IBM InfoSphere DataStage and QualityStage
Version 11 Release 3

Connectivity Guide for Oracle Databases

SC19-4266-00

IBM InfoSphere DataStage and QualityStage
Version 11 Release 3

Connectivity Guide for Oracle Databases

SC19-4266-00

Note

Before using this information and the product that it supports, read the information in "Notices and trademarks" on page 177.

Contents

Chapter 1. Connector Migration Tool

To take advantage of the additional functionality that connectors offer, use the Connector Migration Tool to migrate jobs to use connectors instead of plug-in and operator stages.

The following table lists the stages that can be migrated to connectors and the corresponding connectors that they are migrated to:

Table 1. List of stages and corresponding connectors

Stage	Connector stage
DB2Z stage DB2 UDB API stage DB2 UDB Enterprise stage DB2 UDB Load stage	DB2 Connector
DRS Stage	DRS Connector
Java Client stage Java Transformer stage	Java Integration stage
Netezza Enterprise stage	Netezza Connector
ODBC Enterprise stage ODBC (Server) stage SQLServer Enterprise stage	ODBC Connector
Oracle OCI stage Oracle OCI Load stage Oracle Enterprise stage	Oracle Connector
Teradata API stage Teradata Enterprise stage Teradata Load stage Teradata Multiload stage	Teradata Connector
WebSphere® MQ stage	WebSphere MQ Connector

.

Migrating jobs to use connectors

To migrate jobs to use the connectors, you need to run the Connector Migration Tool.

To run the Connector Migration Tool, start it from the Microsoft Windows **Programs** menu or from the command line. If you start the tool from the command line, additional options that are not provided in the user interface are available.

The user interface leads you through the process of evaluating which jobs, shared containers, and stages to migrate. You select the jobs that you want to migrate, and beside each job name, the tool displays an icon that indicates whether or not the job can be fully migrated, partially migrated, or not migrated at all. To refine the list of jobs to evaluate, you can specify that only jobs that contain specific plug-in and operator stages be listed. The tool gives you a chance to make a backup of a job before you migrate it. You can make a backup copy of the job and then migrate the backup, or you can make a backup copy of the job and then migrate the original job. Either way, your original job is never lost. The job is migrated and

placed in the same folder as the original job, and the log file `CCMigration.log`, which records the results of the migration, is created in the current directory.

The Connector Migration Tool command line options provide the same functionality as the user interface, as well as a few additional options. Using the command line, you can perform these additional tasks:

- Specify a list of job names to be considered for migration.
- Specify a list of shared container names to be considered for migration
- Specify a list of stage type names to limit the jobs that are considered for migration.
- Run a practice migration, where the actual migration does not take place but the possible results of the migration are placed in the log file. You can review the results and then refine the migration as necessary before you run the actual migration.
- Produce a report of jobs and their stages and stage types

Note:

- The Connector Migration Tool does not read environment variables at the operating system level. Environment variables are read only if they are defined within InfoSphere DataStage at the Project level or at the Job level. Project level environment variables are read first, then overwritten by Job environment variables. Environment variables with blank default values are ignored by the Connector Migration Tool. The default values of the environment variables are migrated, but the run-time values are not migrated.
- Throughout this documentation, the term "job" refers to parallel shared containers and server shared containers, as well as IBM® InfoSphere® DataStage® jobs.

Using the user interface to migrate jobs

Use the Connector Migration Tool to view which jobs and stages are eligible for migration and then migrate them to use connectors rather than plug-in and operator stages.

About this task

You use the same project connection details to connect to the Connector Migration Tool as you use to connect to the InfoSphere DataStage and QualityStage® Designer or InfoSphere DataStage and QualityStage Director Client. You must have sufficient user privileges to create and modify the jobs that you are migrating.

Procedure

1. Choose **Start** > **Programs** > **IBM InfoSphere Information Server** > **Connector Migration Tool**.
2. In the Log on window, complete these fields:
   a. In the **Host** field, enter the host name of the services tier. You can specify an optional port by separating it from the host name with a colon. The host name that you specify here is the same one that you specify when you start the Designer client, for example, mymachine:9080).
   b. In the **User name** field, enter your InfoSphere DataStage user name.
   c. In the **Password** field, enter your InfoSphere DataStage password.
   d. In the **Project** field, enter the name of the project. To access an InfoSphere DataStage server that is remote from the domain server, specify the project

name in full as *server*:[*port*]/*project*. As an alternative, you can press the button adjacent to the **Project** field to display a dialog box from which you can select the fully-qualified project name.

e. Click OK. An icon indicates the status of each job. A gray icon indicates that the job cannot be migrated. A gray icon with a question mark indicates that the job might be successfully migrated.

3. Display the jobs and stages to consider for migration:
   - Choose **View** > **View all jobs** to display all of the jobs in the project. This is the default view.
   - Choose **View** > **View all migratable jobs** to display all of the jobs that are in the project and that can be migrated to use connectors. Jobs that do not contain any stages that can be migrated are excluded from the job list.
   - Choose **View** > **View jobs by stage types** to open the Filter by stage type window.
4. Perform the following steps to analyze jobs:
   a. Highlight the job in the job list.
   b. Expand the job in the job list to view the stages in the job.
   c. Select one or more jobs, and click **Analyze**.

   After analysis, the color of the job, stage, or property icon indicates whether or not it can be migrated. A green icon indicates that the job, stage, or property can be migrated. An red icon indicates that the job or stage cannot be migrated. An orange icon indicates that a job or stage can be partially migrated and that a property in a stage has no equivalent in a connector. A gray icon indicates that the job or stage is not eligible for migration.

   Note: The Connector Migration Tool displays internal property names, rather than the names that the stages display. To view a table that contains the internal name and the corresponding display name for each property, from the IBM InfoSphere DataStage and QualityStage Designer client, open the Stage Types folder in the repository tree. Double-click the stage icon, and then click the **Properties** tab to view the stage properties.
5. Click **Preferences** and choose how to migrate the job:
   - Choose **Clone and migrate cloned job** to make a copy of the job and then migrate the copy. The original job remains intact.
   - Choose **Back up job and migrate original job** to make a copy of the job and then migrate the original job.
   - Choose **Migrate original job** to migrate the job without making a backup.
6. Select the jobs and stages to migrate, and then click **Migrate**.

   The jobs and stages are migrated and are placed in the same folder as the original job. If logging is enabled, a log file that contains a report of the migration task is created. After a job is successfully migrated, a green checkmark displays beside the job name in the Jobs list to indicate that the job has been migrated.

Using the command line to migrate jobs

Run the Connector Migration Tool from the command line to use additional options that are not available in the user interface.

About this task

To run the Connector Migration Tool from the command line, you specify the command **CCMigration**, followed by a series of required and optional parameters. If the Connector Migration Tool is started from the command line, its user interface will be displayed if none of the options **-C**, **-M** or **-B** are specified. If any one of these options is specified, then the migration will proceed without any further interaction with the user. The command line options described below can therefore be used whether or not the user interface is displayed.

After a job is successfully migrated, a green checkmark displays beside the job name in the Jobs list to indicate that the job has been migrated.

Procedure

1. From the IBM InfoSphere DataStage client command line, go to the `<InformationServer>\Clients\CCMigrationTool` directory.
2. Enter the command **CCMigration**, followed by the following required parameters:
   - **-h** *host:port*, where *host:port* is the host name and port of the InfoSphere DataStage server. If you do not specify a port, the *port* is 9080 by default.
   - **-u** *user name*, where *user name* is the name of the InfoSphere DataStage user.
   - **-p** *password*, where *password* is the password of the InfoSphere DataStage user
   - **-P** *project*, where *project* is the name of the project to connect to. To specify an InfoSphere DataStage server that is remote from the domain server, specify the fully qualified project name by using the format *server*:[*port*]/*project*.
   - One of the following:
     – **-M** If you specify this parameter, the original jobs are migrated, and backup jobs are not created.
     – **-B** *job name extension*, where *job name extension* is a set of alphanumeric characters and underscores. If you specify this parameter, the Connector Migration Tool creates backup jobs, names the backup jobs *source job name+job name extension*, and then migrates the original jobs. The backup jobs are saved in the same location in the repository as the source jobs.
     – **- C** *job name extension*, where *job name extension* is a set of alphanumeric characters and underscores. If you specify this parameter, the Connector Migration Tool clones the source jobs, names the cloned jobs *source job name+job name extension*, and then migrates the cloned jobs. The cloned jobs are saved in the same location in the repository as the source jobs.

   If you specify one of these options, the migration proceeds without requiring any additional user input. If you do not specify **-M**, **-B**, or **- C**, the user interface is displayed so that you can make additional choices for how to migrate the jobs.
3. Optional: Enter any of the following optional parameters:
   - **-L** *log file*, where *log file* is the file name and path for the log file that records the results of the migration.
   - **-S** *stage types*, where *stage types* is a comma-separated list of stage types. By default, the Connector Migration Tool migrates all stage types. Use this parameter to migrate only jobs that contain the specified stage types. If you specify both the **-S** and **-J** parameters, only the specified stage types within the specified jobs are migrated. If you specify the **-S** parameter and do not specify the **-C**, **-M** or **-B** parameter, only jobs that contain the specified stage

types appear in the job list that is displayed in the user interface. Limiting the jobs that are displayed can significantly reduce the startup time of the Connector Migration Tool.

- **-J** *job names*, where *job names* is a comma-separated list of jobs. By default, the Connector Migration Tool migrates all eligible jobs in the project. Use this parameter to migrate only specific jobs. If you specify the **-J** parameter and do not specify the **-C**, **-M** or **-B** parameter, only the specified jobs appear in the job list that is displayed in the user interface. Limiting the jobs that are displayed can significantly reduce the startup time of the Connector Migration Tool.
- **-c** *shared container names*, where *shared container names* is a comma-separated list of shared containers. By default, the Connector Migration Tool migrates all eligible shared containers in the project. Use this parameter to migrate only specific shared containers. If you specify the **-c** parameter and do not specify the **-C**, **-M**, or **-B** parameter, only the specified shared containers appear in the job list that displays in the user interface. Limiting the shared containers that display might significantly reduce the startup time of the Connector Migration Tool.
- **-R** If you specify this parameter, the Connector Migration Tool reports the details of the migration that would occur if the specified jobs were migrated, but does not perform an actual migration. The details are reported in the log file that is specified by using the **-L** parameter.
- **-a** *auth file*, where *auth file* is the file name that records the user name and password.
- **-A** If you specify this parameter, the Connector Migration Tool adds an annotation to the job design. The annotation describes the stages that were migrated, the job from which the stages were migrated, and the date of the migration.
- **-d***job dump file*, where *job dump file* is the file name and path for a file where a list of jobs, shared containers, and stages is written. Using a job dump file is helpful when you want to determine which jobs are suitable for migration. You can use the **-d** parameter with the **-J**, **-c**, and **-S** parameters to list particular jobs, shared containers, and stage types, respectively.
- **-V** If you specify this parameter, the Connector Migration Tool specifies the target connector variant for migrated stages. The format of the list is a comma-separated list containing *{StageTypeName=Variant}*.
- **-v** If you specify this parameter with the **-d** command, the values of stage properties will be included in the report. If omitted, the report only contains stage names and types, but not the stage properties. This option is useful to identify jobs that have stages with certain property values. If this option is specified, then **–S** is ignored.
- **-T** If you specify this parameter, the Connector Migration Tool enables the variant migration mode. All connector stages found in jobs and containers whose stage type matches those listed by the **–V** command are modified.
- **-U** If you specify this parameter, the Connector Migration Tool enables the property upgrade migration mode. All connector stages found in jobs and containers whose properties match the conditions specified in the `StageUpgrade.xml` file are upgraded.
- **-b** *stage type*, where *stage type* is the built-in stage type to be migrated. This parameter is supported only on the command line, not on the user interface. Currently, only UniData 6 stages are supported. To migrate UniData 6 stages to UniData stages, specify `-b CUDT6Stage`.

Example

The following command starts the Connector Migration Tool, connects to the project billsproject on the server dsserver as user billg, and migrates the jobs db2write and db2upsert:

```
CCMigration -h dsserver:9080 -u billg -p padd0ck
-P billsproject -J db2write,db2upsert -M
```

Deprecated stages

Connectors, which offer better functionality and performance, replace some stages, which were deprecated and removed from the palette. However, you can still use the deprecated stages in jobs and add them back to the palette.

The following stage types were removed from palette for the parallel job canvas:

- DB2Z
- DB2® UDB API
- DB2 UDB Load
- DRS
- Dynamic RDBMS
- Java Client
- Java Transformer
- Netezza Enterprise
- ODBC Enterprise
- Oracle 7 Load
- Oracle OCI Load
- Oracle Enterprise
- Teradata API
- Teradata Enterprise
- Teradata Load
- Teradata Multiload
- WebSphere MQ

The following stage type was removed from the palette for the server job canvas:

- Dynamic RDBMS

When you create new jobs, consider using connectors instead of the deprecated stages. The following table describes the connector to use in place of the deprecated stages:

Table 2. Stages and corresponding connectors

Deprecated stage	Connector stage
DB2Z DB2 UDB API DB2 UDB Enterprise DB2 UDB Load	DB2 Connector
DRS	DRS Connector
Dynamic RDBMS	DB2 Connector Oracle Connector ODBC Connector

Table 2. Stages and corresponding connectors (continued)

Deprecated stage	Connector stage
Java Client Java Transformer	Java Integration stage
Netezza Enterprise	Netezza Connector
ODBC Enterprise	ODBC Connector
Oracle 7 Load Oracle OCI Load Oracle Enterprise	Oracle Connector
Teradata API Teradata Enterprise Teradata Load Teradata Multiload	Teradata Connector
WebSphere MQ	WebSphere MQ Connector

To use any of the deprecated stage types in new jobs, drag the stage type from the repository tree to the canvas or to the palette. From the repository tree, expand **Stage Types**. Under **Stage Types**, expand **Parallel** or **Server**depending on the stage that you want to use. Drag the stage type to the job canvas or to the palette.

Chapter 2. Configuring access to Oracle database

To configure access to an Oracle database, you must install database client libraries and include the path to these installed libraries in the library path environment variable. For more information, see the topic about setting environment variables.

Procedure

1. Install database client libraries.
2. Configure access to Oracle database.

Configuring access to Oracle databases

You can configure access to an Oracle database from the Oracle client system by setting environment variables and by updating Oracle configuration files such as `tnsnames.ora` and `sqlnet.ora`. For more information, see the Oracle product documentation.

Before you begin

- Install client libraries.
- Ensure that your system meets the system requirements and that you have a supported version of the Oracle client and Oracle server. For system requirement information, see http://www.ibm.com/software/data/infosphere/info-server/overview/.
- Ensure that the Oracle client can access the Oracle database. To test the connectivity between the Oracle client and Oracle database server, you can use the Oracle SQL*Plus utility.

About this task

You can use the `dsenv` script to update the environment variables that are used to configure access to Oracle databases. If you use the script, you must restart the server engine and the ASB Agent after you update the environment variables.

Procedure

1. Set either the **ORACLE_HOME** or the **TNS_ADMIN** environment variable so that the Oracle connector is able to access the Oracle configuration file, `tnsnames.ora`.
   - If the **ORACLE_HOME** environment variable is specified, then the `tnsnames.ora` file must be in the *$ORACLE_HOME*`/network/admin` directory.
   - If the **TNS_ADMIN** environment variable is specified, then the `tnsnames.ora` file must be in the *$TNS_ADMIN* directory.
   - If both environment variables are specified, then the **TNS_ADMIN** environment variable takes precedence.
   - Setting these environment variables is not mandatory. However, if one or both environment variables are not specified, then you cannot select a connect descriptor name to define the connection to the Oracle database. Instead, when you define the connection, you must provide the complete connect descriptor definition or specify an Oracle Easy Connect string.

   Note: If you use the Oracle Basic Instant Client or the Basic Lite Instant Client, the `tnsnames.ora` file is not automatically created for you. You must manually

create the file and save it to a directory. Then specify the location of the file in the **TNS_ADMIN** environment variable. For information about creating the `tnsnames.ora` file manually, see the Oracle documentation.

2. Optional: Set the library path environment variable to include the directory where the Oracle client libraries are located. The default location for client libraries are as follows:
   - On Windows, `C:\app\`*`username`*`\product\11.2.0\client_1\BIN`, where *username* represents a local operating system user name. If the complete Oracle database product is installed on the InfoSphere Information Server engine computer instead of just the Oracle client product, then the path would be `C:\app\`*`username`*`\product\11.2.0\dbhome_1\BIN`.
   - On Linux or UNIX, `u01/app/oracle/product/11.2.0/client_1`
3. Set the **NLS_LANG** environment variable to a value that is compatible with the NLS map name that is specified for the job. The default value for the **NLS_LANG** environment variable is `AMERICAN_AMERICA.US7ASCII`.

   The Oracle client assumes that the data that is exchanged with the stage is encoded according to the **NLS_LANG** setting. However, the data might be encoded according to the NLS map name setting. If the **NLS_LANG** setting and the NLS map name setting are not compatible, data might be corrupted, and invalid values might be stored in the database or retrieved from the database. Ensure that you synchronize the **NLS_LANG** environment variable and NLS map name values that are used for the job.

   On Microsoft Windows installations, if the **NLS_LANG** environment variable is not set, the Oracle client uses the value from the Windows registry.

Testing database connections by using the ISA Lite tool

After you establish connection to the databases, test the database connection by running the IBM Support Assistant (ISA) Lite for InfoSphere Information Server tool.

For more information about the ISA Lite tool, see the topic about installation verification and troubleshooting.

Setting the library path environment variable

To apply an environment variable to all jobs in a project, define the environment variable in the InfoSphere DataStage and QualityStage Administrator. The values that are specified for the library path and path environment variables at the project or job level are appended to the existing system values for these variables.

About this task

For example, suppose that directory `/opt/branded_odbc/lib` is specified as the value for the library path environment variable at the project level. Directory `/opt/IBM/InformationServer/Server/branded_odbc/lib`, which contains the same libraries but in a different location is already in the library path that is defined at the operating system level or the `dsenv` script. In this case, the libraries from directory `/opt/IBM/InformationServer/Server/branded_odbc/lib` are loaded when the job runs because this directory appears before directory `/opt/branded_odbc/lib` in the values that are defined for the library path environment variable.

The name of the library path environment variable depends on your operating system.

Operating system	Library path environment variable
Microsoft Windows	**PATH**
HP-UX	**SHLIB_PATH**
IBM AIX®	**LIBPATH**
Other supported Linux and UNIX operating systems, and HP-IA	**LD_LIBRARY_PATH**

On Linux or UNIX operating systems, the environment variables can be specified in the `dsenv` script. InfoSphere Information Server installations on Windows operating system do not include the `dsenv` script.

Setting the library path environment variable in the dsenv file

On Linux or UNIX operating systems, you can specify the library path environment variables in the `dsenv` script. When environment variables are specified in the `dsenv` script, they apply to all InfoSphere DataStage projects that run under the InfoSphere Information Server engine.

Before you begin

Install the client libraries.

Procedure

1. Log in as the DataStage administrator user (`dsadm` if you installed with the default name).
2. Back up the *IS_install_path*`/Server/DSEngine/dsenv` script. *IS_install_path* is the InfoSphere Information Server installation directory (`/opt/IBM/InformationServer` if you installed with the default path).
3. Open the `dsenv` script.
4. Add the path to the directory that contains the client libraries to the library path environment variable.
5. Set up your environment with the updated `dsenv` file.

   ```
   . ./dsenv
   ```
6. Restart the InfoSphere Information Server engine by entering the following commands:

   ```
   bin/uv –admin –stop
   bin/uv –admin –start
   ```
7. Assume root user privileges, directly with the **su** command or through the **sudo** command if the DataStage administrator user is part of the sudoers list.

   ```
   sudo su - root
   ```
8. Change to the ASB Agent home directory by entering the following commands:

   ```
   cd Install_directory/ASBNode/bin
   ```
9. Restart the ASB Agent processes by entering the following commands:

   ```
   ./NodeAgents.sh stopAgent
   ./NodeAgents.sh start
   ```

Results

After you restart the ASB Agent process, the InfoSphere Information Server services take approximately a minute to register the event.

Setting the library path environment variable in Windows

On the Windows operating system, both the library path and **PATH** environment variables are represented by the **PATH** system environment variable. For InfoSphere Information Server engine and ASB Agent processes to detect changes in the environment variables, the changes must be made at the system level and the InfoSphere Information Server engine must be restarted.

Before you begin

Install the client libraries.

Procedure

1. To edit the **PATH** system environment variable, click **Environment Variable** in **Advance System Settings**, and then select **PATH**.
2. Click **Edit**, then specify the path to the directory containing the client libraries.
3. Click OK.
4. Restart the InfoSphere Information Server engine.
5. Restart the ASB Agent processes.

Chapter 3. Oracle connector

Use the Oracle connector to access Oracle database systems and perform various read, write, and load functions.

