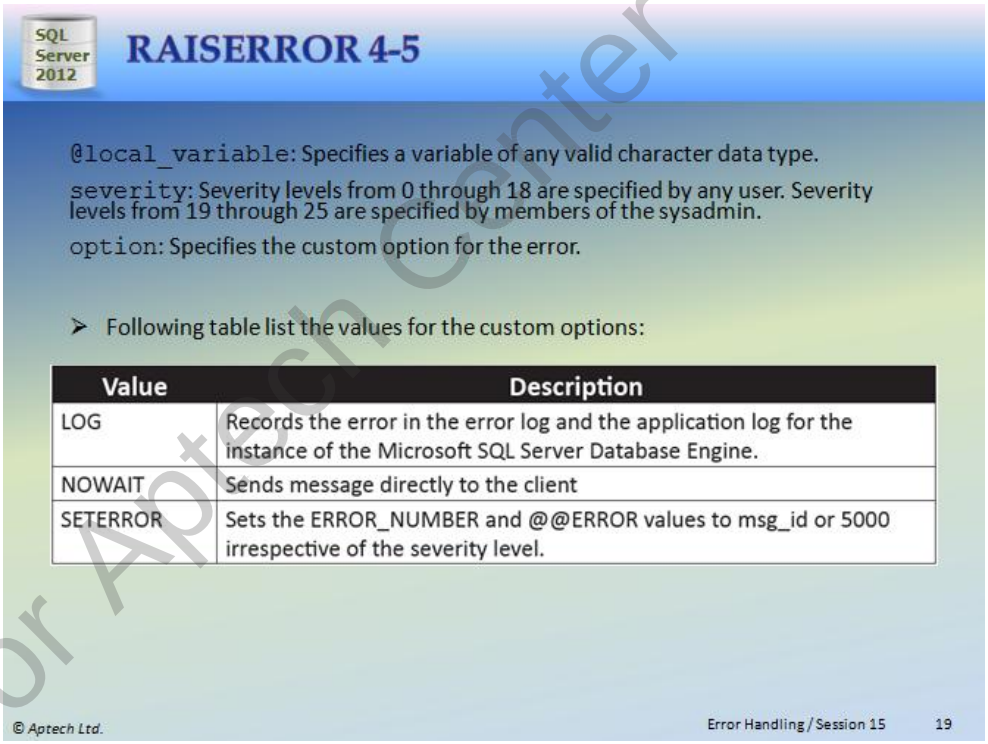
RAISEERROR

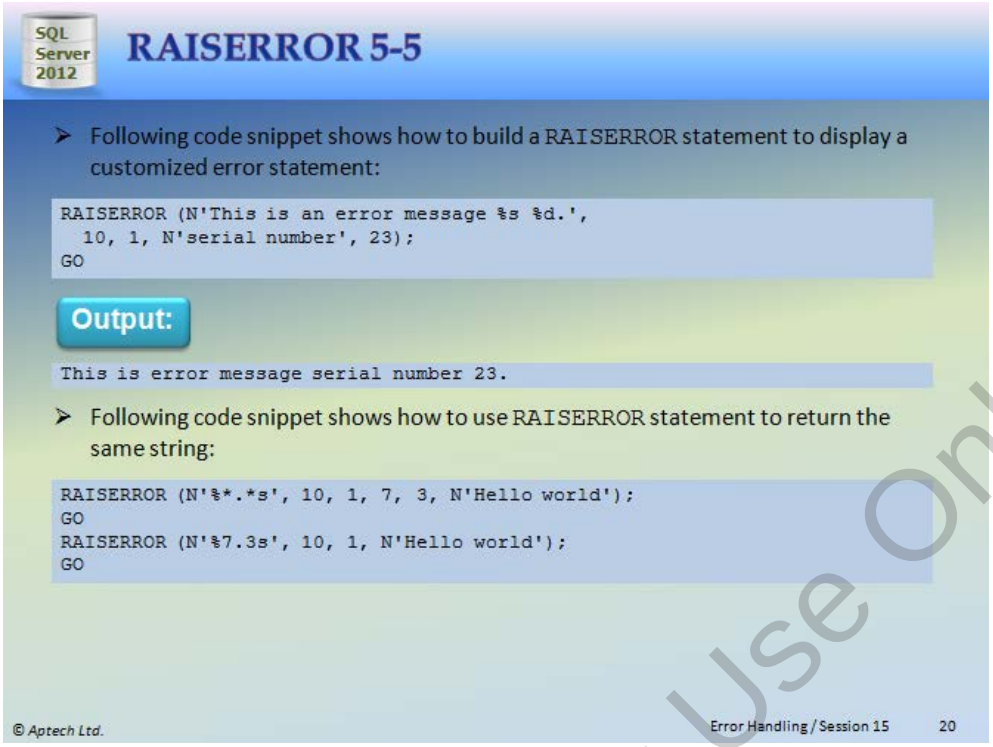
Slides 16 to 20











Using slides 16 to 20, explain the how to alter user-defined.

The RAISERROR statement starts the error processing for a session and displays an error message. RAISERROR can reference a user-defined message stored in the sys.messages catalog view or build dynamic error messages at run-time. The message is returned as a server error message to the calling application or to the associated CATCH block of a TRY…CATCH construct.

Explain the syntax for the RAISERROR statement on slide 17.

Also, explain the parameters that can be used in msg_str and its specifications using slide 18.

Table on slide 19 lists the values for the custom options.

When RAISERROR executes with a severity of 11 or higher in a TRY block, it will transfer the control to the associated CATCH block.

Explain the Code Snippet on slide 20 which demonstrates how to build a RAISERROR statement to display a customized error statement.

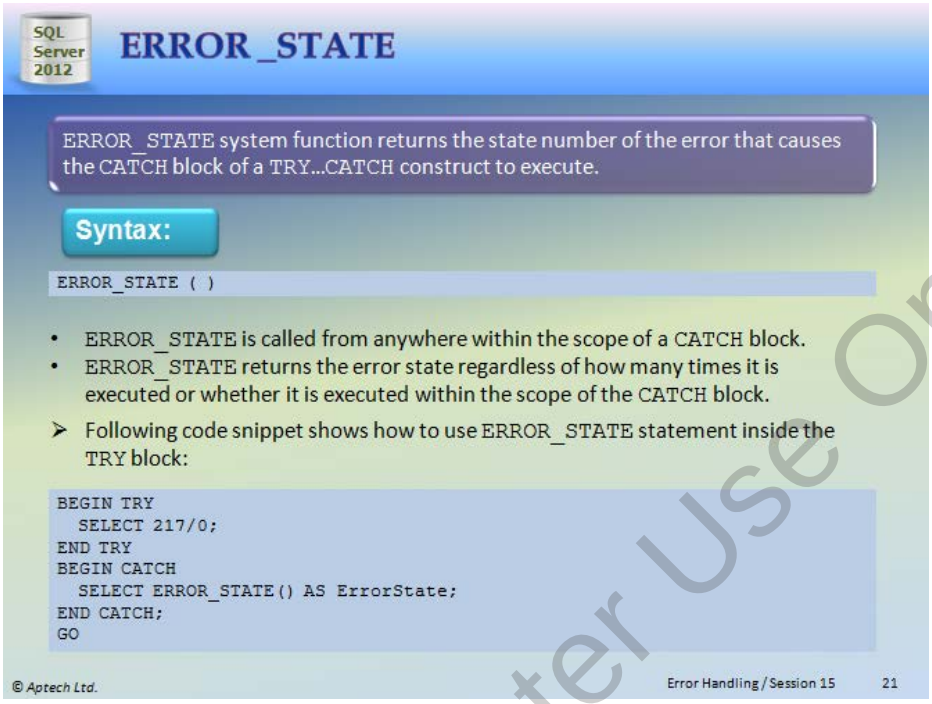
The Code Snippet displays the 'This is error message serial number 23'.

Explain the Code Snippet on slide 20 which demonstrates how to use RAISERROR statement to return the same string.

In the code, the RAISERROR statements return the same string, Hel. The first statement specifies the width and the precision values and the second statement specifies the conversion specification.

ERROR_STATE

Slide 21



Using slide 21, explain the ERROR_STATE.

The ERROR_STATE system function returns the state number of the error that causes the CATCH block of a TRY…CATCH construct to execute.

Explain the syntax for the ERROR_STATE system function on slide 21.

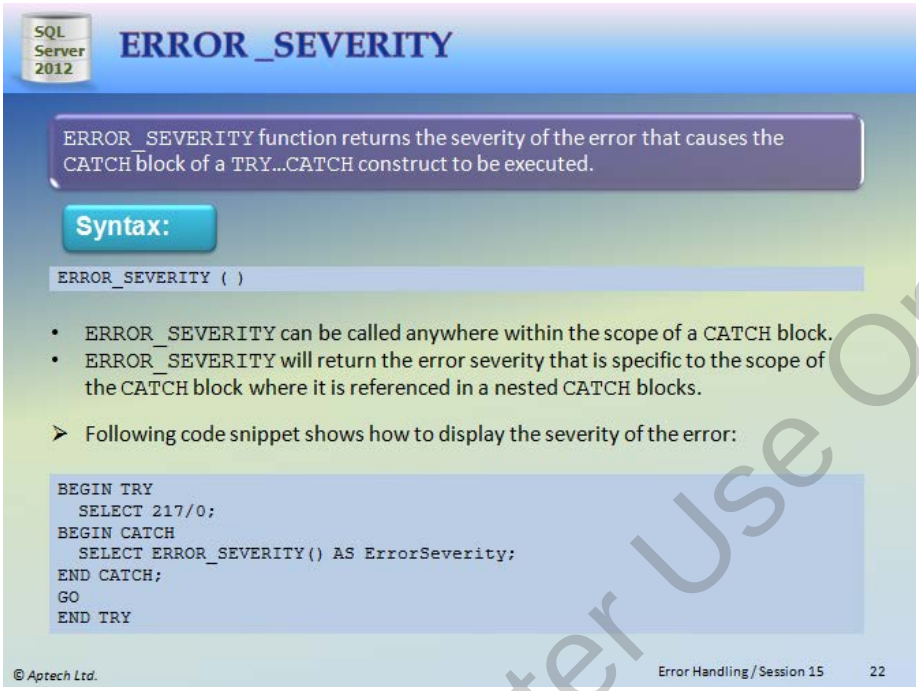
When called in a CATCH block, it returns the state number of the error message that caused the CATCH block to be run. This returns a NULL when it is called outside the scope of a CATCH block.

ERROR_STATE is called from anywhere within the scope of a CATCH block. ERROR_STATE returns the error state regardless of how many times it is executed or whether it is executed within the scope of the CATCH block. This is in comparison with the functions such as @@ERROR that only returns the error number in the statement directly after the one that caused error, or in the first statement of a CATCH block.

Explain the Code Snippet on slide 21 which demonstrates how to use ERROR_STATE statement inside the TRY block.

ERROR _SEVERITY

Slide 22



Using slide 22, explain the ERROR _SEVERITY.

The ERROR_SEVERITY function returns the severity of the error that causes the CATCH block of a TRY…CATCH construct to be executed.

Explain the syntax for ERROR_SEVERITY on slide 22.

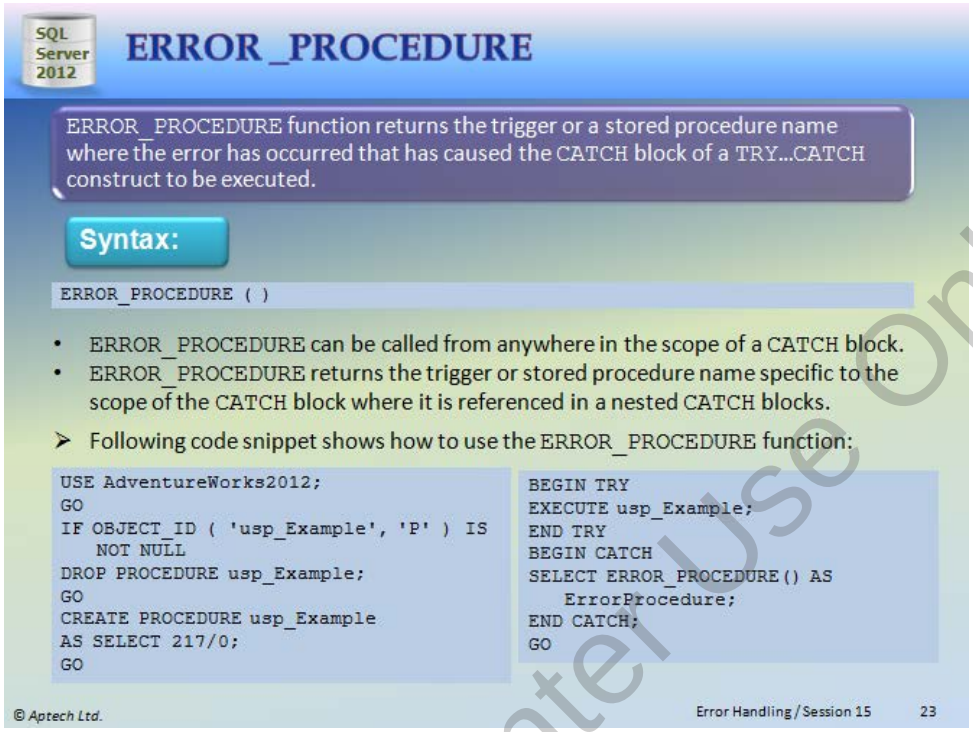
It returns a NULL value if called outside the scope of the CATCH block. ERROR_SEVERITY can be called anywhere within the scope of a CATCH block. In nested CATCH blocks, ERROR_SEVERITY will return the error severity that is specific to the scope of the CATCH block where it is referenced. Users can use the ERROR_SEVERITY function in a CATCH block.

Explain the Code Snippet on slide 22 which shows how to display the severity of the error.

In this code, an attempt to divide by zero generates the error and causes the CATCH block to display the severity error as 16.

ERROR _PROCEDURE

Slide 23



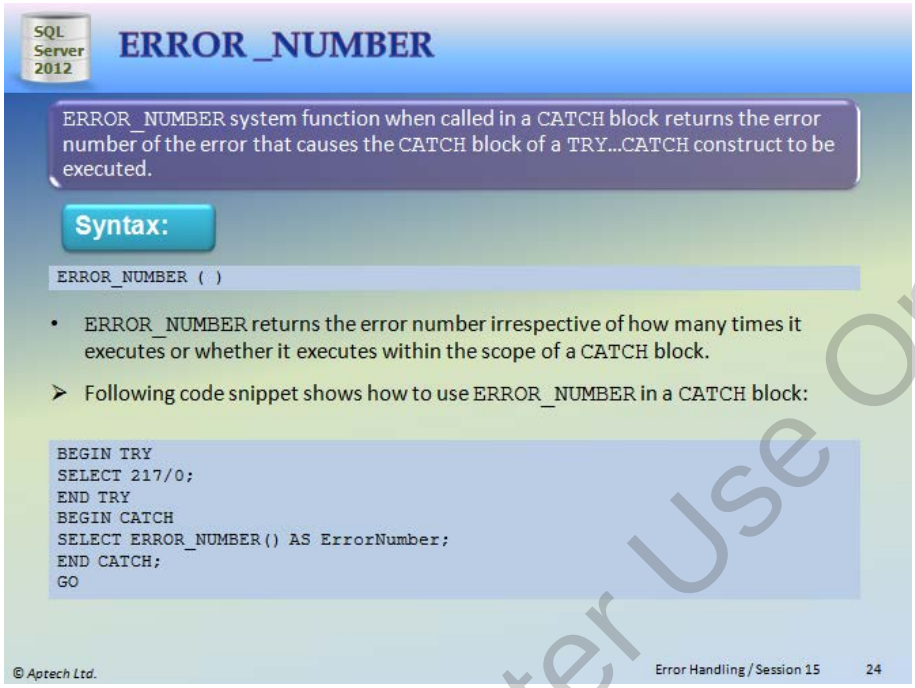
Using slide 23, explain the ERROR _PROCEDURE.
The ERROR_PROCEDURE function returns the trigger or a stored procedure name where the error has occurred that has caused the CATCH block of a TRY…CATCH construct to be executed. Explain the syntax of ERROR_PROCEDURE.

It returns the nvarchar data type. When the function is called in a CATCH block, it will return the name of the stored procedure where the error occurred. The function returns a NULL value if the error has not occurred within a trigger or a stored procedure.
ERROR_PROCEDURE can be called from anywhere in the scope of a CATCH block. The function also returns NULL if this function is called outside the scope of a CATCH block.
In nested CATCH blocks, the ERROR_PROCEDURE returns the trigger or stored procedure name specific to the scope of the CATCH block where it is referenced.

Explain the Code Snippet on slide 23 which shows the use of the ERROR_PROCEDURE function.
In this code, the stored procedure usp_Example generates a divide-by-zero error. The ERROR_PROCEDURE function accordingly returns the name of this stored procedure where the error has occurred.

ERROR_NUMBER

Slide 24



Using slide 24, explain the ERROR_NUMBER.

The ERROR_NUMBER system function when called in a CATCH block returns the error number of the error that causes the CATCH block of a TRY…CATCH construct to be executed.

Explain the syntax of ERROR_NUMBER on slide 24.

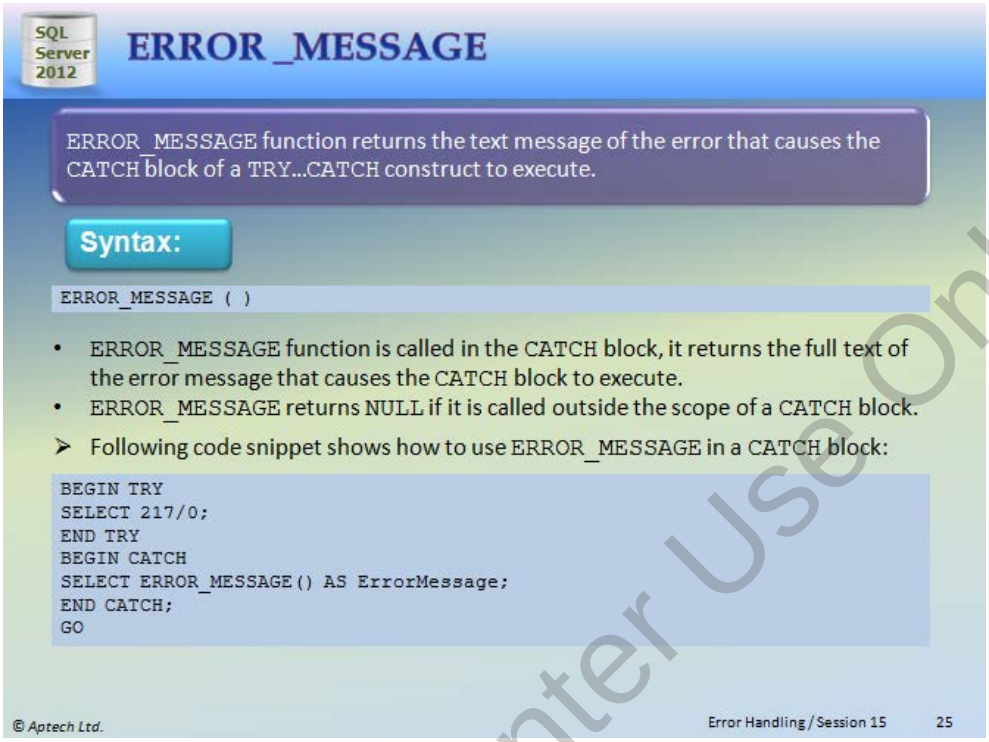
The function can be called from anywhere inside the scope of a CATCH block. The function will return NULL when it is called out of the scope of a CATCH block. ERROR_NUMBER returns the error number irrespective of how many times it executes or whether it executes within the scope of a CATCH block. This is different than the @@ERROR which only returns the error number in the statement immediately after the one that causes error, or the first statement of the CATCH block.

Explain the Code Snippet on slide 24 which demonstrates the use of ERROR_NUMBER in a CATCH block.

As a result of this code, the error number is displayed when the attempted division by zero occurs.

ERROR_MESSAGE

Slide 25



Using slide 25, explain the ERROR_MESSAGE.

The ERROR_MESSAGE function returns the text message of the error that causes the CATCH block of a TRY…CATCH construct to execute.

Explain the syntax of ERROR_MESSAGE.

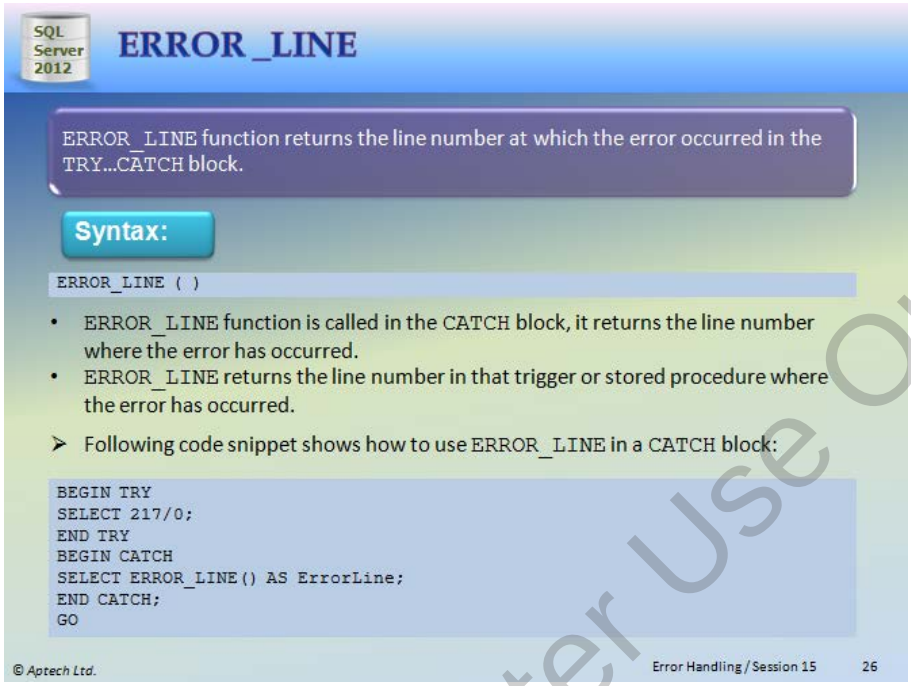
When the ERROR_MESSAGE function is called in the CATCH block, it returns the full text of the error message that causes the CATCH block to execute. The text includes the values that are supplied for any parameter that can be substituted such as object names, times, or lengths. It also returns NULL if it is called outside the scope of a CATCH block.

Explain the Code Snippet on slide 25 which demonstrates the use of ERROR_MESSAGE in a CATCH block.

In this code, similar to other examples, the SELECT statement generates a divide-by-zero error. The CATCH block displays the error message.

ERROR_LINE

Slide 26



Using slide 26, explain the ERROR_LINE.

The ERROR_LINE function returns the line number at which the error occurred in the TRY…CATCH block.

Explain the syntax of ERROR_LINE given on slide 26.

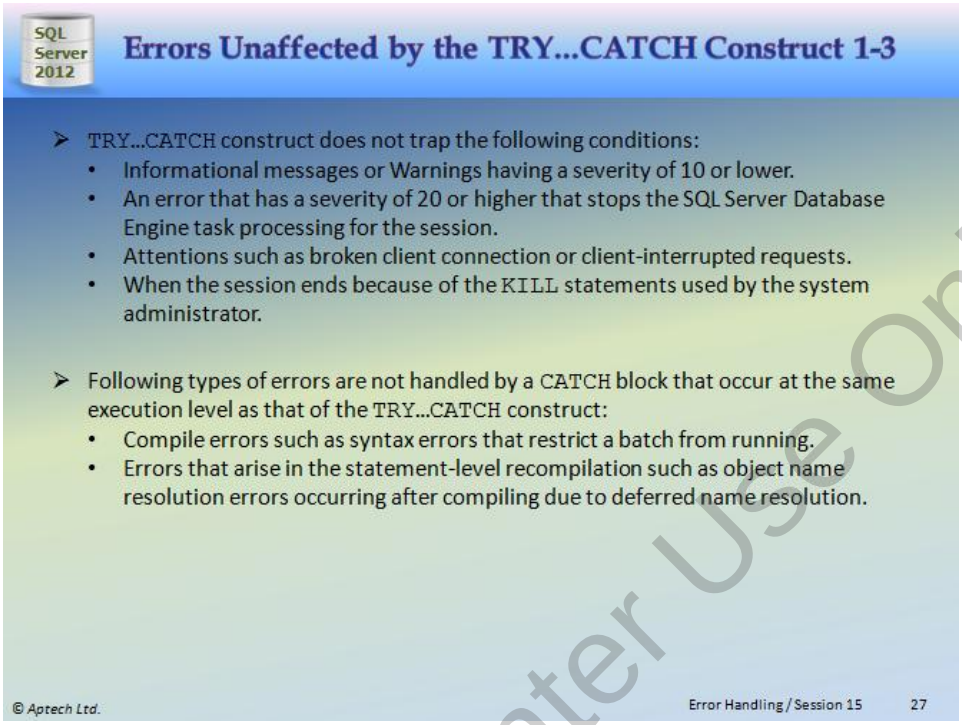
When this function is called in the CATCH block, it returns the line number where the error has occurred. If the error has occurred within a trigger or a stored procedure, it returns the line number in that trigger or stored procedure. Similar to other functions, this function returns a NULL if it is called outside the scope of a CATCH block.

Explain the Code Snippet on slide 26 which demonstrates the use of ERROR_LINE in a CATCH block.

As a result of this code, the line number at which the error has occurred will be displayed.