Setting required user privileges

To run a job that uses the Oracle connector, the user name that the connector uses to connect to the Oracle database must have SELECT access to a set of Oracle dictionary views. The user name must also have access to the Oracle database objects that are required for the operation that the Oracle connector is configured to complete.

Before you begin

- Configure access to Oracle databases.

About this task

The database objects that the user name must have access to and the type of access that is required depend on the operation that the connector is configured to complete. For example, if the connector is configured to insert rows into the TABLE1 table, the user name that the connector uses to connect to the Oracle database must have INSERT access to the TABLE1 table. To grant access to a database object, use the **GRANT** command.

To complete some operations, the Oracle connector accesses Oracle dictionary views. All but one of these views are in the ALL_ or USR_ category, which users have access to by default. Therefore, the user name that the connector uses to connect to the database typically has access to those views. However, you must grant access to the DBA_EXTENTS dictionary view explicitly.

Access to the DBA_EXTENTS dictionary view is required for the rowid range partitioned read method. Rowid range is the default partitioned read method, which the connector uses if you do not select a different partitioned read method. If access to the DBA_EXTENTS dictionary view is not granted to the user name that the connector uses to connect to the database, the connector switches the partitioned read method from rowid range to rowid hash automatically.

Procedure

To grant SELECT access to a dictionary view or other database object, use one of the following methods:

- To grant SELECT access to a single database object, issue the following statement:

  ```
  GRANT SELECT ON database_object TO user_name
  ```

  where *database_object* is the name of the object and *user_name* is the user name with which the connector connects to the database.
- To use a role to grant a user SELECT access to multiple dictionary views or database objects, issue statements that are similar to the following sample statements. These sample statements show how to create a role, grant access to two dictionary views, and then assign the role to a user. To use these sample

statements, replace *role_name*, *dictionary_view*, and *user_name* with the names for your configuration, and issue one GRANT SELECT ON statement for each database object.

```
CREATE ROLE role_name
GRANT SELECT ON dictionary_view1 TO role_name
GRANT SELECT ON dictionary_view2 TO role name
GRANT role_name TO user_name
```

For example, to create the DSXE role, grant access to the DBA_EXTENTS and DUAL dictionary views, and assign the DSXE role to the USER1 user, issue the following statements:

```
CREATE ROLE DSXE
GRANT SELECT ON SYS.DBA_EXTENTS TO DSXE
GRANT SELECT ON SYS.DUAL TO DSXE
GRANT DSXE TO USER1
```

Designing jobs that use the Oracle connector

You can use the Oracle connector to develop jobs that read, write, and load data.

Before you begin

- Configure access to Oracle databases.
- Set required user privileges.
- Verify that the user name for connecting to the Oracle database has the authority and privileges to complete the actions that your job requires.

Procedure

1. Import metadata from an Oracle source.
2. Define a job that contains the Oracle Connector stage.
3. Define a connection to an Oracle database.
4. To set up the Oracle Connector stage to read data from an Oracle database, complete the following steps:
   a. Set up column definitions.
   b. Configure the Oracle connector as a source of data.
   c. Optional: Read partitioned data.
5. To set up the Oracle Connector stage to write data to an Oracle database, complete the following steps:
   a. Set up column definitions.
   b. Configure the Oracle connector as a target of data.
   c. Optional: Create a reject link to manage rejected data.
   d. Optional: Configure the bulk loading of data.
   e. Optional: Write partitioned data.
6. To set up the Oracle Connector stage to look up data in an Oracle database, complete the following steps:
   a. Set up column definitions.
   b. Configure the Oracle connector as a source of data.
   c. Configure normal lookup operations or configure sparse lookup operations.
7. Compile and run the job.

Importing Oracle metadata

Before you use the Oracle connector to read, write, or look up data, you can use InfoSphere Metadata Asset Manager to import the metadata that represents tables and views in a Oracle database. The imported metadata is then saved in the metadata repository.

Before you begin

- Configure access to Oracle databases.

  Configure access to Oracle databases. For more information about accessing Oracle databases, see "Configuring access to Oracle databases" on page 9 in the InfoSphere Information Server Configuration Guide.
- Ensure that the Oracle connector has access to the Oracle "Dictionary views" on page 74 that are required to import metadata.
- Ensure that the Oracle user that is associated with the data connection that you plan to use in InfoSphere Metadata Asset Manager has access to the assets that you want to import. Only assets that the user has access to are shown in InfoSphere Metadata Asset Manager and available to import.

About this task

By using the Oracle connector, you can import metadata about the following types of assets:

- The host computer that contains the Oracle database.
- The database.
- Database schemas.
- Database tables and views. All imported tables are stored in the metadata repository as database tables.
- Column definitions for a table or view.

Procedure

Import metadata by using InfoSphere Metadata Asset Manager. For more information about importing metadata by using InfoSphere Metadata Asset Manager, see the online product documentation in IBM Knowledge Center or the IBM InfoSphere Information Server Guide to Managing Common Metadata.

Defining a job that includes the Oracle connector

To read, write, or look up data in an Oracle database, you can create a job that includes the Oracle connector. Then, you add any additional stages that are required and create the necessary links.

Procedure

1. In the InfoSphere DataStage and QualityStage Designer client, select **File** > **New** from the menu.
2. In the **New** window, select the **Parallel Job** or **Server Job** icon, and then click **OK**.
3. Add the Oracle connector to the job:
   a. In the palette, select the **Database** category.
   b. Locate Oracle in the list of available databases, and click the down arrow to display the available stages.
   c. Drag the Oracle Connector stage to the canvas.

d. Optional: Rename the Oracle Connector stage. Choose a name that indicates the role of the stage in the job.

4. Create the necessary links and add additional stages for the job:
   - For a job that reads Oracle data, create the next stage in the job, and then create an output link from the Oracle connector to the next stage.
   - For a job that writes Oracle data, create one or more input links from the previous stage in the job to the Oracle connector. If you use multiple input links, you can specify the link for the input data and the order for the record processing. If you want to manage rejected records, add a stage to hold the rejected records, and then add a reject link from the Oracle connector to that stage.
   - For a job that looks up Oracle data, create a job that includes a Lookup stage, and then create a reference link from the Oracle connector to the Lookup stage.
5. Save the job.

Defining a connection to an Oracle database

To access data in an Oracle database, you must define a connection that specifies the server, user name, and password.

Before you begin

- Verify that the user name that connects to the Oracle database has the authority and privileges to perform the actions of the job.
- Depending on how you choose to define the connection to the Oracle database, confirm that these Oracle environment variables are set correctly: **TNS_ADMIN**, **ORACLE_HOME**, **ORACLE_SID**, **TWO_TASK**, and **LOCAL**.

Procedure

1. In the job design canvas, double-click the connector stage icon to open the connector properties.
2. On the Properties page, in the **Server** field, complete one of the following steps:
   - Click **Select** to display a list of Oracle services, and then select the Oracle service to connect to. If the list is empty, the connector cannot locate the Oracle `tnsnames.ora` file. The connector tries to locate the file by checking the **TNS_ADMIN** and **ORACLE_HOME** environment variables.
   - Enter the complete content of the connect descriptor in the format that is used in the Oracle `tnsnames.ora` file.
   - Use the following syntax to enter an Oracle Easy Connect string: *host*[:*port*][/*service_name*]
   - To connect to the default local Oracle service, leave the property blank. The **ORACLE_SID** environment variable defines the default local service. The **TWO_TASK** environment variable on Linux or UNIX and the **LOCAL** environment variable on Microsoft Windows define the default remote service. Selecting an Oracle service is preferable to using the **TWO_TASK** or **LOCAL** environment variables.
3. In the **Username** and **Password** fields, specify the user name and password to use to authenticate with the Oracle service. By default, the connector is configured for Oracle database authentication. This form of authentication requires that the name and password that you specify match the credentials that are configured for the user in the Oracle database.
4. Optional: Select the **Use external authentication** check box. This form of authentication requires that the user be registered in Oracle and identified as a

user who is authenticated by the operating system. For information about enabling external authentication to your Oracle server, see the Oracle documentation.

5. Optional: Set the **CC_ORACLECONNECTOR_DEFAULT_CONNECTION_VERSION** environment variable by using the Administrator client for the InfoSphere DataStage project. For example, CC_ORACLECONNECTOR_DEFAULT_CONNECTION_VERSION=11g. The value that is set for the environment variable, for example 11g, is used to populate the **Oracle client version** property when the connector stage is opened for the first time.

Reading data from an Oracle database

You can configure the Oracle connector to connect to an Oracle database and read data from it.

Before you begin

- Import metadata from an Oracle source.
- Define a job that contains the Oracle Connector stage.
- Define a connection to an Oracle database.

About this task

The following figure shows an example of using the Oracle connector to read data. In this example, the Oracle connector reads data from an Oracle database and passes the rows to a Transformer stage, which transforms the data and then sends the data to the ODBC connector. When you configure the Oracle connector to read data, you create only one output link, which in this example transfers rows to the Transformer stage.

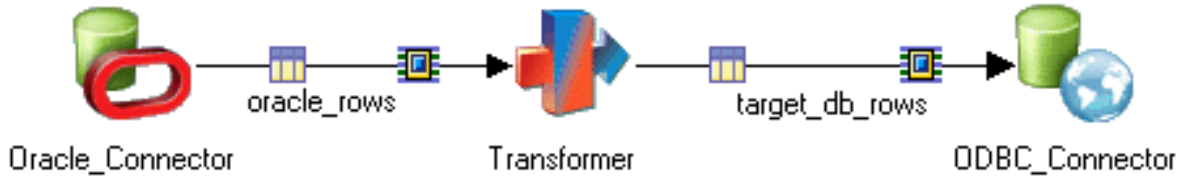


Figure 1. Example of reading data from an Oracle database

Setting up column definitions on a link

Column definitions, which you set on a link, specify the format of the data records that the connector reads from a database or writes to a database.

Procedure

1. From the job design canvas, double-click the connector icon.
2. Use one of the following methods to set up the column definitions:
   - Drag a table definition from the repository view to the link on the job canvas. Then, use the arrow buttons to move the columns between the **Available columns** and **Selected columns** lists.
   - On the **Columns** page, click **Load** and select a table definition from the metadata repository. Then, to choose which columns from the table definition apply to the link, move the columns from the **Available columns** list to the **Selected columns** list.

3. Configure the properties for the columns:
   a. Right-click within the columns grid, and select **Properties** from the menu.
   b. Select the properties to display, specify the order in which to display them, and then click **OK**.
4. Optional: Modify the column definitions. You can change the column names, data types, and other attributes. In addition, you can add, insert, or remove columns.
5. Optional: Save the new table definition in the metadata repository:
   a. On the **Columns** page, click **Save**, and then click **OK** to display the repository view.
   b. Navigate to an existing folder, or create a new folder in which to save the table definition.
   c. Select the folder, and then click **Save**.

Configuring the Oracle connector as a source for reading data

To configure the connector to read rows in an Oracle table or view, you must specify the source table or view or define a complete SELECT statement or PL/SQL block.

About this task

If you specify a SELECT statement, the connector runs the statement only once and sends all of the rows that are returned for that statement to the output link.

If you specify a PL/SQL block, the connector runs the PL/SQL block only once and returns the output bind variables that are specified in the block. A single record is sent to the output link. A PL/SQL block is useful for running a stored procedure that takes no input parameters but that returns values through one or more output parameters.

Procedure

1. From the job design canvas, double-click the Oracle Connector stage.
2. Select the output link to edit. When you edit the output link, you set up the Oracle Connector stage to be the source.
3. Set **Read mode** to **Select** or **PL/SQL**.
4. If you set **Read mode** to **Select**, use one of these methods to specify the source of the data:
   - Set **Generate SQL at runtime** to **Yes**, and then enter the name of the table or view in the **Table name** property. Use the syntax *schema_name.table_name*, where *schema_name* is the owner of the table. If you do not specify *schema_name*, the connector uses the schema that belongs to the user who is currently connected.
   - Set **Generate SQL at runtime** to **No**, and then specify the SELECT statement in the **Select statement** property.
   - Set **Generate SQL at runtime** to **No**, and then enter the fully qualified file name of the file that contains the SQL statement in the **Select statement** property. If you enter a file name, you must also set **Read select statement from file** to **Yes**.
   - Click the **Select statement** property, and then next to the property, click **Build** to start the SQL Builder. To construct the SQL statement, drag table and column definitions that are stored in the repository and choose options for configuring clauses in the SQL statement.

5. If you set **Read mode** to **PL/SQL**, use one of these methods to specify the source of the data:
   - Enter the PL/SQL block manually in the **PL/SQL block** property.
   - Enter the fully qualified file name of the file that contains the PL/SQL block in the **PL/SQL block** property. If you enter a file name, you must also set **Read PL/SQL block from file** to **Yes**.

   The PL/SQL block that you specify must begin with the keyword DECLARE or BEGIN and must end with the keyword END, and you must enter a semicolon after the END keyword.
6. Click **OK**, and then save the job.

Reading partitioned data

In a job that uses multiple nodes, each node that is specified for the stage reads a distinct subset of data from the source table.

Before you begin

To use the default rowid range partitioned read method, the user whose credentials are used to connect to the Oracle database must have SELECT access to the DBA_EXTENTS dictionary view.

About this task

If the connector is configured to run in parallel mode to read data, the connector runs a slightly modified SELECT statement on each node. The combined set of rows from all of the queries is the same set of rows that would be returned if the unmodified user-defined SELECT statement were run once on one node.

Procedure

1. On the job design canvas, double-click the Oracle Connector stage, and then click the **Stage** tab.
2. On the Advanced page, set **Execution mode** to **Parallel**, and then click the **Output** tab.
3. Set **Enable partitioned reads** to **Yes**.
4. Set **Read mode** to **Select**, and then define the SELECT statement that the connector uses at run time:
   - Set **Generate SQL at runtime** to **Yes**, and then enter the name of the table in the **Table name** field. Use the syntax *schema_name.table_name*, where *schema_name* is the owner of the table. If you do not specify *schema_name*, the connector uses the schema that belongs to the currently connected user. The connector automatically generates and runs the SELECT statement.

     To read data from a particular partition of a partitioned table, set the **Table scope** property to **Single partition**, and specify the name of the partition in the **Partition name** property. The connector then automatically adds a PARTITION(*partition_name*) clause to the SELECT statement that is generated. To read data from a particular subpartition of the composite partitioned table, set the **Table scope** property to **Single subpartition** and specify the name of the subpartition in the **Subpartition name** property. The connector then automatically adds a SUBPARTITION(*subpartition_name*) clause to the generated SELECT statement.
   - Set **Generate SQL at runtime** to **No**, and then specify the SELECT statement in the **Select statement** property. You can enter the SQL statement or enter

the fully qualified file name of the file that contains the SQL statement. If you enter a file name, you must also set **Read select statement from file** to **Yes**.

5. Set the **Partitioned reads method** property to the partitioning method that you want to use. The default partitioning method is **Rowid range**.
6. Specify the input values that the partitioned read method uses:
   a. In the **Table name for partitioned reads** property, specify the name of the table that the partitioned read method uses to define the subsets of data that each node reads from the source table.

      If you do not specify a table name, the connector uses the value of the **Generate SQL at runtime** property to determine the table name. If **Generate SQL at runtime** is set to **Yes**, the connector uses the table name that is specified in the **Table name** property. If **Generate SQL at runtime** is set to **No**, the connector looks at the SELECT statement that is specified in the **Select statement** property and uses the first table name that is specified in the FROM clause.
   b. If you choose the **Rowid range** or the **Minimum and maximum range** partitioned read method, in the **Partition or subpartition name for partitioned reads** property, specify the name of the partition or subpartition that the partitioned read methods uses.

      Note: If you do not specify a value for the **Partition or subpartition name for partitioned reads** property, the connector uses the entire table as input for the partitioned read method. When the connector is configured to read data from a single partition or subpartition, you typically specify the name of the partition or subpartition in the **Partition or subpartition name for partitioned reads** property. Then the connector analyzes only the data that belongs to that partition or subpartition. This process typically results in a more even distribution of data and a more efficient use of nodes.
   c. If you choose the **Modulus** or the **Minimum and maximum range** partitioned read method, in the **Column name for partitioned reads**, enter the name of the column from the source table to use for the method. The column must be an existing column in the table, must be of NUMBER(*p*) data type, where *p* is the number precision, and must have a scale of zero.
7. Click **OK**, and then save the job.

Writing data to an Oracle database

You can configure the Oracle connector to connect to an Oracle database and write data to it.

Before you begin

- Import metadata from an Oracle source.
- Define a job that contains the Oracle Connector stage.
- Define a connection to an Oracle database.

About this task

The following figure shows an example of using the Oracle connector to write data. In this example, the ODBC connector reads data from a database and transfers that data to a Transformer stage, which transforms the data and transfers the data to the Oracle connector. The Oracle connector writes the data to an Oracle database. Because this job includes an optional reject link, the Oracle connector transfers rejected records to a stage that stores them. In this example, a Sequential

File stage stores the rejected records.

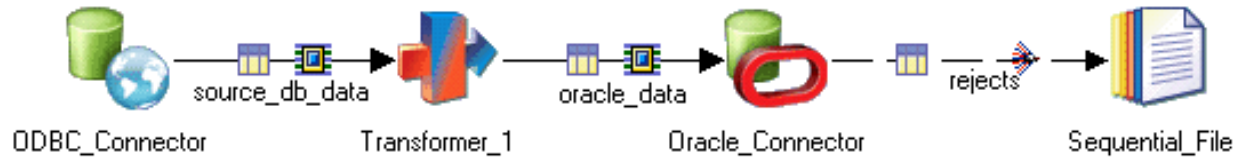


Figure 2. Example of writing data and using a reject link.

Setting up column definitions on a link

Column definitions, which you set on a link, specify the format of the data records that the connector reads from a database or writes to a database.

Procedure

1. From the job design canvas, double-click the connector icon.
2. Use one of the following methods to set up the column definitions:
   - Drag a table definition from the repository view to the link on the job canvas. Then, use the arrow buttons to move the columns between the **Available columns** and **Selected columns** lists.
   - On the **Columns** page, click **Load** and select a table definition from the metadata repository. Then, to choose which columns from the table definition apply to the link, move the columns from the **Available columns** list to the **Selected columns** list.
3. Configure the properties for the columns:
   a. Right-click within the columns grid, and select **Properties** from the menu.
   b. Select the properties to display, specify the order in which to display them, and then click **OK**.
4. Optional: Modify the column definitions. You can change the column names, data types, and other attributes. In addition, you can add, insert, or remove columns.
5. Optional: Save the new table definition in the metadata repository:
   a. On the **Columns** page, click **Save**, and then click **OK** to display the repository view.
   b. Navigate to an existing folder, or create a new folder in which to save the table definition.
   c. Select the folder, and then click **Save**.

Configuring the Oracle connector as a target

To configure the connector to write rows to an Oracle table or writable view, you must specify the target table or view or define the SQL statements or PL/SQL block.

Procedure

1. On the job design canvas, double-click the Oracle Connector stage.
2. Select the input link to edit.
3. Specify how the Oracle connector writes data to an Oracle table or writable view. The following table shows the ways that you can configure the connector to write data.

Table 3. Methods for writing data to an Oracle table or writable view

Method	Procedure
Automatically generate the SQL at run time	1. Set **Generate SQL at runtime** to **Yes**. 2. Set **Write mode** to **Insert**, **Insert new rows only**, **Update**, **Delete**, **Insert then update**, **Update then insert**, or **Delete then insert**. 3. Enter the name of the target table in the **Table name** field.
Enter the SQL manually	1. Set **Generate SQL at runtime** to **No**. 2. Set **Write mode** to **Insert**, **Insert new rows only**, **Update**, **Delete**, **Insert then update**, **Update then insert**, or **Delete then insert**. 3. Enter SQL statements in the fields that correspond to the write mode that you selected. Alternatively, click **Tools** beside each field to view options for generating and validating SQL statements.
Read the SQL statement from a file	1. Set **Generate SQL at runtime** to **No**. 2. Enter the fully qualified name of the file that contains the SQL statement in the **Insert statement**, **Update statement**, **Delete statement**, or **PL/SQL block** field. 3. Set **Read insert statement from file**, **Read update statement from file**, **Read delete statement from file**, or **Read PL/SQL block from file** to **Yes**.
Specify a PL/SQL block	1. Set the **Write mode** to **PL/SQL**. 2. Enter the PL/SQL block in the **PL/SQL block** field. **Note:** The PL/SQL block must begin with the keyword DECLARE or BEGIN and end with the keyword END. You must include a semicolon character after the END keyword.
Bulk load the data	1. Set **Write mode** to **Bulk load**. 2. Enter the name of the table in the **Table name** field. Use the syntax *schema_name.table_name*, where *schema_name* is the owner of the table. If you do not specify *schema_name*, the connector uses the schema that belongs to the currently connected user.

4. Click **OK**, and then save the job.

Rejecting records that contain errors

When the Oracle connector includes a reject link, records that meet specified criteria are automatically routed to the target stage on the reject link. Processing continues for the remaining records.