Errors Unaffected by the TRY..CATCH Construct

Slides 27 to 29



Using slides 27 to 29, explain the errors unaffected by the TRY..CATCH construct.
The TRY…CATCH construct does not trap the following conditions:

- Informational messages or Warnings having a severity of 10 or lower
- An error that has a severity of 20 or higher that stops the SQL Server Database Engine task processing for the session. If errors occur that have severity of 20 or higher and the database connection is not interrupted, the TRY…CATCH will handle the error
- Attentions such as broken client connection or client-interrupted requests
- When the session ends because of the KILL statements used by the system administrator

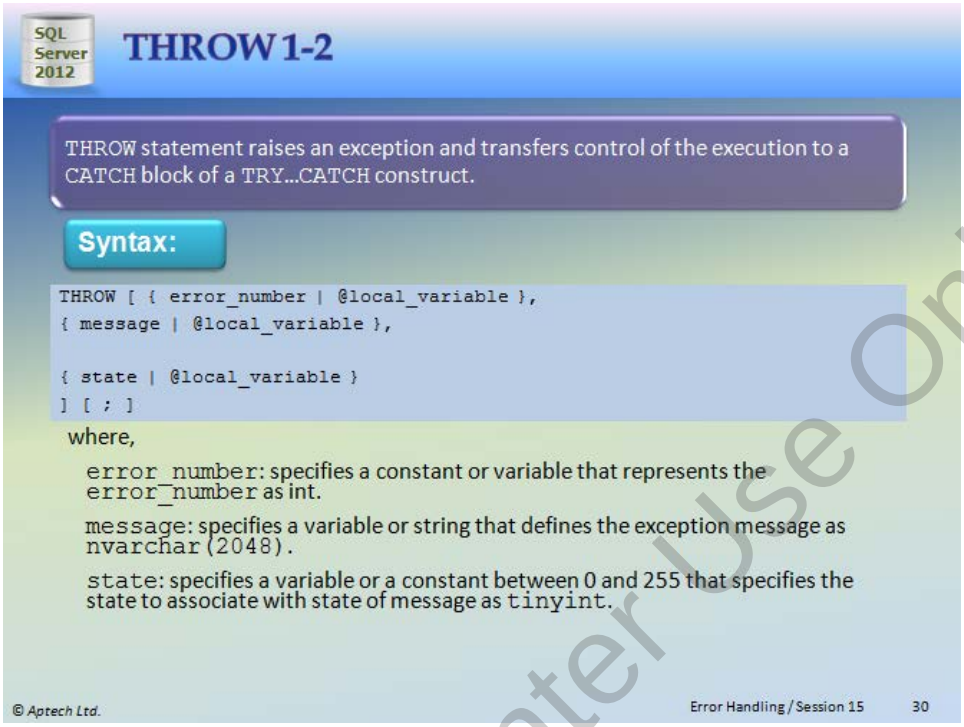
The following types of errors are not handled by a CATCH block that occur at the same execution level as that of the TRY…CATCH construct:

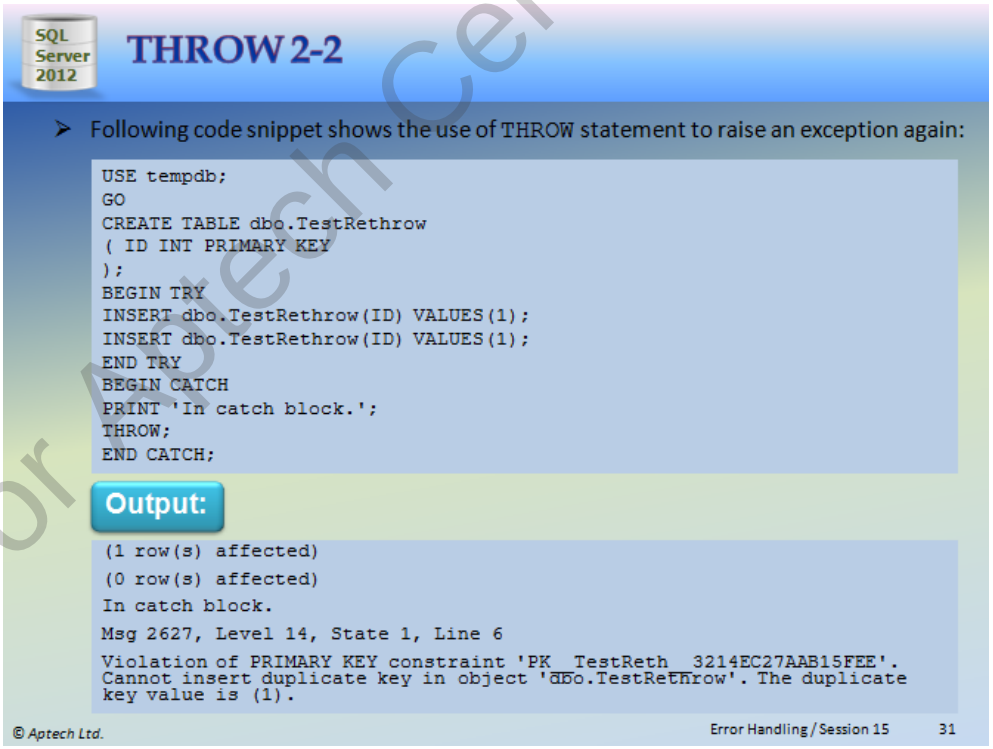
- Compile errors such as syntax errors that restrict a batch from running
- Errors that arise in the statement-level recompilation such as object name resolution errors occurring after compiling due to deferred name resolution

Explain the Code Snippet on slide 28 which demonstrates how an object name resolution error is generated by the SELECT statement.
This code will cause the object name resolution error in the SELECT statement. It will not be caught by the TRY…CATCH construct. Running a similar SELECT statement inside a stored procedure causes the error to occur at a level lower than the TRY block. The error is handled by the TRY…CATCH construct.
Explain the Code Snippet on slide 29 which demonstrates how the error message is displayed in such a case.

THROW

Slides 30 and 31





Using slides 30 and 31, explain the THROW statement.

The THROW statement raises an exception and transfers control of the execution to a CATCH block of a TRY…CATCH construct.

Explain the syntax of the THROW statement shown on slide 30.

Explain the Code Snippet on slide 31 which demonstrates the use of THROW statement to raise an exception again.
In this code, the THROW statement is used to raise once again the exception that had last occurred.
Explain the outcome of the code 31.

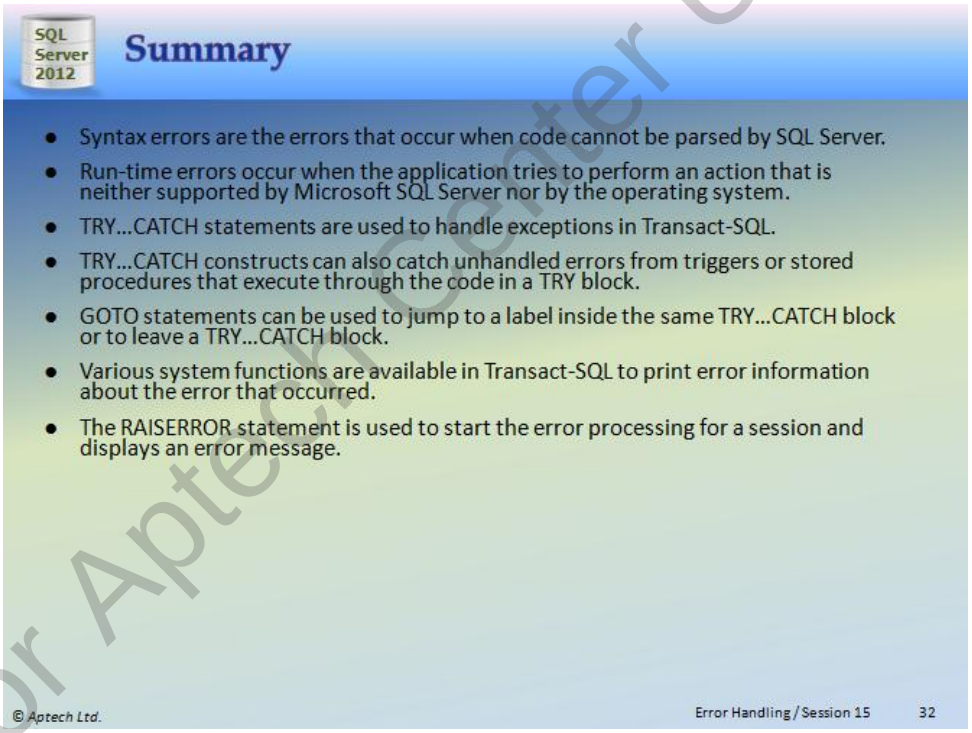
In-Class Question:

What is the use of THROW statement?

Answer:
The THROW statement raises an exception and transfers control of the execution to a CATCH block of a TRY…CATCH construct.

Summarize Session

Slide 32



Using slide 32, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session. Tell the students pointers of the session. This will be a revision of the current session and it will be related to the next session. Explain each of the following points in brief. Tell them that:

- Syntax errors are the errors that occur when code cannot be parsed by SQL Server.
- Run-time errors occur when the application tries to perform an action that is neither supported by Microsoft SQL Server nor by the operating system.
- TRY…CATCH statements are used to handle exceptions in Transact-SQL.
- TRY…CATCH constructs can also catch unhandled errors from triggers or stored procedures that execute through the code in a TRY block.

- GOTO statements can be used to jump to a label inside the same TRY…CATCH block or to leave a TRY…CATCH block.
- Various system functions are available in Transact-SQL to print error information about the error that occurred.
- The RAISERROR statement is used to start the error processing for a session and displays an error message.

15.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 16: Introduction to SQL Server 2016

16.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

16.1.1 Objectives

By the end of this session, learners will be able to:

- Describe an overview of SQL Server 2016
- Outline the new features of SQL Server 2016
- List and describe the different editions of SQL Server 2016

16.1.2 Teaching Skills

To teach this session, you should be well versed with relational database concepts and SQL Server database. You should also be familiar with the SQL Server 2016 database environment, the terminologies used, the new features introduced, enhancements of existing features, and similar other concepts.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
You may go through MSDN library and MSDN blogs before starting the session for gaining additional information. Some hyperlinks are provided in this guide for your reference. It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

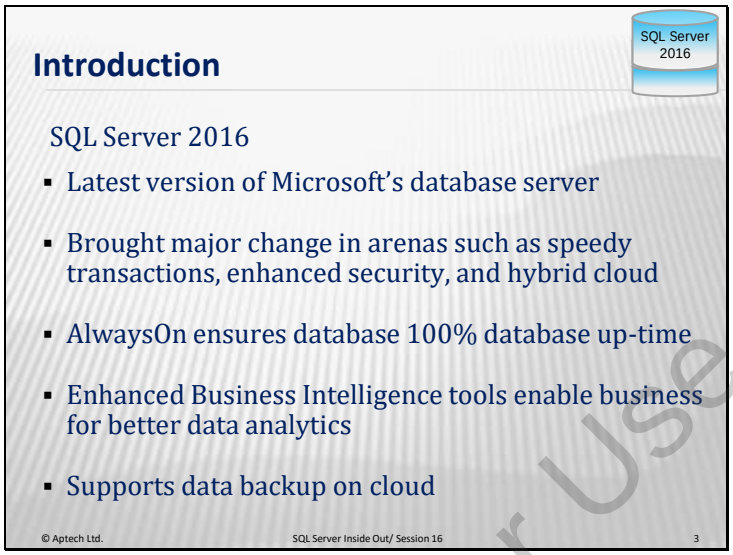
Overview of the Session

Give the students an overview of the current session in the form of session objectives. You may briefly discuss about previous versions of SQL Server. Briefly tell about SQL Server 2016 release date and version. Show the students slide 2 of the presentation.

16.2 In-Class Explanations

Slide 3

Let us understand the need to start using SQL Server 2016.



Using slide 3, give a brief introduction to SQL Server 2016 in general and then explain why it is beneficial to upgrade to SQL Server 2016 or start using SQL Server 2016.

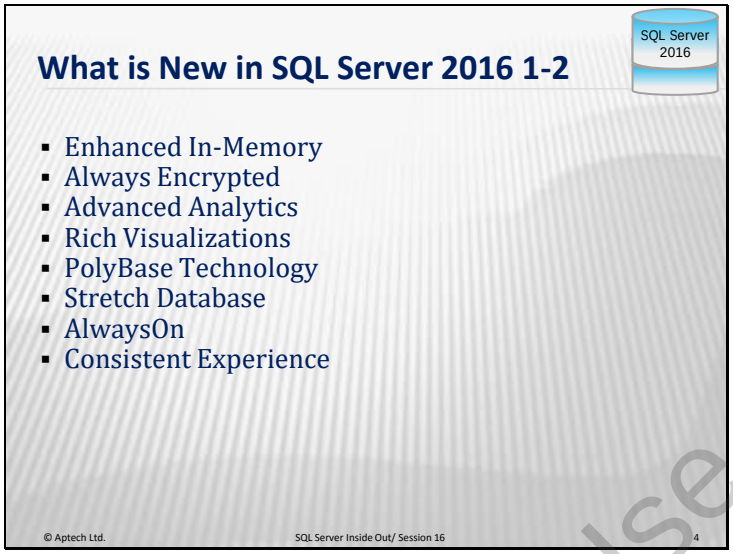
Additional Information:

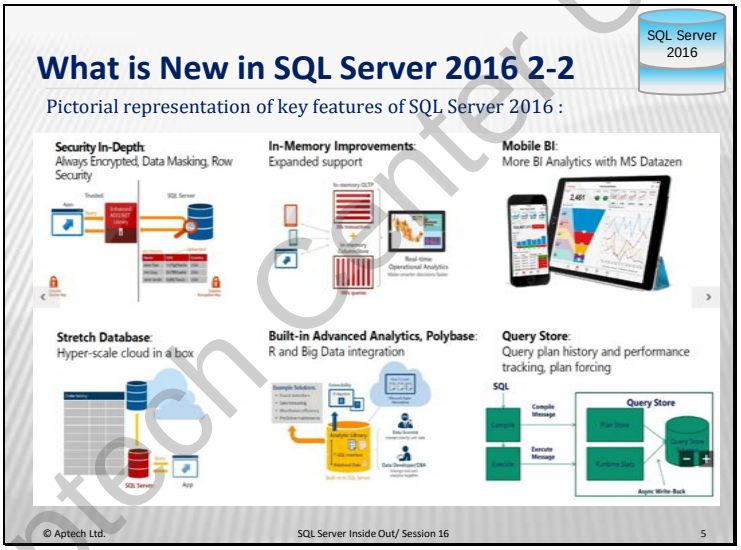
For more information on introduction to SQL Server 2016, you can visit the following link:

https://blogs.technet.microsoft.com/dataplatforminsider/2016/02/03/technical-overview-sql-server-2016-community-technology-preview-3-3-2/

Slides 4 and 5

Let us just take a brief look at the new features of SQL Server 2016.





Using slide 4, list out the new features of SQL Server 2016. Explain in few lines the functionalities and uses of each of the new features.. Use slide 5 to give an insight into the new features of SQL Server 2016 pictorially.

In-Class Question:

After you finish explaining slide 5, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

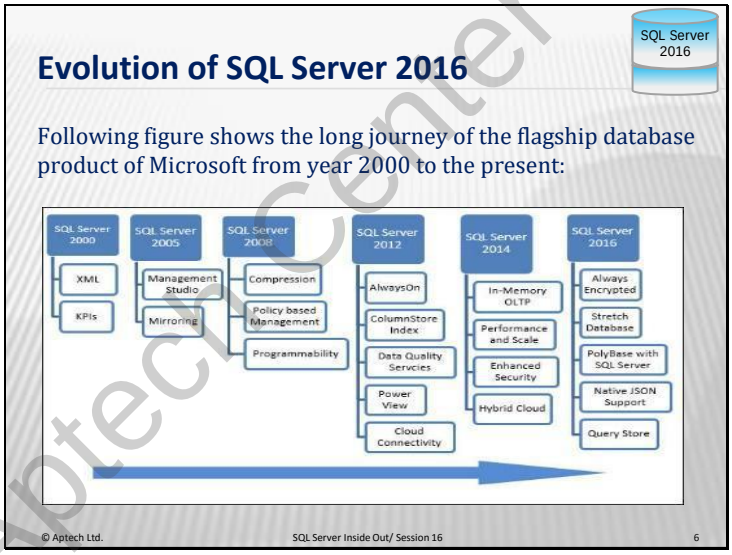
What is PolyBase?

Answer: PolyBase technology is a feature, which uses Transact SQL for querying and enables the users to access data from Hadoop and such noSQL data storages. This simplifies relational and non-relational data management.

Additional Information:
For more information on new features of SQL Server 2016, you can visit the following link:

https://msdn.microsoft.com/en-us/library/bb500435.aspx

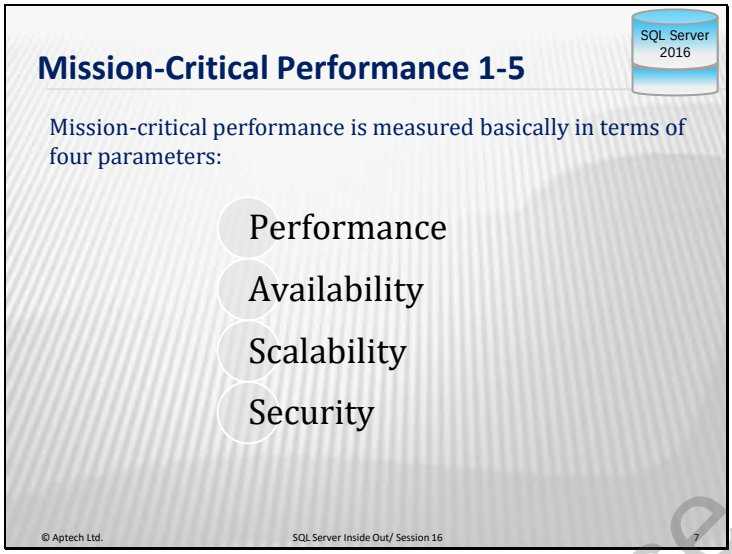
Slide 6
Let us understand the journey of SQL Server 2016.

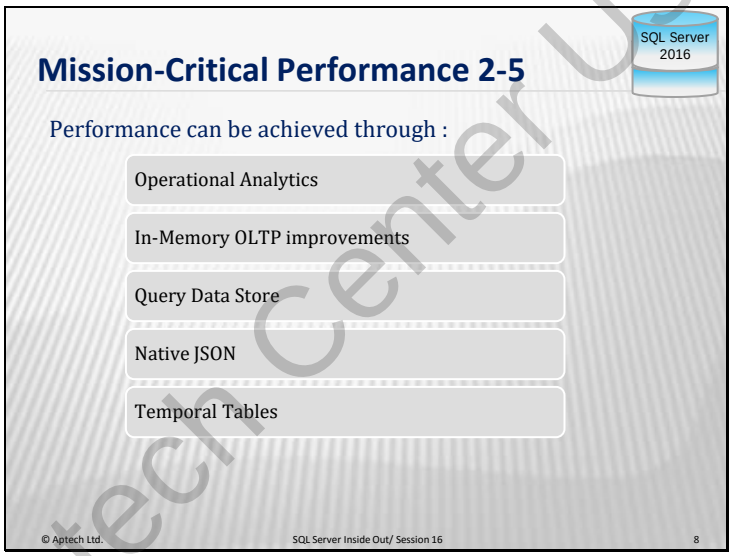


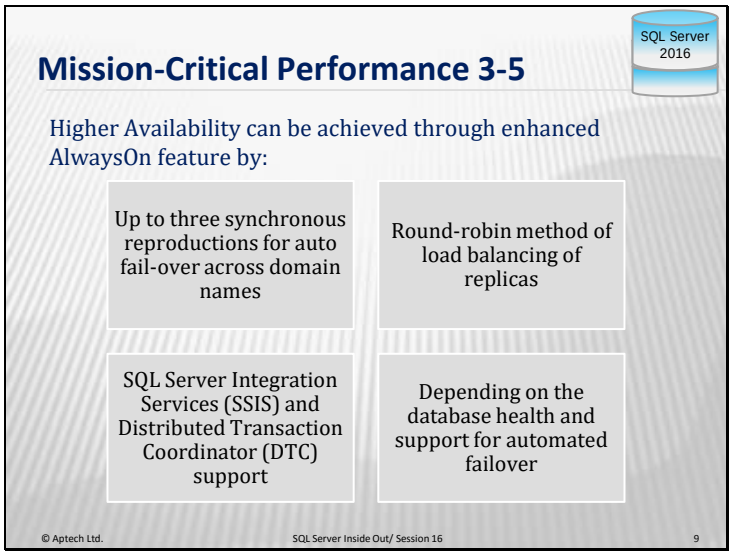
Using slide 6, explain how the SQL Server database has emerged starting from the year 2000 till date. Mention the key milestones in each version as shown on slide 6.

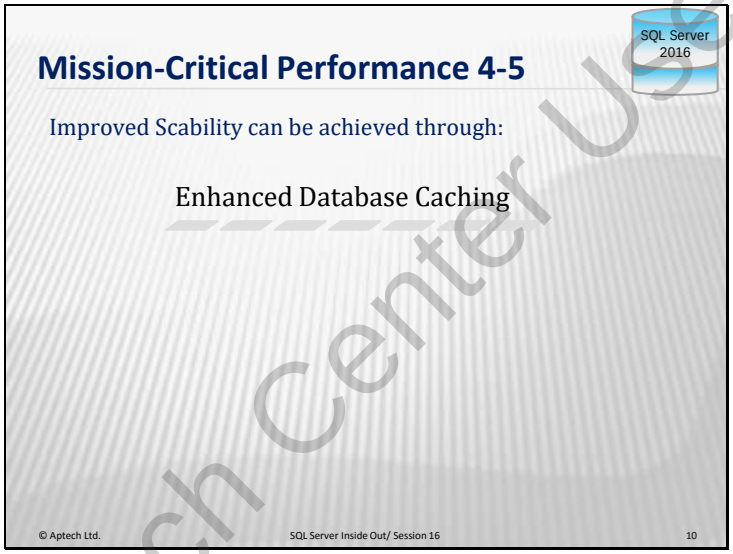
Slides 7 to 11

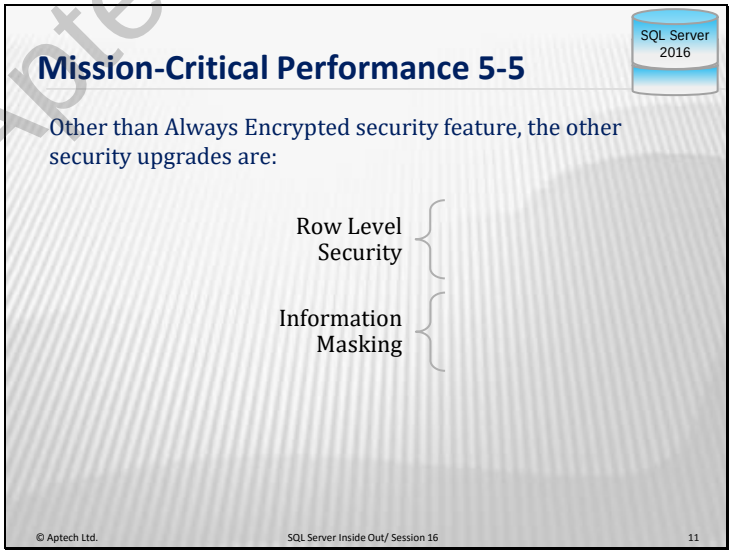
Let us explore the Mission-Critical Performance feature of SQL Server 2016.











Firstly, introduce the concept of mission-critical performance to students. Tell the students that, SQL Server 2016 has set a standard by offering database uptime along with enterprise-level security. As the data in the database becomes massive, it becomes more

complicated to manage the data. This in turn means that the enterprises should adapt a different approach for mission-critical capabilities.

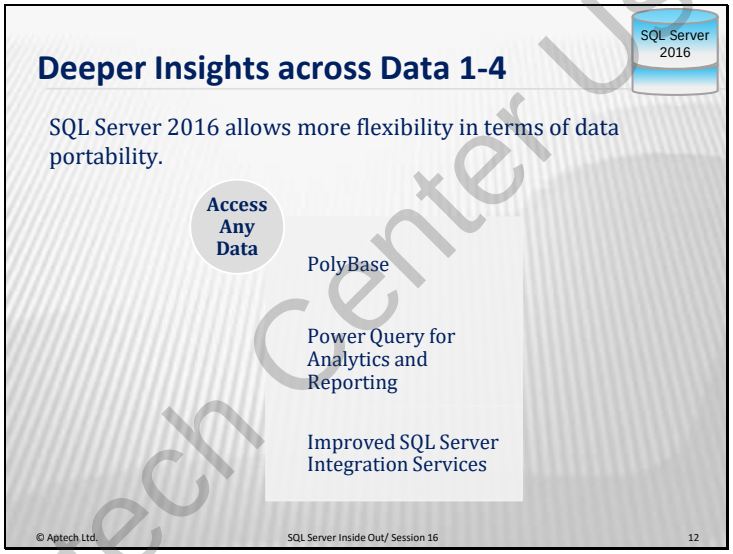
Using slide 7, list out the four basic parameters on which the mission-critical performance is gauged. Using slides 8 to 11, explain how performance, higher availability, improved scalability, and enhanced security can respectively be achieved through the parameters mentioned in each of the slides.

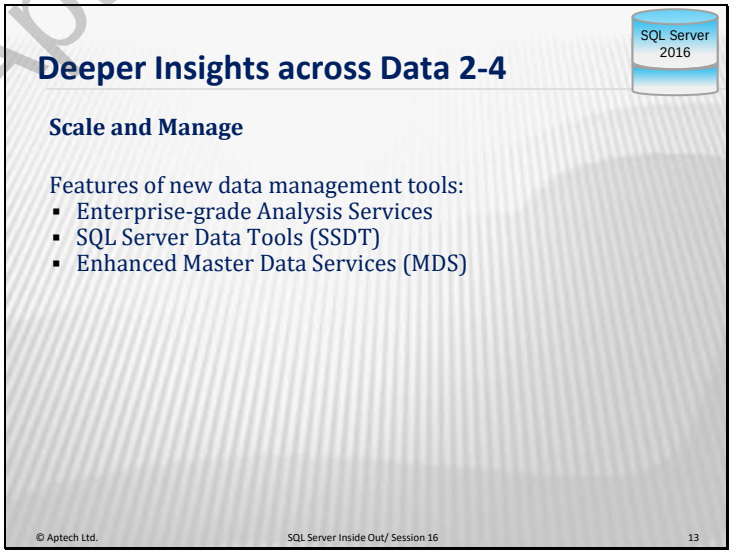
Additional Information:

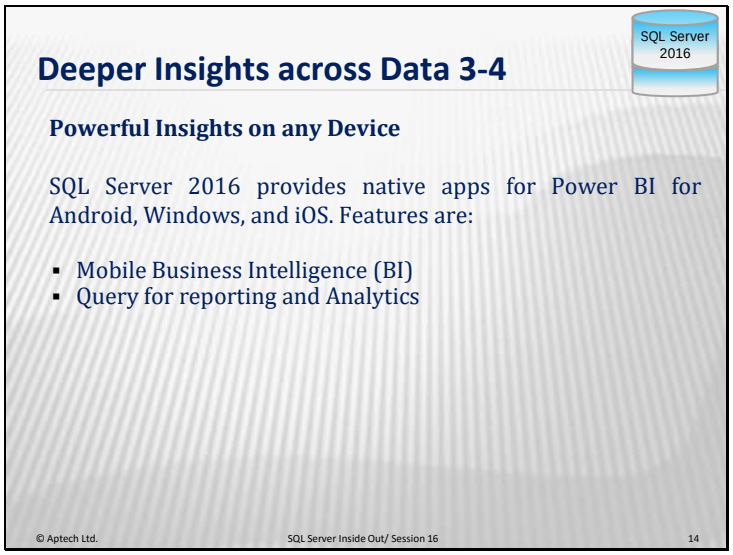
For more information on mission-critical performance, you can visit the following link: http://blogs.microsoft.com/blog/2016/03/10/sql-server-2016-the-database-for-mission-critical-intelligence/#sm.00001dlxje5dwjdlmpxhn7emfte9r

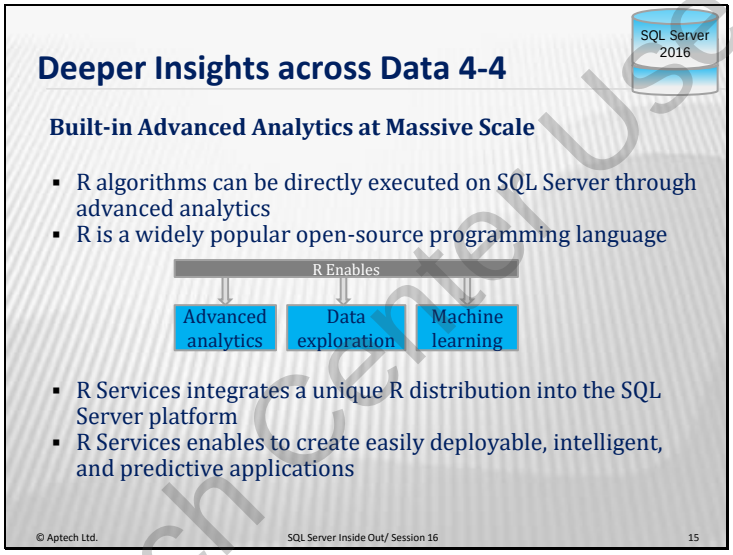
Slides 12 to 15

Let us now explore the flexibility provided by SQL Server 2016 for data porting.