About this task

When you configure a reject link, you select one or more conditions that control when to reject a record and send it to the target stage that receives the rejected records. You can also choose to include the Oracle error code and error message that is generated when a record fails. If you do not define a reject link or if you define a reject link but a failed record does not match any of the specified reject conditions, the connector reports an error and stops the job.

After you run the job, you can evaluate the rejected records and adjust the job and the data accordingly.

Procedure

1. On the job design canvas, add and configure a target stage to receive the rejected records.
2. Right-click the Oracle connector and drag to create a link from the Oracle connector to the target stage.
3. If the link is the first link for the Oracle connector, right-click the link and choose **Convert to reject**. If the Oracle connector already has an input link, the new link automatically displays as a reject link.
4. Double-click the connector to open the stage editor.
5. On the Output page, select the link to the target stage for rejected records from the **Output name** list.
6. Click the **Reject** tab.
7. From the **Reject rows based on selected conditions** list, select one or more conditions to use to reject records.
8. Use one of the methods in the following table to specify when to stop a job because of too many rejected rows.

Method	Procedure
Stop a job based on the percentage of rows that fail.	1. From the **Abort when** list, select **Percent**. 2. In the **Abort after (%)** field, enter the percentage of rejected rows that will cause the job to stop. 3. In the **Start count after (rows)** field, specify the number of input rows to process before calculating the percentage of rejected rows.
Stop a job based on the number of rows that fail.	1. From the **Abort when** list, select **Rows**. 2. In the **Abort after (rows)** field, specify the maximum number of rejected rows to allow before the job stops.

9. Optional: From the **Add to reject row** list, select **ERRORCODE**, **ERRORMESSAGE**, or both. When a record fails, the rejected record includes the Oracle error code and the corresponding message that describes the failure. For a complete list of the Oracle error codes and messages, see the Oracle documentation.
10. Click **OK**, and then save the job.

Configuring bulk loading of data

When you use the Oracle connector to bulk load data to an Oracle database, you can enable or disable constraints and triggers. You can also configure the date cache, manage indexes, set options for bulk record loading, and enable manual mode.

Before you begin

Choose the bulk load write method and specify the table to write to.

About this task

In the Oracle Connector stage, you can set properties that apply only when you use the connector to bulk load data. The values for these properties can affect the load performance and prevent issues that might occur during the bulk load.

For example, during a bulk load, enforcing table constraints and triggers might result in additional I/O overhead and prevent a successful load operation. To avoid these issues, disable Oracle constraints and triggers before a bulk load.

To improve load performance, you can configure the Oracle date cache.

If you do not want the stage to load data directly to the Oracle database, you can enable manual mode. When manual mode is enabled, the connector creates control and data files that can be used to load data to the database by using the Oracle SQL*Loader utility.

Procedure

1. Configure the connector to disable constraints before it bulk loads data and enable constraints after it bulk loads data:
   a. Set **Perform operations before bulk load** to **Yes**.
   b. Set **Disable constraints** to **Yes**.
   c. Set **Perform operations after bulk load** to **Yes**.
   d. Set **Enable constraints** to **Yes**.
   e. In the **Exceptions table name** field, enter the name of the exceptions table. If the exceptions table does not exist, the connector creates it. If the exceptions table already exists, the connector deletes any data that is in the table and then uses it.
   f. Set **Process exception rows** to **Yes**. When **Process exception rows** is set to **Yes**, the connector deletes from the target table the rows that fail the constraint checks. If you defined a reject link for the connector and enabled the **SQL error - constraint check** reject condition, the connector sends the deleted rows to the reject link. If **Process exception rows** is set to **No** and rows fail a constraint check, the job stops.
2. Configure the connector to disable triggers before it bulk loads data and enable triggers after it bulk loads data:
   a. Set **Perform operations before bulk load** to **Yes**.
   b. Set **Disable triggers** to **Yes**.
   c. Set **Perform operations after bulk load** to **Yes**.
   d. Set **Enable triggers** to **Yes**.
3. To control how to handle table indexes during a bulk load, set the **Index maintenance option** property.

4. To rebuild indexes after a bulk load:
   a. Set **Perform operations after bulk load** to **Yes**.
   b. Set **Rebuild indexes** to **Yes**.
   c. Optional: To enable or disable parallelism and logging to the redo log when the index is rebuilt, specify nondefault values for the **Parallel clause** and **Logging clause** properties. By default, parallel and logging clauses are not included in the ALTER INDEX statement.
   d. Optional: To stop the job if an index rebuild statement fails, set **Fail on error for index rebuilding** to **Yes**. If an index rebuild fails, the connector logs a fatal error.
5. If you plan to bulk load data into tables that contain DATE or TIMESTAMP columns, enable and configure the date cache:
   a. Set **Use Oracle date cache** to **Yes**.
   b. Optional: In the **Cache size** property, enter the maximum number of entries that the cache stores. The default is 1,000.
   c. Optional: Set **Disable cache when full** to **Yes**. When the number of entries in the cache reaches the number that is specified in the **Cache size** property and the next lookup in the cache results in a miss, the cache is disabled.
6. Set options to control bulk record loading:
   a. Set **Array size** to a value 1 - 999,999,999. The default is 2,000.
   b. Set **Buffer size** to a value 4 - 100,240, which represents the buffer size in KB. The default is 1,024.
   c. Set the **Allow concurrent load sessions** property depending on your requirement.
7. To enable manual mode:
   a. Set **Manual mode** to **Yes**.
   b. Optional: In the **Directory for data and control files** property, specify a directory to save the control and data files to.
   c. Optional: In the **Control file name** property, specify a name for the control file. If you do not specify a value for the control file name, the connector generates the name in the *servername_tablename*.ctl format, where *servername* is the value that specified for the **Server** property and *tablename* is the value specified in the **Table name** property.
   d. In the **Data file name** property, specify the name of the data file. If you do not specify a value for the data file name, the connector generates the name in the *servername_tablename*.dat format.
   e. In the **Load options** property, specify the bulk load options to include in the control file that the connector generates. The value contains parameters that are passed to the Oracle SQL*Loader utility when the utility is invoked to process the control and data files. The default value is OPTIONS(DIRECT=FALSE,PARALLEL=TRUE).

      The DIRECT=FALSE parameter tells the Oracle SQL*Loader to use the conventional path load instead of the direct path load. The PARALLEL=TRUE parameter tells the utility that the data can be loaded in parallel from multiple concurrent sessions. For more information about these options and other load options, see the Oracle product documentation.

      The word *OPTIONS* and the parentheses must be included in the value that is specified for the property. The connector saves this property value to the control file that is generated and does not check the syntax of the value.

Writing partitioned data

In a job that uses multiple nodes, records that arrive on the input link of the connector are distributed across multiple nodes. Then, the records are written in parallel from all of the nodes to the target database.

About this task

The default partition type is **Auto**, which selects the partition type based on the various settings for the stages in the job. In general, instead of using **Auto**, it is better to select a partition type based on your knowledge about the actual data and the target table that the connector writes to at run time. In particular, if the target table is range-partitioned or list-partitioned, select **Oracle connector**. When the Oracle connector partition type is selected, the connector partitions the input records so that each node writes rows to the partition that is associated with that node.

Procedure

1. On the job design canvas, double-click the Oracle Connector stage.
2. On the Input page, select the input link.
3. On the Partitioning page, select a partition type.

Looking up data in an Oracle database

You can configure the connector to complete a normal lookup or a sparse lookup on an Oracle database.

Before you begin

- Import metadata from an Oracle source.
- Define a job that contains the Oracle Connector stage.
- Define a connection to an Oracle database.

About this task

In the following figure, a Lookup stage extracts data from an Oracle database based on the input parameter values that the Lookup stage provides. Although the reference link appears to go from the Oracle connector to the Lookup stage, the link transfers data both to and from the Oracle connector. Input parameters are transferred from the input link on the Lookup stage to the reference link, and output values that the Oracle connector provides are transferred from the Oracle connector to the Lookup stage. The output values are routed to the columns on the output link of the Lookup stage according to the column mappings that are defined for the Lookup stage.

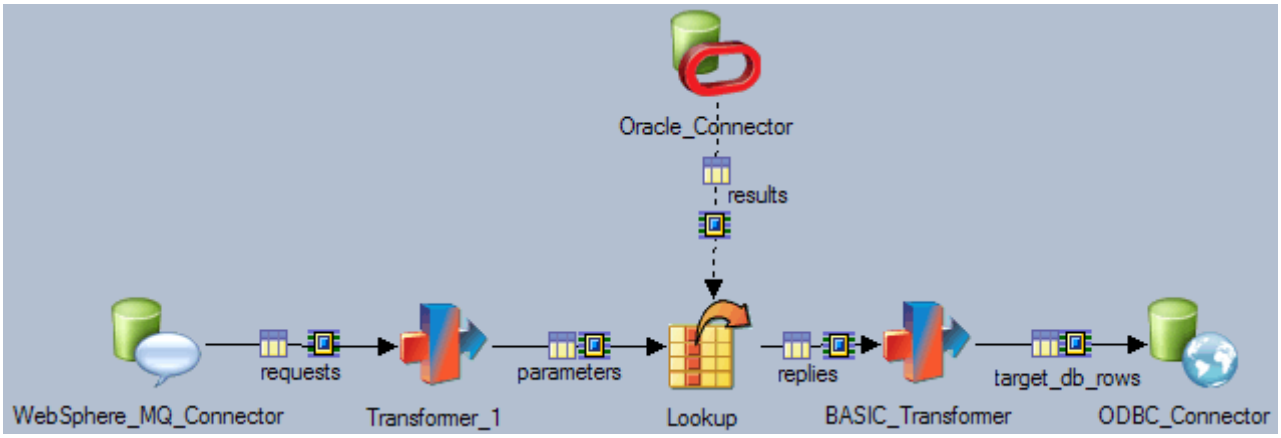


Figure 3. Example of using the Oracle connector with a Lookup stage.

Setting up column definitions on a link

Column definitions, which you set on a link, specify the format of the data records that the connector reads from a database or writes to a database.

Procedure

1. From the job design canvas, double-click the connector icon.
2. Use one of the following methods to set up the column definitions:
   - Drag a table definition from the repository view to the link on the job canvas. Then, use the arrow buttons to move the columns between the **Available columns** and **Selected columns** lists.
   - On the **Columns** page, click **Load** and select a table definition from the metadata repository. Then, to choose which columns from the table definition apply to the link, move the columns from the **Available columns** list to the **Selected columns** list.
3. Configure the properties for the columns:
   a. Right-click within the columns grid, and select **Properties** from the menu.
   b. Select the properties to display, specify the order in which to display them, and then click **OK**.
4. Optional: Modify the column definitions. You can change the column names, data types, and other attributes. In addition, you can add, insert, or remove columns.
5. Optional: Save the new table definition in the metadata repository:
   a. On the **Columns** page, click **Save**, and then click **OK** to display the repository view.
   b. Navigate to an existing folder, or create a new folder in which to save the table definition.
   c. Select the folder, and then click **Save**.

Configuring the Oracle connector as a source for looking up data

To configure the connector to look up rows in an Oracle table or view, you must specify the source table or view or define a complete SELECT statement or PL/SQL block.

About this task

If you define a PL/SQL block for a normal lookup operation, when you run the job, the connector runs the specified PL/SQL block only once and returns a single record to the Lookup stage. For each record on the input link to the Lookup stage, the Lookup stage completes a lookup operation on the single record that is returned by the connector.

If you define a PL/SQL block for a sparse lookup operation, the connector runs the specified PL/SQL block one time for each record on the input link to the Lookup stage.

Procedure

1. From the job design canvas, double-click the Oracle Connector stage.
2. Select the output link to edit. When you edit the output link, you set up the Oracle Connector stage to be the source.
3. Set **Read mode** to **Select** or **PL/SQL**.
4. If you set **Read mode** to **Select**, use one of these methods to specify the source of the data:
   - Set **Generate SQL at runtime** to **Yes**, and then enter the name of the table or view in the **Table name** property. Use the syntax *schema_name.table_name*, where *schema_name* is the owner of the table. If you do not specify *schema_name*, the connector uses the schema that belongs to the user who is currently connected.
   - Set **Generate SQL at runtime** to **No**, and then specify the SELECT statement in the **Select statement** property.
   - Set **Generate SQL at runtime** to **No**, and then enter the fully qualified file name of the file that contains the SQL statement in the **Select statement** property. If you enter a file name, you must also set **Read select statement from file** to **Yes**.
   - Click the **Select statement** property, and then next to the property, click **Build** to start the SQL Builder. To construct the SQL statement, drag table and column definitions that are stored in the repository and choose options for configuring clauses in the SQL statement.
5. If you set **Read mode** to **PL/SQL**, use one of these methods to specify the source of the data:
   - Enter the PL/SQL block manually in the **PL/SQL block** property.
   - Enter the fully qualified file name of the file that contains the PL/SQL block in the **PL/SQL block** property. If you enter a file name, you must also set **Read PL/SQL block from file** to **Yes**.

   The PL/SQL block that you specify must begin with the keyword DECLARE or BEGIN and must end with the keyword END, and you must enter a semicolon after the END keyword.
6. Click **OK**, and then save the job.

Configuring normal lookup operations

You configure the Oracle connector to perform a normal lookup on an Oracle database.

Before you begin

- To specify the format of the data records that the Oracle connector reads from an Oracle database, set up column definitions on a link.

- Configure the Oracle connector as a source for the reference data.

About this task

In a normal lookup, the connector runs the specified SELECT statement or PL/SQL block only one time; therefore, the SELECT statement or PL/SQL block cannot include any input parameters. The Lookup stage searches the result set data that is retrieved and looks for matches for the parameter sets that arrive in the form of records on the input link to the Lookup stage. A normal lookup is also known as an in-memory lookup because the lookup is performed on the cached data in memory.

Typically you use a normal lookup when the target table is small enough that all of the rows in the table can fit in memory.

Procedure

1. Add a Lookup stage to the job design canvas, and then create a reference link from the Oracle Connector stage to the Lookup stage.
2. Double-click the Oracle Connector stage.
3. From the **Lookup Type** list, select **Normal**.
4. To save the changes, click **OK**.
5. Double-click the Lookup stage.
6. To specify the key columns, drag the required columns from the input link to the reference link. The columns from the input link contain values that are used as input values for the lookup operation.
7. Map the input link and reference link columns to the output link columns and specify conditions for a lookup failure:
   a. Drag or copy the columns from the input link and reference link to your output link.
   b. To define conditions for a lookup failure, click the **Constraints** icon in the menu.
   c. In the Lookup Failure column, select a value, and then click **OK**. If you select **Reject**, you must have a reject link from the Lookup stage and a target stage in your job configuration to capture the rejected records.
   d. Click **OK**.
8. Save, compile, and run the job.

Configuring sparse lookup operations

You configure the Oracle connector to perform a sparse lookup on an Oracle database.

Before you begin

- To specify the format of the data records that the Oracle connector reads from an Oracle database, set up column definitions on a link.
- Configure the Oracle connector as a source for the reference data.

About this task

In a sparse lookup, the connector runs the specified SELECT statement or PL/SQL block one time for each parameter set that arrives in the form of a record on the input link to the Lookup stage. The specified input parameters in the statement must have corresponding columns defined on the reference link. Each input record includes a set of parameter values that are represented by key columns. The Oracle

connector sets the parameter values on the bind variables in the SELECT statement or PL/SQL block, and then the Oracle connector runs the statement or block. The result of the lookup is routed as one or more records through the reference link from the Oracle connector back to the Lookup stage and from the Lookup stage to the output link of the Lookup stage. A sparse lookup is also known as a direct lookup because the lookup is performed directly on the database.

Typically, you use a sparse lookup when the target table is too large to fit in memory. You can also use the sparse lookup method for real-time jobs.

You can use the sparse lookup method only in parallel jobs.

Procedure

1. Add a Lookup stage to the job design canvas, and then create a reference link from the Oracle Connector stage to the Lookup stage.
2. Double-click the Oracle Connector stage.
3. From the **Lookup Type** list, select **Sparse**.
4. Specify the key columns:
   a. If you set **Generate SQL** to **Yes** when you configured the connector as a source, specify the table name, and then specify the key columns on the Columns page.
   b. If you set **Generate SQL** to **No** when you configured the connector as a source, specify a value for the **Select statement** property. In the select part of the SELECT statement, list the columns to return to the Lookup stage. Ensure that this list matches the columns on the Columns page.
5. On the Properties page, specify a table name, and then specify a WHERE clause for the lookup. Key columns that follow the WHERE clause must have the word *ORCHESTRATE* and a period added to the beginning of the column name. *ORCHESTRATE* can be all uppercase or all lowercase letters, such as `ORCHESTRATE.Field001`. The following SELECT statement is an example of the correct syntax of the WHERE clause: `select Field002,Field003 from MY_TABLE where Field001 = ORCHESTRATE.Field001`. The column names that follow the word *ORCHESTRATE* must match the column names on the Columns page.
6. To save the changes, click **OK**.
7. Double-click the Lookup stage.
8. Map the input link and reference link columns to the output link columns and specify conditions for a lookup failure:
   a. Drag or copy the columns from the input link and reference link to your output link.
   b. To define conditions for a lookup failure, click the **Constraints** icon in the menu.
   c. In the Lookup Failure column, select a value, and then click **OK**. If you select **Reject**, you must have a reject link from the Lookup stage and a target stage in your job configuration to capture the rejected records.
   d. Click **OK**.
9. Save, compile, and run the job.

Generating SQL statements in the connector at design time

You can configure the connector to generate SQL statements at design time in their statement properties.

Before you begin

Create a job that includes a connector as a source or target.

About this task

You can generate the SQL statement text only for those statement properties that have the **Generate SQL statement** option in the Build list.

Note: Under some circumstances, the connector requires a connection to generate SQL statements. When a user name and password are not supplied and a connection is required, a connection is made by using the user who is running the ASB Agent service.

Procedure

1. Double-click the connector on the job canvas to open the stage editor.
2. In the navigator, click the output or input link, depending on the type of job that you create.
3. Set **Generate SQL at runtime** to No.
4. In the **Table name** property, type the name of the table for the SQL statement.
5. For jobs in target context (input links), select the type of statement you want to generate in the **Write mode** property.
6. On the **Columns** page, define the columns to use in the SQL statement.
7. Click the **Properties** tab.
8. Click the **Build** button that is associated with the statement property, and select **Generate SQL statement** from the list.

   Note: The **Generate SQL statement** option will only be available for statements which that connector supports generating at design time. In some cases a connector may only support generating the SQL at runtime during job execution.
9. Click **OK** to save the job.

Validating SQL statements in the connector at design time

After you generate or write a SQL statement, you can validate the statement during job design.

About this task

You can validate the SQL statement text only for those statement properties that have the **Validate SQL** option in the Build list.

Note: Under some circumstances, the connector requires a connection to validate SQL statements. When a user name and password are not supplied and a connection is required, a connection is made by using the user who is running the ASB Agent service.

Procedure

1. Save the job.
2. Click the **Build** button that is associated with the statement property, and select **Validate SQL**. The **Validate SQL** option is enabled only if the statement property contains a value and this option will only be available for statements which the target RDBMS supports validating.

Results

The connector validates the SQL statement by preparing the statement with the RDBMS it supports. If the SQL contains error, an error message is shown.

Troubleshooting the Oracle connector

You can use the troubleshooting and support information to isolate and resolve problems with the Oracle connector.

Oracle environment logging

The Oracle connector can log debug messages that contain information about the current Oracle environment settings. These messages are useful for diagnosing problems.

By default, debug messages are not displayed in the log file. To view debug messages in the log file, set the **CC_MSG_LEVEL** environment variable to 2.

The Oracle connector logs the following environment information:

Oracle client version and Oracle server version
The Oracle connector uses the following syntax to log the current version: *A.B.C.D.E*, where *A* is the major version, *B* is the minor version, *C* is the update number, *D* is the patch number, and *E* is the port update number. The Oracle client version is logged from the conductor node and from all processing nodes. The Oracle server version is logged only from the conductor node.

NLS session parameters
The connector logs a message that contains the name and value of each NLS session parameter. The values are logged from the conductor node and from all processing nodes.

NLS database parameters
The Oracle connector logs a message that contains the name and value of each NLS database parameter. The values are logged only from the conductor node.

NLS_LANG
The Oracle connector logs a message that contains the value of the **NLS_LANG** environment variable, as seen by the Oracle client library. This value might not match the value of the **NLS_LANG** environment variable that you specify or configure in the Microsoft Windows registry because Oracle replaces or adds to incorrect or missing parts of the value with default values for the current client environment, if necessary. The connector logs the **NLS_LANG** value from the conductor node and from all processing nodes.

Debug and trace messages

Debug and trace messages provide detailed information that you can use to troubleshoot problems.

Debug messages

The Oracle connector has only one generic debug message, which has up to four arguments. IIS-CONN-ORA-005001 has the message text CCORA DEBUG: {0}{1}{2}{3}{4}. The content of the debug message is useful for troubleshooting a job.

Trace messages

The Oracle connector has two trace messages. One specifies that a method was entered, and the other specifies that a method was exited. Both messages include the name of the class that defines the method, if applicable, and the name of the method.

Table 4. Trace message numbers and the corresponding message text

Message number	Message text
IIS-CONN-ORA-006001	->{0}::{1}
IIS-CONN-ORA-006002	<-{0}::{1}

Oracle connector runs in sequential mode when a reject link has a constraint violation reject condition

When an Oracle Connector stage is configured to bulk load data and has a reject link where the **SQL error - constraint violation** reject condition is selected, the connector runs in sequential mode.

Symptoms

When you run the job that contains the Oracle Connector stage, you get the following message:

```
[IIS-CONN-ORA-003004] The connector was configured to load data in parallel
but the reject condition for checking constraints was selected for the
reject link. This combination is not supported. The connector will run in
sequential mode.
```

Causes

When the following conditions are met, the Oracle connector must run in sequential mode:

- The connector is configured to write data in bulk load mode
- A reject link is defined for the stage
- The **SQL error - constraint violation** reject condition is specified for the reject link

Suppose that a stage is configured to reject rows that violate the constraints, to disable constraints before the load, and to enable them after the load. The following steps occur:

1. The connector disables the constraints before the load and then loads the data.
2. The ROWID values of the rows that violated the constraints are stored in the exceptions table on the Oracle database.
3. The connector sends the rows that failed the constraints to the reject link.
4. The connector deletes the rows that failed the constraints from the target table and enables the constraints.

All rows that violate the constraints are rejected. For example, suppose that two rows that have the same primary key value are loaded. Because this condition violates the primary key constraint, both rows are rejected.

In this scenario, the Oracle connector must run in sequential mode because of the way that parallel jobs that contain the Oracle connector work. A parallel job uses one conductor process and one or more player processes for each stage. When the

Oracle connector uses player processes, the processes are independent of each other and cannot detect when other player processes start or end. Only the conductor process can detect when the player processes are complete, and rows cannot be rejected until all the player processes are complete. However, only player processes can access the reject link for a stage. As a result, the connector must run in sequential mode and use only one player process to load the data.

Resolving the problem

Complete one of the following tasks:

- If constraints are not defined for the target table, clear the **SQL error - constraint violation** reject condition for the reject link. The job can then run in parallel mode.
- Use the insert write mode instead of the bulk load write mode. When the connector uses the insert write mode, constraints remain enabled while the player processes insert data to the table. Constraint violations are reported immediately to the player processes, and the player processes can send rows that violate constraints to the reject link.
- Enforce constraints after data is bulk loaded to the target table. Instead of configuring the stage to reject rows that violate the constraints automatically, complete the following steps:
  1. Disable the constraints on the table.
  2. To bulk load data to the table, run the job.
  3. Process the exceptions table and enable the constraints manually, or specify a PL/SQL block in the **After SQL** property of the stage.
- On the Advanced page for the stage, set the **Execution mode** property to **Sequential**. The stage runs in sequential mode, and message IIS-CONN-ORA-003004 is not logged if the **SQL error - constraint violation** reject condition is selected on a reject link.

Reference

To use the Oracle connector successfully, you might need detailed information, such as information about data type mappings, stage properties, and supported read and write methods.

Runtime mappings between InfoSphere DataStage columns and SQL statement parameters

When the connector exchanges data with an Oracle database, the connector assumes that the data for each column conforms with the data type definition that is specified for that column on the link.

The data type definition includes the SQL type, length, scale, nullable, and extended attributes. If data type conversion is required, the connector relies on the Oracle database to accept or reject the conversion. If the conversion is rejected because of data type incompatibility, data truncation, or some other issue, the Oracle database reports an error and the connector acts based on how it was configured.

For example, suppose that when the connector inserts records into the database, the database reports an error for the data type conversion of a field in a record. Depending on how the connector is configured, the connector might reject records that fail data type conversion or log an error and stop the job.

When the **Read mode** property is set to **Select** or **PL/SQL** and the connector is configured to read Oracle data and provide the data on an output link, the connector tries to match the names of the result set columns with the output link columns. The order of the columns on the link and in the Oracle database is irrelevant.

If the **Read mode** property is set to **PL/SQL** and the **Lookup type** is set to **Sparse**, the connector matches by name the reference link columns with the parameters in the PL/SQL block. The connector maps the columns that are marked as key columns to PL/SQL input/output parameters and maps the remaining columns to the PL/SQL output parameters. If the connector cannot match the names, the connector attempts to use the column order to associate link columns and parameters. Therefore, the connector associates the first column on the link with the first parameter, associates the second column on the link with the second parameter, and so on.

When the **Write mode** property is set to **Insert**, **Update**, **Delete**, or **PL/SQL**, the connector maps the columns on the input link to the input parameters that are specified in the SQL or PL/SQL statement.

Two formats are available for specifying parameters in the statement: InfoSphere DataStage syntax and Oracle syntax. The following list describes how the connector maps the columns, based on the format that you use to specify the parameters:

InfoSphere DataStage syntax
: The InfoSphere DataStage syntax is ORCHESTRATE.*parameter_name*. If you use InfoSphere DataStage syntax to specify parameters, the connector uses name matching. Therefore, every parameter in the statement must match a column on the link, and the parameter and the column must have the same name. If the connector cannot locate a matching column for a parameter, an error message is logged and the operation stops.

Oracle syntax
: The Oracle syntax is *name*, where *name* is the parameter name or parameter number. If you use the Oracle syntax to specify parameters, the connector first tries name matching. If name matching fails because some or all of the names do not match, the connector checks whether the name values are integers. If all of the name values are integers, the connector uses these integers as 1-based ordinals for the columns on the link. If all of the name values are integers but some or all of the integer values are invalid, meaning smaller than 1 or larger than the total number of columns on the link, an error message is logged and the operation stops. If some or all of the name values are not integers, the connector maps columns based on column order.

After completing the mapping, the connector removes any output link columns that were not mapped. If the job later references one of the unmapped columns, a runtime error occurs. For example, if the statement `SELECT COL1, COL2 FROM TABLE1` is specified for the stage and the output link defines the columns COL1, COL2, and COL3, the connector completes the following tasks:

1. Binds column COL1 from the statement to column COL1 on the link.
2. Binds column COL2 from the statement to column COL2 on the link.
3. Removes column COL3 from the link at run time because the COL3 column is unmapped.

Data type mapping and Oracle data types

When the Oracle connector imports a table definition, the connector converts Oracle data types to IBM InfoSphere DataStage data types. When the Oracle connector creates a table by issuing an SQL statement that is specified in the **Create table statement** property, the connector converts InfoSphere DataStage data types to Oracle data types.

Oracle datetime data types

The Oracle connector can read from and write to columns that use the Oracle datetime data types DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE.

The way that the connector handles Oracle datetime data types depends on whether the design-time schema specifies datetime columns or text columns. In a job, columns of DATE, TIME, and TIMESTAMP data types are datetime columns, while columns of CHAR, VARCHAR, LONGVARCHAR, NCHAR, NVARCHAR, and LONGNVARCHAR are text columns.

When the table definition on a link specifies a column in a text data type, the text values that the connector writes must match the format that is specified in the Oracle NLS session parameters. In Oracle, the following session parameters control the format of dates and time stamps:

- NLS_CALENDAR
- NLS_DATE_FORMAT
- NLS_DATE_LANGUAGE
- NLS_TIME_FORMAT
- NLS_TIME_TZ_FORMAT
- NLS_TIMESTAMP_FORMAT
- NLS_TIMESTAMP_TZ_FORMAT

You can specify the session parameters in one of the following ways:

- Alter the current session by including ALTER SESSION SET *parameter = value* statements in the **Before SQL statement (node)** property. This method is preferred.
- Set the environment variables that have the same names as the session parameters. If you use this method, you must also define the **NLS_LANG** environment variable.

When the Oracle connector forwards datetime values to the Oracle client as text, the Oracle client assumes that the values match the format that the NLS session parameters specify. If the format does not match, the Oracle client returns an error for the values, and the connector logs a message. For example, if the **NLS_DATE_FORMAT** session parameter is set to MM/DD/YYYY, then the text values that the connector writes to a column of DATE data type must adhere to that format. In this case, the value 12/03/2008 is acceptable, but the value 03-DEC-2008 is not.

When the design-time schema specifies a column in a datetime data type, the Oracle connector ignores the Oracle NLS settings and converts the values into the Oracle datetime data type.

You can configure the Oracle connector to log debug messages that contain information about the current settings for the Oracle NLS session parameters, NLS

database parameters, and the **NLS_LANG** environment variable. By default, debug messages are not shown in the log file. To view debug messages in the log file, set the **CC_MSG_LEVEL** environment variable to 2.

When the table definition on the output link specifies a column in a text data type, the values that the connector provides on the output link automatically match the format that the Oracle NLS session parameters specify. This matching occurs because Oracle automatically converts datetime values to text values in the specified format. When the table definition on the output link specifies a column in a datetime data type, the Oracle connector performs the conversion between the two datetime data types and ignores the Oracle NLS settings.

Oracle LOB and XMLType data types

The Oracle connector supports reading and writing the XMLType data type and the Oracle LOB data types BFILE, BLOB, CLOB, NCLOB, LONG RAW, RAW.

When you configure the Oracle connector to read data from a database table that contains LOB columns, you specify how to produce the LOB field values on the output link. The choices are inline or by reference.

When you use the inline form for LOB field values, the connector produces the actual values. Because the actual values are transferred on the link, use the inline form when the LOB values are relatively small, typically not more than a few hundred KB. To configure the connector to use the inline form, set **Enable LOB references** to **No**.

Use the reference form to transfer LOB values that are relatively large, typically more than 1 MB, from the source stage to the target stage. However, when you use the reference form, interim stages cannot process the actual values. For example, if you add a Transformer stage to a job, the Transformer stage cannot perform operations on the actual LOB values because only the reference strings, not the actual values, are transferred through the job.

To configure the Oracle connector to use the reference form, set **Enable LOB references** to **Yes**. Then, in the **Columns for LOB references** property, select the columns to pass by reference. Only link columns of LongVarChar, LongNVarChar and LongVarBinary data types are available for selection.

When a downstream LOB-aware stage receives the reference string on its input link, the stage engages the Oracle connector to retrieve the actual value that the reference string represents. The stage then processes that actual value. The connector outputs these reference strings as the values of the fields. When a downstream LOB-aware stage requires the values, the connector uses the information in the reference strings to retrieve the actual values and then passes them to the downstream stage, which loads the values into the target table. The LOB-aware stages include the DB2 connector, WebSphere MQ connector, ODBC connector, Teradata connector, and Oracle connector. If you specify a target stage that is not LOB-aware, the target stage cannot recognize the reference string as a special locator value and treats the reference string as ordinary data.

Consider these potential issues when you configure the connector to read and write LOB data:

- The connector supports both the inline and reference form to transfer BFILE, BLOB, CLOB, NCLOB, and XMLType columns.

- The connector supports only the inline form to transfer LONG and LONG RAW columns. The length attribute for the column on the link must be set to the maximum expected length for the actual data at run time.
- If at run time Oracle connector dynamically adds a column to the link that has the **Runtime Column Propagation** setting enabled and the link column corresponds to a LONG or LONG RAW table column in the database, the connector sets the link column length to the maximum value that meets both of the following conditions:
  - The value does not exceed 999999.
  - When the value is multiplied by the value that is specified in the **Array size** property for the stage, the product does not exceed 10485760 (the number of bytes in 10 MB).
- When you configure the Oracle connector to read data from a BFILE column, you can transfer the actual file contents, or you can transfer only a reference to the file location. If you transfer the file contents of a BFILE, set the **Transfer BFILE contents** property to **Yes**. By default, **Transfer BFILE contents** is set to **No**, and the connector transfers only the reference to the file location.
- When you configure the connector to read XMLType data and manually create the SELECT statement, you must use an alias to reference the table. Also, the XMLType column must use the Oracle GETCLOBVAL() or GETBLOBVAL() member function to get the actual XML content as BLOB or CLOB. If the column on the output link is defined as LongVarChar or LongNVarChar and passed inline, use the Oracle GETCLOBVAL() member function. If the column is defined as LongVarBinary and passed inline, use the GETBLOBVAL() member function. Do not use the GETCLOBVAL() and GETBLOBVAL() member functions when you pass XMLType columns as LOB references. To read from an XMLType object table or object view, use the OBJECT_VALUE pseudonym for the column name.
- When you configure the connector to write XMLType data, if the column on the input link is defined as Binary, VarBinary, or LongVarBinary, you must use the Oracle SYS.XMLTYPE.CREATEXML() member function in the SQL statement to create the XML content.

Example: Writing to an XMLType column

The following statement is the table definition:

```
CREATE TABLE TABLE1 (COL1 NUMBER(10), COL2 XMLTYPE) XMLTYPE COL2 STORE
  AS BINARY XML;
```

To write the binary XML value to the XMLType column, enter this INSERT statement in the **Insert statement** property in the connector:

```
INSERT INTO TABLE1 (COL1, COL2) VALUES (ORCHESTRATE.COL1,
 SYS.XMLTYPE.CREATEXML(ORCHESTRATE.COL2, 1, NULL, 1, 1));
```

In this example, the second parameter of the SYS.XMLTYPE.CREATEXML function specifies the character set ID for the US7ASCII character set in Oracle. The third parameter is an optional schema URL that forces the input to conform to the specified schema. The fourth parameter is a flag that indicates that the instance is valid according to the specified XML schema. The fifth parameter is a flag that indicates that the input is well formed.

Example: Reading XMLType data from a standard table or view

The following statement is the table definition:

```
CREATE TABLE TABLE1 (COL1 NUMBER(10), COL2 XMLTYPE)
 XMLTYPE COL2 STORE AS CLOB;
```

To retrieve the XML value as a CLOB value, enter this SELECT statement in the **Select statement** property in the connector:

```
SELECT COL1, T.COL2.GETCLOBVAL() FROM TABLE1 T;
```

To retrieve the XML value as a BLOB value that uses the character encoding AL32UTF8, enter this SELECT statement in the **Select statement** property in the connector:

```
SELECT COL1, T.COL2.GETBLOBVAL(893) FROM TABLE1 T;
```

The number 893 is the character set ID for the AL32UTF8 character set in Oracle. Oracle defines a character set ID for each character encoding that it supports. For information about the supported character encodings and IDs, see the Oracle documentation.

Example: Reading XMLType data from an object table

The following statement is the table definition:

```
CREATE TABLE TABLE1 OF XMLTYPE XMLTYPE
 STORE AS BINARY XML;
```

To retrieve the XML value as a CLOB value, enter this SELECT statement in the **Select statement** property in the connector:

```
SELECT T.OBJECT_VALUE.GETCLOBVAL() FROM TABLE1 T;
```

To retrieve the XML value as a BLOB value that uses the US7ASCII character encoding, enter this SELECT statement in the **Select statement** property in the connector:

```
SELECT T.OBJECT_VALUE.GETBLOBVAL(1) FROM TABLE1 T;
```

The number 1 is the character set ID for the US7ASCII character set in Oracle.

Example: Reading XMLType data from an object view

This example uses the TABLE1 table, which was defined in the previous example. The following statement is the view definition:

```
CREATE VIEW VIEW1 AS SELECT * FROM TABLE1;
```

To retrieve the XML value from the VIEW1 view as a CLOB value, enter this SELECT statement in the **Select statement** property in the connector:

```
SELECT V.OBJECT_VALUE.GETCLOBVAL() FROM VIEW1 V;
```

Data type mappings from Oracle to InfoSphere DataStage

When importing metadata, the Oracle connector converts Oracle data types to InfoSphere DataStage data types.

The following table shows the mapping between Oracle data types and InfoSphere DataStage data types. In the table, the following abbreviations are used:

- *n* – size
- *p* – precision
- *fsp* – precision for fractions of a second
- *yp* – year precision

- *dp* – day precision
- *sp* – second precision

Single-byte and multibyte character sets are specified in the table. For a single-byte character set, the **NLS_CHARACTERSET** database parameter is set to a single-byte character set such as WE8MSWIN1252. For a multibyte character set, the **NLS_CHARACTERSET** database parameter is set to a multibyte character set such as AL32UTF8.

Table 5. Oracle data types and corresponding InfoSphere DataStage data types

Oracle data type	InfoSphere DataStage SQL type	InfoSphere DataStage length	InfoSphere DataStage scale	InfoSphere DataStage extended
CHAR(*n* BYTE)	CHAR	*n*	unset	unset
CHAR(*n* CHAR) single-byte	CHAR	*n*	unset	unset
CHAR(*n* CHAR) multibyte	NCHAR	*n*	unset	unset
CHAR single-byte	If the **NLS_LENGTH_SEMANTICS** database parameter is set to CHAR, then see CHAR(n CHAR) single-byte. Otherwise, see CHAR(n BYTE). In both cases, assume that n = 1.			
CHAR multibyte	If the **NLS_LENGTH_SEMANTICS** database parameter is set to CHAR, then see CHAR(n CHAR) multibyte. Otherwise, see CHAR(n BYTE). In both cases, assume that n = 1.			
VARCHAR2(*n* BYTE)	VARCHAR	*n*	unset	unset
VARCHAR2(*n* CHAR) single-byte	VARCHAR	*n*	unset	unset
VARCHAR2(*n* CHAR) multibyte	NVARCHAR	*n*	unset	unset
CLOB single-byte	LONGVARCHAR	unset	unset	unset
CLOB multibyte	LONGNVARCHAR	unset	unset	unset
LONG single-byte	LONGVARCHAR	unset	unset	unset
LONG multibyte	LONGNVARCHAR	unset	unset	unset
NCHAR(*n*)	NCHAR	*n*	unset	unset
NCHAR	See NCHAR(*n*) and assume *n* = 1.			
NVARCHAR2(*n*)	NVARCHAR	*n*	unset	unset
NCLOB	LONGNVARCHAR	unset	unset	unset
NUMBER	DOUBLE	unset	unset	unset
NUMBER (*p*, *s*) {*p*>=*s*} {*s*>=0}	DECIMAL	*p*	*s*	unset
NUMBER(*p*, *s*) {*p*<*s*} {*s*>=0}	DECIMAL	*s*	*s*	unset
NUMBER(*p*, *s*) {*s*<0}	DECIMAL	*p*-*s*	unset	unset
FLOAT(*p*) {1 <=*p* <=63}	FLOAT	unset	unset	unset

Table 5. Oracle data types and corresponding InfoSphere DataStage data types (continued)

Oracle data type	InfoSphere DataStage SQL type	InfoSphere DataStage length	InfoSphere DataStage scale	InfoSphere DataStage extended
FLOAT(*p*) {64 <=*p* <= 126}	DOUBLE	unset	unset	unset
BINARY_FLOAT	FLOAT	unset	unset	unset
BINARY_DOUBLE	DOUBLE	unset	unset	unset
LONG RAW	LONGVARBINARY	unset	unset	unset
RAW(*n*)	VARBINARY	*n*	unset	unset
BLOB	LONGVARBINARY	unset	unset	unset
BFILE	VARCHAR	285	unset	unset
DATE	TIMESTAMP	unset	unset	unset
TIMESTAMP(*fsp*)	TIMESTAMP	unset	*fsp*	Microseconds
TIMESTAMP(*fsp*) WITH TIME ZONE	TIMESTAMP	unset	*fsp*	Microseconds
TIMESTAMP(*fsp*) WITH LOCAL TIME ZONE	TIMESTAMP	unset	*fsp*	Microseconds
TIMESTAMP	See TIMESTAMP(*fsp*) and assume *fsp*=6.			
TIMESTAMP WITH TIME ZONE	See TIMESTAMP(*fsp*) WITH TIME ZONE and assume *fsp*=6.			
TIMESTAMP WITH LOCAL TIME ZONE	See TIMESTAMP(*fsp*) WITH LOCAL TIME ZONE and assume *fsp*=6.			
INTERVAL YEAR (*yp*) TO MONTH	VARCHAR	*yp*+4	unset	unset
INTERVAL DAY TO SECOND (*sp*)	VARCHAR	*sp*+13	unset	unset
INTERVAL DAY (*dp*) TO SECOND	VARCHAR	*dp*+17	unset	unset
INTERVAL DAY (*dp*) TO SECOND (*sp*)	VARCHAR	*dp*+*sp*+11	unset	unset
INTERVAL YEAR TO MONTH	See INTERVAL YEAR (*yp*) TO MONTH and assume *yp*=2.			
INTERVAL DAY TO SECOND	See INTERVAL DAY (*dp*) TO SECOND (*sp*) and assume *dp*=2 and *sp*=6.			
ROWID	CHAR	18	18	unset
UROWID(*n*)	VARCHAR	*n*	unset	unset
UROWID	See UROWID(*n*) and assume *n*=4000.			
XMLType stored as CLOB or OBJECT_ RELATIONAL single-byte	See CLOB single-byte.			
XMLType stored as CLOB or OBJECT_ RELATIONAL multibyte	See CLOB multibyte.			