Using slide 12, explain the Access Any Data option. You can introduce features such as PolyBase, PowerQuery, and Integration Services that enable portability in various ways. As users these days tend to work across platforms and perform a variety of tasks including querying and reporting, they need to have features that can help them in this regard.

Use slide 13 to explain scalability. Tell that, enhancements to scalability feature is achieved through the memory-optimized tables that are on stored disk. Multiple concurrent threads, multi-threaded recovery are some of the enhancements to scalability. Also, introduce the data management tools such as SSDT and MDS. Tell the students that SSDT is a development tool that allows you to design and deploy SQL content and it is as simple as developing an application with any Microsoft development tool. On top of it, this tool can be downloaded for free. MDS of SQL Server 2016 comes with a new layout for Web. New features such as controlled logging configuration, data compression for entities, enhanced security functions, comprehensive security model, and creation of jobs for log and index maintenance are introduced.

Use slide 14 to explain about the BI and reporting and analytics. Tell the students that Power BI is a cloud-based Business Intelligence (BI) technology by Microsoft that is part of

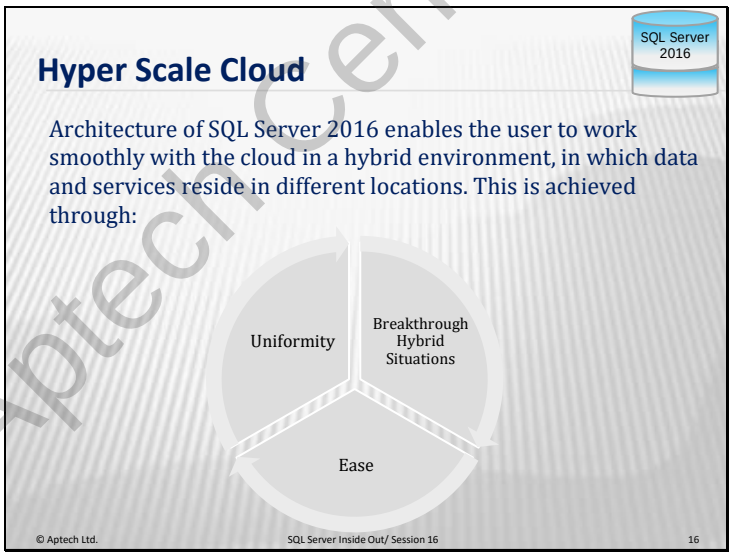
the Office 365 suite, the cloud-based suite of productivity applications. It aids business users in accessing and analyzing data in user-friendly formats, such as dashboards, reports, and datasets. Power BI is designed to enable business and enterprise users to gain insights from their data.

Use slide 15 to introduce R Services. Tell the students that R is a programming language and environment that supports statistical computing and graphics.

Explain in brief about R, highlighting how it helps in building analytic applications in collaboration with SQL Server 2016.

Additional Information:
For more information on deeper insights across data, visit the following links:
https://sqlserverjunction.wordpress.com/2016/03/24/whats-new-in-sql-server-2016-deep-insights-across-data/
http://searchsqlserver.techtarget.com/feature/Four-tips-on-boosting-SQL-Server-scalability
https://blogs.technet.microsoft.com/dataplatforminsider/2015/10/29/microsoft-business-intelligence-our-reporting-roadmap/

Slide 16
Let us understand the Hyper Scale Cloud.

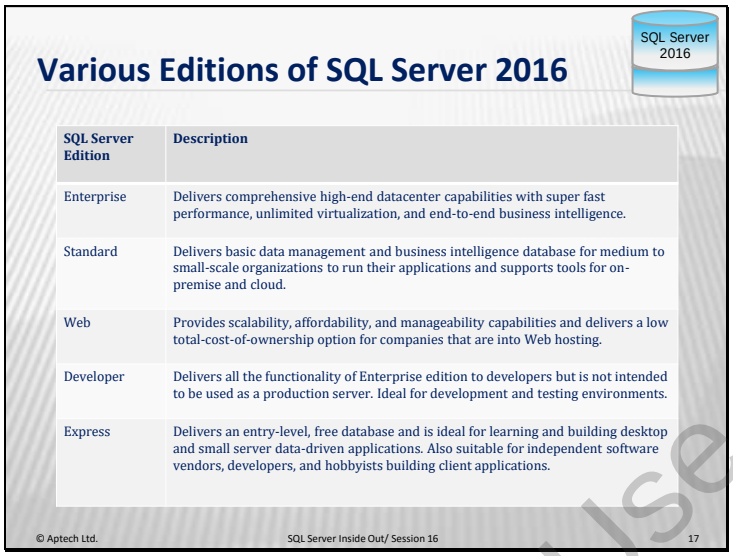


Start with explaining the concept of cloud to the students. Then give a brief introduction about SQL Servers Cloud capabilities.

Tell the students that starting from SQL Server 2012 version, you can see continual enhancements to the hyper-scale cloud capabilities. SQL Server database is aiming towards simplest integration with data sources other than SQL Server, also keeping in mind easy administration. The cloud features added in SQL Server 2014 have been enhanced significantly in SQL Server 2016. SQL Server can be deployed in public, private, or hybrid cloud environment thus empowering SQL Server for cloud integration.

Slide 17

Let us understand the various editions of SQL Server 2016.



Use the table given on slide 17 to brief about various SQL Server editions and their salient features.

Additional Information:
For additional information on SQL Server 2016 editions, you can visit the following link:
https://msdn.microsoft.com/en-us/library/ms144275.aspx

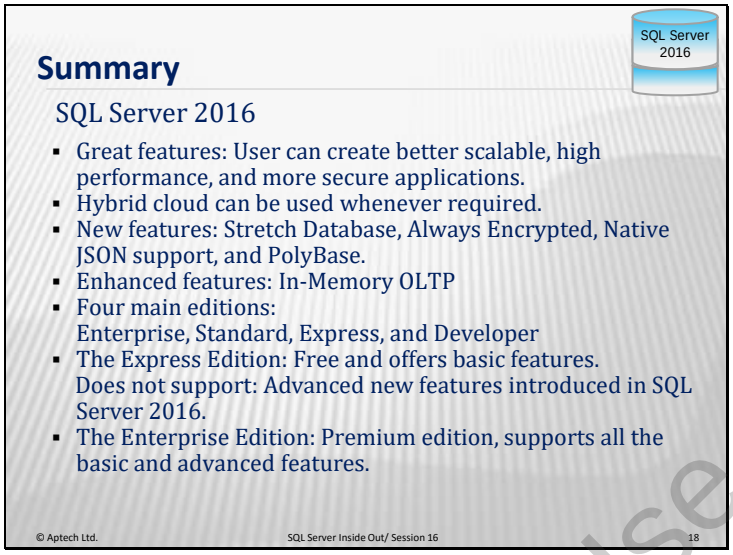
In-Class Question:

After you finish explaining slide 17, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

Which one of the SQL Server editions is most suited to be used as a development and testing server?
Answer: SQL Server Developer Edition.

Slide 18

Let us summarize the session.



Using slide 18 summarize the session. Explain each of the following points in brief. Tell them that:

- SQL Server 2016 is a database that has some excellent features using which developers can develop scalable, more secure applications with high performance.
- The use of hybrid cloud makes it more flexible.
- Point out the new features.
- Point out the enhanced features.
- Brief out editions of SQL Server 2016.
- One or two sentences about express edition as it is free.
- One or two sentences about enterprise edition.

16.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session.

Tips: You can also check the **Articles/Blogs/Expert Videos** uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 17: New Features of SQL Server 2016

17.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

17.1.1 Objectives

By the end of this session, learners will be able to:

- List new features of SQL Server 2016
- Explain Real-time Operational Analytics
- Describe native JSON support
- Explain AlwaysOn feature
- Describe enhancements made to In-Memory OLTP (Hekaton)

17.1.2 Teaching Skills

To teach this session, you should be familiar with new features of SQL Server 2016. You should know the concepts of OLAP, SSAS, Column Store Indexes, and such concepts. You should be well versed with features such as real-time operational analytics, JSON support, AlwaysOn, and In-Memory OLTP enhancements.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

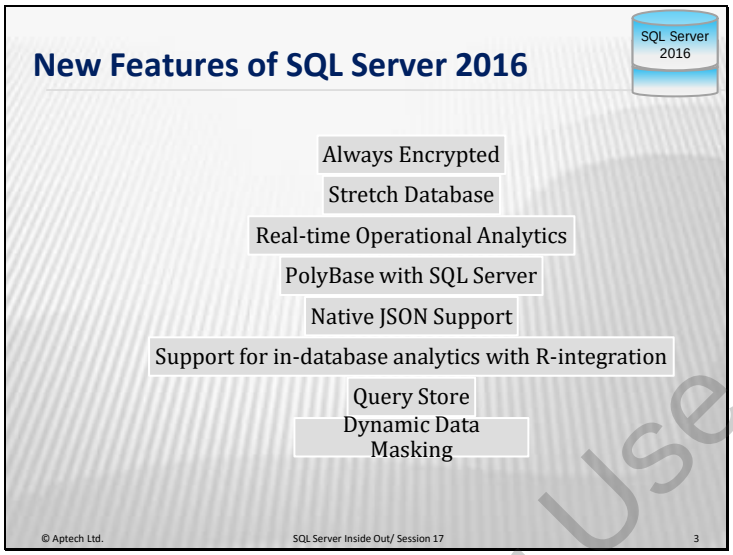
Overview of the Session

Give the students an overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

17.2 In-Class Explanations

Slide 3

Let us explore the new features of SQL Server 2016.



Using slide 3, list out the new features of SQL Server 2016. Tell the students that various features including real-time operational analytics and native JSON support have been introduced in SQL Server 2016.

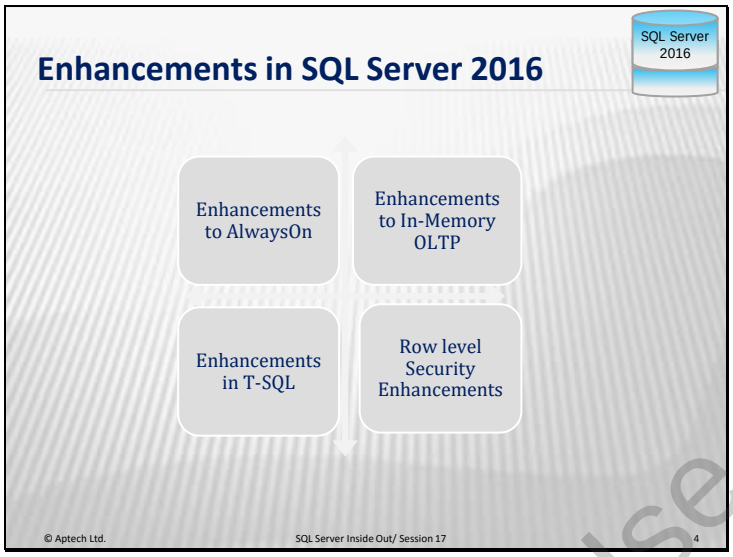
Additional Information:

For more information on new features of SQL Server 2016, you can visit the following links:

http://www.databasejournal.com/features/mssql/slideshows/10-new-features-worth-exploring-in-sql-server-2016.html
http://www.infoworld.com/article/3013601/application-development/new-features-in-sql-server-2016.html

Slide 4

Let us explore enhancements to existing features in SQL Server 2016.



Using slide 4, list out the enhanced features of SQL Server 2016.

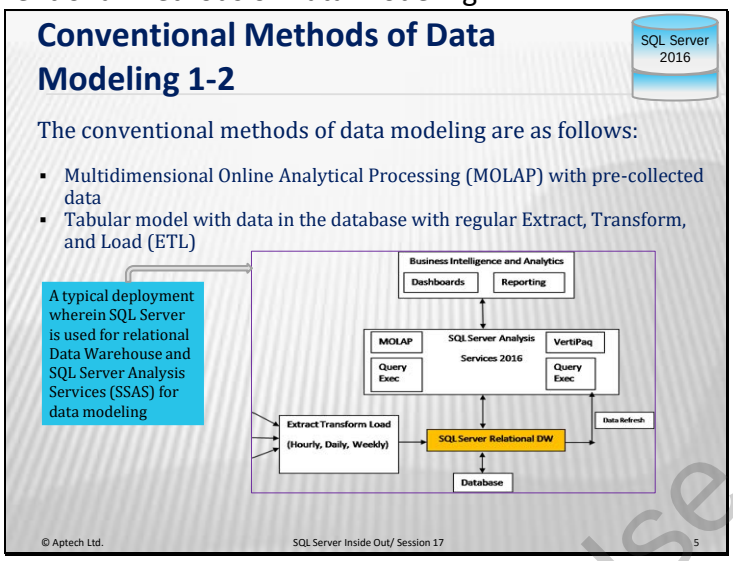
Additional Information:

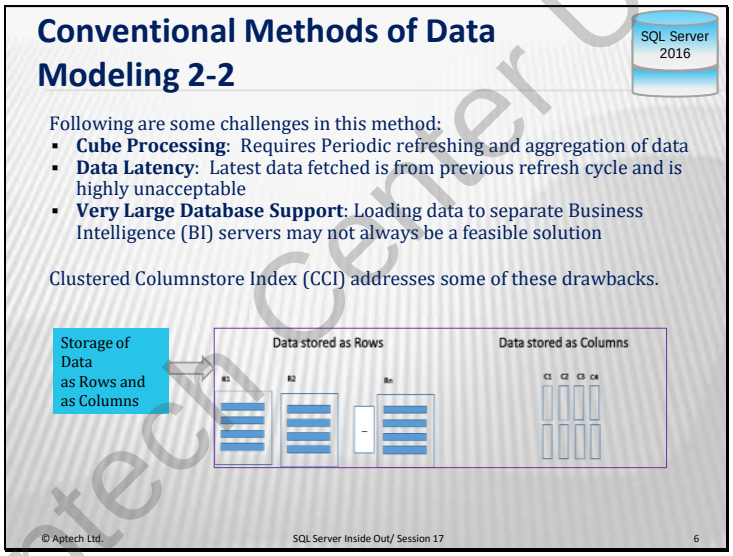
For more information on enhancements to SQL Server 2016, you can visit the following links:

https://msdn.microsoft.com/en-us/library/bb500435.aspx
https://msdn.microsoft.com/en-us/library/bb510411.aspx
http://www.databasejournal.com/features/mssql/slideshows/10-new-features-worth-exploring-in-sql-server-2016.html

Slides 5 and 6

Let us explore conventional methods of Data Modeling.





Using slide 5, explain the two conventional modeling methods namely MOLAP and Tabular model. Explain the diagram. Tell the students that Multidimensional is a mature technology built on open standards, adapted by numerous vendors of BI software but can be hard to master. Tabular model offers a relational modeling approach that many developers find easy to use.

Using slide 6, mention the challenges involved in conventional data modeling. Explain what is cube processing, data latency is, and how huge amount of data will affect BI.

Tell the students about:

Cube processing:
Before browsing the data stored in the cube, you should first process the cube if any modifications are made to the cube structure. To obtain correct results it is suggested to process cube if any data is added or changed in the cube.

Data latency:
The time taken to fetch data from data warehouse is data latency, in terms of business intelligence.

Explain CCI:
Columnstore:
Columnstore is defined as the data that is actually stored in column-wise format but logically stored as a table having columns and rows.

Rowstore:
Rowstore is defined as the data that is actually stored in row-wise format, but logically stored as a table having columns and rows.

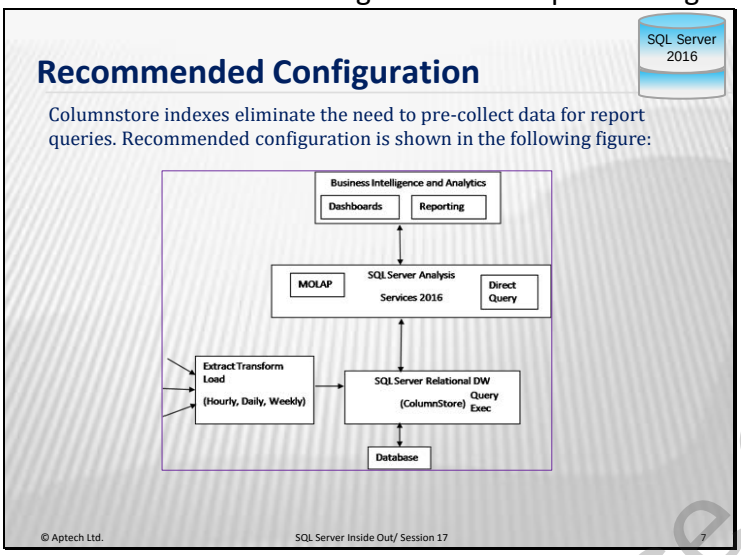
Columnstore index:
Columnstore index is a technology that uses columnstore to store, retrieve, and manage data.
Querying on huge data warehouse uses columnstore index as the standard technology for data operations such as query and store, which enables 10x boost in query performance.

Later on, mention how some of the challenges involved in conventional data modeling are taken care of with CCI implementation.

Additional Information:
For more information on conventional methods of data modeling, you can visit the following links:
https://msdn.microsoft.com/en-us/library/hh212940.aspx
https://technet.microsoft.com/en-us/library/aa216366(v=sql.80).aspx
https://msdn.microsoft.com/en-us/library/gg492088.aspx

Slide 7

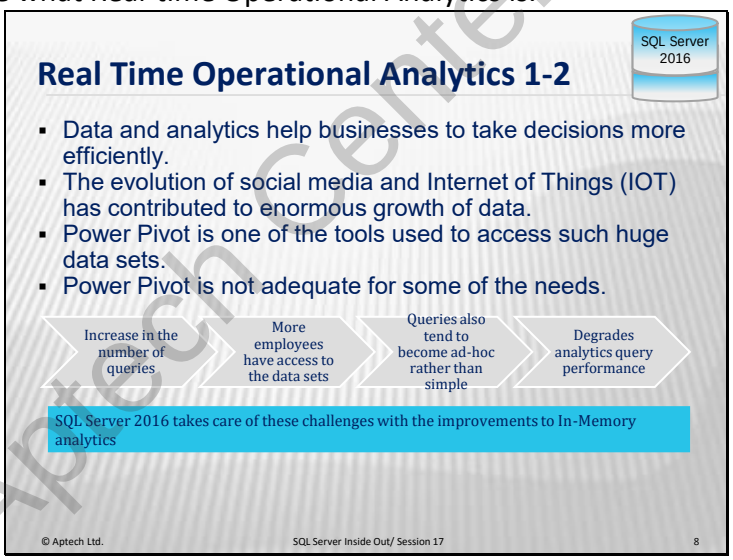
Let us now look into one recommended configuration for implementing columnstore index.

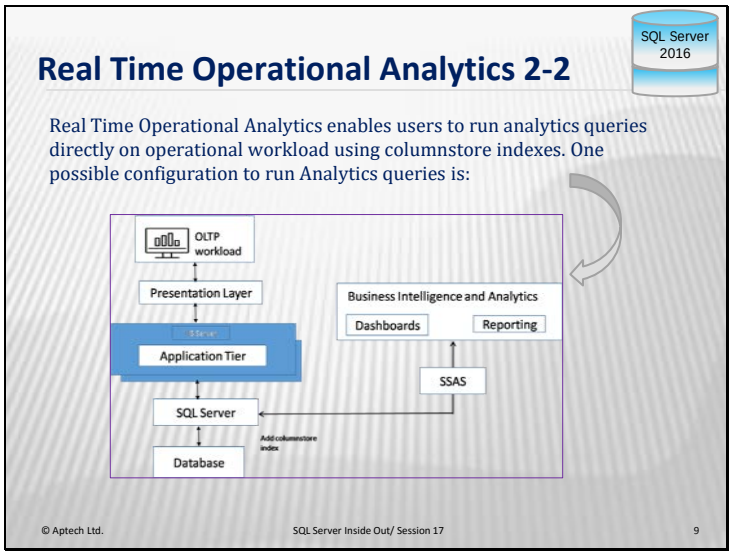


Using slide 7, explain the diagram.

Slides 8 and 9

Let us now explore what Real-time Operational Analytics is.





Firstly, tell the students,

- How important it is to have a highly scalable and available transaction system for any business.
- The role of analytics for such transactions.
- The drawback in terms of complexity and cost, if data and analytics are isolated.

Using slide 8, explain the need for real-time operational analytics.

Using slide 9, explain what real-time operational analytics is and also explain the diagram which shows a sample configuration to implement real-time operational analytics.

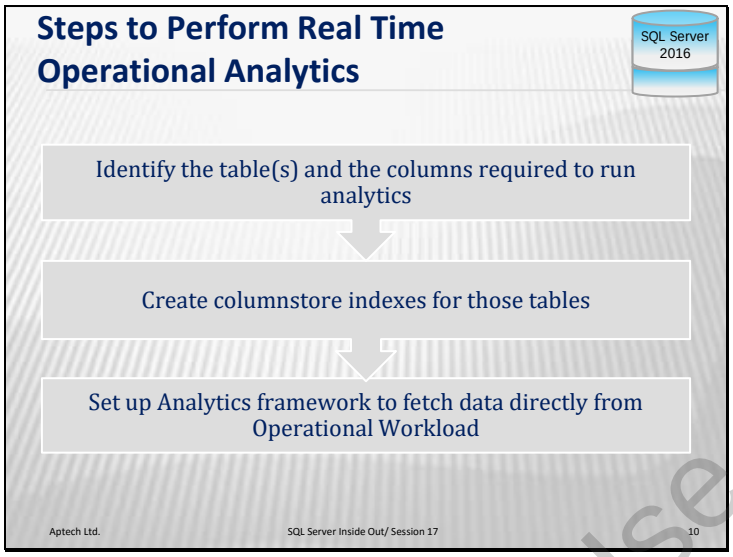
Tell the students that:
Real-time analytics allows using all the available data and resources when needed and enables dynamic analysis and reporting on the data.
SQL Server 2016 introduces real-time operational analytics, the ability to run both analytics and OLTP workloads on the same database tables at the same time. Besides running analytics in real-time, you can also eliminate the need for ETL and a data warehouse.

Additional Information:

For more information on real-time operational analytics, you can visit the following link:
https://msdn.microsoft.com/en-IN/library/dn817827.aspx

Slide 10

Let us examine the steps involved in implementing real-time operational analytics.



Explain the step-by-step procedure to implement real-time operational analytics.

Tell the students that before deploying operational analytics, you may consider the following factors:

- Clearly state the objective for the analytics
- Gather data for analyzing
- Design a prototype
- Interact with the customer one-to-one and discuss the prototype with them
- Test the prototype extensively and redesign the prototype as needed
- Think of outsourcing if it suits better in terms of cost

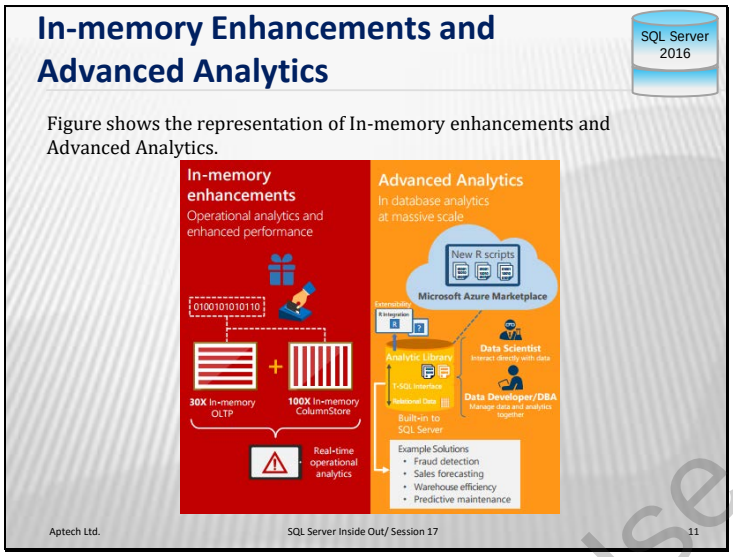
Additional Information:

For more information on real-time operational analytics, you can visit the following link:

https://www.dialog-direct.com/media-room/article/operationalizing-analytics-six-steps-to-an-effective-revenue-generating-strategy/

Slide 11

Let us now explore In-memory enhancements and Advanced Analytics.



Using the diagram given on slide 11, explain In-memory enhancements and Advanced Analytics.

Tell the students that In-memory enhancements enables faster transactions and faster queries when compared to relational databases that are on disk. Advanced Analytics enables running the analytics algorithms directly on the SQL Server transactional database.

After you finish explaining slide 11, you can ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

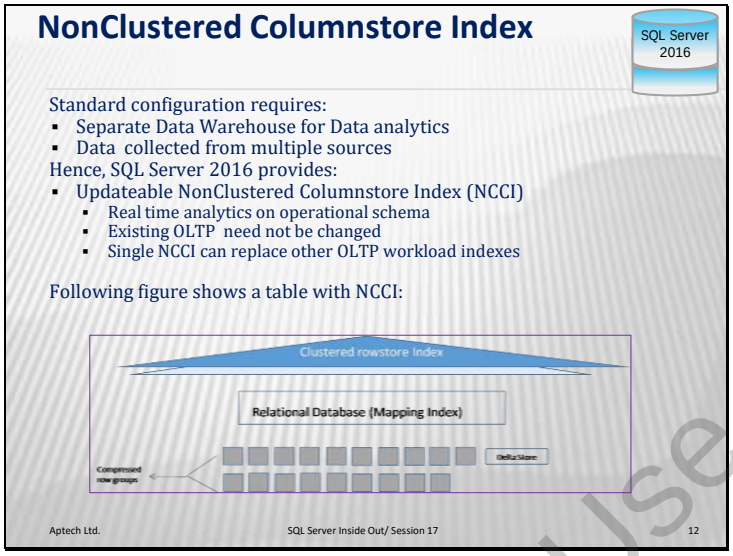
In-Class Question:

Which third-party tool enables the user to simulate different workloads against the In-Memory OLTP Engine of SQL Server?

Answer: In-Memory OLTP Simulator.

Slide 12

Now let us understand NCCI, nonclustered columnstore index and its advantage in real-time analytics implementation.



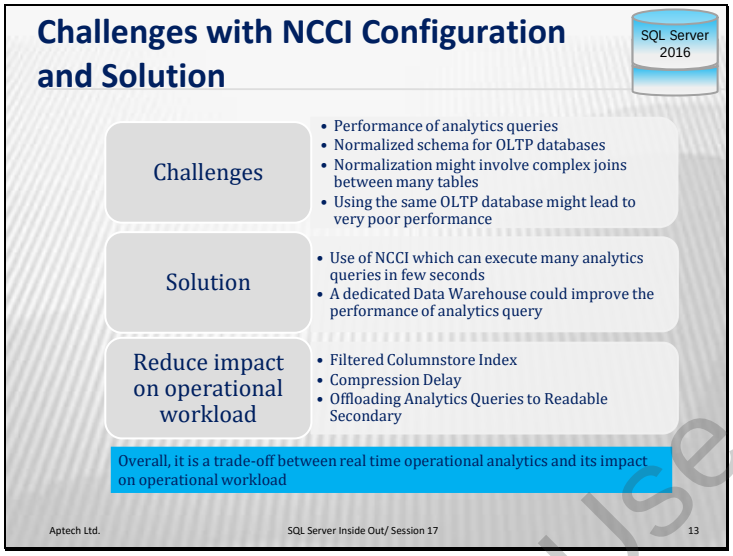
Using slide 12, tell the students that the introduction of updateable NonClustered Columnstore Index, has made possible real-time analytics on operational schema. All you need is to create NCCI on one or more tables on which analytics is to be performed. Explain the diagram that depicts table with NCCI.