Table 5. Oracle data types and corresponding InfoSphere DataStage data types (continued)

Oracle data type	InfoSphere DataStage SQL type	InfoSphere DataStage length	InfoSphere DataStage scale	InfoSphere DataStage extended
XMLType stored as BINARY XML	See BLOB.			
Other	UNKNOWN	unset	unset	unset

Data type mappings for creating a table

When you use the **Table action** property to create a table, the connector maps InfoSphere DataStage column definitions to Oracle column definitions.

The following table lists the mappings and uses these abbreviations:

- *n* – size
- *p* – precision
- *sp* – second precision
- *s* – scale

Table 6. InfoSphere DataStage column definitions and corresponding Oracle column definitions

InfoSphere DataStage column definition	Oracle column definition
Data type: Bit Length: any Scale: any Extended: not applicable	NUMBER(5,0)
Data type: Char Length: unset Scale: any Extended: unset	CHAR(2000)
Data type: Char Length: *n* Scale: any Extended: unset	CHAR(*n*)
Data type: VarChar Length: unset Scale: any Extended: unset	VARCHAR2(4000)
Data type: VarChar Length: *n* Scale: any Extended: unset	VARCHAR2(*n*)
Data type: LongVarChar Length: any Scale: any Extended: unset	CLOB
Data type: Char Length: unset Scale: any Extended: Unicode	NCHAR(1000)

Table 6. InfoSphere DataStage column definitions and corresponding Oracle column definitions (continued)

InfoSphere DataStage column definition	Oracle column definition
Data type: Char Length: *n* Scale: any Extended: Unicode	NCHAR(*n*)
Data type: VarChar Length: unset Scale: any Extended: Unicode	NVARCHAR2(2000)
Data type: VarChar Length: *n* Scale: any Extended: Unicode	NVARCHAR2(*n*)
Data type: LongVarChar Length: *n* Scale: any Extended: Unicode	NCLOB
Data type: NChar Length: unset Scale: any Extended: not applicable	NCHAR(1000)
Data type: NChar Length: *n* Scale: any Extended: not applicable	NCHAR(*n*)
Data type: NVarChar Length: unset Scale: any Extended: not applicable	NVARCHAR2(2000)
Data type: NVarChar Length: *n* Scale: any Extended: not applicable	NVARCHAR2(*n*)
Data type: LongNVarChar Length: any Scale: any Extended: not applicable	NCLOB
Data type: Binary Length: unset Scale: any Extended: not applicable	RAW(2000)
Data type: Binary Length: *n* Scale: any Extended: not applicable	RAW(*n*)
Data type: VarBinary Length: unset Scale: any Extended: not applicable	RAW(2000)

Table 6. InfoSphere DataStage column definitions and corresponding Oracle column definitions (continued)

InfoSphere DataStage column definition	Oracle column definition
Data type: VarBinary Length: *n* Scale: any Extended: not applicable	RAW(*n*)
Data type: LongVarBinary Length: any Scale: any Extended: not applicable	BLOB
Data type: Decimal Length: *p* Scale: unset Extended: not applicable	NUMBER(*p*)
Data type: Decimal Length: *p* Scale: *s* Extended: not applicable	NUMBER(*p*,*s*)
Data type: Double Length: any Scale: any Extended: not applicable	BINARY_DOUBLE
Data type: Float Length: any Scale: any Extended: not applicable	BINARY_FLOAT
Data type: Real Length: any Scale: any Extended: not applicable	BINARY_FLOAT
Data type: TinyInt Length: any Scale: any Extended: unset	NUMBER(3,0)
Data type: SmallInt Length: any Scale: any Extended: unset	NUMBER(5,0)
Data type: Integer Length: any Scale: any Extended: unset	NUMBER(10,0)
Data type: BigInt Length: any Scale: any Extended: unset	NUMBER(19,0)
Data type: TinyInt Length: any Scale: any Extended: unsigned	NUMBER(3,0)

Table 6. InfoSphere DataStage column definitions and corresponding Oracle column definitions (continued)

InfoSphere DataStage column definition	Oracle column definition
Data type: SmallInt Length: any Scale: any Extended: unsigned	NUMBER(5,0)
Data type: Integer Length: any Scale: any Extended: unsigned	NUMBER(10,0)
Data type: BigInt Length: any Scale: any Extended: unsigned	NUMBER(20,0)
Data type: Numeric Length: *p* Scale: unset Extended: not applicable	NUMBER(*p*)
Data type: Numeric Length: *p* Scale: *s* Extended: not applicable	NUMBER(*p*,*s*)
Data type: Date Length: any Scale: any Extended: not applicable	DATE
Data type: Time Length: any Scale: unset Extended: unset	DATE
Data type: Time Length: any Scale: *sp* Extended: unset	TIMESTAMP(*sp*)
Data type: Timestamp Length: any Scale: unset Extended: unset	DATE
Data type: Timestamp Length: any Scale: *sp* Extended: unset	TIMESTAMP(*sp*)
Data type: Time Length: any Scale: unset Extended: Microseconds	TIMESTAMP(6)
Data type: Time Length: any Scale: *sp* Extended: Microseconds	TIMESTAMP(*sp*)

Table 6. InfoSphere DataStage column definitions and corresponding Oracle column definitions (continued)

InfoSphere DataStage column definition	Oracle column definition
Data type: Timestamp Length: any Scale: unset Extended: Microseconds	TIMESTAMP(6)
Data type: Timestamp Length: any Scale: *sp* Extended: Microseconds	TIMESTAMP(*sp*)
Data type: Unknown Length: any Scale: any Extended: any	NCLOB

Properties for the Oracle connector

Use these options to modify how the connector reads and writes data.

Enable quoted identifiers property

To maintain the case-sensitivity of Oracle schema object names, you can manually enter double quotation marks around each name or set the **Enable quoted identifiers** property to **Yes**.

Usage

The Oracle connector automatically generates and runs SQL statements when either of these properties are set:

- **Generate SQL at runtime** is set to **Yes**.
- **Table action** is set to **Create**, **Replace**, or **Truncate**.

In these cases, the generated SQL statements contain the names of the columns and the name of the table on which to perform the operation. The column names in the database table match the column names that are specified on the link for the stage. The table name matches the table that is specified in the **Table name** property.

By default, the Oracle database converts all object names to uppercase before it matches the names against the Oracle schema object names in the database. If the Oracle schema object names all use uppercase, then how you specify the names in the connector properties, by using uppercase, lowercase, or mixed case, has no effect on schema matching. The names will match. However, if the Oracle schema object names use all lowercase or mixed case, you must specify the names exactly as they appear in the Oracle schema. In this case, you must manually enter double quotation marks around each name or set the **Enable quoted identifiers** property to **Yes**.

Examples

For example, assume that the **Enable quoted identifiers** property is set to **No** and that you want to create a table that contains one column and use a SELECT statement that references the column. The statement `CREATE TABLE Table2b (Col1 VARCHAR2(100))` creates the table TABLE2B, which contains one column, COL1. The statement `SELECT Col1 FROM tABLE2B` runs successfully because the Oracle database

automatically changes the Col1 and tABLE2B names in the statement to the uppercase versions COL1 and TABLE2B and matches these names with the actual schema object name and column name in the database.

Now assume that you use the statement `CREATE TABLE "Table2b" ("Col1" VARCHAR2(100))` to create the table Table2b, which contains one column, Col1. Case-sensitivity is preserved because you enclosed the table and column names in double quotation marks. Now the statement `SELECT Col1 FROM tABLE2B` fails because the Oracle database automatically changes Col1 and Table2b to the uppercase versions COL1 and TABLE2B, and these names do not match the actual names, Col1 and Table2b, in the database. However, the statement `SELECT "Col1" FROM "Table2b"` runs successfully.

Now consider an example that illustrates the effect of the **Enable quoted identifiers** property on table and column creation. Assume that the **Table name** property is set to john.test. The input link contains the columns Col1, Col2, and Col3, all of which are of VarChar(10) data type. The **Table action** property is set to **Create**. If the **Enable quoted identifiers** property is set to **No**, the connector generates and runs these SQL statements at runtime and creates the table JOHN.TEST with the columns COL1, COL2, and COL3:

```
CREATE TABLE john.test(Col1 VARCHAR2(10),Col2 VARCHAR2(10),Col3 VARCHAR2(10));
```

However, if the **Enable quoted identifiers** property is set to **Yes**, the connector generates and runs this SQL statement at runtime and creates the table john.test with the columns Col1, Col2, and Col3:

```
CREATE TABLE "john"."test"("Col1" VARCHAR2(10),"Col2" VARCHAR2(10),
 "Col3" VARCHAR2(10));
```

Isolation level property

Use the **Isolation level** property to configure how the connector manages statements in transactions.

Usage

As soon as the connector establishes a connection to the Oracle database and issues the first transactional statement, the connector implicitly starts a transaction that uses the specified isolation level. All of the operations that the connector performs on the database are part of the current transaction. When the transaction ends, either through a commit or a rollback, and the connector issues the next transactional statement, the connector again implicitly starts a new transaction on the connection.

Oracle cannot roll back some database operations, even if the transaction to which they belong is rolled back. For example, DDL operations cannot be rolled back.

The following table describes the different options for the **Isolation level** property.

Table 7. Options for the isolation level property

Option	Description
Read committed	Each SELECT statement that runs in the transaction sees the rows that were committed when the current statement started.

Table 7. Options for the isolation level property (continued)

Option	Description
Serializable	Each SELECT statement that runs in the transaction sees only the rows that were committed when the transaction started.
Read only	Each SELECT statement that runs in the transaction sees only the rows that were committed when the transaction started. However, the DML statements INSERT, UPDATE, DELETE and MERGE are not allowed in the transaction. This isolation level prevents the PL/SQL block from running DML statements. However, if the PL/SQL block overrides the isolation level, the block can run DML statements, even if you set the isolation level to **Read only**.

Array size, buffer size, and record count properties

Use the array size, buffer size, and record count properties to control the number of records to read from a database or write to a database at one time.

Usage

You set the **Array size** and **Record count** properties together. The array size specifies the number of records to include in each batch that the read and write operations on the database process. The record count specifies the number of records to process in each transaction.

If the value that you specify for the **Record count** property is not 0 and is not a multiple of the value that you specify for the **Array size** property, the connector automatically chooses an array size so that the record count is a multiple of it. When the connector chooses the array size, the connector attempts to find a value that is close to the value that you specified. If the connector cannot find that value, it chooses the value 1 or the value that matches the record count value, whichever is closer to the value that you specified. Then, the connector logs an informational message to inform you that it modified the value of the **Array size** property.

If you configure row prefetching, when a SELECT statement runs, the connector fetches the number of rows that is specified by the **Array size** property. In addition, the Oracle client fetches the number of rows that is specified by the **Prefetch row count** property.

To control when the connector bulk loads buffered records into a target table, set an array size and a buffer size. When the connector stage is configured to run in parallel on more than one processing node, each of the processing nodes establishes a separate Oracle session and loads data to the target table concurrently with the other processing nodes.

The connector always tries to load data in chunks, where each chunk contains the number of rows that is specified in the **Array size** property. The **Buffer size** property controls the maximum size of the buffer that holds each chunk of records in KB.

Based on the types and lengths of the columns that are defined on the input link, the connector calculates whether the specified array size can always fit into the

specified buffer size. If the buffer is too small to accommodate the number of records specified for the array size, the connector automatically resets the array size to the maximum number of records that fit in the buffer.

The following table shows the values that you can set these properties to.

Table 8. Values for the array size, buffer size, and record count properties

Property	Unit	Available values	Default
Array size	Records	1 - 999999999	2000
Buffer size	KB	4 - 100240	1024
Record count	Records	0 - 999999999 If you enter 0, the connector processes all records before it commits the transaction.	2000

How waves affect these properties

You can use the **Mark end of wave** property to specify whether to insert an end-of-wave marker after the number of records that are specified in the **Record count** property are processed. When the end-of-wave marker is inserted, any records that the Oracle connector buffered are released from the buffer and pushed into the job flow so that downstream stages can process them.

When an upstream stage provides records to the Oracle connector in the form of waves, each wave includes an end-of-wave marker. In this case, the array size and the record count apply to each separate wave of records. If not enough records are available to fill the buffer to the specified array size value, the connector loads the incomplete buffer of records as a batch and then processes the next wave of records. When records do not arrive in waves and instead all arrive in a single wave, the array size and the record count apply to that single wave.

Properties to run an SQL statement before or after processing data

Use the **Run before and after SQL statements** property to configure the connector to run an SQL statement before or after processing data. You can configure the connector to run the SQL statements before or after processing any data in a job or to run an SQL statement once before or after processing the data on each node.

Usage

Running an SQL statement before or after processing data is useful when you need to perform operations that prepare database objects for data access. For example, you might use an SQL statement to create a target table and add an index to it. The SQL statement that you specify is performed once for the whole job, before any data is processed.

After the connector runs the statement that is specified in the **Before SQL statement** property or **After SQL statement** property, the connector explicitly commits the current transaction. For example, if you specify a DML statement, such as INSERT, UPDATE, DELETE, or MERGE, in the **Before SQL statement** property, the results of the DML statement are visible to individual nodes.

To run an SQL statement on each node that the connector is configured to run on, use the **Before SQL (node) statement** property or the **After SQL (node) statement** property. The connector runs the specified SQL statement once before any data is processed on each node or once after any data is processed on each node. Then, the connector explicitly commits the current transaction. For example, to set the data format to use for the client session on a node, you specify the ALTER SESSION statement in the **Before SQL (node)** property.

When you specify the statement to run before or after processing, enter the SQL or PL/SQL statement, or enter the fully qualified path to the file that contains the SQL or PL/SQL statement. Do not include input bind variables or output bind variables in the SQL or PL/SQL statement. If the statement contains these types of variables, the connector logs a fatal message, and the operation stops. If you specify a file name, the file must be on the computer where the InfoSphere Information Server engine tier is installed, and you must set the **Read Before SQL statement from file** or **Read After SQL statement from file** property to **Yes**.

When the connector is used to write records to the database and is configured to perform a table action on the target table before writing data, you can use the **Run table action first** property to control whether the SQL statement or the table action is performed first.

Properties that control job failure

You can control whether to stop a job when certain SQL statements do not successfully complete or when the first warning message is reported.

Stopping a job in the middle of a process is useful when you want to receive prompt notification that something you expected to work failed. By design, a job stops when a fatal message is reported. The following list contains the properties that control job failure:

- **Abort when create table statement fails**
- **Abort when drop table statement fails**
- **Abort when truncate table statement fails**
- **Fail on error for Before SQL statement**,
- **Fail on error for After SQL statement**
- **Fail on error for Before SQL (node) statement**
- **Fail on error for After SQL (node) statement**
- **Fail on error for index rebuilding**

By default, all of the properties except **Fail on error for drop table statement** and **Fail on error for index rebuilding** are set to **Yes**. If a property is set to **Yes** and an error occurs, the message is reported to the log file, and the job stops. If a property is set to **No** and an error occurs, the corresponding message is reported to the log file, and the job continues.

If you set the property **Process warning messages as fatal errors** to **Yes**, the job stops when the first warning message is issued, and the connector reports the error in the log. By default, this property is set to **No**. In this case, when the first warning message is issued, it is sent to the log and the job continues.

Transparent application failover properties

You can configure the Oracle connector to receive messages that describe when the Oracle client starts transparent application failover (TAF) and how TAF progresses.

Usage

When a database connection is enabled for TAF, the application that is connected to the database is transparently reconnected to an alternative database instance if the original connection fails. Because the reconnection occurs transparently, the connector might seem to unexpectedly stop running and hang while the reconnection occurs. For this reason, you might want to configure the connector to receive notifications about TAF. You can also specify how long the Oracle client side of the connection waits for TAF to complete.

To configure the connector for TAF notifications, set these properties:

- Set **Manage application failover** to **Yes**.
- Set **Number of retries** to the number of times to attempt application failover.
- Set **Time between retries** to the number of seconds to wait between subsequent attempts to failover.

If the RETRIES and DELAY values are specified as part of the FAILOVER_MODE configuration in the `tnsnames.ora` file, the connector ignores these values and instead uses the values that are specified for the **Number of retries** and **Time between retries** properties.

The two types of TAF are SESSION and SELECT. If you want the connector to continue fetching data for the SELECT statement that is interrupted when failover occurs, enable the SELECT failover type.

When TAF starts, the connector takes the following steps:

1. The connector logs a warning message that indicates that TAF began. This message includes the type of TAF that is taking place, either SESSION or SELECT.
2. Each time that the Oracle client attempts application failover, the connector logs a warning message to indicate the failover attempt.
3. If the TAF succeeds, the connector logs a warning message to indicate that TAF completed successfully.
4. If the **Before SQL statement** property is set to **Yes**, the connector reruns the statement that is specified in the **Before SQL statement** property. If the **Replay Before SQL (node) statement** property is set to **Yes**, the connector reruns the statement that is specified in the **Before SQL (node) statement** property once on each node.
5. If all of the TAF attempts fail or if the Oracle client indicates that TAF cannot be completed, the connector logs a warning message, and the operation stops because the connector does not have a valid connection to the database.

Example: Multiple database connections are configured, and application failover is not enabled

For this example, the connector is configured in the following way:

- The connector is configured to run a SELECT statement that reads 1,000,000 rows from a table.
- The **Manage application failover** property is set to **No**.
- The connector is configured to connect to an Oracle RAC system.
- The connector specifies *ORCL_1* as the connect descriptor to use to connect to the *orcl1* database instance.
- The `tnsnames.ora` configuration file contains the following connect descriptors:

```
ORCL_1 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = tcp)(HOST = orcl1-server)(PORT = 1521))
    (CONNECT_DATA = (SERVICE_NAME = orcl)(INSTANCE_NAME = orcl1)
      (FAILOVER_MODE = (BACKUP = ORCL_2)(TYPE = select)(METHOD = preconnect))))

ORCL_2 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = orcl2-server)(PORT = 1521))
    (CONNECT_DATA = (SERVICE_NAME = orcl)(INSTANCE_NAME = orcl2)
      (FAILOVER_MODE = (BACKUP = ORCL_1)(TYPE = select)(METHOD = preconnect))))
```

The connection that is established through the ORCL_1 connect descriptor has the following characteristics:

- The Oracle client connects to the listener on host *orcl1-server* and port 1521 and attaches to the service *orcl* and the instance *orcl1*.
- The FAILOVER_MODE specifies that if the *orcl1* instance becomes unavailable while the application is connected to it, the SELECT type of TAF takes place.
- The BACKUP option specifies the backup connect descriptor that the Oracle client uses if failover occurs.
- The METHOD option specifies when the Oracle client connects to the backup instance. The value PRECONNECT specifies that the backup connection be established at the same time that the primary connection is established. Then, if the primary connection fails, the failover to the backup connection occurs.

If the connection to the instance *orcl1* fails while the connector is fetching data from a table, the connector stops processing data until the failover to the instance *orcl2* takes place. Because **Manage transparent application failover** is set to **No**, the connector does not receive any notification when failover starts or completes. Because the connection to the backup instance is established at the same time that the primary connection is established, the failover occurs quickly and might occur so quickly that the delay is not noticeable. After the failover completes, the connector continues fetching data because the failover TYPE is set to SELECT.

Suppose that the connector was configured to write data and was running an INSERT statement when the connection to the instance failed. After the failover completed and the connector attempted to insert new data or commit the data that was inserted just prior to the instance failing, the statement fails. The connector logs an error message, and the job stops.

Example: A single database connection is configured, and application failover is enabled

In this example, there is only one database instance, and failover occurs only after the Oracle administrator restarts the instance. For this example, the Oracle connector is configured in the following way:

- The connector is configured to run a SELECT statement that reads 1,000,000 rows from a table.
- The **Manage application failover** property is set to **Yes**.
- The connector is configured to connect to a single database instance.
- The connector specifies *ORCL* as the connect descriptor to use to connect to the *orcl* database instance.
- The `tnsnames.ora` configuration file contains the following connect descriptor:

```
ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = orcl-server)(PORT = 1521))
```

```
    (CONNECT_DATA = (SERVICE_NAME = orcl)
      (FAILOVER_MODE = (TYPE=select)(METHOD=basic)(RETRIES=20)(DELAY=5)
      )
    )
  )
```

The connection that is established through the ORCL connect descriptor has the following characteristics:

- The Oracle client connects to the listener on host *orcl-server* and port 1521 and attaches to service *orcl*, which implements a single instance.
- The FAILOVER_MODE specifies that if the instance becomes unavailable while the application is connected to it, the SELECT type of TAF takes place.
- The METHOD option, which is set to BASIC, specifies that the attempt to reconnect to the instance happens when the failover occurs.

If the connection to the instance fails while the connector is fetching data from a table, the connector receives a notification that failover is taking place because **Manage transparent application failover** is set to **Yes**. Each time that the Oracle client attempts to reestablish the connection, the Oracle client notifies the connector, and the connector logs a message. The Oracle client ignores the RETRIES and DELAY options because the **Number of retries** and **Time between retries** properties are configured for the connector.

Suppose that the connector was configured to write data and was running an INSERT statement when the connection to the instance failed. After failover completed, the connector can try to recover from the error and continue to write records to the database. To configure the connector to attempt to resume the write operation after failover completes, set the **Resume write** property to **Yes**.

Properties for managing connections

Use properties to manage how the connector reconnects to an Oracle database after losing the connection or closing an inactive connection.

Usage

If the connection to the Oracle database is lost, the connector can attempt to reconnect to the database for a specified number of tries. When the connection is reestablished, data can be processed from the point where it left off. The connector attempts to reconnect when the situation is feasible, such as after a session timeout or a network outage. However, in some cases the connector might not be able to reconnect.

To preserve connection resources to the database, you can configure the connector to automatically close the connection to an Oracle database if the connection is inactive for a specified period. For example, you might want the connector to disconnect if the job is processing records in transaction waves, and a long interval between the waves exists. If the connection to the database is closed during that time, other client applications can connect to the database.

The following table shows the properties for managing connections.

Table 9. Properties for managing connections

Property	Description
Reconnect	To reconnect to an Oracle database after losing the connection, set this property to **Yes**. This property applies to all links on the stage and cannot be configured separately for individual links.
Number of retries	Enter the number of times to try to establish a connection after a connection is lost.
Interval between retries	Enter the time in seconds to wait between retries to establish a connection.
Disconnect	To close an inactive connection, set this property to **Period of inactivity**.
Inactivity period	Enter the time in seconds after which an idle connection must be closed.

Read properties

Use these properties to modify how the connector reads data.

Prefetch properties:

Use the **Prefetch row count** and **Prefetch buffer size** properties to enable prefetching for SELECT statements. If row prefetching is enabled, the connector fetches the number of rows that is specified by the **Array size** property. In addition, the Oracle client fetches a number of rows that is based on the values of the **Prefetch row count** and **Prefetch buffer size** properties.

Usage

You can set the **Prefetch row count** property, the **Prefetch buffer size** property, or set both properties. If you set both properties to a value that is greater than 0, the Oracle client tries to prefetch the number of rows that is specified for the **Prefetch row count** property. If the number of rows cannot fit in the memory size that is specified for the **Prefetch buffer size** property, the Oracle client prefetches as many rows as can fit into the buffer.

When you set the **Prefetch row count** or **Prefetch buffer size** property to 0, the type of row prefetching that is controlled by that property is disabled.

The Oracle client immediately provides the rows that are fetched based on the value of the **Array size** property to the connector. The Oracle client caches the rows that are fetched based on the values of the **Prefetch row count** and **Prefetch buffer size** properties. As the connector continues to request data for the currently running SELECT statement, the fetch requests are optimized because the prefetched rows are cached.

The following table shows the values that you can set these properties to.

Table 10. Values for the prefetch row count and prefetch buffer size properties

Property	Available values	Default
Prefetch row count	0 - 999999999	1

Table 10. Values for the prefetch row count and prefetch buffer size properties (continued)

Property	Available values	Default
Prefetch buffer size (KB)	0 - 100240	0 By default, row prefetching based on buffer size is disabled.

Write properties

Use these properties to modify how the connector writes data.

Table action property:

Use the **Table action** property to configure the connector to complete create, replace, and truncate actions on a table at run time. These actions are completed before any data is written to the table.

Usage

You can set the **Table action** property to the values that are listed in the following table.

Table 11. Values of the **Table action** *property*

Value	Description
Append	No action is completed on the table. This option is the default.
Create	Create a table at run time. Use one of these methods to specify the CREATE TABLE statement: • Set **Generate create table statement at runtime** to **Yes** and enter the name of the table to create in the **Table name** property. In this case, the connector automatically generates the CREATE TABLE statement from the column definitions on the input link. The column names in the new table match the column names on the link. The data types of columns in the new table are mapped to the column definitions on the link. • Set **Generate create table statement at runtime** to **No**, and enter the CREATE TABLE statement in the **Create table statement** property.

Table 11. Values of the **Table action** *property (continued)*

Value	Description
Replace	Replace a table at run time. Use one of these methods to specify the DROP TABLE statement: • Set **Generate drop table statement at runtime** to **Yes**, and enter the name of the table to drop in the **Table name** property. • Set **Generate drop table statement at runtime** to **No**, and enter the DROP TABLE statement in the **Drop table statement** property. Use one of these methods to specify the CREATE TABLE statement: • Set **Generate create table statement at runtime** to **Yes**, and enter the name of the table to create in the **Table name** property. • Set **Generate create table statement at runtime** to **No**, and enter the CREATE TABLE statement in the **Create table statement** property.
Truncate	Truncate a table at run time. Use one of these methods to specify the TRUNCATE TABLE statement: • Set **Generate truncate table statement at runtime** to **Yes**, and enter the name of the table to truncate in the **Table name** property. • Set **Generate truncate table statement at runtime** to **No**, and enter the TRUNCATE TABLE statement in the **Truncate table statement** property.

To configure the job to fail when the statement that is specified by the table action fails, you can set the appropriate property to **Yes**:

- **Abort when create table statement fails**
- **Abort when drop table statement fails**
- **Abort when truncate table statement fails**

Otherwise, when the statement fails, the connector logs a warning message, and the job continues.

Drop unmatched fields property:

Use the **Drop unmatched fields** property to specify how to handle unused columns on the input link.

Usage

When you create a job that writes data from the input link to the database, you can use the **Drop unmatched fields** property to control how to handle any unused columns (fields) on the input link. Unused columns on the input link can be the following types of columns:

- Columns that the connector did not pair with any parameter in the target SQL or PL/SQL statement
- If **Bulk load** is specified as the write mode, columns that the connector did not pair with any target table column

You can set the **Drop unmatched fields** property to the values that are listed in the following table.

Table 12. Values of the **Drop unmatched fields** *property*

Value	Description
Yes	The connector drops any unused columns on the input link. For each dropped column, the connector writes an informational message in the job log to indicate that the column and its associated values were ignored.
No	When the connector encounters an unused column on the input link, the connector logs an error message and stops the job.

You use the **Enable quoted identifiers** property to specify whether the name matching between the input link columns and target SQL statement parameters or table columns is case sensitive.

Example

For example, consider the following job:

- The connector stage is configured to use bulk load as the write mode.
- The target table in the database contains these columns: FIRSTNAME, LASTNAME and DATEOFBIRTH.
- The input link of the connector contains these columns: FirstName, LastName, Address, DateofBirth, Phone, and Email.

The following table shows how the values of the **Drop unmatched fields** and **Enable quoted identifiers** properties affect the results of the job.

Table 13. How the values of the **Drop unmatched fields** *and* **Enable quoted identifiers** *properties affect the results of the job*

Drop unmatched fields	Enable quoted identifiers	Result
No	No	The connector logs an error message to indicate that the Address column from the input link is not used, and the job stops.

Table 13. How the values of the **Drop unmatched fields** *and* **Enable quoted identifiers** *properties affect the results of the job (continued)*

Drop unmatched fields	Enable quoted identifiers	Result
No	Yes	The connector logs an error message to indicate that the FirstName column from the input link is not used, and the job stops.
Yes	No	The connector logs informational messages to indicate that the Address, Phone, and Email columns from the input link are not used. The connector loads only the data that is provided for the FirstName, LastName and DateofBirth input link columns.
Yes	Yes	All columns are dropped. Because the Oracle database requires a minimum of one column in the records that are written to the database, the job fails and the connector logs an error message.

Preserve trailing blanks property:

Use the **Preserve trailing blanks** property to specify whether the stage preserves trailing white space characters in the text field values of the records that it passes to the database.

Usage

This property is available for all modes that are available in the **Write mode** property, including the bulk load mode. The property applies to the input link columns and key columns on the reference link that have the character data types, such as VarChar or NVarChar.

You can set the **Preserve trailing blanks** property to the values that are listed in the following table.

Table 14. Values of the **Preserve trailing blanks** *property*

Value	Description
Yes	The trailing white space characters are treated the same as any other characters. They are preserved along with the other characters, and the data is passed to the database in its original form. This behavior is the default for the connector.

Table 14. Values of the **Preserve trailing blanks** *property (continued)*

Value	Description
No	The stage removes trailing white space characters from the text field values. The trimmed values are passed to the database. Any leading white space characters in the values are preserved.

Fail on row error property:

Use the **Fail on row error** property to log an error message and stop the job when an error occurs while writing a record to the database.

Usage

This property is not available if the **Write mode** property is set to **Bulk load**.

You can set the **Fail on row error** property to the values that are listed in the following table.

Table 15. Values of the **Fail on row error** *property*

Value	Description
Yes	When a record is not written to the database, the connector logs an unrecoverable error, and the job stops.
No	When a record is not written to the database, the connector logs a warning message and continues to process the remaining input records.

The default value for the property depends on the type of job in which the connector stage is running. For parallel jobs, the default value is **Yes**. If a reject link is defined for the stage, this property is not available and automatically defaults to **Yes**.

For server jobs, the default value is **No**. By default, if an error occurs when writing a record to the database, a warning message is logged, and the job continues. If the input link comes from a Transformer stage that is configured to reject rows that the Oracle Connector stage could not write to the database, the **Fail on row errors** property must be set to **No**. The Transformer stage can send the rows that the Oracle Connector stage cannot write to the database to the reject link.

Logging properties:

Use the logging properties to specify how the Oracle connector logs the values that are in each column when an SQL statement fails to insert, update, or delete a row.

Usage

Each node that fails to insert, update, or delete rows prints the first row that failed on that node. The logging properties are not available when the **Write mode** property is set to **Bulk load**.

The following table shows the logging properties.

Table 16. Logging properties

Property	Values	Description
Log column values on first row error	• Yes • No	If you choose **Yes**, the connector logs column values for the first row that failed on each node. Also, the **Log key values only** and **Column delimiter** properties are enabled. The default value is **No**.
Log key values only	• Yes • No	If you choose **Yes**, the connector logs the values of key columns only. The default value is **No**.
Column delimiter	• Space • Newline • Tab • Comma	Specify the delimiter that is used between column values in the log.

Allow concurrent load sessions property:

Use the **Allow concurrent load sessions** property to specify whether multiple applications, such as multiple processing nodes of the Oracle Connector stage, can load data to the table, partition, or subpartition segments concurrently.

Usage

You can set the **Allow concurrent load sessions** property when the Oracle Connector stage is configured to load data to a table, partition, or subpartition segment from a single processing node. If you set the property to **No**, other applications cannot load data to the same segment while the connector loads data. Other applications might include external applications or other InfoSphere DataStage jobs.

If the Oracle Connector stage is configured to run in parallel on more than one processing node, each of the processing nodes establishes a separate Oracle session and loads data to the target table concurrently. In this scenario, if the **Allow concurrent load sessions** property is set to **No**, multiple processing nodes cannot load data concurrently to the same segment in the database. This situation might lead to the Oracle error ORA-00054, where the processing nodes try to load data to a segment while another processing node is loading data to the same segment. To avoid this situation, set the **Allow concurrent load sessions** property to **Yes**.

Sometimes, the Oracle Connector stage is configured to load data from multiple processing nodes to a partitioned Oracle table, and the stage is configured to partition the input data. If the table supports the specified partitioning type, each processing node loads data to its assigned partition segment or a set of subpartition segments, and the processing nodes do not compete for access to the segment. In this scenario, setting **Allow concurrent load sessions** property to **No** does not prevent the Oracle Connector stage from loading data in parallel from multiple processing nodes. However, the setting prevents other applications from concurrently loading data to the segments that are accessed by this Oracle Connector stage.

Index maintenance option property:

Set the **Index maintenance option** property to control how to handle table indexes during a bulk load.

Usage

The following table shows the values for the **Index maintenance option** property.

Value	Description
Do not skip unusable	When the connector loads rows into the table, the connector tries to maintain indexes. If an index on the table is unusable, the bulk load fails.
Skip unusable	The connector skips indexes that are unusable and maintains indexes that are usable. If the property is set to this value when the connector bulk loads into a partitioned table that has a global index defined, the bulk load fails.
Skip all	The connector skips all indexes. Any index that is usable before the load is marked unusable after the load.

Lookup properties

Use these properties to modify how the connector looks up data.

Log multiple matches property:

When the Oracle Connector stage runs in a parallel job and in lookup, it is connected with a reference link to the Lookup stage, and the Lookup stage provides support for handling multiple lookup matches. Use the **Log multiple matches** property when the Oracle Connector stage runs in a server job and in the lookup mode of operation. You can use the property to log a message when a lookup statement returns multiple matching records for the input key record.

Usage

In this mode, one or more reference links connect the Oracle Connector stage with a Transformer stage. Each input record is checked separately. Even if the lookup statement in the connector returns multiple rows, only the first row is provided by the connector on the reference link. This property controls whether to log a message if such a situation occurs.

The following table shows the values for the **Log multiple matches** property.

Value	Description
None	The connector does not log a message for multiple matches.
Informational	The connector logs a message of informational severity.
Warning	The connector logs a message of warning severity.

Value	Description
Fatal	The connector logs a message of fatal severity and stops the job.

Runtime column propagation

Use runtime column propagation to have the connector automatically add missing columns to the link schema when the job runs.

Usage

Before you can enable runtime column propagation in a stage, runtime column propagation must be enabled for parallel jobs at the project level from the InfoSphere DataStage Administrator client. To enable runtime column propagation for the output link of the stage, select the **Runtime column propagation** check box on the Columns page.

When runtime column propagation is enabled, the connector inspects at run time the columns in the result set of the query statement that it ran on the database. The connector compares those columns to the columns that are defined on the output link. Columns that are in the result set but not on the output link are added to the link. Columns that are on the output link but not in the query result set are removed from the link.

When the Oracle connector dynamically adds a column to the link at run time in a job that has runtime column propagation enabled and the link column corresponds to a LONG or LONG RAW table column in the database, the connector sets the link column length to the maximum possible value that meets both of these conditions:

- The value does not exceed 999999.
- When the value is multiplied by the value that is specified in the **Array size** property for the stage, the product does not exceed 10485760 (the number of bytes in 10 MB).

When runtime column propagation is enabled, a SELECT statement contains an SQL expression for a column name, and no alias is specified for the column, the connector automatically adds a new column to the link and specifies a column name that matches the SQL expression.

The following rules explain how the column name is derived from the SQL expression:

- Non-alphanumeric characters and underscores (_) are replaced with a pair of underscore characters.
- The dollar sign ($) is replaced with __036__.
- The number sign (#) is replaced with __035__.
- White space characters are removed.
- If any character replacement is performed, the prefix CC_*N*_ is appended to the column name, where *N* is the index of the SQL expression column in the SELECT statement list. The first column in the SELECT statement list has index 1, the second column has index 2, and so on.

Example

The following example illustrates how runtime column propagation works. Assume that runtime column propagation is enabled for the stage, that the statement `SELECT COL1, RPAD (COL2, 20, '*') FROM TABLE1` is specified in the stage, and that the output link defines two columns, COL1 and COL2. Because runtime column propagation is enabled, the connector tries to match columns only by name, not by position. The COL1 column from the SELECT statement is mapped to the COL1 column on the output link, but the SQL expression RPAD (COL2, 20, '*') is not mapped to any column on the output link. Therefore, the connector adds the following column to the link: CC_2_RPAD_COL2__20_____. In the new column name, the number 2 is used in the column name prefix because the SQL expression appears as the second column in the SELECT statement list. Each non-alphanumeric character (, ' *) is replaced by two underscore characters. The white spaces in the SQL expression are removed. Finally, the connector removes the COL2 column from the output link because that column is unmapped.

If runtime column propagation is not enabled, the connector performs matching by position. Consequently, the COL1 and COL2 columns remain on the link, and COL2 on the link represents the values of the SQL expression from the SELECT statement. If the column alias COL2 is used for the SQL expression and runtime column propagation is enabled, the mapping by name is successful, and the two existing link columns, COL1 and COL2, are used. The SELECT statement in this case is `SELECT COL1, RPAD(COL2, 20, '*') COL2 FROM TABLE1`.

Partitioned read methods

The Oracle connector supports these partitioned read methods: rowid range, rowid round robin, rowid hash, modulus, minimum and maximum range, and Oracle partitions.

For all partitioned read methods except the Oracle partitions method, the connector modifies the WHERE clause in the specified SELECT statement. If the WHERE clause is not included in the specified SELECT statement, the connector adds a WHERE clause.

For the Oracle partitions method, the connector modifies the specified SELECT statement by adding a PARTITON(partition_name) clause. When the specified SELECT statement contains subqueries, the connector modifies the first SELECT...FROM subquery in the SELECT statement.

Rowid range partitioned read method

The rowid range partitioned read method uses values from the ROWID pseudo-column to determine the rows to read. The ROWID pseudo-column, which is included in every Oracle table, contains a rowid value that uniquely identifies each row in the table.

When you use the rowid range method, the connector completes these steps:

1. The connector queries the DBA_EXTENTS dictionary view to obtain storage information about the source table.
2. The connector uses the information from the DBA_EXTENTS dictionary view to define a range of rowid values for each node.
3. At run time, each node runs the specified SELECT statement with a slightly modified WHERE clause. The modified WHERE clause ensures that the node

reads only the rows that have rowid values in its assigned range. If the specified SELECT statement does not have a WHERE clause, the connector adds it.

The connector does not support the rowid range method in these cases:

- SELECT access is not granted on the DBA_EXTENTS dictionary view for the currently connected user.
- The connector reads from an index-organized table.
- The connector reads from a view.

In these cases, the connector logs a warning message and uses the rowid hash method, which does not have these restrictions.

These are the advantages of using the rowid range method instead of using the rowid round robin method:

- The SELECT statement for each node is less complex because it does not require as many SQL functions.
- The rowid range method provides a better distribution of rows across the nodes because the distribution is based on the physical collocation of the rows.

Example of using the rowid range partitioned read method

For this example, the Oracle connector is configured in the following way:

- The **Select statement** property is set to `SELECT * FROM TABLE1 WHERE COL1 > 10`.
- The **Table name for partitioned reads** property is set to `TABLE1`.
- The connector is configured to run in parallel mode on four nodes.
- The **Partitioned reads method** property is set to **Rowid range**.

In this example, the connector calculates the rowid range for each processing node and runs a SELECT statement on each node. For each node, the SELECT statement specifies the rowid range that is assigned to that node. The SELECT statements are similar to the following statements, but the actual rowid range values will vary:

Node 1

```
SELECT * FROM TABLE1 WHERE TABLE1.ROWID BETWEEN 'AAARvrAAEAAAAVpAAA' AND
 'AAARvrAAEAAAAVuH//' AND (COL1 > 10)
```

Node 2

```
SELECT * FROM TABLE1 WHERE TABLE1.ROWID BETWEEN 'AAARvrAAEAAAAVvAAA' AND
 'AAARvrAAEAAAAV0H//' AND (COL1 > 10)
```

Node 3

```
SELECT * FROM TABLE1 WHERE TABLE1.ROWID BETWEEN 'AAARvrAAEAAAAV1AAA' AND
 'AAARvrAAEAAAAV6H//' AND (COL1 > 10)
```

Node 4

```
SELECT * FROM TABLE1 WHERE TABLE1.ROWID BETWEEN 'AAARvrAAEAAAAV7AAA' AND
 'AAARvrAAEAAAAWAH//' AND (COL1 > 10)
```

Rowid round robin partitioned read method

The rowid round robin method uses the ROWID_ROW_NUMBER function from the DBMS_ROWID package to obtain the row number of the row within the table block where the row resides. The method uses the MOD function on the row number to distribute rows evenly among the nodes.

These are the advantages of using the rowid round robin method instead of using the rowid range method:

- The currently connected user does not require SELECT access on the DBA_EXTENTS dictionary view.
- The rowid round robin method supports reading data from an index-organized table.
- The rowid round robin method supports reading data from a view. The rows in the view must correspond to the physical rows of the table. The rowid round robin method cannot read rows from a view that is derived from a join operation on two or more tables.

Example of using the rowid round robin partitioned read method

For this example, the Oracle connector is configured in the following way:

- The **Select statement** property is set to `SELECT * FROM TABLE1 WHERE COL1 > 10`.
- The **Table name for partitioned reads** property is set to `TABLE1`.
- The connector is configured to run in parallel mode on four nodes.
- The **Partitioned reads method** property is set to **Rowid round robin**.

The connector runs these SELECT statements on the nodes:

Node 1

```
SELECT * FROM TABLE1 WHERE MOD(DBMS_ROWID.ROWID_ROW_NUMBER(TABLE1.ROWID), 4) = 0 AND
 (COL1 > 10)
```

Node 2

```
SELECT * FROM TABLE1 WHERE MOD(DBMS_ROWID.ROWID_ROW_NUMBER(TABLE1.ROWID), 4) = 1 AND
 (COL1 > 10)
```

Node 3

```
SELECT * FROM TABLE1 WHERE MOD(DBMS_ROWID.ROWID_ROW_NUMBER(TABLE1.ROWID), 4) = 2 AND
 (COL1 > 10)
```

Node 4

```
SELECT * FROM TABLE1 WHERE MOD(DBMS_ROWID.ROWID_ROW_NUMBER(TABLE1.ROWID), 4) = 3 AND
 (COL1 > 10)
```

Rowid hash partitioned read method

The rowid hash method is similar to the rowid round robin method. However, instead of using the ROWID_ROW_NUMBER function to obtain the row number, the rowid hash method uses the ORA_HASH function to obtain a hash value for the rowid value of each row. Then, the rowid hash method applies the MOD function on the row number to distribute rows evenly among the nodes.