Additional Information:

For more information, visit the following link:
https://blogs.msdn.microsoft.com/sqlserverstorageengine/2016/02/29/real-time-operational-analytics-using-nonclustered-columnstore-index/

Slide 13

Let us see the challenges, solution, and the features available in SQL Server 2016 to reduce impact on operational workload.



Using slide 13, tell the students about:

Challenges in NCCI:

- How to ascertain good performance for analytics when database schema is configured for OLTP.
- Highly normalized schema, which means poor performance due to complex joins between the tables.
- Reduce or totally remove the impact of analytics on transactional workload.

Solution to those challenges:

Enable real-time operational analytics in your workload using NCCI without changing transactional workload.

Reducing impact on operational workload:

- **Filtered Columnstore Index**: Enables the customer to create NCCI with a filtered condition.
  Show the SQL command to create NCCI.
- **Compression delay**: Enables you to define the duration till when the data rows can be hot.
- **Offload analytics to Alwayson readable secondary**: In this case the only overhead is the additional NCCI at primary replica.

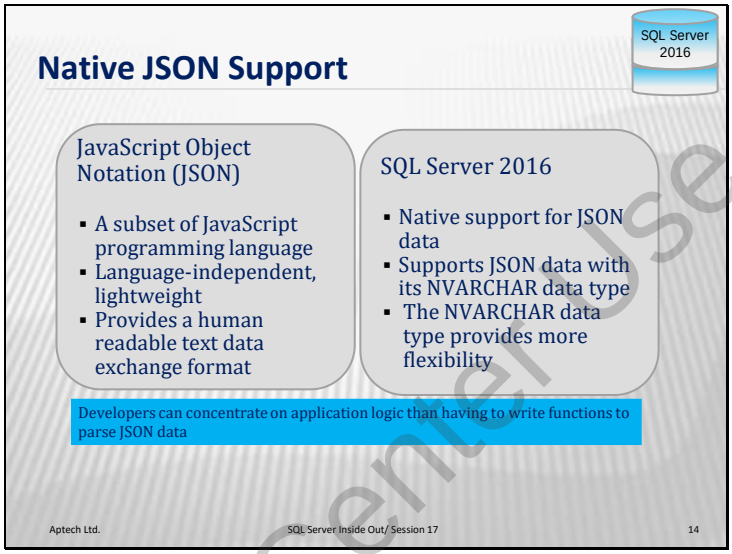
Conclude real-time operational analytics stating that, the columnstore index and memory optimization can be collaborated together to obtain the best OLTP performance and analytics query.

Additional Information:

For more information on real-time operational analytics using NCCI, you can visit the following link:
https://blogs.msdn.microsoft.com/sqlserverstorageengine/2016/02/29/real-time-operational-analytics-using-nonclustered-columnstore-index/

Slide 14

Let us understand what is JSON and how SQL Server 2016 native supports JSON for data interchange.



Explain the students that JSON is:

- A lightweight data-interchange format.
- It is easy for humans to read and write.
- It is easy for machines to parse and generate.
- It is based on a subset of the JavaScript Programming Language.

In comparison with earlier version of SQL Server, the native JSON support provided by SQL Server 2016 enable developers to focus on application logic. The XML support still continues.

After you finish explaining slide 14, you can ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

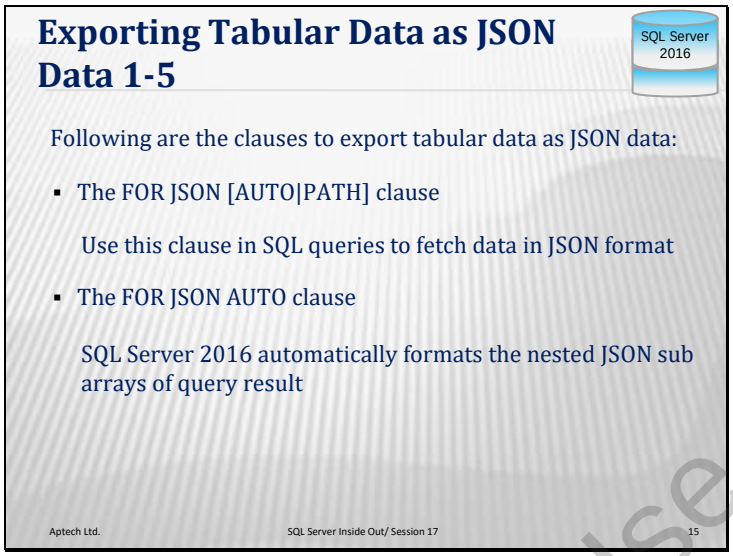
In-Class Question:

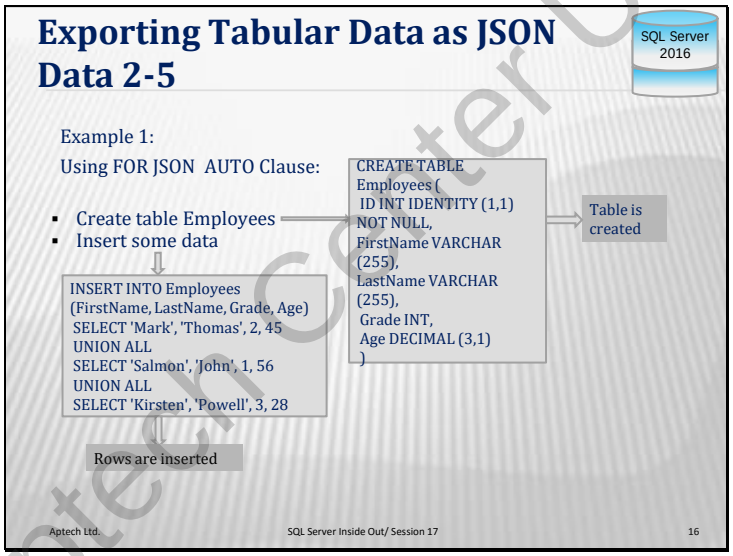
What is the advantage of using JSON over XML?

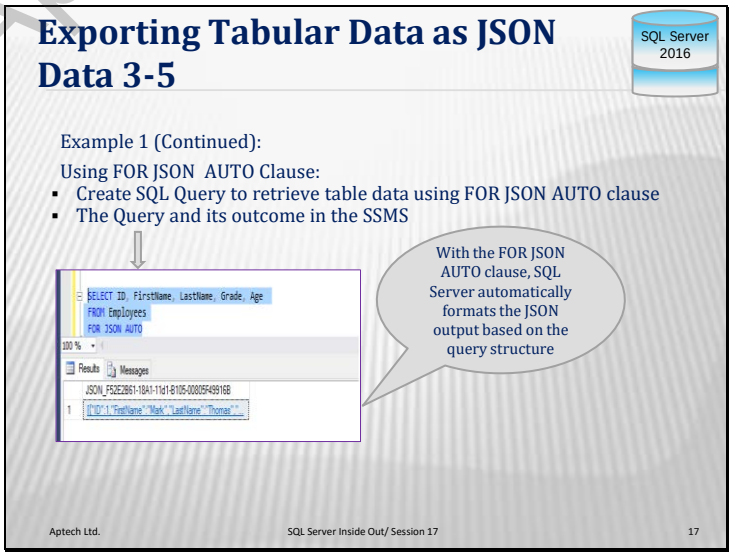
Answer: Compared to XML, JSON is lighter and faster. JSON objects are typed while XML data is not.

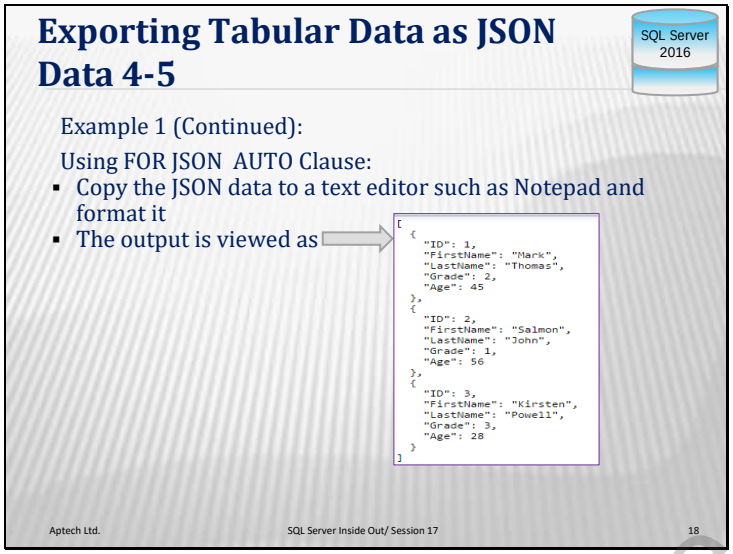
Slides 15 to 19

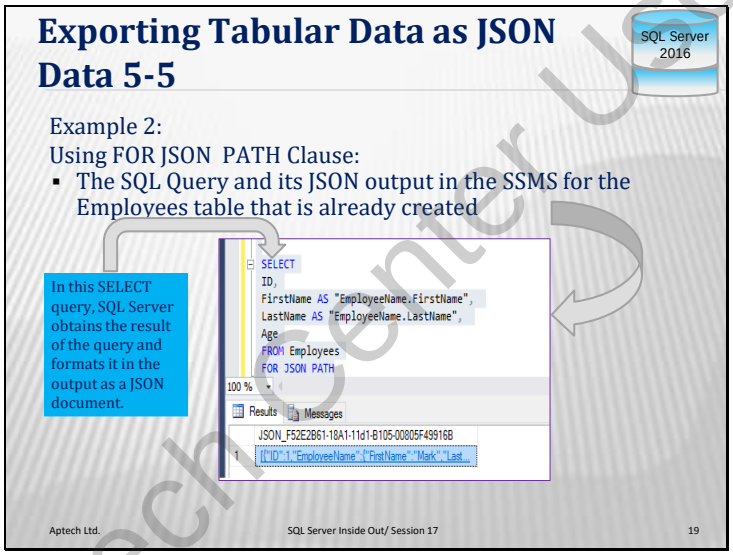
Let us understand converting tabular data to JSON data and vice-versa.











Using slides 15 to 19, demonstrate the following to the students with examples:

- FOR JSON [AUTO|PATH] clause
- FOR JSON AUTO clause

Explain the use of each clause.

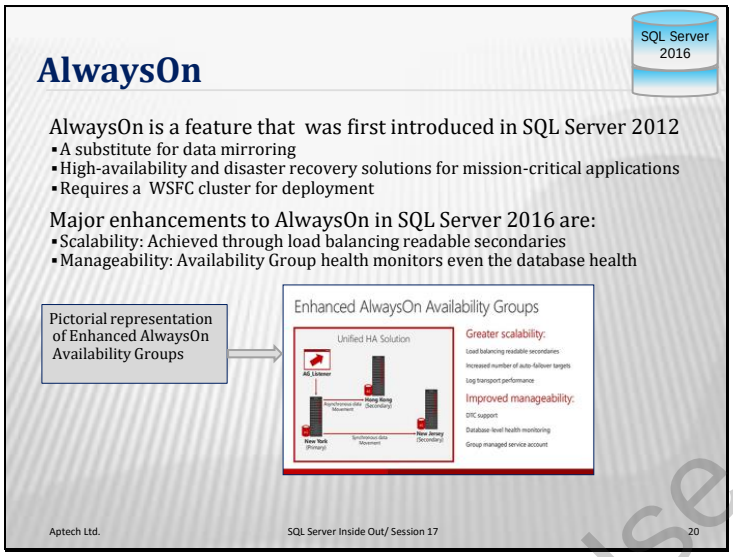
Additional Information:

For more information, visit the following links:

http://www.tutorialspoint.com/json/json_overview.htm
https://blogs.msdn.microsoft.com/jocapc/2015/05/16/json-support-in-sql-server-2016/

Slide 20

Let us understand AlwaysOn feature.

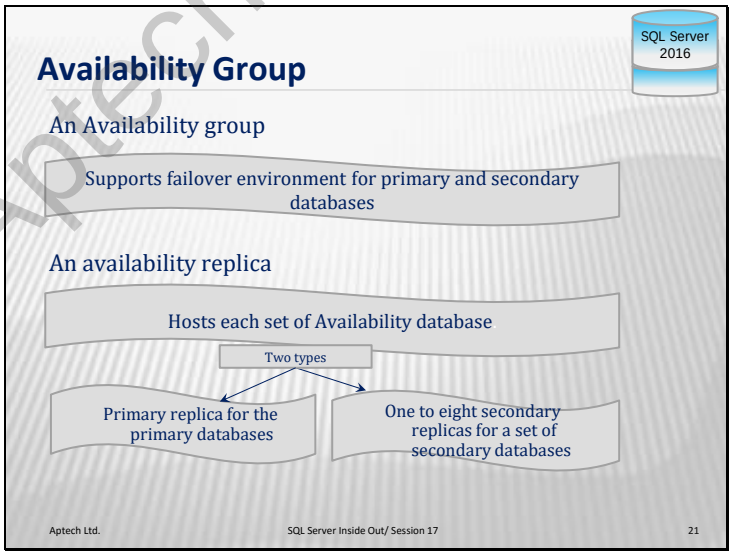


Explain what AlwaysOn is and enhanced features of AlwaysOn. Explain the concept of failover.

Mention that the MSDN Website states that 'Database mirroring will be removed in future versions of SQL Server and hence it is recommended to use AlwaysOn Availability groups instead'.

Slide 21

Let us understand what is an Availability group and its relation to AlwaysOn.

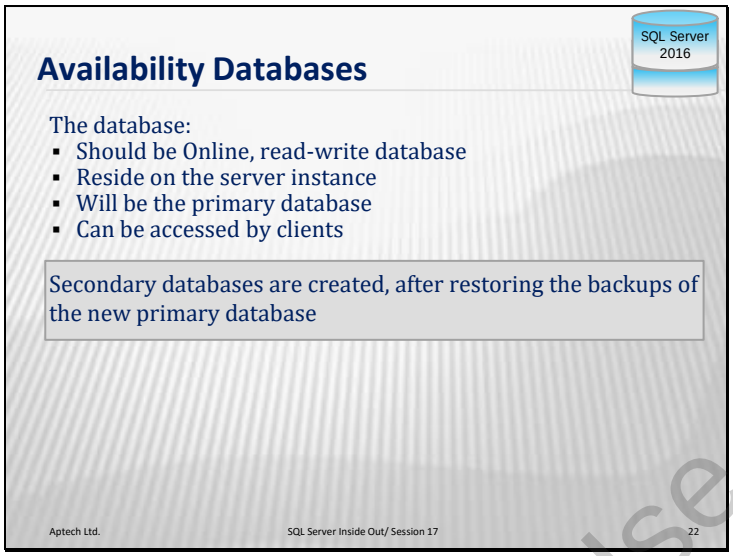


Explain what AlwaysOn Availability group is.

Tell the students that AlwaysOn Availability Groups are a fundamental component of the availability story for SQL Server and provide a robust disaster recovery solution as well.

Slide 22

Let us understand what an Availability database is.

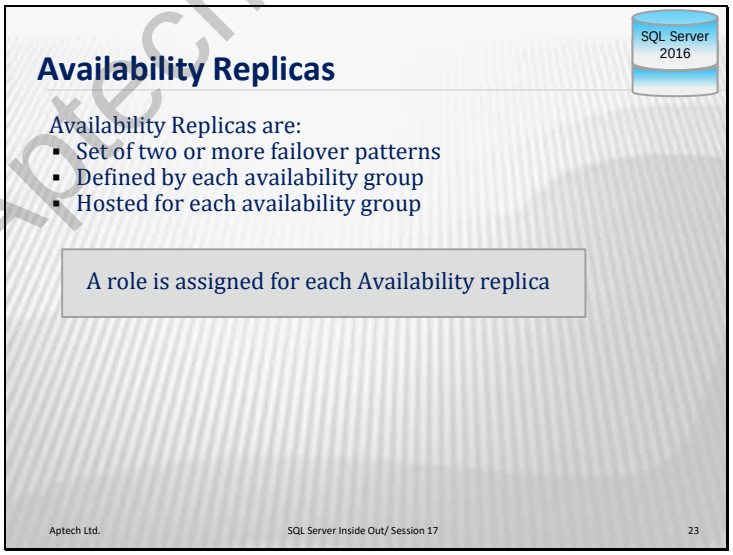


Using slide 22, explain the students that:

To add a database to an availability group, you should be connected to the server instance that hosts primary replica. The database also should be on the server instance that hosts the primary replica. This will be the primary database, which will be accessed by the clients. You can make use of SSMS to Transact SQL to add a database to the availability group. Secondary databases also can be added to the availability group.

Slide 23

Let us now understand about Availability replicas.



Tell the students that, Availability replicas are components of Availability groups. They are the failover patterns defined by the availability group. Explain in brief the two types of availability replicas, the primary replica and the secondary replica. Brief about the roles, that is, primary role for the primary replica that hosts read-write primary database and secondary role for the secondary replica that hosts read-only secondary database.

After you finish explaining slide 23, you can ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

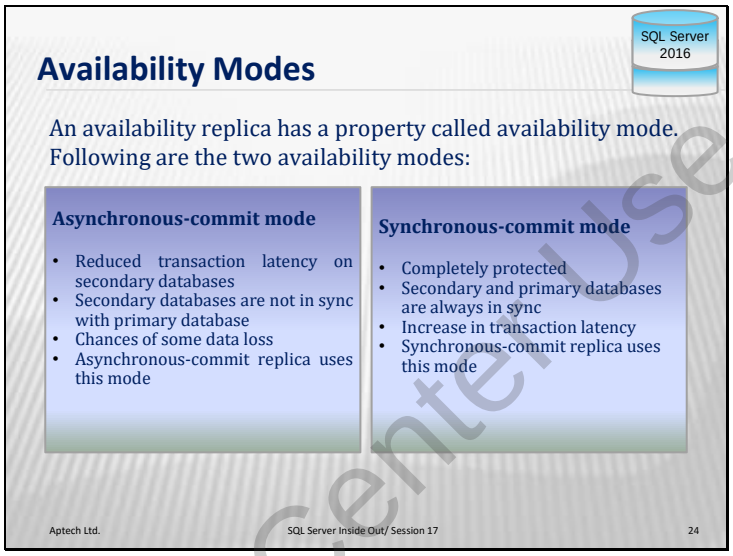
In-Class Question:

How many primary replicas can be there in an AlwaysOn Availability group?

Answer: One

Slide 24

Let us now understand about Availability modes.

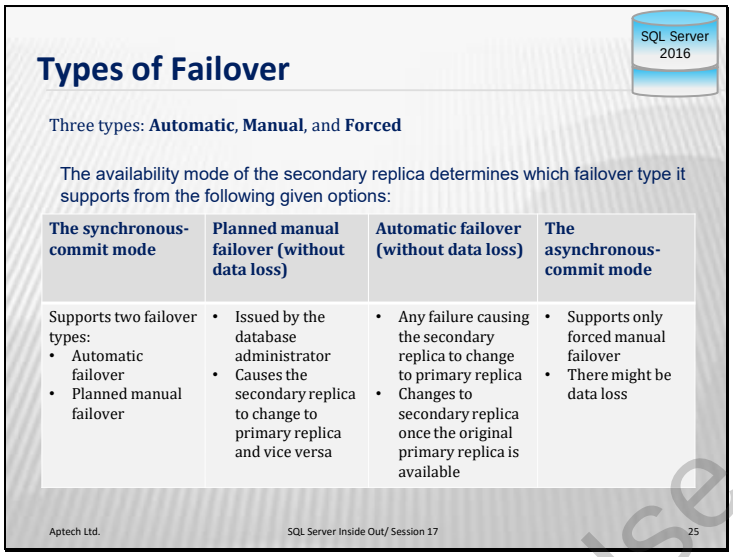


Using slide 24, tell the students that:

In Asynchronous commit mode, the primary replica does not wait for the secondary replica to write the transaction log, that is primary replica is not in sync with secondary replica. At the same time, the primary replica does not wait for the primary replica to write the transaction log. This can result in data loss. In synchronous commit mode, as the name itself suggests, the primary and secondary replicas are always in sync with each other but this may lead to transaction latency.

Slide 25

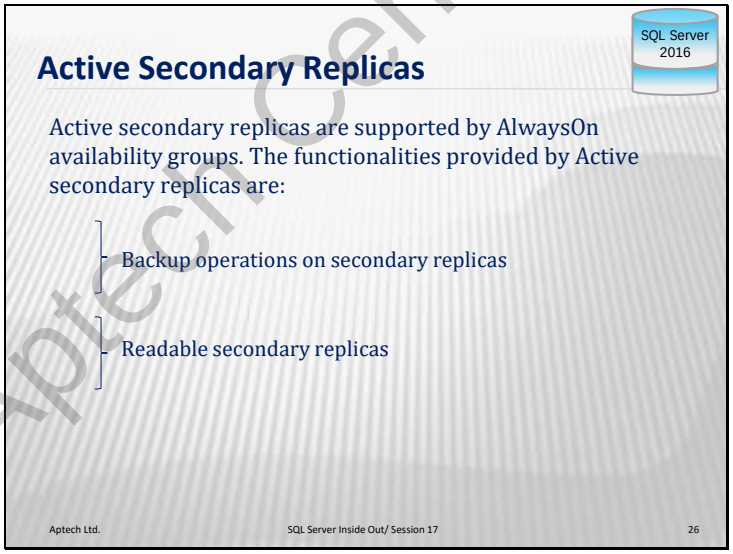
Let us now explore the types of failovers.



With the help of table given on slide 25, explain the types of failover.

Slide 26

Let us now explore Active Secondary replicas.



Using slide 26, explain the advantages of backup operations on secondary replicas saying that:

- Backup operation can put significant strain on I/O and CPU
- Offloading backup operations on secondary replicas allow you to use the resources on server instance that hosts the primary replica for tier-1 workloads.

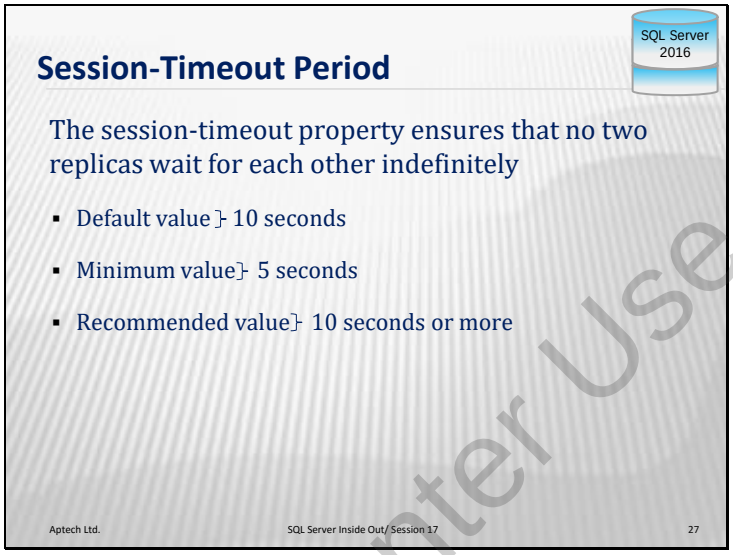
Explain readable secondary replica:

- A readable secondary replica allows read-only access to all its secondary databases.
- However, readable secondary databases are not set to read-only. They are dynamic.

Additional Information:

For more information, visit the following links:
https://msdn.microsoft.com/en-us/library/hh245119.aspx
https://msdn.microsoft.com/en-us/library/ff878253.aspx

Slide 27
Let us understand the session-timeout property.



Explain the students about session-timeout period.

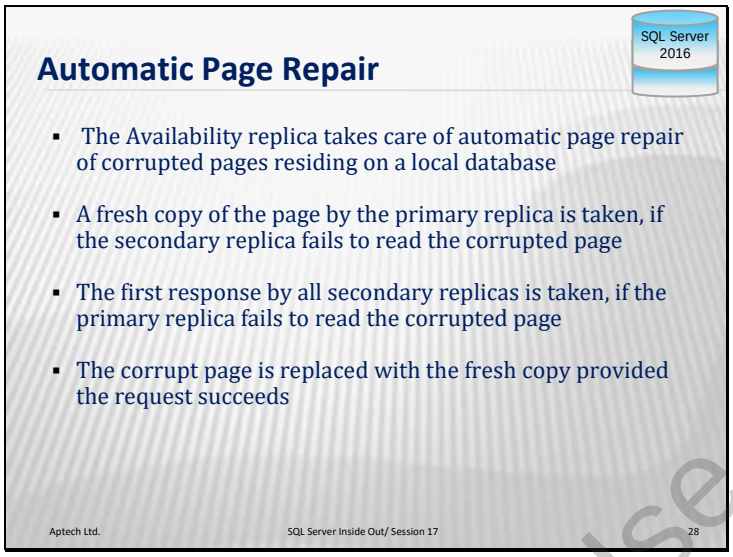
The session-timeout is the property of a replica. It determines the duration in seconds that an availability replica waits before considering that the connection is failed. The ping occurs only between the primary replica and the secondary replica of the availability group.

Additional Information:

For more information, visit the following link:
https://msdn.microsoft.com/en-IN/library/hh213612.aspx

Slide 28

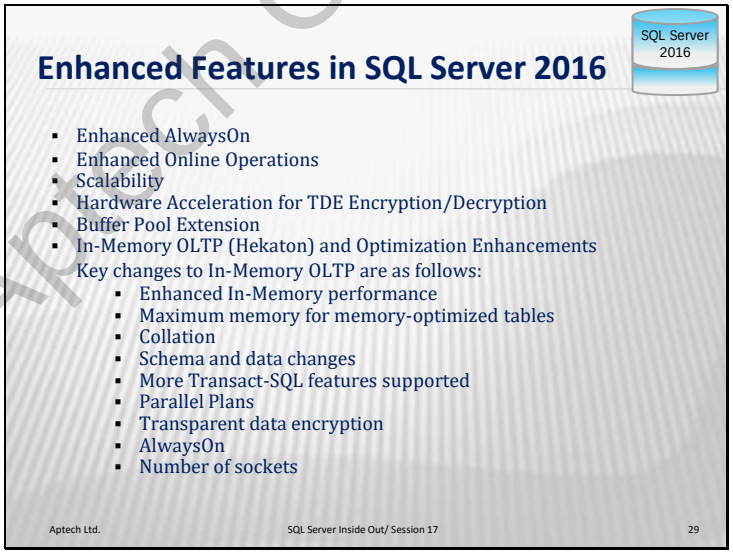
Let us understand the term Automatic Page Repair.



Using slide 28, explain how automatic page repair takes place. Tell the students that: Whenever certain types of errors lead to corrupt page making it unreadable, availability replica tries to automatically recover the page. If the replica also fails to read the page, it requests a fresh copy of the page from another replica, obtaining a readable copy.

Slide 29

Let us now explore the enhanced features in SQL Server 2016.



Explain all the enhanced features given on slide 29:

- **Enhanced AlwaysOn:**
  1. Round-robin load balancing in readable secondaries
  2. Increased number of auto-failover targets
  3. Enhanced log replication throughput and redo speed
  4. Support for group-managed service
  5. Support for Distributed Transactions (DTC)

6. Basic HA in Standard edition
7. Direct seeding of new database replicas

- **Enhanced Online Operations:**
  SQL Server 2016 provides DBAs the flexibility to terminate the processes which can block them from issuing table locks. SQL Server 2016 also enables enhanced online database operations thus providing 100 percent uptime.

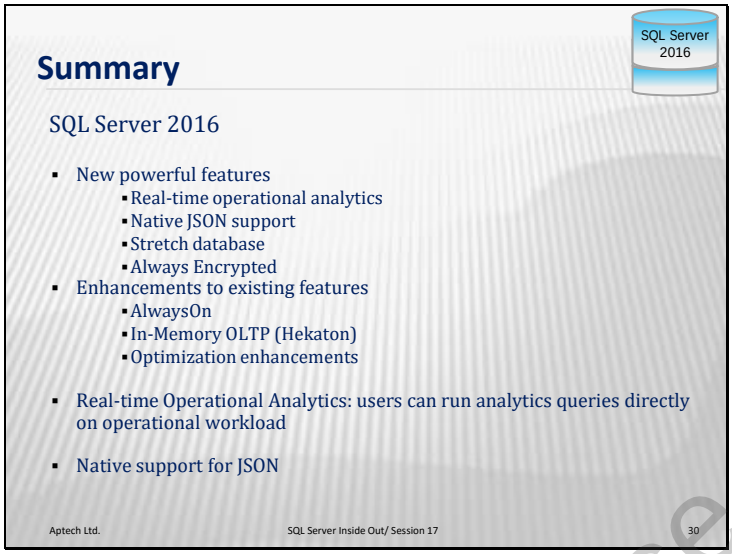
- **Scalability:**
  Scalability factor is directly proportional to the interaction between Microsoft SQL Server and Microsoft Windows Server.
- **Hardware acceleration for TDE Encryption/Decryption:**
  The enhancements in TDE encryption/decryption algorithms along with hardware acceleration work in collaboration to provide better security.
- **Buffer Pool Extension:**
  The use of solid-state drives reduces burden on SQL Server memory, improves query performance, and there is no data loss.
- In-Memory OLTP and Optimization Enhancements are as follows:
  - Improve performance of OLTP queries by moving select tables to memory
  - Compiling stored procedure to native x86 code
  - OLTP engine is designed to work at RAM speed
  - The Analysis Migrate Report (AMR) tool to smoothly migrate applications to In-Memory OLTP

Additional Information:
For more information on enhanced features in SQL Server 2016, visit the following links:

http://searchsqlserver.techtarget.com/feature/Whats-new-in-2016s-SQL-Server-AlwaysOn-Availability-Groups
https://channel9.msdn.com/events/DataDriven/SQLServer2016/InMemoryOLTP

Slide 30

Let us now summarize the session.



Use slide 30, summarize the session. Highlight all the important points described in the session.

17.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session.

Tips: You can also check the **Articles/Blogs/Expert Videos** uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 18: Enhancements in SQL Server 2016

18.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

18.1.1 Objectives

By the end of this session, learners will be able to:

- Describe the importance of configuring tempdb database files
- Describe how to configure tempdb during installation of an instance of SQL Server 2016
- Outline the allocation and autogrowth of tempdb files in SQL Server 2016
- List the differences between two execution plans
- Explain how to enable system-versioning on tables

18.1.2 Teaching Skills

To teach this session, you should be well versed with the new features and enhancements in SQL Server 2016. You also need to know about SQL Server 2014 and how some of the earlier features are improved in SQL Server 2016. You should be familiar with:

- Tempdb, its configuration, and related concepts such as file allocations and autogrowth handling
- Comparison of execution plans
- Temporal tables, concept of storing and retrieving different versions of data

You should teach the concepts in the theory class using the slides and images on the slides that support the content of the slide. Elaborate on the points shown on the slides and explain any examples provided on the slides to enable better understanding.

Tips:
Interact with the class, refresh any previous concepts, if required. You can use your own experience and knowledge, suggestions provided in this guide or any Websites that give more information. Some references are provided in this guide.
It is recommended that you test the understanding of the students by asking questions in between the class.

Overview of the Session

Using slide 2 of the presentation, provide an overview of the session.

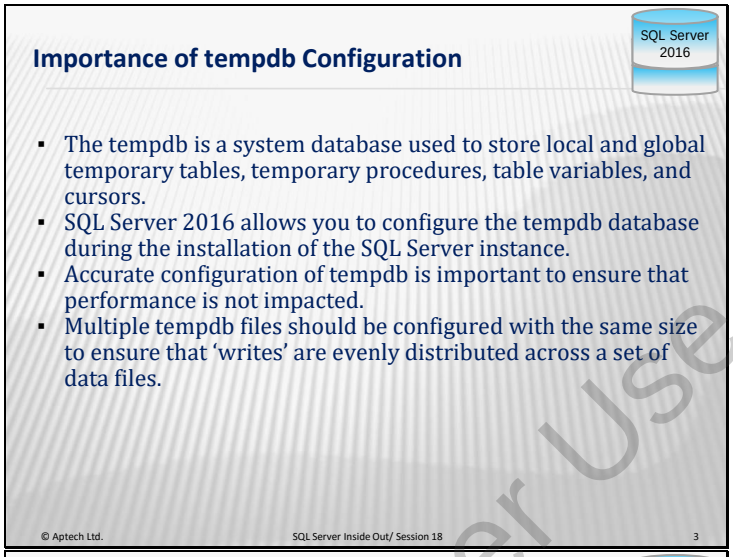
- Begin the session by introducing students to few drawbacks in previous versions of SQL Server: configuration of tempdb files manually after SQL Server installation, inability to compare execution plans, and inability to retrieve different versions of data. State that SQL Server 2016 comes with enhancements to overcome these issues.

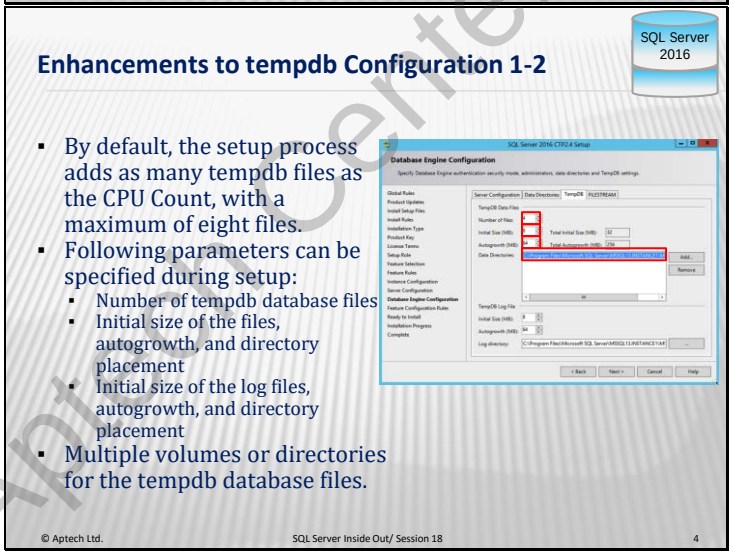
- Conclude by listing the objectives on the screen

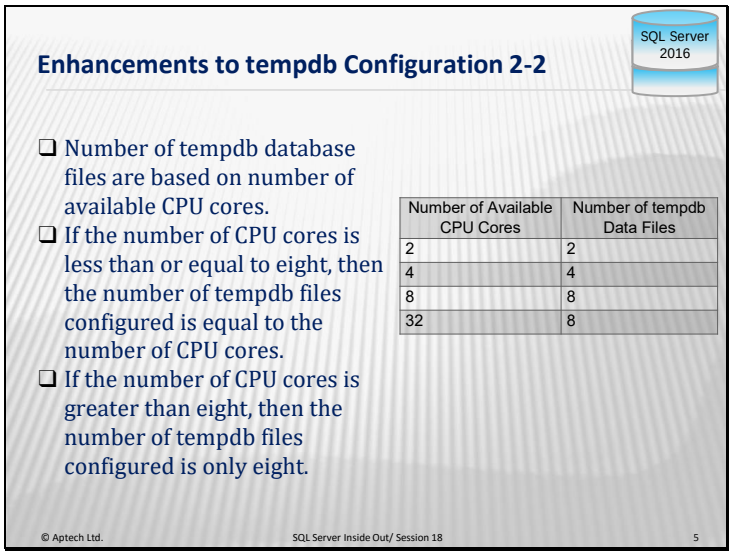
18.2 In-Class Explanations

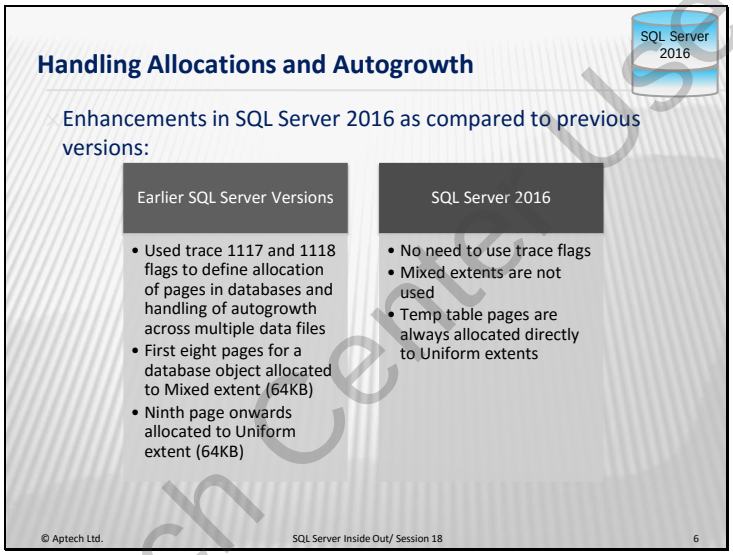
Slides 3-6

Let us understand the importance of configuring tempdb accurately.









❖ Using **slide 3**,

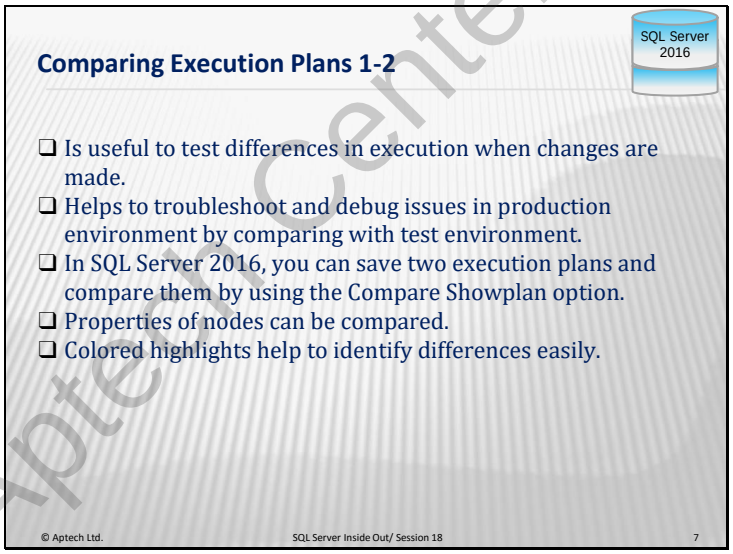
- Cover the functions of tempdb:
  - Store local and global temporary tables, temporary procedures, table variables, and cursors.
  - Database engine in SQL Server uses the tempdb database to store the intermediate sort results during the indexing process.
- Explain the proportional fill method used by SQL for 'writes' and how it aims for even distribution of writes.
  - **Tip:** You could ask anyone in the class to refresh knowledge about SQL Server using round-robin and proportional fill method to write to data files.
- Explain the impact of having different sized tempdb files.
  - Uneven sized files would result in SQL Server choosing only one with higher percentage of free space for writes. This results in overload of one file leading to poor performance.
- Conclude this slide by stressing the importance of tempdb configuration in the performance of SQL Server.

- Using **slide 4**,
  - Point out the image on the slide and list all the tempdb parameters that can be configured during SQL Server 2016 Setup itself.
- Using **slide 5**,
  - Explain how the number of tempdb files is allocated based on the CPU count.
- Using **slide 6,**
  - Compare how pages were allocated and autogrowth was handled in previous SQL Server versions and SQL Server 2016.
  - Refresh concepts of mixed extents and uniform extents, and how trace Flag 1117 and 1118 were used to handle autogrowth in SQL Server 2014 and explain that in SQL Server 2016, this is not required.

Additional Information:

For more information, visit the following links:
http://www.sqlservercentral.com/articles/tempdb/139206/
https://msdn.microsoft.com/en-GB/library/ms190768.aspx

Slides 7 and 8
Let us learn about comparing execution plans, a new feature in SQL Server 2016.



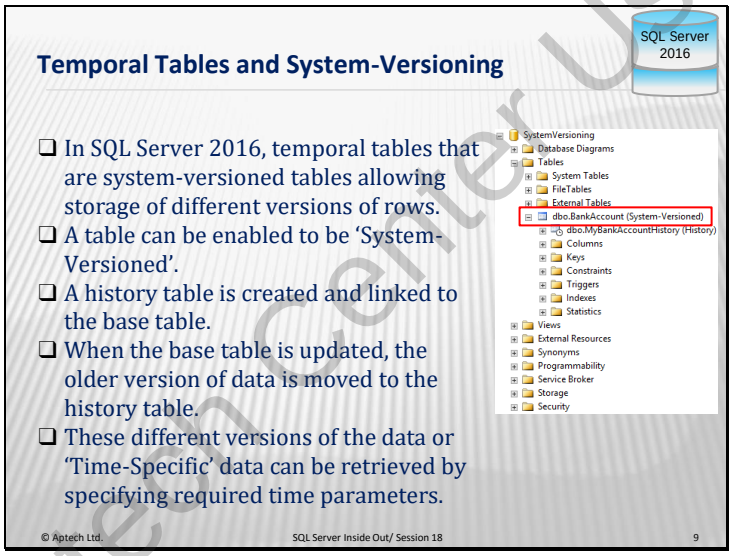
- Using **slide 7**,
- List the scenarios where comparing two execution plans is useful.
  - **Tip:** You could ask participants to call out about any experiences where there was a necessity to compare execution plans.
  - You could use an example of execution in a test and production environment and highlight how understanding the change gave significant help to improve performance or debug an error. For example, an upgrade might not have impacted some queries in the test environment but some issue is occurring in the production environment. To analyze this, the execution plans can be compared.

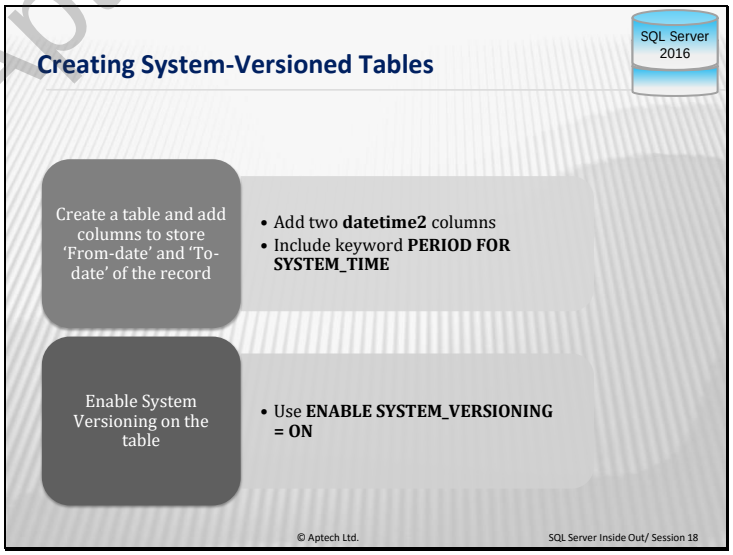
- Using **slide 8,**
  - Using the image, point out to the Compare Showplan option provided in SQL Server 2016 that provides a detailed comparison of two execution plans.
  - Explain that the execution plans must first be saved and then compared using the option.
  - State that the differences in the two execution plans are highlighted and two changed nodes can be compared to identify the better option.

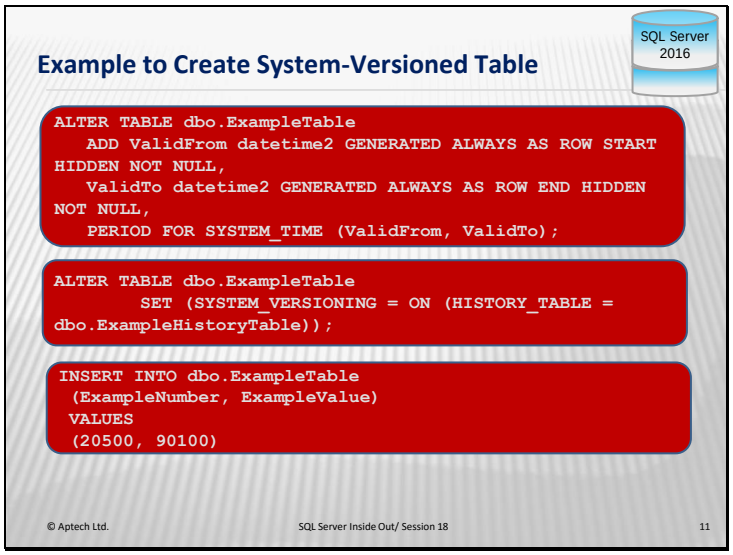
Additional Information:

For more information, visit the following link:
https://blogs.msdn.microsoft.com/sql_server_team/comparison-tool-released-with-latest-ssms/

Slides 9 to 11
Let us understand versioning of row data, the concept of temporal tables and history tables, and how to create system-versioned tables.







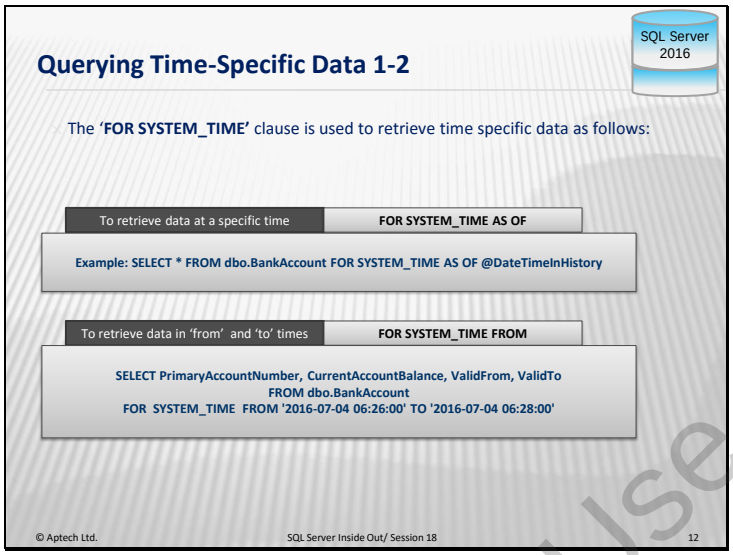
- Using **slide 9**,
  - Talk about how previously retrieving data at different points in time was a challenge and how temporal tables overcomes this.
  - Using the image on the slide, explain how to enable system-versioning on a table, which is called a temporal table.
  - Explain that once system-versioning is enabled on a table, such a table is called a base table or temporal table and that a history table is created and linked to it. Whenever new update happens to the base table on a row, the older version is pushed to the history table. This data in different versions can be accessed.
  - Mention the uses of being able to access different versions of row data, such as analyzing trends, support business decisions, and revert to previous version of data in case of errors.
- Using **slide 10,**
  - Explain the two important inclusions to be made to code to create system-versioned tables.
- Using **slide 11,**
  - Explain the code to create system-versioned tables using the example on the slide.

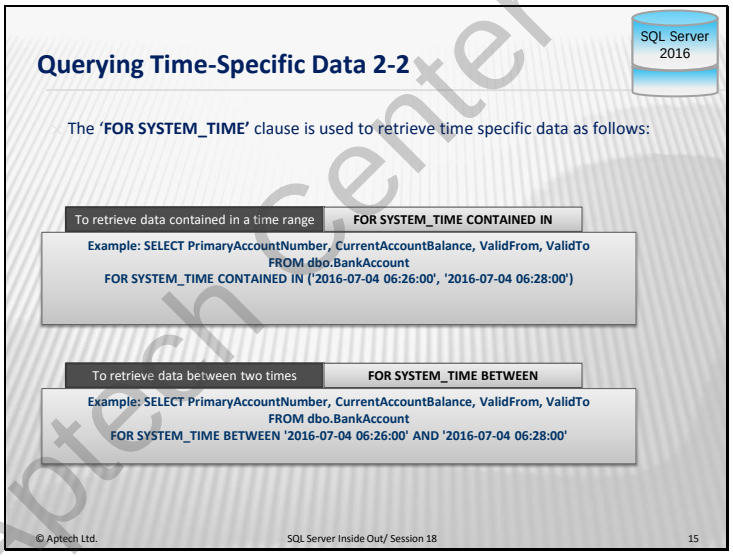
Additional Information:

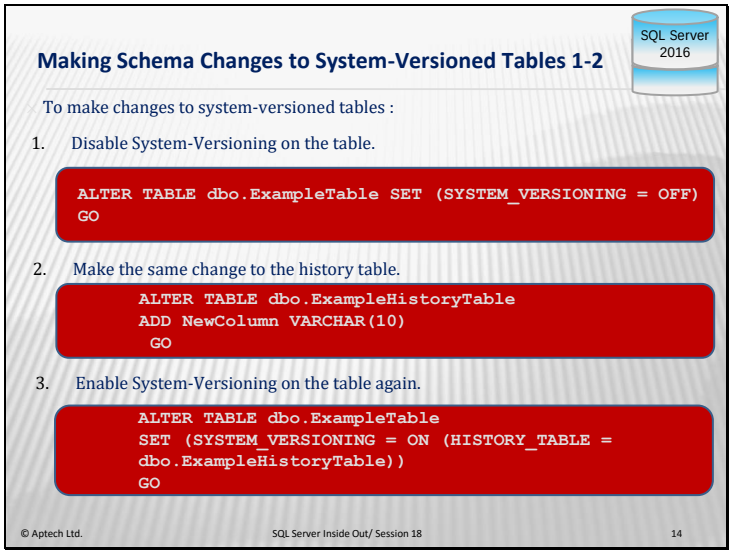
For more information, visit the following link:
https://msdn.microsoft.com/en-GB/library/dn935015.aspx

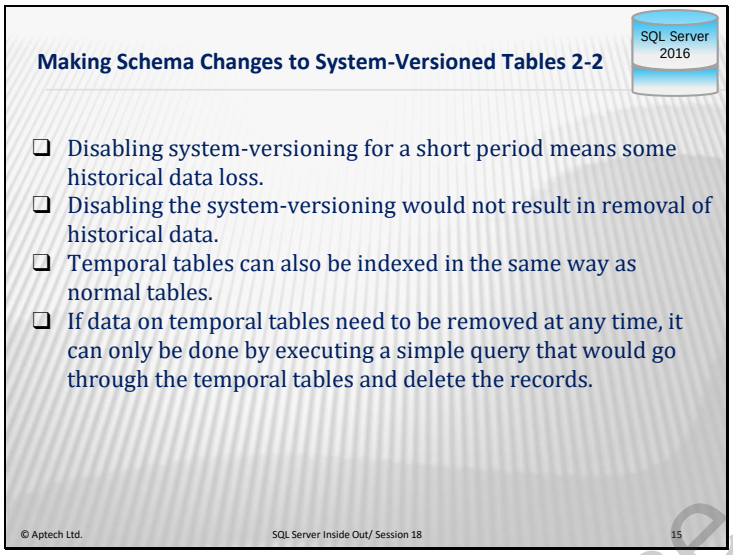
Slides 12 to 15

Let us understand how to retrieve versions of data and how to make any schema changes to temporal tables.









- Using **slide 12**,
  - Explain how time-specific data can be queried for by including the **For-System-time clause.**
    Use the example on the slide pertaining to retrieval of bank account information at a specific time or by providing 'from' and 'to' timings.
- Using **slide 13,**
  - Explain with the examples on the slide how to retrieve information in a time range and between two specific timings.
- Using **slide 14,**
  - Explain that a straight-forward change to a system-versioned table is not possible as it throws errors.
  - If schema changes are made to the base table, same changes must be made to the history table.
  - Outline how schema changes can be made to system-versioned tables, use the steps on the slide and explain the code at each step.

- Using **slide 15,**
  - Mention that by disabling system-versioning, for a short period, history data is not stored.
  - Also mention that previously stored historical data would not be lost when system-versioning is disabled.
  - Add that indexing on a system-versioned table can be done in the same way as a normal table and that this would improve performance on querying history data.
  - Mention that no clean-up process is available at present to delete data on temporal tables.

In Class Question:

After you finish explaining slide 15, you will ask the students a few In-Class questions. This will help you in reviewing their understanding of the topic.

 In an employee database, if a list of employees who were logged-in to the company system on 25-01-2016 in the time range 10:AM and 10:30AM needs to be retrieved, which clause relevant to retrieval of versions of data should be used?

Answer: FOR SYSTEM_TIME BETWEEN

If the number of available CPU cores is 8, what would be the number of tempdb database files by default?

Answer: 4

Would historical data maintained in the history table would be lost when system-versioning is disabled?

Answer: No. Data already stored would not be lost

Name the feature to use if one wants to troubleshoot a problem in production environment, which had not occurred earlier in the test environment.

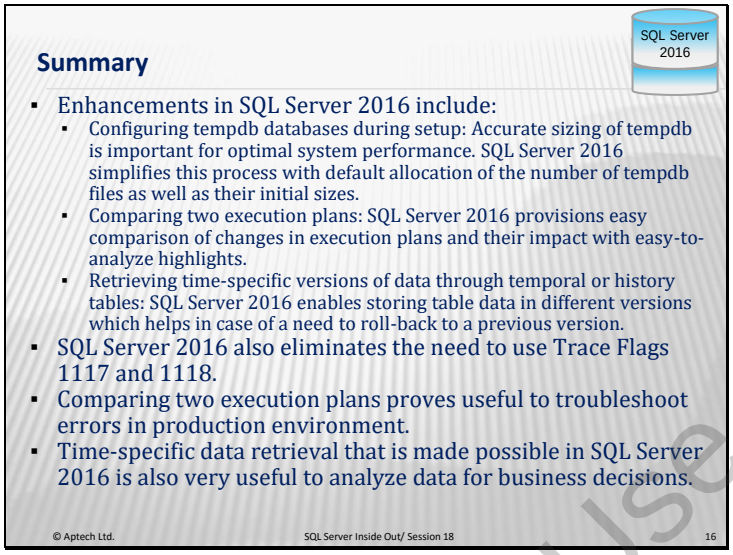
Answer: Compare Showplan

Does SQL Server 2016 use Mixed extents or Uniform extents?

Answer: Uniform extents

Slide 16

Let us summarize the session.



Using **slide 16**,

- Summarize the session. Explain each of the points on the slide in brief.

18.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session, which covers more enhancements made in SQL Server 2016.

You can refer to Microsoft articles on SQL Server 2016 pertaining to concepts covered in the session for detailed information that would help you answer any queries in the class.

Tips: You can also check the **Articles/Blogs/Expert Videos** uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 19: Security Upgrades and Working with JSON

19.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

19.1.1 Objectives

By the end of this session, learners will be able to:

- Explain how to use SQL Server features to protect data at rest and in motion
- Describe security enhancements in SQL Server 2016
- Explain how to work with JSON data

19.1.2 Teaching Skills

To teach this session, you should be well versed with features offered by SQL Server to protect data at rest and in motion, security enhancements of SQL Server 2016, and converting JSON data to relational form using OPNEJSON function.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities
Follow the order given here during In-Class activities.