Example of using the rowid hash partitioned read method

For this example, the Oracle connector is configured in the following way:

- The **Select statement** property is set to `SELECT * FROM TABLE1 WHERE COL1 > 10`.
- The **Table name for partitioned reads** property is set to `TABLE1`.
- The connector is configured to run in parallel mode on four nodes.
- The **Partitioned reads method** property is set to **Rowid hash**.

The connector runs these SELECT statements on the nodes:

Node 1

```
SELECT * FROM TABLE1 WHERE MOD(ORA_HASH(TABLE1.ROWID), 4) = 0 AND (COL1 > 10)
```

Node 2

```
SELECT * FROM TABLE1 WHERE MOD(ORA_HASH(TABLE1.ROWID), 4) = 1 AND (COL1 > 10)
```

Node 3

```
SELECT * FROM TABLE1 WHERE MOD(ORA_HASH(TABLE1.ROWID), 4) = 2 AND (COL1 > 10)
```

Node 4

```
SELECT * FROM TABLE1 WHERE MOD(ORA_HASH(TABLE1.ROWID), 4) = 3 AND (COL1 > 10)
```

Modulus partitioned read method

When this method is selected, for each node, the connector reads the rows that satisfy the following condition: MOD(*column_name*, *number_of_nodes*) = *node_number*. In this condition, MOD is the modulus function, *column_name* is the name of the column that is specified in the **Column name for partitioned reads** property, *number_of_nodes* is the total number of nodes on which the stage runs, and *node_number* is the index of the current node.

The indexes are zero-based. Therefore, the first node has index 0, the second node has index 1, and so on.

To use this method, you must specify a column name from the input table in the **Column name for partitioned reads** property. The column that you specify must be of the data type NUMBER(*p*), where *p* is a value in the range 1 - 38. The specified column must exist in the table that is specified in the **Table name for partitioned reads** property, the **Table name** property, or the **Select statement** property. The value for the **Select statement** property is used only if you do not explicitly specify the table name in one of the other two properties.

Example of using the modulus partitioned read method

For this example, the Oracle connector is configured in the following way:

- The **Select statement** property is set to `SELECT * FROM TABLE1 WHERE COL1 > 10`.
- The **Table name for partitioned reads** property is set to `TABLE1`.
- The connector is configured to run in parallel mode on four nodes.
- The **Partitioned reads method** property is set to **Modulus**.
- The **Column name for partitioned reads** property is set to `COL2`, and COL2 is defined as NUMBER(5) in TABLE1.

The connector runs the following SELECT statements on the nodes:

Node 1

```
SELECT * FROM TABLE1 WHERE MOD(TABLE1.COL2, 4) = 0 AND (COL1 > 10)
```

Node 2

```
SELECT * FROM TABLE1 WHERE MOD(TABLE1.COL2, 4) = 1 AND (COL1 > 10)
```

Node 3

```
SELECT * FROM TABLE1 WHERE MOD(TABLE1.COL2, 4) = 2 AND (COL1 > 10)
```

Node 4

```
SELECT * FROM TABLE1 WHERE MOD(TABLE1.COL2, 4) = 3 AND (COL1 > 10)
```

Minimum and maximum range partitioned read method

When this method is specified, the connector calculates the minimum and maximum value for the specified column and then divides the calculated range into subranges. Each subrange is then assigned to a node; the number of subranges equals the number of nodes that are configured for the stage. On each node, the connector runs a SELECT statement that returns the rows where the value in the specified column is in the subrange that was assigned to that node.

To use this method, you must specify a column name from the input table in the **Column name for partitioned reads** property. The column that you specify must be of the data type NUMBER(*p*), where *p* is a value in the range 1 - 38. The specified column must exist in the table that is specified in the **Table name for partitioned reads** property, the **Table name** property, or the **Select statement** property. The value for the **Select statement** property is used only if you do not explicitly specify the table name in one of the other two properties.

Example of using the minimum and maximum range partitioned read method

For this example, the Oracle connector is configured in the following way::

- The **Select statement** property is set to `SELECT * FROM TABLE1 WHERE COL1 > 10`.
- The **Table name for partitioned reads** property is set to `TABLE1`.
- The connector is configured to run in parallel mode on four nodes.
- The **Partitioned reads method** property is set to **Minimum and maximum range**.
- The **Column name for partitioned reads** property is set to `COL2`, and COL2 is defined as NUMBER(5) in TABLE1.

The connector determines the minimum and maximum value for column COL2. If the minimum value is -20 and maximum value is 135, the connector runs the following SELECT statements on the nodes:

Node 1

```
SELECT * FROM TABLE1 WHERE TABLE1.COL2 <= 18 AND (COL1 > 10)
```

Node 2

```
SELECT * FROM TABLE1 WHERE TABLE1.COL2 BETWEEN 19 AND 57 AND (COL1 > 10)
```

Node 3

```
SELECT * FROM TABLE1 WHERE TABLE1.COL2 BETWEEN 58 AND 96 AND (COL1 > 10)
```

Node 4

```
SELECT * FROM TABLE1 WHERE TABLE1.COL2 >= 97 AND (COL1 > 10)
```

Oracle partitions partitioned read method

When this method is specified, the connector determines the number of partitions in the table and dynamically configures the number of nodes to match the number of table partitions. The connector associates each node with one table partition. For each node, the connector reads the rows that belong to the partition that is associated with that node.

To perform this operation, the connector adds the PARTITION(*partition_name*) clause to the SELECT statement where *partition_name* is the name of the partition

that is associated with the current node. Consequently, when you specify a value for the **Select statement** property, do not include a PARTITION or SUBPARTITION clause.

The connector can dynamically adjust the number of nodes on which it runs. However, for this process to work, do not use the Advanced page of the Stage window to constrain the node configuration at design time. If the node configuration is constrained at design time and the resulting number of nodes does not match the number of partitions in the table, the connector returns an error and the job fails.

Example of using the Oracle partitions partitioned read method

For this example, the Oracle connector is configured in the following way:

- The **Select statement** property is set to `SELECT * FROM TABLE1 WHERE COL1 > 10`.
- The **Table name for partitioned reads** property is set to `TABLE1`.
- The connector is configured to run in parallel mode on five nodes.
- The **Partitioned reads method** property is set to **Oracle partitions**.
- TABLE1 has four partitions:

```
CREATE TABLE TABLE1
(
  COL1 NUMBER(10),
  COL2 DATE
)
PARTITION BY RANGE (COL2)
(
  PARTITION PART1 VALUES LESS THAN (TO_DATE('01-JAN-2006','DD-MON-YYYY')),
  PARTITION PART2 VALUES LESS THAN (TO_DATE('01-JAN-2007','DD-MON-YYYY')),
  PARTITION PART3 VALUES LESS THAN (TO_DATE('01-JAN-2008','DD-MON-YYYY')),
  PARTITION PART4 VALUES LESS THAN (MAXVALUE)
);
```

The connector determines that TABLE1 has four partitions: PART1, PART2, PART3, and PART4. The connector concludes that the stage must run on four processing nodes. Because the stage was configured to run on five nodes, the connector removes the fifth node from the list of nodes and logs an informational message to indicate that the list of nodes was adjusted and that the stage will run on four nodes.

The connector runs the following SELECT statements on the nodes:

Node 1

```
SELECT * FROM TABLE1 PARTITION(PART1) WHERE COL1 > 10
```

Node 2

```
SELECT * FROM TABLE1 PARTITION(PART2) WHERE COL1 > 10
```

Node 3

```
SELECT * FROM TABLE1 PARTITION(PART3) WHERE COL1 > 10
```

Node 4

```
SELECT * FROM TABLE1 PARTITION(PART4) WHERE COL1 > 10
```

Oracle connector partition type

For writes to a range-partitioned, list-partitioned or interval-partitioned table, the Oracle connector partition type ensures that the distribution of input records matches the organization of the partitions in the table.

When the Oracle connector partition type is selected, the connector first gets the partitioning information for the table. In most cases, the connector uses the partitioning information from the table to which the connector writes the data; the name of this table is usually specified in the **Table name** property or is implicitly specified in the INSERT, UPDATE, or DELETE SQL statement. To configure the connector to use the partitioning information from one table but write the data to a different table, you specify the table name in the **Table name for partitioned writes** property.

After the connector determines the table name for the partitioned write, the connector determines the set of nodes on which to run. The connector determines the number of partitions that are on the table and associates one node with each partition. The number of partitions must match the number of nodes. A mismatch between the number of nodes and the number of partitions can occur in the following situations:

- The configuration of the parallel processing nodes specifies a resource constraint. If the configuration specifies a constraint, the connector cannot dynamically modify the set of processing nodes. As a result, the connector reports an error and stops the operation.
- The list of nodes that are configured for the stage contains more nodes than the number of partitions in the table. In this case, the connector removes the excess nodes from the end of the list.
- The list of nodes that are configured for the stage contains fewer nodes than the number of partitions in the table. In this case, the connector adds nodes to the end of the list. The definition for each node that is added matches the definition of the last node in the original list.

Next, the connector determines the node to send each input record to. For each incoming record, the connector inspects the data in the fields that correspond to the table columns that constitute the partition key for the table. The connector compares those values to the boundary values that are specified for the individual partitions of the table and determines the partition that will store the records. Because the number of nodes matches the number of partitions and each partition has only one node assigned to it, the connector routes the records to the node that is associated with each partition, and the node writes the records into the database.

For the connector to determine both the number of partitions in a table and the partitioning type that was used to partition the table, the table must exist in the database before you run the job. The only exception to this rule is when the **Table action** property is set to **Create** or **Replace** and the **Create statement** property specifies a CREATE TABLE statement. In this case, the connector analyzes the CREATE TABLE statement to determine the number of partitions and the partition type that the table will have when it is created at run time. The connector uses this information to determine the number of nodes that the stage will run on.

Conditions that cause the stage to run in sequential mode, report errors, or both

If the table uses a supported partition type but the partition key in the table includes a virtual column, the connector does not force sequential execution.

Instead, the connector runs on the number of nodes that is equal to the number of table partitions. However, because only one node processes the data, the connector effectively runs in sequential mode.

If the **Table action** property is set to **Create** or **Replace** and the **Generate create statement at runtime** property is set to **Yes**, the connector does not create the table as a partitioned table. Therefore, the connector cannot associate the table partitions with the nodes. In this case, the connector logs a warning and runs the stage in sequential mode.

If the table does not exist and the **Before SQL statement** property or the **Before SQL (node) statement** property specifies the CREATE TABLE statement, the connector reports an error. The error is reported because the connector tries to determine the number of partitions and the partition type before it runs the before SQL statement that creates the table.

When the **Table scope** property is set to **Single partition** or **Single subpartition**, the connector runs the stage in sequential mode and logs a warning. In this case, the connector is explicitly configured to write data to only one partition or subpartition; therefore, only one node is assigned to that partition or subpartition.

Support for standard Oracle partition types

When you use the Oracle connector partition type in the Oracle Connector stage, you can write to range-partitioned, list-partitioned, or interval-partitioned tables on Oracle databases.

The following table shows the standard Oracle partition types that are supported by the Oracle connector partition type. The table also describes the actions that the Oracle connector takes when the Oracle connector partition type is selected and the connector writes data to tables that are partitioned in each way.

Table 17. Oracle partition types that are supported and unsupported when you use the Oracle connector partition type in the Oracle Connector stage

Oracle partition type	Support	Actions that the connector takes
• Range • Composite range-range • Composite range-list • Composite range-hash	Supported	The connector inspects the values of the record fields that correspond to the partition key columns, determines the partition to which the record belongs, and redirects the record to the node that is associated with that table partition.
• List • Composite list-range • Composite list-list • Composite list-hash	Supported	The connector inspects the value of the record that corresponds to the partition key column, determines the partition to which the record belongs, and redirects the record to the node that is associated with that table partition.
Hash	Unsupported	The connector runs the stage in sequential mode and logs a warning message.

Table 17. Oracle partition types that are supported and unsupported when you use the Oracle connector partition type in the Oracle Connector stage (continued)

Oracle partition type	Support	Actions that the connector takes
• Interval • Composite interval-range • Composite interval-list • Composite interval-hash	Supported	The connector inspects the value of the record that corresponds to the partition key column and determines the partition to which the record belongs. If the record belongs to one of the partitions that existed when the job started, the connector redirects the record to the node that is associated with that table partition. Otherwise, the connector redirects the record to a special node that is reserved for loading records into partitions that are new and were created dynamically.
Reference	Unsupported	The connector runs the stage in sequential mode and logs a warning message.
Virtual	Unsupported	The connector runs the stage in sequential mode and logs a warning message.
System	Unsupported	The connector runs the stage in sequential mode and logs a warning message.

Supported write methods

When you configure the Oracle connector as a target, you can use the supported write methods to write rows to an Oracle table or writable view.

The following table lists the write modes and describes the operations that the connector completes on the target table for each write mode.

Table 18. Write modes and descriptions

Write mode	Description
Insert	The connector attempts to insert records from the input link as rows into the target table.
Update	The connector attempts to update rows in the target table that correspond to the records that arrive on the input link. Matching records are identified by the values that correspond to link columns that are marked as key columns.

Table 18. Write modes and descriptions (continued)

Write mode	Description
Delete	The connector attempts to delete rows in the target table that correspond to the records that arrive on the input link. Matching records are identified by the values that correspond to link columns that are marked as key columns.
Insert new rows only	The behaviour of this write mode is very similar to the Insert write mode. However, when this write mode is selected, records that cannot be written to the database because of a primary key or unique constraint are ignored, and the connector processes the remaining records. When any error other than a primary key or unique constraint violation occurs, the connector still logs a fatal error and stops the job.
Insert then update	For each input record, the connector first tries to insert the record as a new row in the target table. If the insert operation fails because of a primary key or unique constraint, the connector updates the existing row in the target table with the new values from the input record.
Update then insert	For each input record, the connector first tries to locate the matching rows in the target table and to update them with the new values from the input record. If the rows cannot be located, the connector inserts the record as a new row in the target table.
Delete then insert	For each input record, the connector first tries to delete the matching rows in the target table. Regardless of whether rows were actually deleted or not, the connector then runs the insert statement to insert the record as a new row in the target table.
PL/SQL block	For each input record, the connector runs the specified PL/SQL block.
Bulk load	The connector uses the Oracle direct path load method to bulk load data.

Reject conditions

When you use the Oracle connector as a target in a job, you can add a reject link and send rejected records to a target stage. Reject conditions determine when a record is rejected.

You can set the following reject conditions:

Row not updated - update mode

The connector checks for this condition only when the **Write mode** property is set to **Update**. The connector attempts to update a row in the target table and the operation succeeds, but the database reports that zero

rows were updated. For example, if key field values in the input record do not match key column values of any row in the target table, this reject condition is satisfied.

This condition does not have a corresponding Oracle error code and error message.

Row not updated - insert then update mode

The connector checks for this condition only when the **Write mode** property is set to **Insert then Update**. The connector attempts to update a row in the target table and the operation succeeds, but the database reports that zero rows were updated.

For example, the following situation results in a row not being updated. Suppose that the key field values in the input record do not match the key column values of any row in the target table. However, field values for one or more of the remaining fields violate the unique or primary key constraint in the table. In this case, the INSERT statement fails because of the constraint violation. Also, the UPDATE statement does not update any rows because no matching rows in the table meet the condition that is specified in the WHERE clause of the UPDATE statement.

This condition does not have a corresponding Oracle error code and error message.

Row not deleted

The connector checks for this condition only when the **Write mode** property is set to **Delete**. The connector attempts to delete a row in the target table and the operation succeeds, but the database reports that no data was deleted. This situation can occur when the key field values in the input record do not match the key column values of any row in the target table.

This condition does not have a corresponding Oracle error code and error message.

SQL error – constraint check

This condition occurs when an operation cannot be completed because of a constraint check. In some situations, this SQL error does not result in a record being sent to the reject link. For example, when the **Write mode** property is set to **Insert then update** and the insert operation fails because of a primary key constraint, the connector attempts to update the row, rather than send the record to the reject link. However, if the update operation fails for one of the selected reject conditions, the connector sends the input record to the reject link.

SQL error – type mismatch

This condition occurs when a data value in the record is not compatible with the data type of the corresponding column in the target table. In this case, Oracle cannot convert the data and returns an error.

SQL error – data truncation

This condition occurs when the data types of the columns on the link are compatible with the column data types in the target table, but data is lost because of a size mismatch.

SQL error – character set conversion

This condition occurs when the record contains Unicode data in one or more of NChar, NVarChar or LongNVarChar columns and conversion errors happen when that data is converted to the database character set that is specified by the **NLS_CHARACTERSET** database parameter.

SQL error – partitioning
This condition occurs when the connector tries to write a record to a particular partition in the partitioned table, but the specified partition is not the partition to which the record belongs.

SQL error – XML processing
This condition occurs when a record that contains an XML data document cannot be inserted into an XMLType column in a table because the XML data contains errors. For example, if the specified XML document is not well-formed or if the document is invalid in relation to its XML schema, this error condition occurs.

SQL error – other
This condition covers all SQL errors that are not covered explicitly by one of the other reject conditions.

White space characters, NULL values, and empty string values

When the Oracle connector reads data from a database or writes data to a database, the connector always preserves white space characters such as SPACE, TAB, CR (carriage return), and LF (line feed). In addition, the connector does not trim leading or trailing white space characters from text values unless the **Preserve trailing blanks** property is set to **No**.

The Oracle database does not support empty string values in text columns. Instead, the Oracle database treats these values as NULL values.

Before writing values into fixed-size text columns, the Oracle database pads all non-empty values with space characters.

For example, assume that you use the following statement to create a target table named TABLE1 and configure the connector to insert or bulk load data into this table:

```
CREATE TABLE TABLE1 (COL1 VARCHAR2(10) NULL, COL2 CHAR(3) NULL);
```

The following table shows the input data for the COL1 and COL2 columns and the corresponding values that are stored in TABLE1. In the table, an en dash (-) represents a space character.

Table 19. Example input column values and corresponding table values that are stored in the database

Column values	Table values
"VAL1-1-", "V1-"	"VAL1-1-", "V1-"
"V2--", "2-"	"V2--", "2--"
"-", "-"	"-", "---"
"3", NULL	"3", NULL
NULL, "4"	NULL, "4--"
"", ""	NULL, NULL
NULL, NULL	NULL, NULL

Dictionary views

To complete specific tasks, the Oracle connector requires access to a set of Oracle dictionary views.

The following table describes how the Oracle connector uses each dictionary view.

Table 20. How the Oracle connector uses Oracle dictionary views

Dictionary view	Use	Required for these tasks
ALL_CONSTRAINTS	Obtain the list of constraints for a table	• Importing a table definition • Enabling and disabling constraints
ALL_INDEXES	Obtain the list of indexes for a table	• Importing a table definition • Determining the list of indexes to rebuild • Determining how a table is organized, either as a heap table or by index
ALL_OBJECTS	Obtain additional metadata, such as table names and view names, for the objects that you specify	Depends on the objects that you specify. For example, for a parallel read that is based on Oracle partitions, the connector accesses this view to determine the object type, either table or view, and the partitions and subpartitions.
ALL_PART_COL_ STATISTICS	Determine the boundary (high) value for each partition in a table	Writing to a partitioned table
ALL_PART_KEY_COLUMNS	Determine the list of columns that are in the partition key for a table	Writing to a partitioned table
ALL_PART_TABLES	Determine the partitioning method that the table uses. When the Oracle connector partition type is selected, the Oracle connector uses the information from this view to determine the partition to which each record belongs and then to direct each record to the node that is associated with that partition.	Writing to a partitioned table
ALL_TAB_COLS	• Determine column metadata such as data type, length, precision, and scale to determine if a column is a virtual column • Determine if a column exists and if it is of the correct data type when the **Modulus** or the **Minimum and Maximum range** partitioned read method is specified	Completing actions on a partitioned table

Table 20. How the Oracle connector uses Oracle dictionary views (continued)

Dictionary view	Use	Required for these tasks
ALL_TAB_PARTITIONS	Determine the number and names of the partitions in a partitioned table	Completing actions on a partitioned table
ALL_TAB_SUBPARTITIONS	Determine the number and names of all subpartitions in a composite-partitioned table	Completing actions on a partitioned table
ALL_TABLES	Determine the list of tables that are accessible by the current user	• Importing a table definition • Identifying the users that have tables with the SYSTEM or SYSAUX table space as the default table space for the user • Determining if a specified table is partitioned
ALL_VIEWS	Determine the views that are accessible by the current user	Identifying the views that you can import
ALL_XML_TAB_COLS	Determine the XML storage option that was specified in the column definitions	Importing metadata for tables that contain XMLType columns
ALL_XML_TABLES	Determine the XML storage option that was specified in the table definitions	Importing metadata for tables that contain XMLType columns
DBA_EXTENTS	Gather information about the table storage organization	Reading from partitioned tables by using the rowid range partitioned read method. If select access is not granted to this view, the connector automatically switches to the rowid hash partitioned read method.
DUAL	Obtain and calculate various intermediate values that the connector needs for its operation	Completing actions on a table
USER_TAB_PRIVS	Determine if the current user was granted select privilege on a particular dictionary view such as the DBA_EXTENTS view. If the current user was not granted select privilege, the connector takes corrective action.	Accessing a dictionary view

Exceptions table

If you configure the connector to enable constraints after bulk loading data, the connector stores the ROWID values for any rows that violate the constraints in an exceptions table.