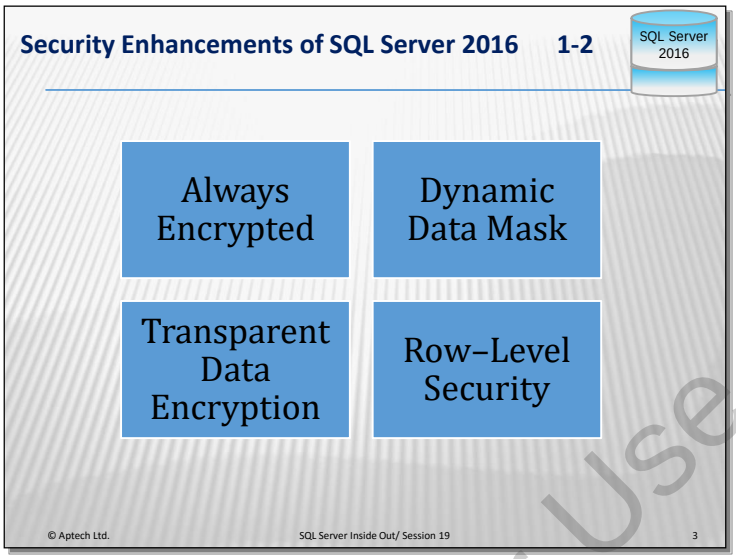
Overview of the Session

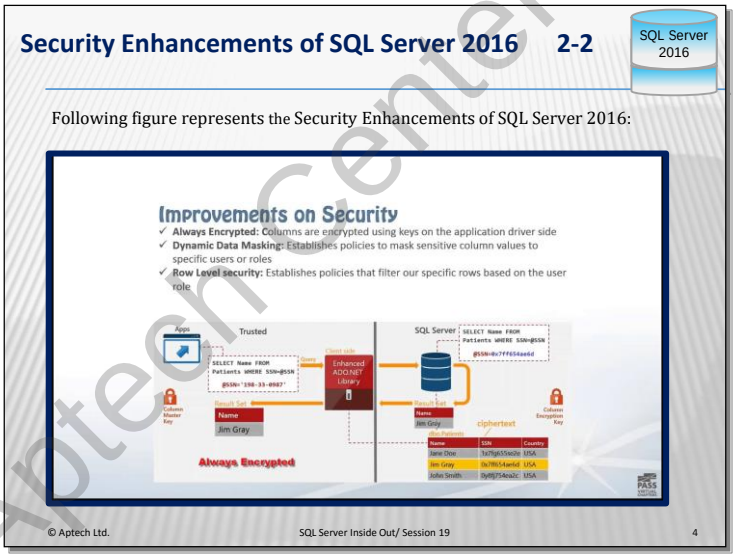
Give the students an overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

19.2 In-Class Explanations

Slides 3 and 4

Let us have a glance at security enhancements provided by SQL Server 2016.

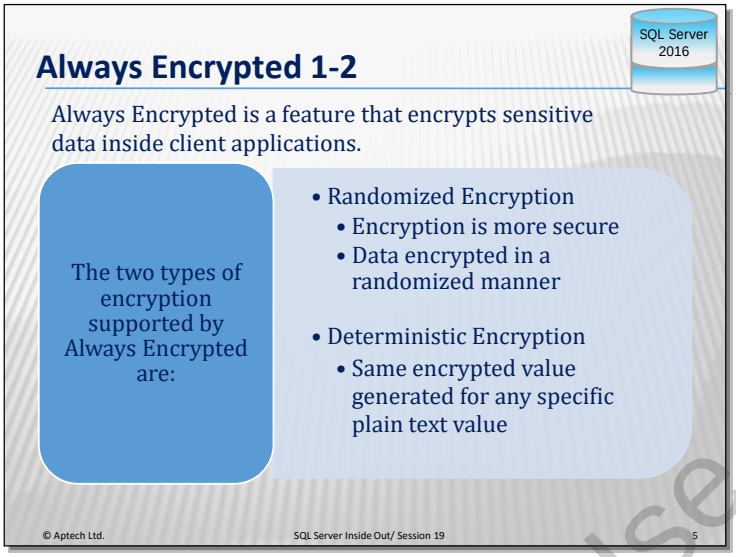


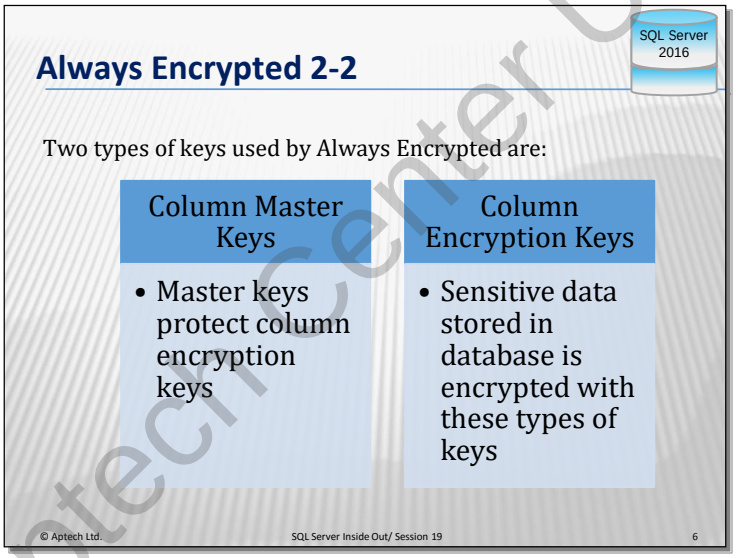


Using slide 3, give one or two line introduction about all the four security features. Then, using slide 4, explain the pictorial representation of the security improvements.

Slides 5 and 6

Let us understand Always Encrypted feature of SQL Server 2016.





Using slide 5, explain the students about Always Encrypted feature in detail. Describe the two types of encryption supported by Always Encryption, namely, Randomized Encryption, and Deterministic Encryption.

Tell the students that:

Always Encrypted is one of the new feature introduced in SQL Server 2016. It enables encryption of both data in use and data that is stored. The data is encrypted while in memory, enabling protection from unauthorized access. Always Encrypted allows you to encrypt only required columns instead of the entire database, unlike TDE. Always Encrypted also ensures that the data is revealed as plain text only within the application tier and nowhere else. One limitation imposed by Always Encrypted is that, **.NET Framework 4.6** should be installed on the machine that runs client application.

You can refer to the following figure to understand this concept better. You can also share the figure with students for their better understanding.

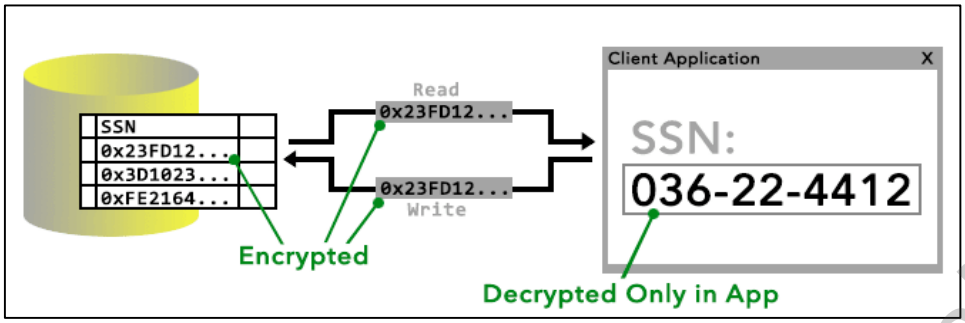


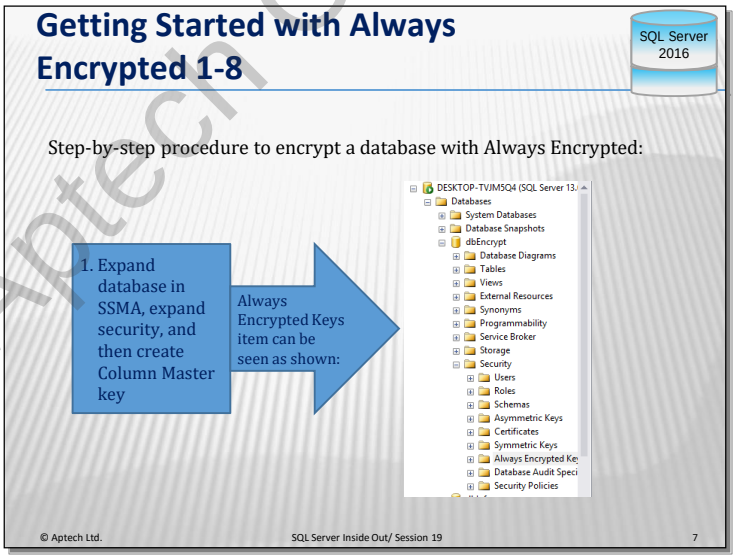
Image courtesy: https://www.mssqltips.com/sqlservertip/4011/sql-server-2016-always-encrypted/

Using slide 6, describe the two types of keys namely Column Master Keys and Column Encryption Keys.

Additional Information:
For more information on Always Encrypted feature of SQL Server 2016, you can visit the following link:

https://www.mssqltips.com/sqlservertip/4011/sql-server-2016-always-encrypted/

Slides 7 to 14
Let us now see the step-by-step procedure to encrypt a database using Always Encrypted feature.



Getting Started with Always Encrypted 2-8

SQL Server 2016

2. Expand 'Always Encrypted Keys' option

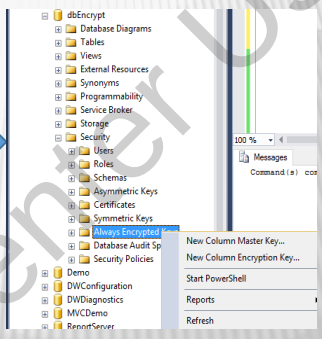
This gives more options as shown:

Getting Started with Always Encrypted 3-8

SQL Server 2016

3. To create Column Master Key, click the 'Column Master Keys' item

This displays options as shown:

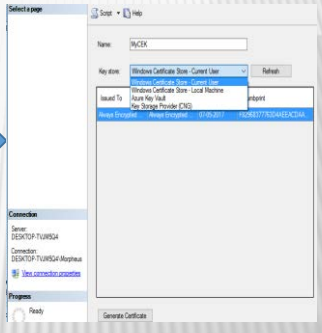


Getting Started with Always Encrypted 4-8

SQL Server 2016

4. Click 'New Column Master Key' item in the drop-down list.

Screen is displayed as shown:

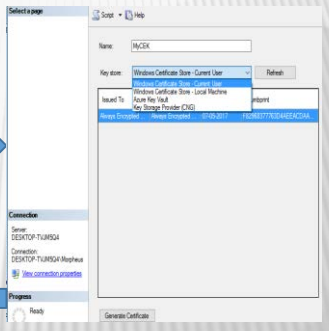


Getting Started with Always Encrypted 5-8

SQL Server 2016

5. Enter a name for Column Master Key in the name section. Expand 'Key store' drop-down box.

Different key store options are displayed as shown:



The encrypted master key 'MyCEK' has been created.

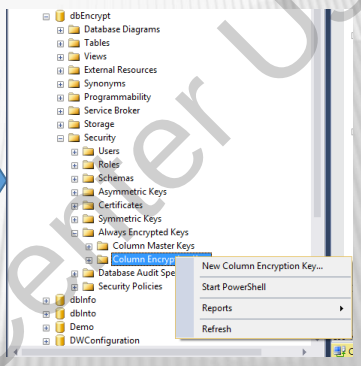
© Aptech Ltd. SQL Server Inside Out/ Session 19 11

Getting Started with Always Encrypted 6-8

SQL Server 2016

6. To create a column encryption key, right-click the 'Column Encryption Key' item in the Object Explorer and then select 'New Column Encryption Key'

The screen is displayed as shown:

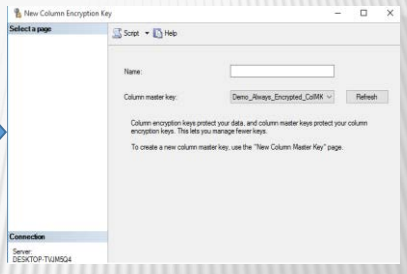


© Aptech Ltd. SQL Server Inside Out/ Session 19 12

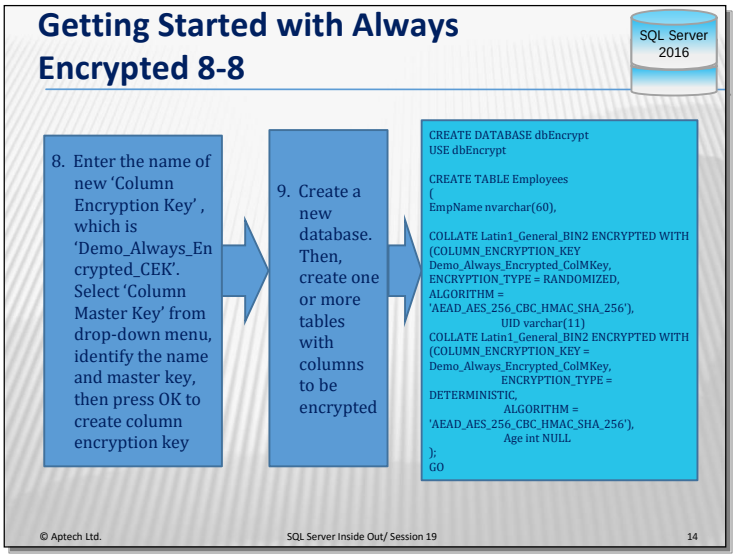
Getting Started with Always Encrypted 7-8

SQL Server 2016

7. Then the 'New Column Encryption Key' is displayed



© Aptech Ltd. SQL Server Inside Out/ Session 19 13



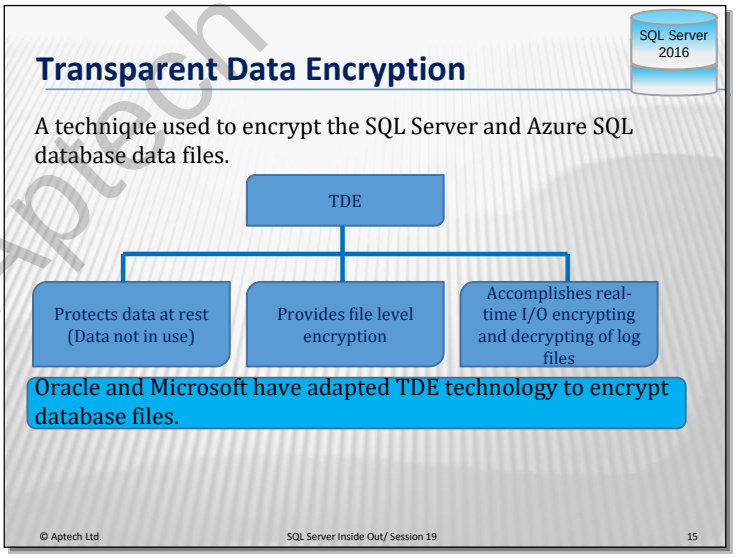
As slides 7 to 14 are self explanatory, use them and show the step-by-step procedure to encrypt the data. Show the example given on the slides to demonstrate the explanations.

Explain using slide 14, the creation of table named Employee and define columns to be encrypted.

The example shows both types of encryption namely **Randomized** and **Deterministic**.

Slide 15

Let us explore another security feature of SQL Server 2016, Transparent Data Encryption, shortly called TDE.



Explain main functionalities of TDE using slide 15.

Tell the students that:

TDE is the main SQL Server encryption possibility that was first introduced in SQL Server 2008. TDE allows you to encrypt the entire database and database backups that use TDE are

also naturally encrypted. TDE uses AES and 3DES encryption algorithms. The implementation of TDE is completely transparent and you need not change any code.

In-Class Question:

After you finish explaining slide 15, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

Consider that a company 'ABC' sells brand 'XYZ' electronic equipment. It has stored sales data from the year 2012 till date in a database namely 'Sales_db2012_tilldate'. Recently, the company moved the data from year 2012 to year 2015 to the database namely 'Sales_db2012_2015', which is rarely used. To encrypt the sensitive data in Sales_db2012_2015, which encryption method would you use?

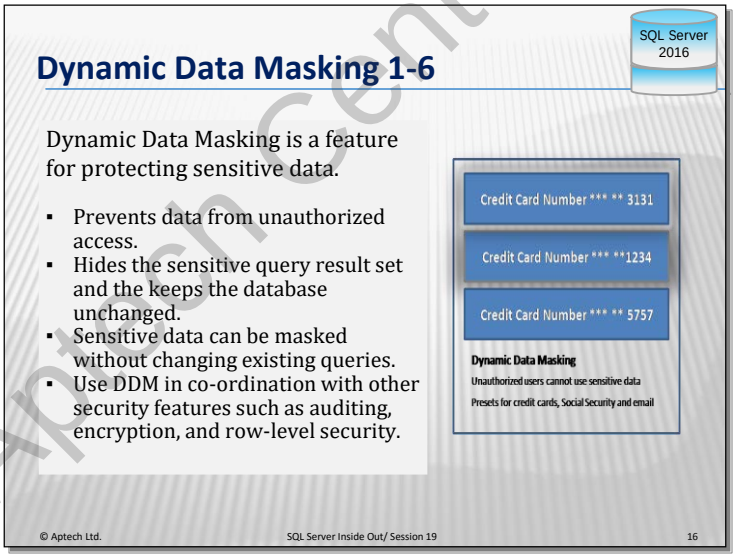
Answer: TDE

Additional Information:

For more information on TDE, you can visit the following link:
https://msdn.microsoft.com/en-in/library/bb934049.aspx

Slides 16 to 21

Let us examine another security feature of SQL Server 2016, Dynamic Data Masking (DDM).



Dynamic Data Masking 2-6

SQL Server 2016

Limitations and restrictions of DDM

The column types that cannot have a masking rule defined on them are:

- Encrypted columns
- Filestream
- COLUMN_SET
- Sparse column that is part of a COLUMN_SET
- Computed column
- FULLTEXT index key

Dynamic Data Masking 3-6

SQL Server 2016

Creating a DDM

```
CREATE TABLE NewGroup
 (UID int IDENTITY PRIMARY KEY,
 FNM varchar(100) MASKED WITH (FUNCTION = 'partial(1,'XXXXXXX',0)') NULL,
 FatherName varchar(100) NOT NULL,
 Mobile varchar(12) MASKED WITH (FUNCTION = 'default()') NULL,
 PersonalEmail varchar(100) MASKED WITH (FUNCTION = 'email()') NULL);
 INSERT NewGroup (FNM, FatherName, Mobile, PersonalEmail) VALUES
('Johnson', 'Flanagan', '12345688', 'JohnnsonFlan@yahoo.com'),
('Rossell', 'Geller', '76543218', 'RossGeller@hotmail.com'),
('Brook', 'Darwin', '88956585', 'Brookdarwin@gmail.com');
SELECT * FROM NewGroup
```


Dynamic Data Masking 4-6

SQL Server 2016

Granting SELECT Permission

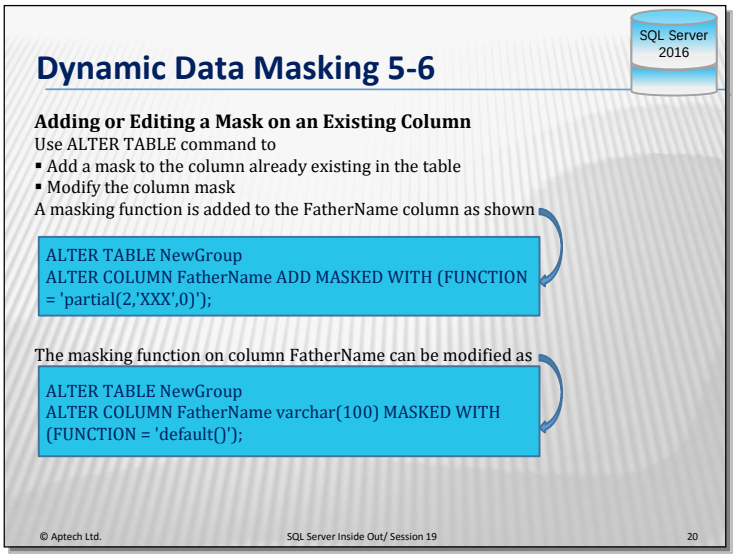
```
CREATE USER DemoUser WITHOUT LOGIN;
GRANT SELECT ON NewGroup TO DemoUser;

EXECUTE AS USER = 'DemoUser';
SELECT * FROM NewGroup;
REVERT;
```

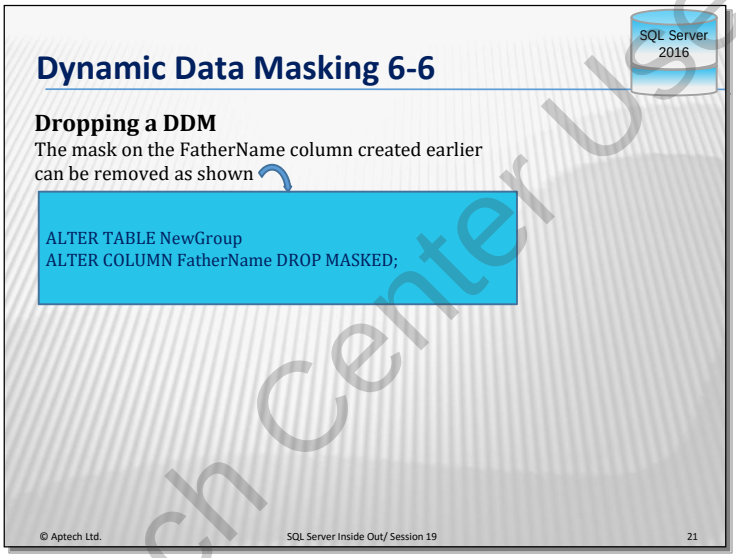
The data is changed from:

1 John Flanagan 12345688 Johnson@yahoo.com

To:

1 JXXXXXXX Flanagan xxxx JXXXX@XXXX.com





Explain all the points given on slide 16 about DDM.

Tell the students that:
Data masking is a security feature that hides sensitive data from certain users. The level of security and privacy of the databases can be enhanced by adopting data masking. The privileged users will get results with the actual data and other users will see the data for which they have permission based on the policies defined by the DBAs. Nevertheless, the data in the database is not changed.

Let us now understand the limitation of DDM.

With the help of slide 17, explain the limitations of DDM.

Let us now learn the SQL commands to create a DDM, granting permission, and altering a DDM.

With the help of slides 18 to 21, explain the following with example:

- Creating a DDM
- Granting SELECT permission to user
- Adding or editing a mask on a column
- Dropping a DDM

Also, explain the four types of masks namely: Default, E-mail, Custom String, and Random

In-Class Question:

After you finish explaining Dynamic Data Mask feature, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

 Company 'PQR', stores its customer database that includes sensitive data such as 'email-id' and 'mobile-number' of the customer. Which security policy would you recommend to secure the columns 'email-id' and 'mobile-number'? How would you implement it?
Answer: Dynamic Data Mask.
Implementation: Define mask types:
'**E-mail**' for column 'email-id'
'**Random**' for column 'mobile-number'

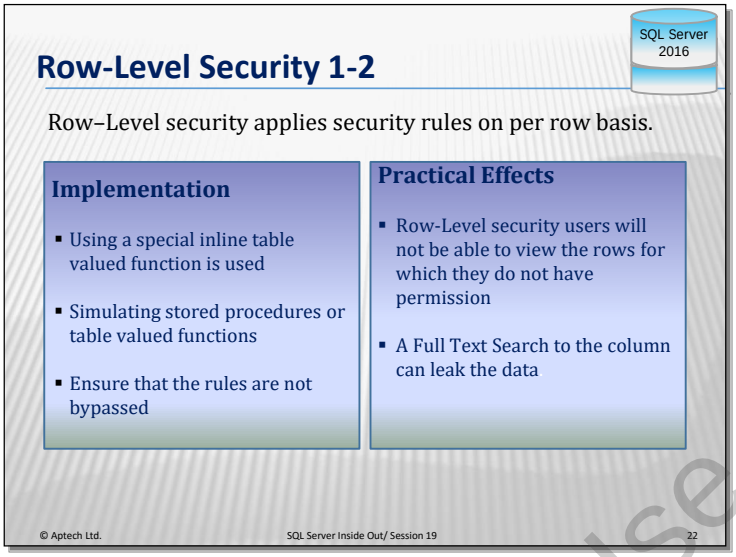
Additional Information:

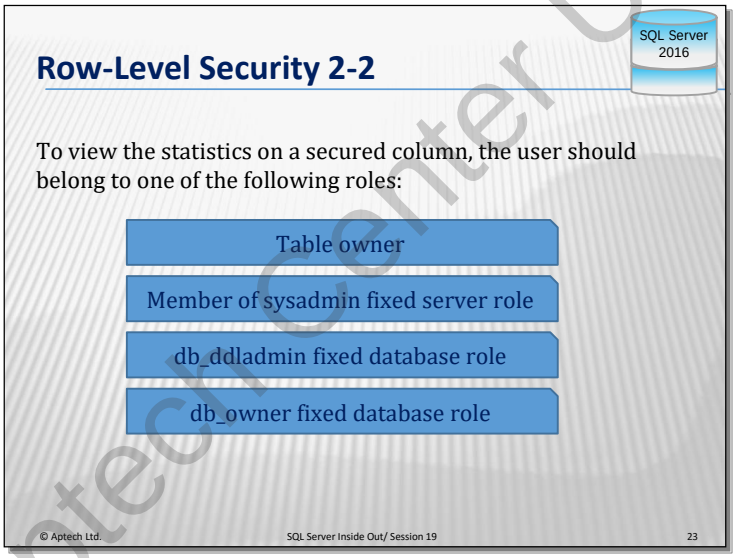
For more information on Dynamic Data Mask, you can visit the following links:

https://blog.oraylis.de/2015/12/sql-server-2016-dynamic-data-masking/
http://www.sqlservercentral.com/blogs/the-dba-who-came-in-from-the-cold/2016/04/13/sql-2016-dynamic-data-masking/

Slides 22 and 23

Let us understand the row-level security feature of SQL Server 2016.





Firstly explain what row-level security is. With the help of slide 22, describe its implementation and its practical impacts.

Tell the students that:
Row-level Security is another security feature provided by SQL Server 2016 that enables the user to define security at the row-level within the database in concern. Row-level Security (RLS) can be implemented using a special inline table value function and hence the developer need not change the code in his or her application.

Slide 23 shows the user roles that can access secured columns defined by RLS.

Additional Information:

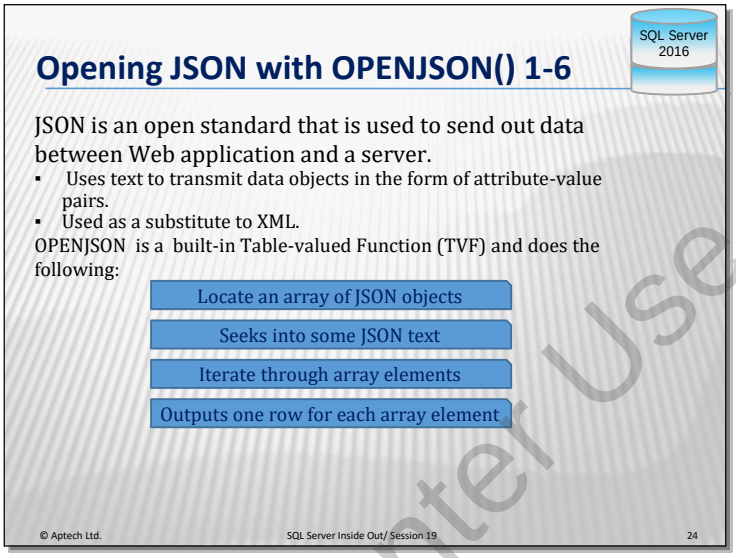
For more information on RLS, you can visit the following links:

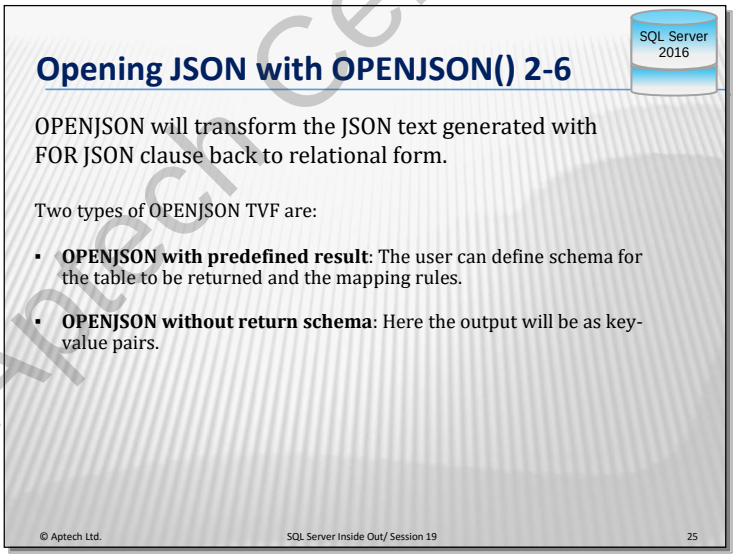
http://www.databasejournal.com/features/mssql/row-level-security-with-sql-server-2016.html

https://channel9.msdn.com/Shows/Data-Exposed/SQL-Server-2016-Row-Level-Security

Slides 24 to 29

Let us now understand more about JSON that is using OPENJSON function.





Opening JSON with OPENJSON() 3-6

SQL Server 2016

OPENXML is used as the framework for OPENJSON. Though the usage looks identical, there are elemental differences:

OPENJSON	OPENXML
Accepts text input Drawback: When two identical JSON text are passed to different OPENJSON calls, they will be parsed each time.	Accepts handle.
Return row with (key, value) schema.	Parses entire XML tree without distinct schema. Returns a flat table with all nodes.

Opening JSON with OPENJSON() 4-6

SQL Server 2016

Converting JSON to Rowset Data Using the OPENJSON Function:
The OPENJSON rowset function is used to convert the data to relational format. It returns following three values:

- **Key**: Key is either the index of an array element or property name within the object.
- **Value**: Value specifies the array element value or the value of the property within the object.
- **Type**: The data type of the value. This is depicted numerically as shown:

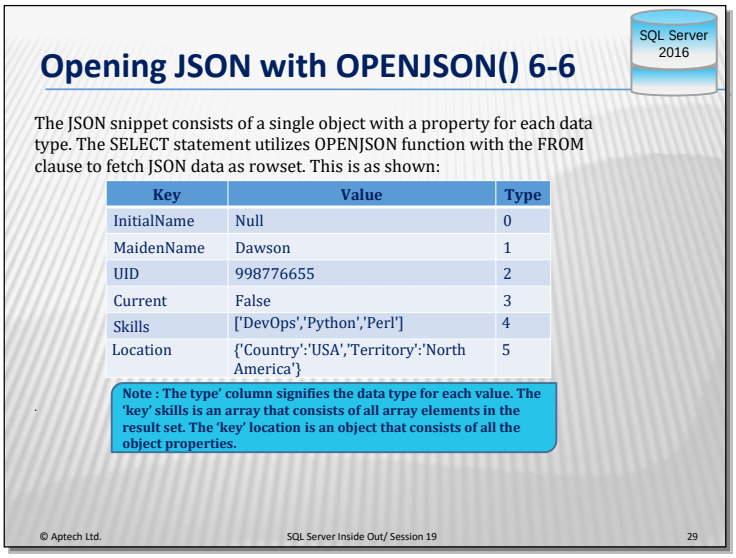
Numeric Value	Data Type
0	null
1	string
2	int
3	True or false
4	array
5	object

Opening JSON with OPENJSON() 5-6

SQL Server 2016

Converting JSON to Rowset Data Using the OPENJSON Function:
The user can allocate JSON snippet to a variable and then use OPENJSON function to call the variable as shown in the example:

```
DECLARE @JSON NVARCHAR (MAX) = N'

{
'InitialName': null,
'MaidenName': 'Dawson ',
'UID': 9988776655,
'Current': false,
'Skills': ['DevOps', 'Python', 'Perl'],
'Region': {'Country':'USA','Territory':'North America'}
}';

SELECT * FROM OPENJSON (@json);
```



Using slides 24 and 25, firstly revise about what JSON does.

Then explain about the OPENJSON function:

- OPENJSON is one of the new JSON function introduced in SQL Server 2016, much identical to the OPENXML function.
- This table valued function can be used to transform the JSON text to one or many rows. Basically, it provides a row-set view of the input JSON string.

Describe about the two types of OPENJSON Table Valued Function (TVF).
Show examples for both OPENJSON with predefined result and OPENJSON without return schema.

Using slide 26, differentiate between OPENJSON and OPNEXML functions.

Slide 27 explains the values returned by the OPNJSON function while converting JSON data to rowset data.

Now let us see OPENJSON in action:
Use slides 28 and 29 to demonstrate with an example the conversion of JSON data to rowset data.

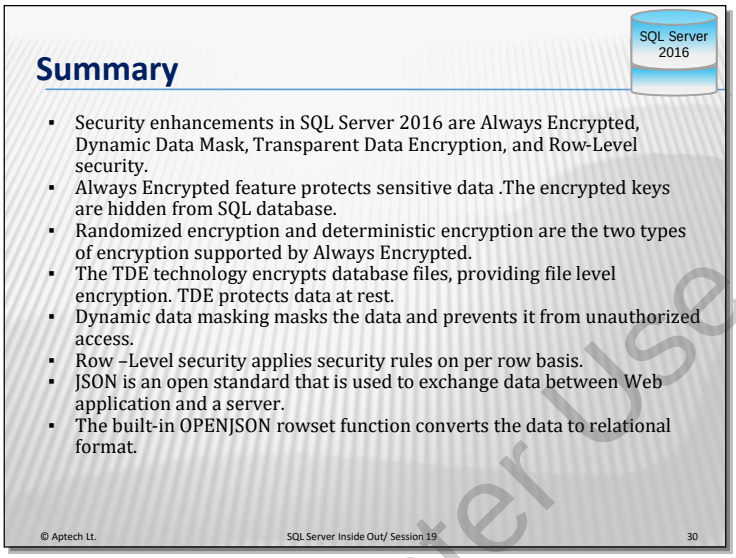
In-Class Question:

After you finish explaining OPENJSON function, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

What do you do to obtain more readable output while using OPENJSON function?
Answer: Use the WITH clause to SELECT statement to specify an explicit schema, and create column definitions to obtain more readable column names.

Additional Information:
For more information on OPENJSON function, you can visit the following links:
http://sqlhints.com/2015/11/22/openjson-function-in-sql-server-2016/
https://msdn.microsoft.com/en-us/library/dn921879.aspx

Slide 30
Let us summarize the session.



Using slide 30, summarize the session. Explain each of the points given on the slide in brief.

Conclude the session by saying that:
In this session, the students learnt the enhanced security features of SQL Server 2016, implementation, the practical impacts it may have, and the OPENJSON function.

19.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session.

Tips: You can also check the **Articles/Blogs/Expert Videos** uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 20: PolyBase, Query Store, and Stretch Database

20.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

20.1.1 Objectives

By the end of this session, learners will be able to:

- Describe PolyBase
- Explain features and advantages of PolyBase
- Define and describe Query Store
- Explain how to dynamically stretch warm and cold transactional data from SQL Server to Azure
- Describe how to tune workload performance with Query Store

20.1.2 Teaching Skills

To teach this session, you should be well versed with the new features and enhancements in SQL Server 2016. You also need to know about SQL Server 2014 and how some of the earlier features are improved in SQL Server 2016. You should be familiar with:

- PolyBase; accessing and working with external databases such as Hadoop and related concepts
- Query Store feature and performance tuning
- Stretch database

You should teach the concepts in the theory class using the slides and images on the slides that support the content of the slide. Elaborate on the points shown on the slides and explain any examples provided on the slides to enable better understanding.

Tips:
Interact with the class, refresh any previous concepts, if required. You can use your own experience and knowledge, suggestions provided in this guide or any Websites that give more information. Some references are provided in this guide.

It is recommended that you test the understanding of the students by asking questions in between the class.

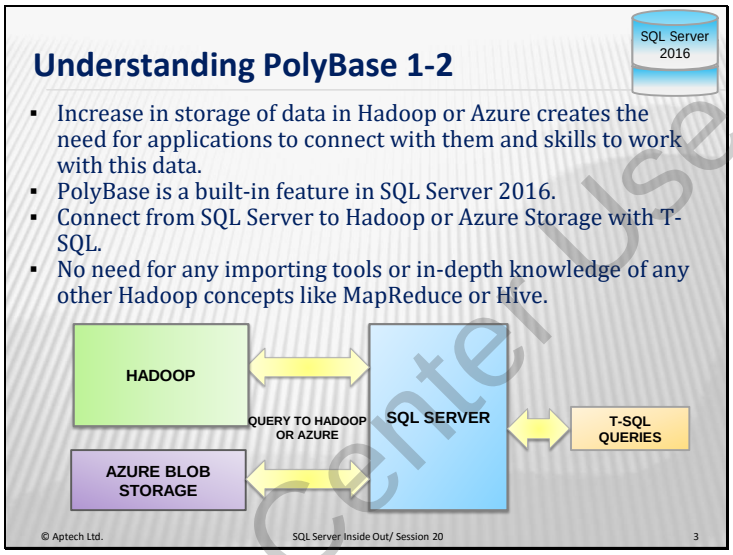
Overview of the Session

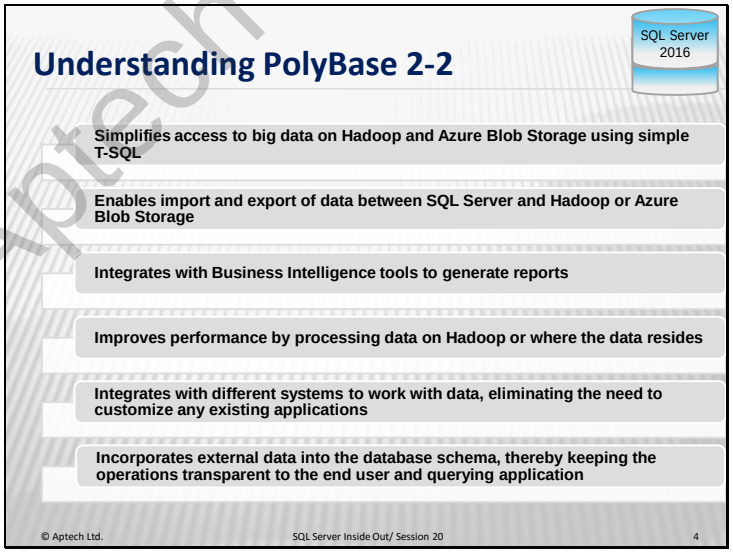
Give the students an overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

20.2 In-Class Explanations

Slides 3 to 7

Let us understand the PolyBase feature, its architecture, and learn how to install the feature in SQL Server 2016.





PolyBase Architecture

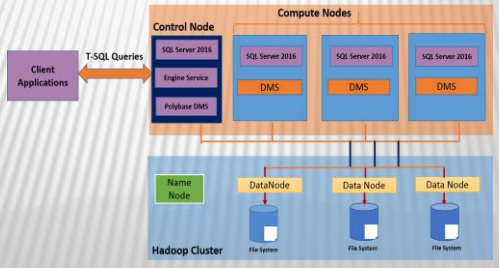
SQL Server 2016

CONTROL NODE

- Similar to Hadoop's Name Node/Job tracker and SQL Server instance acts as the Control Node
- Runs the PolyBase Engine Service
- Manages queries between applications and external data sources

COMPUTER NODE

- Similar to Hadoop's Data Node/Task tracker
- Runs PolyBase Data Movement Service
- Brings data temporarily to SQL Server to handle the parallel query executions

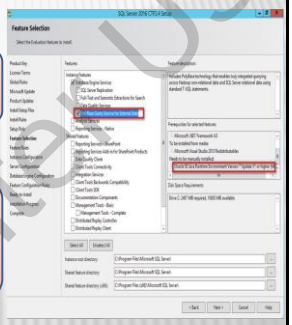


Installing and Setting up PolyBase 1-2

SQL Server 2016

1. Select PolyBase feature during SQL Server set up
2. Ensure JRE installation
3. Configure interaction with Hadoop or Azure Storage

```
EXEC sp_configure , hadoop connectivity, 5;
GO
RECONFIGURE;
GO
```



4. Create a data source object that defines the connection to the external data source:

```
CREATE EXTERNAL DATA SOURCE HDP
WITH
(
TYPE='HADOOP'
Location='hdfs://sandbox.hortonworks.com:8020';
```


Installing and Setting up PolyBase 2-2

SQL Server 2016

5. Define a file format object to specify the format of external data by using the statement: CREATE EXTERNAL FILE FORMAT.

```
CREATE EXTERNAL FILE FORMAT TSV
WITH (
        FORMAT_TYPE = DELIMITEDTEXT,
         FORMAT_OPTIONS (
               FIELD_TERMINATOR = , \T',
               DATE_FORMAT = , MM/DD/YYYY'
```

6. Create external tables using the CREATE EXTERNAL TABLE statement to facilitate a schema-based structure.

```
CREATE EXTERNAL TABLE sample_1
(
Code nvarchar (255),
Description nvarchar (255),
Total_emp int,
Salary nvarchar (255)
)
WITH
(
LOCATION = '/APPS/HIVE/WAREHOUSE/SAMPLE_07',
DATA_SOURCE = HDP2,
FILE_FORMAT=TSV,
REJECT_TYPE=value,
REJECT_VALUE=0
)
```

7. Verify creation of External Resources and External tables folders.

- Using **slide 3**,
  - Cover how previously Microsoft analytics in SQL Server Parallel Data Warehouse was used to connect to data in Hadoop and how working knowledge of MapReduce or Hive applications was required.
  - Explain how the new built-in feature PolyBase that overcomes these challenges by enabling direct interaction using simple T-SQL commands.
  - Cover the concept of combining relational data on SQL server and non-relational data on Hadoop.
  - Use the image given on the slide to illustrate the interaction between Hadoop, SQL Server, and usage of T-SQL commands.
- Using **slide 4**,
  - List the advantages of PolyBase as given on the slide.
  - Explain that this feature creates an external tables folder which lists any external tables.
  - Discuss that queries are run on Hadoop, where the data resides, thus, eliminating the need to import the data to SQL Server.
  - List or interact to touch upon business intelligence tools such as PowerPivot which can use the data in Hadoop to generate comprehensive reports.
- Using **slide 5**,
  - Explain using the image on the slide that PolyBase architecture is similar to Hadoop architecture because its main function is to interact with Hadoop data.
  - Elaborate on the two main components – Control Node and Compute Node.
  - Explain their functions as listed on the slide.
- Using **slides 6 and 7**,
  - Explain the steps to set up PolyBase during SQL Server installation.
  - Explain specifying Hadoop flavors and list the flavors supported (Cloudera and Hortonworks).
  - Mention file formats supported.
  - Explain using the clauses: sp_configure, CREATE EXTERNAL DATA SOURCE, CREATE EXTERNAL FILE FORMAT, and CREATE EXTERNAL TABLE.
  - Mention formation of new folder External Resources is created under the databases folder. The data sources and file formats are listed under this folder. External Tables containing defined external tables is created under the Tables folder.

In-Class Questions:

After you finish explaining the PolyBase feature, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

 Can you name the feature in SQL Server 2016 that should be used to combine relational data in local database and non-relational data on Hadoop and generate BI Reports with ease?

Answer: PolyBase

What is an important pre-requisite to PolyBase setup?
Answer: JRE Installation

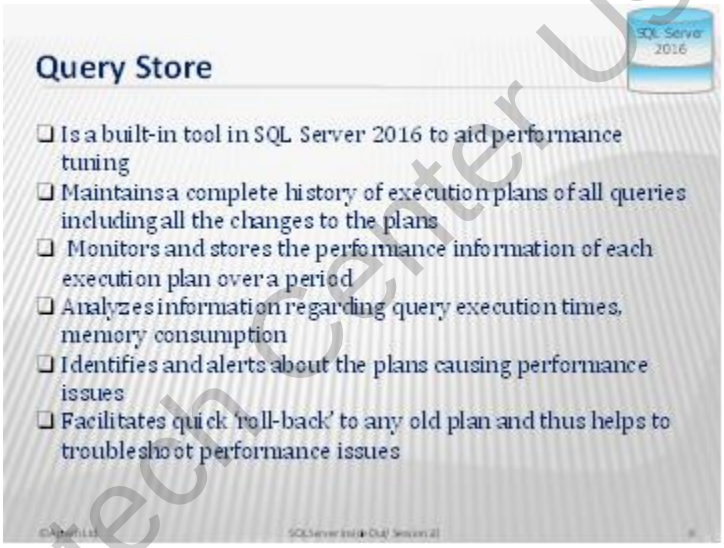
Which part of Hadoop architecture is similar to the control node in PolyBase?

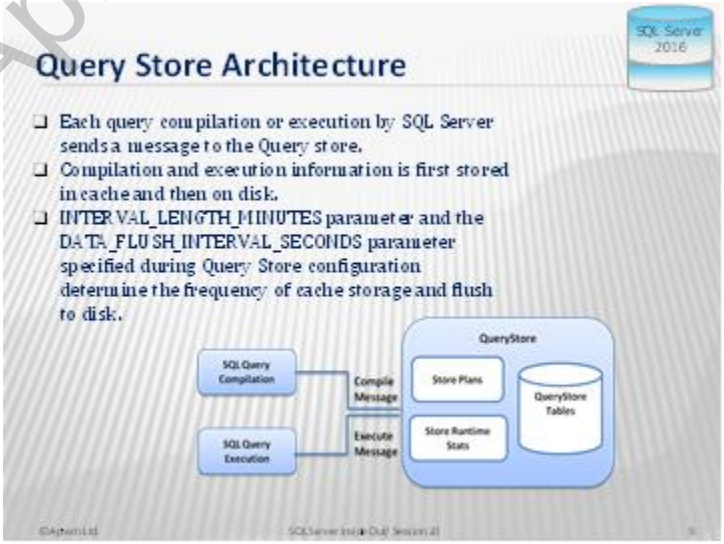
Answer: Name node

Additional Information:

For more information, visit the following links:
www.cs.put.poznan.pl/events/Microsoft-PolyBase.pdf
http://www.jamesserra.com/archive/2014/02/polybase-explained/

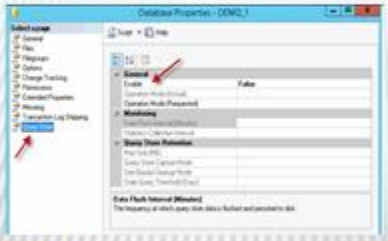
Slides 8 to 13
Let us learn about the query store feature and its advantages.





Enabling Query Store

- Two methods to enable the Query Store on a database:
  - Using GUI inside SQL Server Management Studio (SSMS)

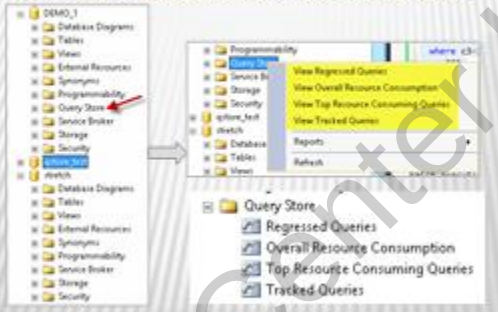


  - Using T-SQL:

```
ALTER DATABASE [DEMO] SET QUERY_STORE = ON
GO
```

Viewing Query Information

- To view information about queries right- click or expand the Query Store container and select the required options.

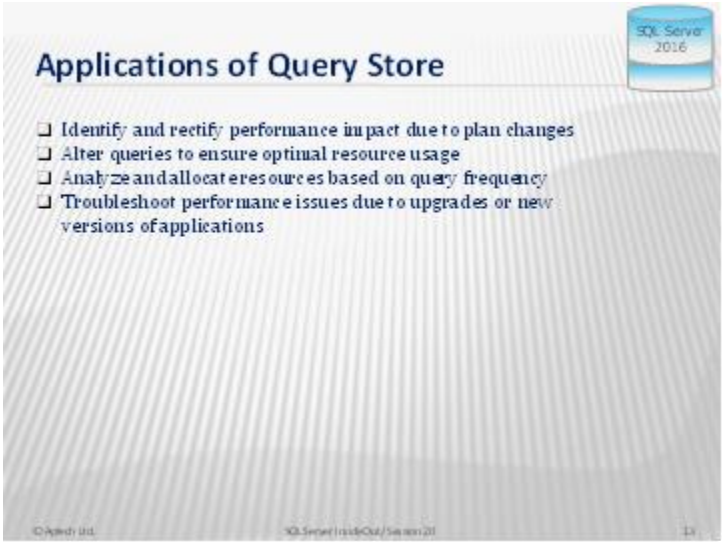


Configuring Query Store

- Setup parameters using SSMS

- Setup parameters using T-SQL

```
ALTER DATABASE [DEMO_1]
SET QUERY_STORE (OPERATION_MODE = READ_ONLY,
                 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
                 DATA_FLUSH_INTERVAL_SECONDS = 900,
                 INTERVAL_LENGTH_MINUTES = 60,
                 MAX_STORAGE_SIZE_MB = 100,
                 QUERY_CAPTURE_MODE = AUTO,
                 SIZE_BASED_CLEANUP_MODE = AUTO)
GO
```



- Using **slide 8**,
  - Explain the query store feature and highlight its uses with the help of slide.

- Using **slide 9**,
  - Explain the query store architecture where the query information - execution plans and runtime stats are stored by Query Store.
  - Explain that when a query gets submitted against a database that has the Query Store enabled, the compiled query Execution Plan is written to the Query Store Plan Store. Also tell that the runtime information of the query is recorded in the Runtime Stats Store.
  - Explain that initially the information is stored in cache.
  - Tell the students that after a specific interval the information is written to disk.
  - Explain that frequency is based on the specified parameters.
  - Explain that information can be retrieved anytime, even after server restart.

- Using **slides 10 and 11**,
  - Explain about setting the query store enable to 'True' in SSMS or 'ON' with T-SQL.
  - List and explain the information about queries that can be viewed using the SSMS.
  - Give an instance or application in real life when a list of regressed queries or resource consumption details are required or encourage participants to share any such requirements in their experience.
- Using **slide 12**,
  - Describe how to specify the values to configure Query store. Use the image given on the slide to show the SSMS options and the SQL Code.
- Using **slide 13**,
  - Go through the Query Store applications as listed on the screen.
  - Optionally, interact with the class to state any personal experiences of issues with troubleshooting and difficulty to identify them.

In-Class Questions:

After you finish explaining the Query Store feature, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

Which feature in SQL Server 2016 helps to view a list of all queries that are consuming major resources?

Answer: Query Store feature and View Top Resource Consuming Queries option

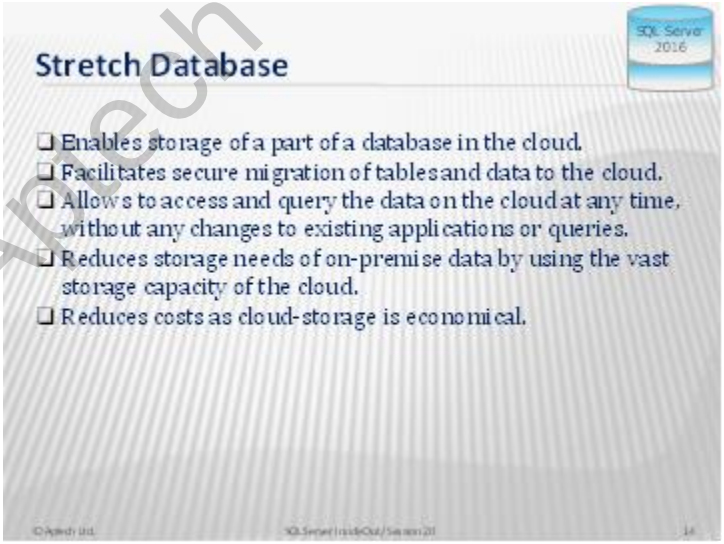
Which parameter must be configured in the Query Store feature to specify the duration of keeping the information in the cache and the time when it is stored on the disk?

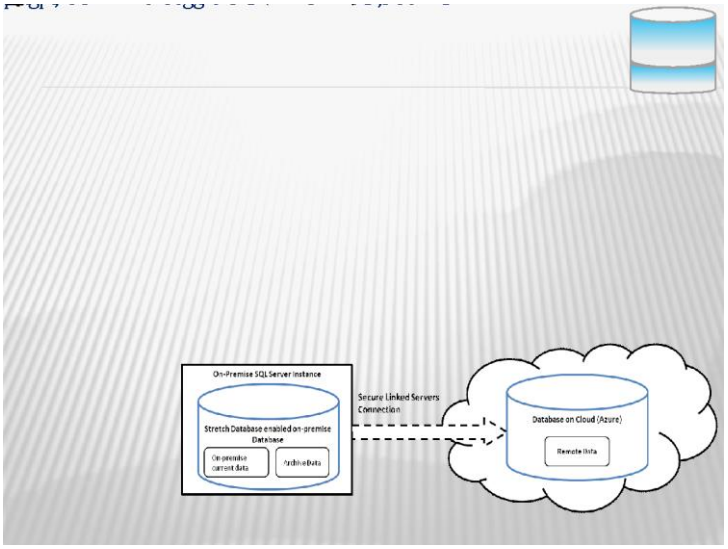
Answer: INTERVAL_LENGTH_MINUTES parameter and DATA_FLUSH_INTERVAL_SECONDS

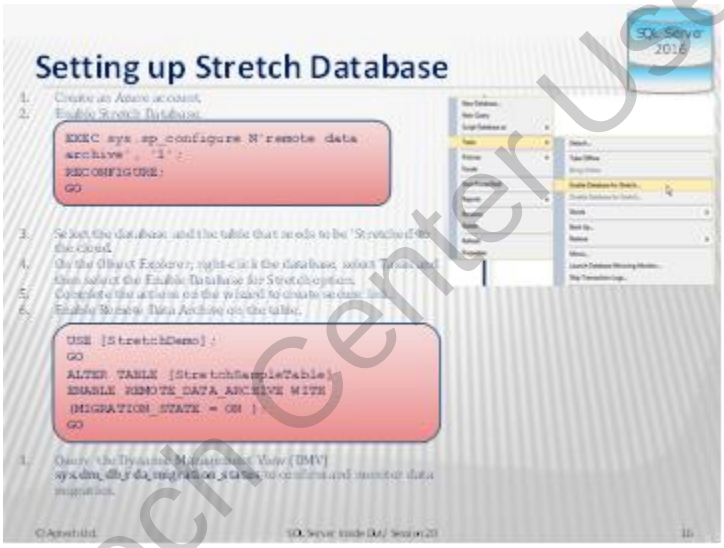
Additional Information:

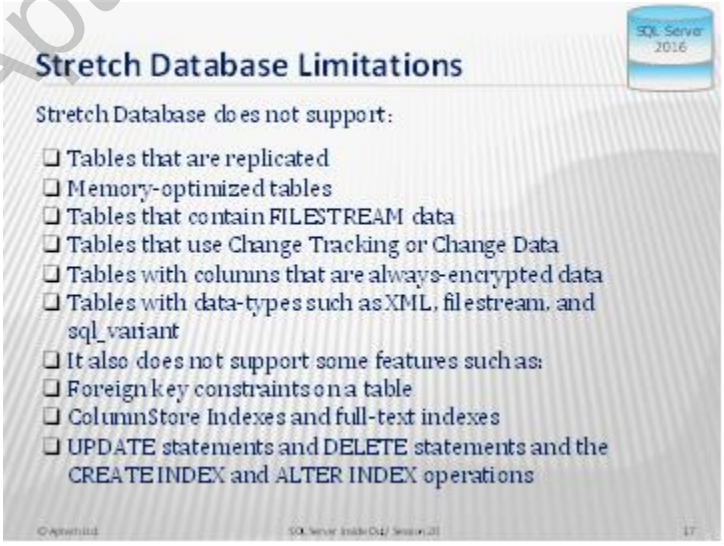
For more information, visit the following links:
http://slavasql.blogspot.in/2015/09/sql-server-2016-query-store.html
http://windowsitpro.com/sql-server-201s6/what-sql-server-2016-query-data-store
http://www.infoq.com/news/2015/07/SQL-Server-Query-Store

Slides 14 to 17
Let us understand the Stretch database feature and the concept of migrating and storing data on the cloud using SQL Server 2016.









- Using **slide 14**,
  - Explain how storing data for long periods on-premise is costly and how the facility to move it to the cloud and also query the data with the same ease as with on-premise data is useful.
  - List the advantages of the Query store feature. Mention the uses of being able to access different versions of row data, such as analyzing trends, support business decisions, and revert to previous version of data in case of errors.
- Using **slide 15**,
  - Explain how a secure connection is created that helps interaction with the database on the cloud.
  - With the help of the image given on the slide, explain that data is migrated to the cloud when Stretch database is enabled.
- Using **slide 16**,
  - List the steps to enable Stretch database with the help of the slide.
  - Explain the SQL code to enable migration to of data to the database on cloud/Azure.
  - Tell about using the dynamic management view to confirm whether the data is migrated and to monitor the migration.
- Using **slide 17**,
  - List the features that are not supported by Stretch database as shown on the slide.
  - State from experience or knowledge about any issues encountered when using these features.

In-Class Question:

After you finish explaining the Stretch Database feature, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

 Can memory-optimized tables be migrated to an Azure database?

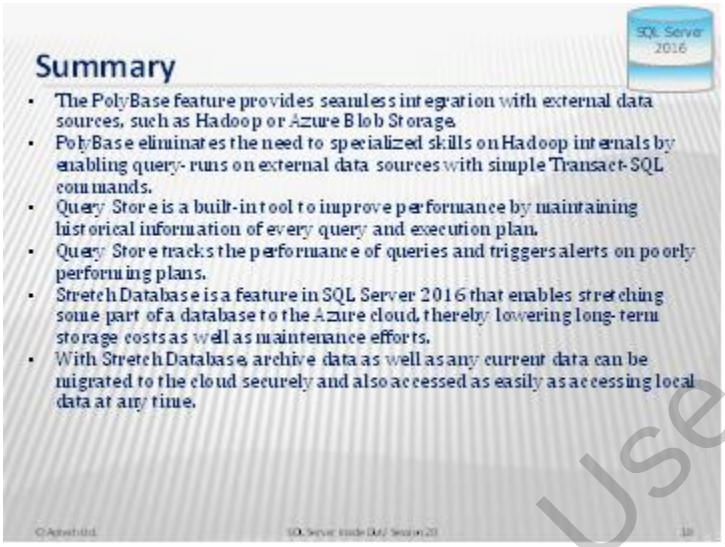
Answer**:** No. Memory-optimized tables are not supported by Stretch database that is the feature enabling migration of data to Azure

Additional Information:

For more information, visit the following links:
http://www.thewhir.com/web-hosting-news/new-microsoft-hybrid-storage-capabilities-stretch-databases-between-on-prem-and-azure-cloud
http://www.databasejournal.com/features/mssql/getting-started-with-stretch-database-functionality-in-sql-server-2016-part-1.html

Slide 18

Let us summarize the session.



Using **slide 18**,

- Summarize the session. Explain each of the points on the slide in brief.

20.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session, which covers more performance tools and T-SQL enhancements made in SQL Server 2016.

You can refer to Microsoft articles on SQL Server 2016 pertaining to concepts covered in the session for detailed information that would help you answer any queries in the class.

Tips: You can also check the **Articles**/**Blogs**/**Expert Videos** uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 21: Improved Performance Tools and Transact – SQL Enhancements

21.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

21.1.1 Objectives

By the end of this session, learners will be able to:

- Explain how to start and use Database Engine Tuning Advisor in SQL Server 2016
- Explain the set of tools for monitoring events in SQL Server 2016 and for tuning the physical database design
- Outline the enhancements in Transact-SQL

21.1.2 Teaching Skills

To teach this session, you should be well versed with the new features and enhancements in SQL Server 2016. You also need to know about SQL Server 2014 and how some of the earlier features are improved in SQL Server 2016. You should be familiar with:

- Memory-optimized tables, their use, and how SQL Server 2016 supports working with these tables
- Different performance tuning tools and their applications
- Transact SQL statements and clauses that are supported in SQL Server 2016

You should teach the concepts in the theory class using the slides and images on the slides that support the content of the slide. Elaborate on the points shown on the slides and explain any examples provided on the slides to enable better understanding.

Tips:
Interact with the class, refresh any previous concepts, if required. You can use your own experience and knowledge, suggestions provided in this guide or any Websites that give more information.

Some references are provided in this guide.

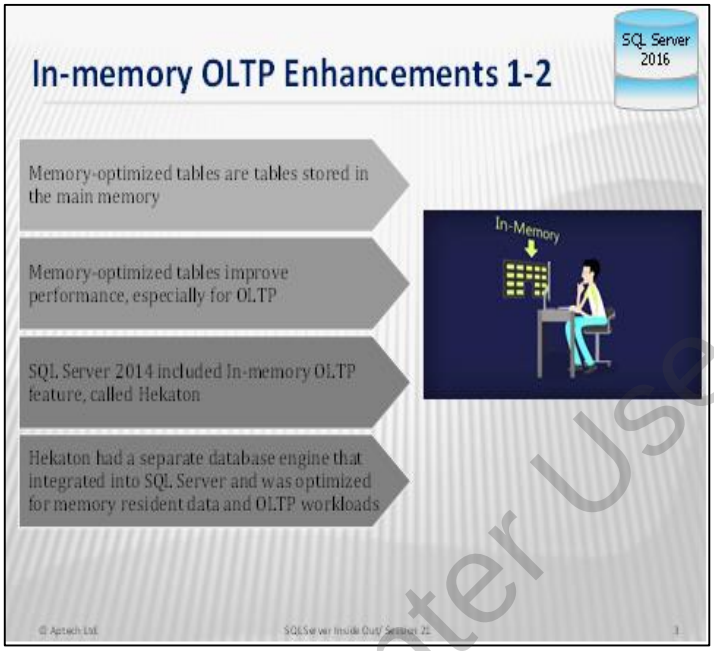
It is recommended that you test the understanding of the students by asking questions in between the class. Use the In-Class questions to review the understanding of a topic.

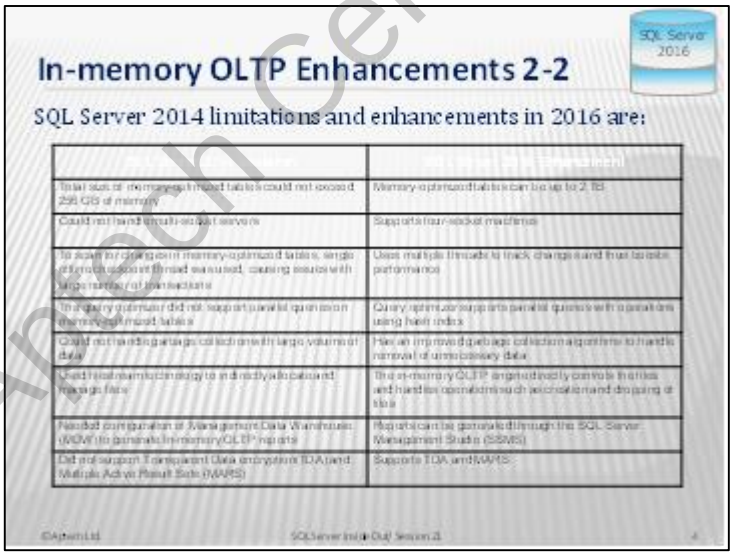
Overview of the Session
Give the students an overview of the current session in the form of session objectives. You may briefly discuss about previous versions of SQL Server. Briefly tell about SQL Server 2016 release date and version. Show the students slide 2 of the presentation.

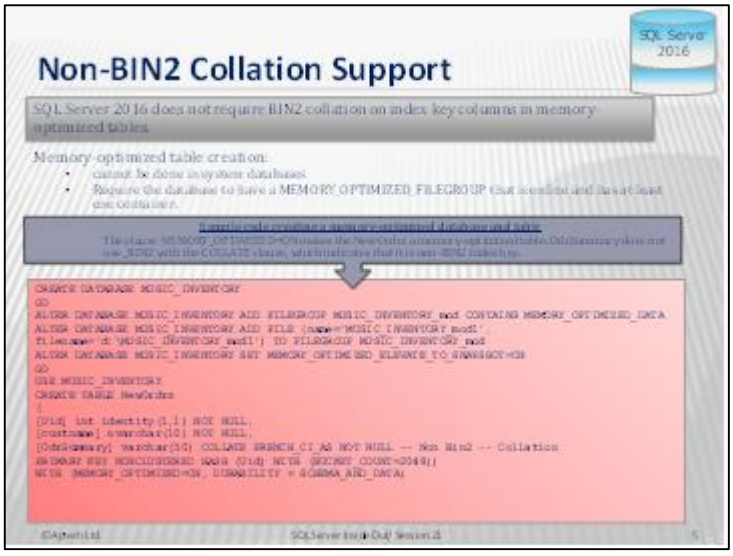
21.2 In-Class Explanations

Slides 3 to 5

Let us understand about using memory-optimized tables and In-memory OLTP enhancements in SQL Server 2016.







- Using **slide 3**,
  - Explain memory-optimized tables and their advantages – faster access and improved performance.
  - Interact with students to recap about how these features were used in SQL Server 2014.

- Using **slide 4**,
  - Explain how scalability and performance are improved for memory-optimized tables in the current version as compared to 2014 version that could not handle large sizes of these tables.
  - In-memory OLTP engine used a single offline checkpoint thread per database to scan the transaction log for changes related to memory-optimized tables that meant for large number of transactions, the thread could fall behind. Whereas in 2016 version, multiple log-reader threads help to boost performance.
  - For hash-indexed operations and for procedures that are not natively compiled, query optimizer 2016 version supports parallel querying.
  - Briefly, recap by interacting about TDA and MARS.

- Using **slide 5**,
  - Interact with students to recap about why collate BIN2 clause is used and how index columns in memory-optimized tables were not supported unless they were BIN2 collated.
  - Go through code given on the slide to explain Non-BIN2 support for memory-optimized tables in SQL Server 2016.

After you finish explaining slide 5, you can ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

In-Class Question:

 What is the size of memory-optimized tables that SQL Server 2016 supports?

Answer: 2 TB

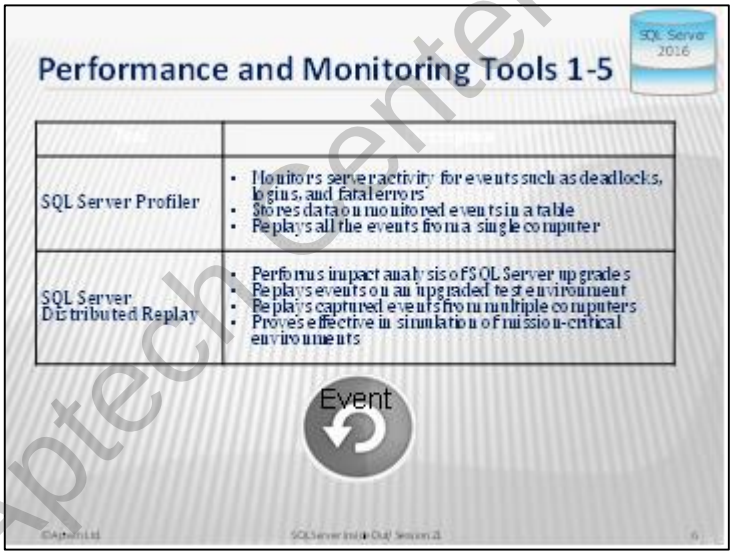
Additional Information:

For more information, visit the following links:

- http://sqlperformance.com/2015/11/sql-server-2016/in-memory-oltp-enhancements
- http://logicalread.solarwinds.com/sql-server-2016-inmemory-oltp-pd01/#.VxNaxfl97cc

Slides 6 to 12

Let us learn about the performance and monitoring tools in SQL Server 2016.



Performance and Monitoring Tools 2-5

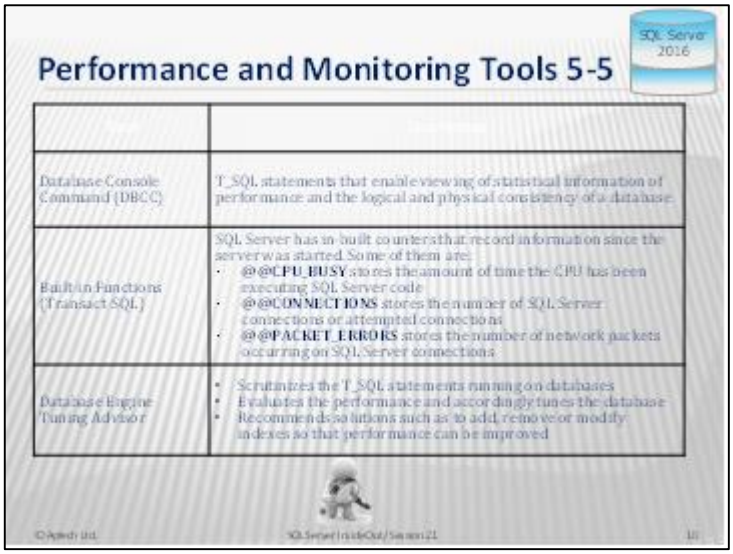
Tool	Description
Monitor Resource Usage (System Monitor)	• Monitors resource usage by server processes • Captures server performance using system-defined counter • Captures event information using user-defined counters • Sets thresholds on counters • Triggers alerts as required • Monitors SQL Server and Windows operating system simultaneously to understand any interrelated issues
Open Activity Monitor (SQL Server Management Studio)	• Available in SQL Server Management Studio • Monitors processes on the SQL Server • Reports information about user activity, any blocked processes and locks

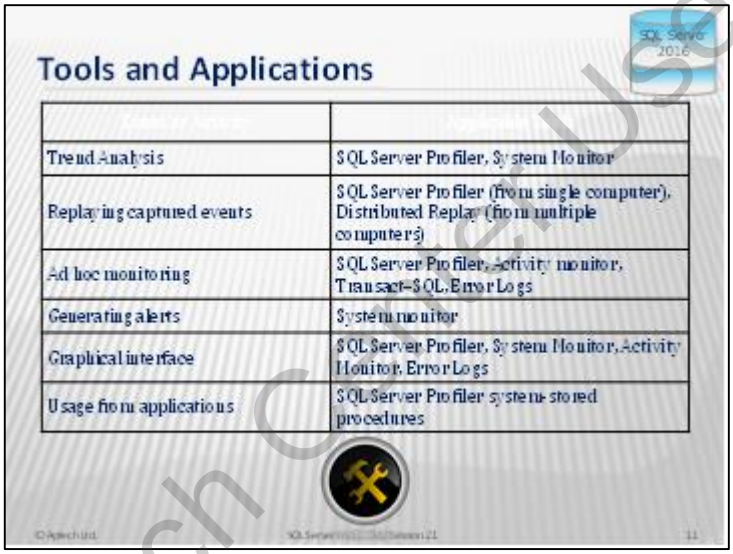
Performance and Monitoring Tools 3-5

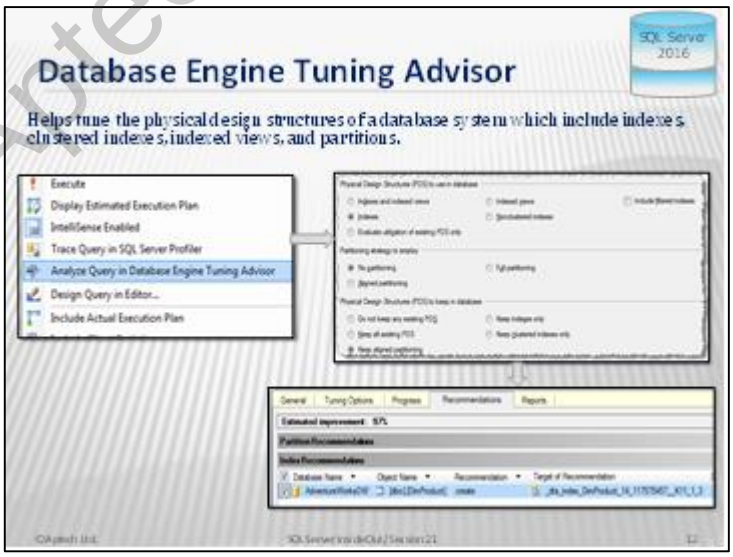
Tool	Description
Live Query Statistics	• Troubleshoots performance issues of queries • Generates statistical information about on-going queries • Allows viewing of real-time query statistics on SSMS
SQL Trace	• Are T-SQL procedures that create traces on a SQL Server Database engine. • Can run from the applications directly without the need to use the SQL Server Profiler. The procedures are · sp_trace_create · sp_trace_generateevent · sp_trace_setevent · sp_trace_setfilter · sp_trace_setstatus

Performance and Monitoring Tools 4-5

Tool	Description
Error Logs	• Help debug server problems. • Log information about Windows Server and OS events and also about events occurring in SQL Server and full-text searches.
System Stored Procedures (Transact-SQL)	Monitor and provide information: · sp_who: Stores users and processes information. Information about the on-going statement and reports if it is blocked · sp_lock: Store information on locks such as object ID, index ID, type of lock, and type of resource on which the lock is applied · sp_spaceused: Logs information about the amount of disk space used by tables or by database · sp_monitor: Logs information about CPU usage, I/O usage, and idle time between executions of two sp_monitor statements







- Using **slide 6**,
  - List the main features of SQL Server profiler and SQL Server Distributed Replay.
    Explain with any real life application example how traces can be managed and replayed with these tools.
    For example, an administrator to can review any of the auditing events, such as success and failure of a login attempt or of permissions in accessing statements and objects.
  - Explain how performance information can be gathered by tracing events during development phase of a software implementation.
  - Explain that the tools can be used during troubleshooting by capturing events on the production system and replaying them on the test system, so that the production system can be used without interference.
  - Highlight the difference between the two tools that Distributed Replay can replay events from multiple computers unlike SQL Server Profiler.
- Using **slide 7**,
  - Explain the features of Monitor Resource Usage and the Open Activity Monitor.
  - Give an example such as monitoring events when 'n' number of users are logged on so that the information can be used to improve performance or to debug on receiving alerts for specific events.
- Using **slide 8**,
  - Explain how obtaining live statistics will help troubleshoot issues immediately.
  - Explain how traces help in obtaining event information and work as an alternative for using SQL Server Profiler.
- Using **slide 9**,
  - Discuss and ask participants to provide any examples of using error logs.
  - Explain how the system stored procedures help to get information by constantly monitoring the processes.
- Using **slide 10**,
  - Explain functions of built-in counters that record information since starting SQL Server.
  - Explain uses of Database Engine Tuning Advisor (DTA) to do performance tuning.
- Using **slide 11**,
  - Explain how the tools can be decided based on the requirement.
- Using **slide 12,**
  - Go through the steps to use DTA with the images on the slide.
  - Show the selection of the Analyze Query in DTA option and then selection of the required performance information.

After you finish explaining slide 12, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

In-Class Question:

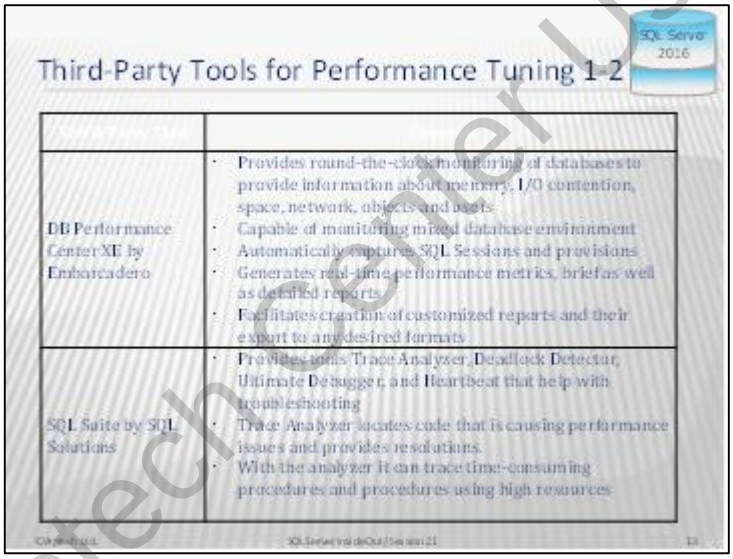
Which tool is good for traces and event replay from single computer?

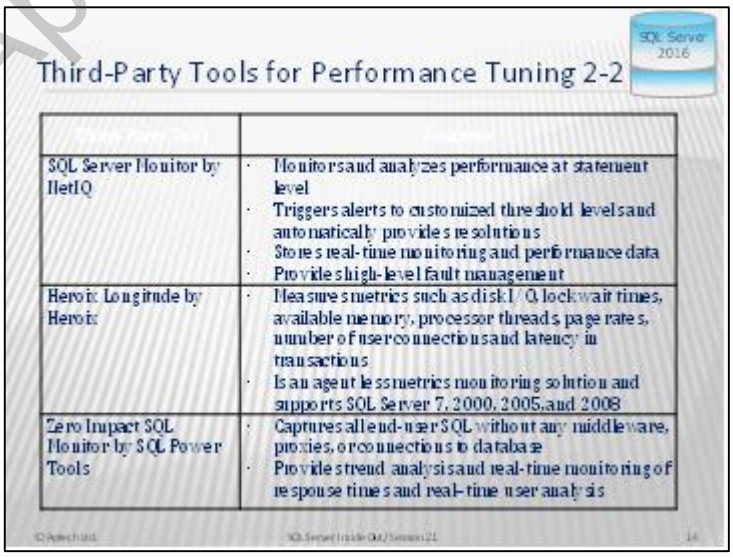
Answer: SQL Server Profiler

Additional Information:

For more information, visit the following link:
https://msdn.microsoft.com/en-gb/library/ms179428.aspx

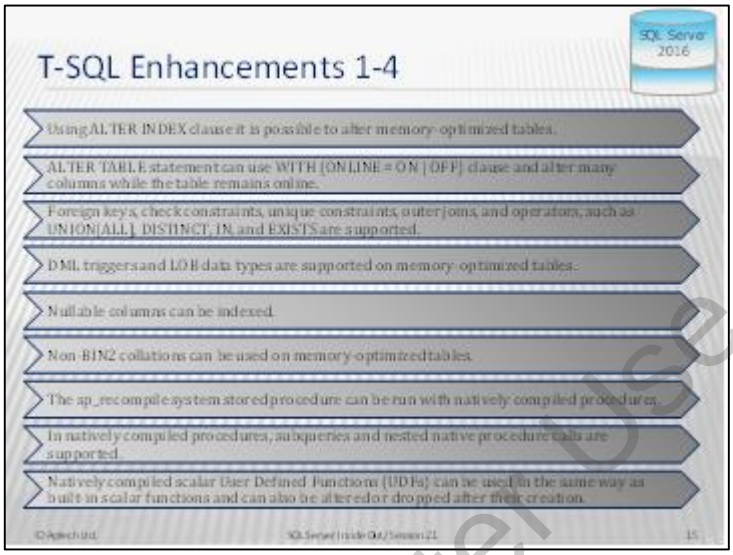
Slides 13 and 14
Let us know the third-party tools that can be used with SQL Server 2016.

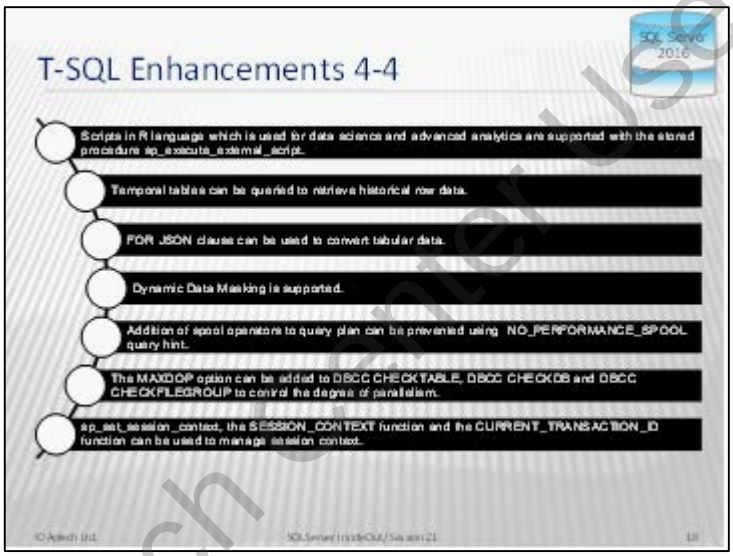




- Using **slides 13 and 14**,
  - List the third-party tools and explain how they can be used to tune performance.

Slides 15 to 18





- Using **slide 15**,
  - List the different T-SQL statements and features that are supported in SQL Server 2016.
  - Explain that earlier memory-optimized tables had to be dropped and re-created if any schema changes were required but now the ALTER INDEX and ALTER TABLE clauses are supported to make schema changes.
- Using **slide 16**,
  - Go through the code on the slide to explain how to truncate large partitioned tables.
  - Explain that this simplifies maintenance on large partitioned tables. It is also much faster than deleting the rows from a partition using the DELETE statement.
- Using **slide 17**,
  - Go through the code on the slide to explain how the existence of a table, column, index, view, and database can be checked before dropping.

- Using **slide 18**,
  - State that users can execute scripts written in another language in SQL Server 2016 and R-Language scripts are supported. Interact with the class to discuss about R Language and how being able to execute these scripts is an advantage.
  - Ask questions to revise from previous sessions how dynamic data masking and JSON clause is supported.
  - Explain how **max degree of parallelism** option can be used to limit the number of processors to use in parallel plan execution.

After you finish explaining slide 18, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

In-Class Questions:

 What external scripts can be executed through T-SQL commands in SQL Server 2016?

Answer: R Language

 Which clause is supported to verify the existence of a table before dropping it?

Answer: DROP IF

 Can the index of a memory-optimized table be altered without dropping it in SQL Server 2016?

Answer: Yes. The ALTER INDEX clause is supported on memory-optimized tables.

Additional Information:

For more information, visit the following link:
https://www.mssqltips.com/sqlservertip/4108/tsql-enhancements-in-sql-server-2016/

Slide 19

Let us summarize the session.

Using slide 19,

- Summarize the session. Explain each of the points on the slide in brief.

21.3 In-Class Activities

You can refer to Microsoft articles on SQL Server 2016 pertaining to concepts covered in this session for detailed information that would help you answer any queries in the class.

Tips: You can also check the **Articles/Blogs/Expert Videos** uploaded on the OnlineVarsity site to gain additional information related to the topics covered in this session.