The format of the exceptions table is specified in the `utlexcpt.sql` and `utlexcpt1.sql` scripts, which are in the Oracle installation directory. For example,

for installations on Microsoft Windows, the scripts are in the %ORACLE_HOME%\RDBMS\ADMIN directory. The utlexcpt.sql script defines the format for exceptions tables that accept the physical ROWID values that conventional tables use. The utlexcpt1.sql script defines the format for exceptions tables that accept the universal ROWID (UROWID) values that both conventional and index-organized tables use.

When a database already has an exceptions table, the table must use the format that is specified in the script that corresponds to the type of the target table; otherwise, the connector reports a fatal error about the table format, and the job stops.

If you do not specify an exceptions table, the following actions occur:

- The connector tries to enable the constraint. The operation fails if the table contains rows that violate the constraint.
- The connector cannot be configured to automatically delete the rows that violate the constraint.
- If you define a reject link and select the **SQL Error - constraint violation** condition for the reject link, the job fails, and the message IIS-CONN-ORA-001058 is written to the job log, indicating that an exceptions table is required.

Environment variables that the Oracle connector uses

In addition to the environment variables that affect how the Oracle connector operates, the Oracle connector queries and uses Oracle environment variables and environment variables for the local operating system.

Library path

This variable must include the directory where the Oracle client libraries are stored. The following table lists the name of the library path variable for each operating system.

Operating system	Name of the library path variable
HP-UX	**LD_LIBRARY_PATH** or **SHLIB_PATH**
IBM AIX	**LIBPATH**
Linux	**LD_LIBRARY_PATH**
Microsoft Windows	**PATH**

LOCAL

This Oracle environment variable specifies the default remote Oracle service. When this variable is defined, the connector connects to the specified database by using an Oracle listener that accepts connection requests. This variable is for use on Microsoft Windows only. Use the **TWO_TASK** environment variable for Linux and UNIX.

ORACLE_HOME

This Oracle environment variable specifies the location of the home directory of the Oracle client installation. The connector uses the variable to locate the tnsnames.ora configuration file, which is required to make a connection to an Oracle database. The connector looks for the tnsnames.ora file in the ORACLE_HOME/network/admin directory.

ORACLE_SID

This Oracle environment variable specifies the default local Oracle service. When this variable is defined, the connector connects to the specified

database and does not use an Oracle listener. On Microsoft Windows, you can specify this environment variable in the Windows registry.

If both **ORACLE_SID** and **TWO_TASK** or **LOCAL** are defined, **TWO_TASK** or **LOCAL** takes precedence.

TWO_TASK
This Oracle environment variable specifies the default remote Oracle service. When this variable is defined, the connector connects to the specified database by using an Oracle listener that accepts connection requests. This variable is for use on Linux and UNIX only. Use the **LOCAL** environment variable for Microsoft Windows.

If both **ORACLE_SID** and **TWO_TASK** or **LOCAL** are defined, **TWO_TASK** or **LOCAL** takes precedence.

TNS_ADMIN
This Oracle environment variable specifies the location of the directory that contains the `tnsnames.ora` configuration file. When this variable is specified, it takes precedence over the value of the **ORACLE_HOME** environment variable when the Oracle connector tries to locate the configuration file. The connector looks for the `tnsnames.ora` file directly under the `TNS_ADMIN` directory.

Chapter 4. Oracle Enterprise stage

The Oracle Enterprise stage is a database stage that you can use to read data from and write data to an Oracle database. It can also be used in conjunction with a Lookup stage to access a lookup table that is hosted by an Oracle database.

When you use IBM InfoSphere DataStage to access Oracle databases, you can choose from a collection of connectivity options. For most new jobs, use the Oracle Connector stage, which offers better functionality and performance than the Oracle Enterprise stage.

If you have jobs that use the Oracle Enterprise stage and want to use the connector, use the Connector Migration Tool to migrate jobs to use the connector.

The Oracle Enterprise stage can have a single input link and a single reject link, or a single output link or output reference link.

The stage performs one of the following operations:

- Updates an Oracle table using INSERT or UPDATE or both as appropriate. Data is assembled into arrays and written using Oracle host-array processing.
- Loads an Oracle table (by using Oracle fast loader).
- Reads an Oracle table.
- Deletes rows from an Oracle table.
- Performs a lookup directly on an Oracle table.
- Loads an Oracle table into memory and then performs a lookup on it.

When you use an Oracle stage as a source for lookup data, there are special considerations about column naming. If you have columns of the same name in both the source and lookup data sets, note that the source data set column will go to the output data. If you want this column to be replaced by the column from the lookup data source, you need to drop the source data column before you perform the lookup (you can, for example, use a Modify stage to do this).

For more information about performing lookups, see the *IBM InfoSphere DataStage and QualityStage Parallel Job Developer's Guide*.

When you edit a Oracle Enterprise stage, the Oracle Enterprise stage editor appears.

The stage editor has up to three pages, depending on whether you are reading or writing a database:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Inputs Page**. This is present when you are writing to a Oracle database. This is where you specify details about the data being written.
- **Outputs Page**. This is present when you are reading from a Oracle database, or performing a lookup on an Oracle database. This is where you specify details about the data being read.

Note: For Oracle direct path load, the client version must be the same as or earlier than the server version. You should have read and execute permissions to use

libraries in the $ORACLE_HOME/lib and $ORACLE_HOME/bin directories and read permissions on all files in the $ORACLE_HOME directory. Otherwise, you might experience problems using Oracle enterprise stage to connect to Oracle.

Accessing Oracle databases

Before you can use the Oracle Enterprise stage, you must set values for environment variables and ensure that you have the roles and privileges that are required.

Before you begin

Install the Oracle standard client. You cannot use the stage if only Oracle Instant Client is installed.

Procedure

1. Create the user defined environment variable ORACLE_HOME and set this to the $ORACLE_HOME path (for example, /disk3/oracle10g).
2. Add *ORACLE_HOME/bin* to your PATH and *ORACLE_HOME/lib* to your LIBPATH, LD_LIBRARY_PATH, or SHLIB_PATH.
3. Have login privileges to Oracle using a valid Oracle user name and corresponding password. These must be recognized by Oracle before you attempt to access it.
4. Have SELECT privilege on:
   - DBA_EXTENTS
   - DBA_DATA_FILES
   - DBA_TAB_PARTITONS
   - DBA_TAB_SUBPARTITONS
   - DBA_OBJECTS
   - ALL_PART_INDEXES
   - ALL_PART_TABLES
   - ALL_INDEXES
   - SYS.GV_$INSTANCE (Only if Oracle Parallel Server is used)

   Note: *APT_ORCHHOME/bin* must appear before *ORACLE_HOME/bin* in your PATH.

   You can create a role that has the appropriate SELECT privileges, as follows:

   ```
   CREATE ROLE DSXE;
   GRANT SELECT on sys.dba_extents to DSXE;
   GRANT SELECT on sys.dba_data_files to DSXE;
   GRANT SELECT on sys.dba_tab_partitions to DSXE;
   GRANT SELECT on sys.dba_tab_subpartitions to DSXE;
   GRANT SELECT on sys.dba_objects to DSXE;
   GRANT SELECT on sys.all_part_indexes to DSXE;
   GRANT SELECT on sys.all_part_tables to DSXE;
   GRANT SELECT on sys.all_indexes to DSXE;
   ```

   Once the role is created, grant it to users who will run the IBM InfoSphere DataStage and QualityStage jobs, as follows:

   GRANT DSXE to <*oracle userid*>;

Handling special characters (# and $)

The number sign (#) and dollar sign ($) characters are reserved in IBM InfoSphere DataStage. If you connect to Oracle databases that use these characters in column names, you must complete steps to ensure that the characters are handled correctly.

About this task

InfoSphere DataStage converts these characters into an internal format, then converts them back as necessary.

To take advantage of this facility, you need to perform the following task:

- Avoid using the strings __035__ and __036__ in your Oracle column names. __035__ is the internal representation of # and __036__ is the internal representation of $.

When you use this feature in your job, import metadata by using the Plug-in Meta Data Import tool, and avoid entering metadata manually.

After the table definition is loaded, the internal column names are displayed rather than the original Oracle names both in table definitions and in the Data Browser. They are also used in derivations and expressions. Because the original names are used in generated SQL statements, however, use them if you enter SQL in the job manually.

Generally, in the Oracle Enterprise stage, enter external names everywhere except when you refer to stage column names, where you use names in the form ORCHESTRATE.*internal_name*.

When using the Oracle Enterprise stage as a target, enter external names as follows:

- For Load options, use external names for select list properties.
- For Upsert option, for update and insert, use external names when referring to Oracle table column names, and internal names when referring to the stage column names. For example:

```
INSERT INTO tablename (A#, B$#) VALUES
(ORCHESTRATE.A__036__A__035__, ORCHESTRATE.B__035__035__B__036__)

UPDATE tablename SET B$# = ORCHESTRATE.B__035__035__B__036__ WHERE (A# =
ORCHESTRATE.A__036__A__035__)
```

When you use the Oracle Enterprise stage as a source, enter external names as follows:

- For Read using the user-defined SQL method, use external names for Oracle columns for SELECT: For example:

```
SELECT M#$, D#$ FROM tablename WHERE (M#$ > 5)
```

- For Read using Table method, use external names in select list and where properties.

When you use the Oracle Enterprise stage in parallel jobs to look up data, enter external or internal names as follows:

- For lookups that use the user-defined SQL method, use external names for Oracle columns for SELECT, and for Oracle columns in any WHERE clause you might add. Use internal names when referring to the stage column names in the WHERE clause. For example:

```
SELECT M$##, D#$ FROM tablename
WHERE (B$# = ORCHESTRATE.B__035__ B __036__)
```

- For lookups that use the Table method, use external names in select list and where properties.
- Use internal names for the key option on the Inputs page **Properties** tab of the Lookup stage to which the Oracle Enterprise stage is attached.

Loading tables

If you use the Load method (which uses the Oracle SQL*Loader utility) to load tables with indexes, you must specify a value for the **Index Mode** property.

By default, the stage sets the following options in the Oracle load control file:

- DIRECT=TRUE
- PARALLEL = TRUE

This action causes the load to run using parallel direct load mode. To use the parallel direct mode load, the table must not have indexes, or you must set the **Index Mode** property to **Rebuild** or **Maintenance**. If the only index on the table is from a primary key or unique key constraint, you can instead use the **Disable Constraints** property, which disables the primary key or unique key constraint and enable it again after the load.

If you set the **Index Mode** property to **Rebuild**, the following options are set in the file:

- SKIP_INDEX_MAINTENANCE=YES
- PARALLEL=TRUE

If you set the **Index Mode** property to **Maintenance**, the following option is set in the file:

- PARALLEL=FALSE

You can use the environment variable **APT_ORACLE_LOAD_OPTIONS** to control the options that are included in the Oracle load control file. You can load a table with indexes without using the **Index Mode** or **Disable Constraints** properties by setting the **APT_ORACLE_LOAD_OPTIONS** environment variable appropriately. You need to set the Direct option or the PARALLEL option or both to FALSE, for example:

```
APT_ORACLE_LOAD_OPTIONS='OPTIONS(DIRECT=FALSE,PARALLEL=TRUE)'
```

In this example, the stage runs in parallel, however, since DIRECT is set to FALSE, the conventional path mode rather than the direct path mode would be used.

If **APT_ORACLE_LOAD_OPTIONS** is used to set PARALLEL to FALSE, then you must set the execution mode of the stage to run sequentially on the **Advanced** tab of the Stage page.

If loading index-organized tables (IOTs), do not set both DIRECT and PARALLEL to true as direct parallel path load is not allowed for IOTs.

Data type conversion for writing to Oracle

When the Oracle Enterprise stage writes or loads data, it automatically converts the IBM InfoSphere DataStage data types to Oracle data types.

The data types are shown in the following table:

Table 21. Data type conversion for writing data to an Oracle database

InfoSphere DataStage SQL data type	**Underlying data type**	**Oracle data type**
Date	date	DATE
Time	time	DATE (does not support microsecond resolution)
Timestamp	timestamp	DATE (does not support microsecond resolution)
Timestamp with Extended=Microseconds	timestamp[microseconds]	TIMESTAMP (6)
Decimal Numeric	decimal (*p*, *s*)	NUMBER (*p*, *s*)
TinyInt	int8	NUMBER (3, 0)
TinyInt with Extended=Unsigned	uint8	NUMBER (3, 0)
SmallInt	int16	NUMBER (5, 0)
SmallInt with Extended=Unsigned	uint16	NUMBER (5, 0)
Integer	int32	NUMBER (10, 0)
Integer with Extended=Unsigned	uint32	NUMBER (10, 0)
BigInt	int64	NUMBER (19)
BigInt with Extended=Unsigned	uint64	NUMBER (20)
Float Real	sfloat	BINARY_FLOAT
Double	dfloat	BINARY_DOUBLE
Binary with Length undefined	raw	RAW (2000)
VarBinary with Length undefined LongVarBinary with Length undefined	raw[]	RAW (2000)
Binary with Length=n	raw[n]	RAW (n)
VarBinary with Length=n LongVarBinary with Length=n	raw[max=n]	RAW(n)
Char with Extended undefined and Length undefined	string	CHAR (32)
NChar with Length undefined Char with Extended=Unicode and Length undefined	ustring	NVARCHAR (32)
Char with Extended undefined and Length=n	string[n]	CHAR (n)
NChar with Length=n Char with Extened=Unicode and Length=n	ustring[n]	NCHAR (n)

Table 21. Data type conversion for writing data to an Oracle database (continued)

InfoSphere DataStage SQL data type	Underlying data type	Oracle data type
Bit	uint16	NUMBER (5)
Unknown	fixed-length string in the form string[*n*] and ustring[*n*]; length <= 255 bytes	NVARCHAR(*32*)
LongVarChar with Extended undefined and Length undefined VarChar with Extended undefined and Length undefined	string[]	VARCHAR2 (32)
NVarChar with Length undefined LongNVarChar with Length undefined LongVarChar with Extended=Unicode and Length undefined VarChar with Extended=Unicode and Length undefined	ustring[]	NVARCHAR2 (32)
LongVarChar with Extended undefined and Length=n VarChar with Extended undefined and Length=n	string[max=n]	VARCHAR2 (n)
NVarChar with Length=n LongNVarChar with Length=n LongVarChar with Extended=Unicode and Length=n VarChar with Extended=Unicode and Length=n	ustring[max=n]	NVARCHAR2 (n)

The default length of VARCHAR is 32 bytes. That is, 32 bytes are allocated for each variable-length string field in the input data set. If an input variable-length string field is longer than 32 bytes, the stage issues a warning.

Data type conversion for reading from Oracle

When the Oracle Enterprise stage reads data, it automatically converts Oracle data types to the IBM InfoSphere DataStage data types.

The data types are shown in the following table:

Table 22. Data type conversion for reading data from an Oracle database

InfoSphere DataStage SQL data type	Underlying data type	Oracle data type
Unknown Char LongVarChar VarChar NChar NVarChar LongNVarChar	string[*n*] or ustring[*n*] Fixed length string with length = *n*	CHAR(*n*)
Unknown Char LongVarChar VarChar NChar NVarChar LongNVarChar	string[max = *n*] or ustring[max = *n*] variable length string with length = *n*	VARCHAR(*n*)
Timestamp	Timestamp	DATE
Decimal Numeric	decimal (38,10)	NUMBER
Integer Decimal Numeric	int32 if precision (p) <11 and scale (s) = 0 decimal[p, s] if precision (p) =>11 and scale (s) > 0	NUMBER(*p, s*)
not supported	not supported	• LONG • CLOB • NCLOB • BLOB • INTERVAL YEAR TO MONTH • INTERVAL MONTH TO DAY • BFILE

Examples

These examples describe how the Oracle Enterprise stage looks up data from or updates data in an Oracle table.

Looking up an Oracle table

In this example, the Oracle Enterprise stage looks up data in an Oracle table. The stage looks up the interest rate for each customer based on the account type.

Here is the data that arrives on the primary link:

Table 23. Example of Looking up an Oracle table

Customer	accountNo	accountType	balance
Latimer	7125678	plat	7890.76
Ridley	7238892	flexi	234.88
Cranmer	7611236	gold	1288.00

Table 23. Example of Looking up an Oracle table (continued)

Customer	accountNo	accountType	balance
Hooper	7176672	flexi	3456.99
Moore	7146789	gold	424.76

Here is the data in the Oracle lookup table:

Table 24. Example of Looking up an Oracle table

accountType	InterestRate
bronze	1.25
silver	1.50
gold	1.75
plat	2.00
flexi	1.88
fixterm	3.00

Here is what the lookup stage will output:

Table 25. Example of Looking up an Oracle table

Customer	accountNo	accountType	balance	InterestRate
Latimer	7125678	plat	7890.76	2.00
Ridley	7238892	flexi	234.88	1.88
Cranmer	7611236	gold	1288.00	1.75
Hooper	7176672	flexi	3456.99	1.88
Moore	7146789	gold	424.76	1.75

The job is illustrated in the following figure. The stage editor that you use to edit this stage is based on the generic stage editor. The Data_set stage provides the primary input, the Oracle_8 stage provides the lookup data, Lookup_1 performs the lookup and outputs the resulting data to Data_Set_3. In the Oracle stage, specify that you are going to look up the data directly in the Oracle database, and the name of the table you are going to lookup. In the Lookup stage, you specify the column that you are using as the key for the lookup.

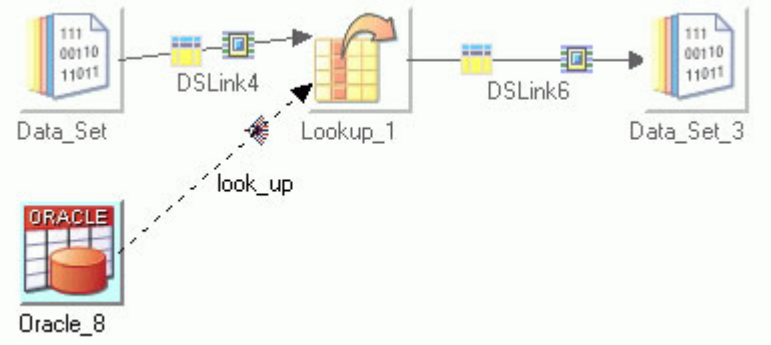


Figure 4. Example look up job

The properties for the Oracle stage are given in the following table:

Table 26. Properties for Oracle stage

Property name	Setting
Lookup Type	Sparse
Read Method	Table
Table	interest

Updating an Oracle table

In this example, the Oracle Enterprise stage updates an Oracle table with three new columns.

The database records the horse health records of a large stud. Details of the worming records are added to the main table and populated with the most recent data by using the existing column "name" as a key. The metadata for the new columns is as follows:

Table 27. Column metadata on the Properties tab

Column name	Key	SQL type	Extended	Length	Scale	Nullable
name	Yes	Char				No
wormer_type		Char	Unicode			No
dose_interval		Char	Unicode			No
dose_level		Char	Unicode			No

Specify upsert as the write method and select User-defined Update & Insert as the upsert mode. The existing name column is not included in the INSERT statement. The properties (showing the INSERT statement) are shown below. The INSERT statement is as generated by the IBM InfoSphere DataStage, except the name column is removed.

```
INSERT
INTO
horse_health
(wormer_type, dose_interval, dose_level)
VALUES
(ORCHESTRATE.name,
ORCHESTRATE.wormer_type,
ORCHESTRATE.dose_interval,
ORCHESTRATE.dose_level)
```

The UPDATE statement is as automatically generated by the InfoSphere DataStage:

```
UPDATE
horse_health
SET
wormer_type=ORCHESTRATE.wormer_type,
dose_interval=ORCHESTRATE.dose_interval,
dose_level=ORCHESTRATE.dose_level
WHERE
(name=ORCHESTRATE.name)
```

Must Do's

The steps to use an Oracle Enterprise stage in a job depend on the operation that you want to complete with the stage.

This section specifies the minimum steps to take to get a Oracle Enterprise stage functioning. The InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Updating an Oracle database

To update an Oracle database, you specify and configure an update method and ensure that column metadata is specified for the update operation.

Procedure

1. In the Input link **Properties** tab, under the Target category specify the update method as follows:
   a. Specify a Write Method of Upsert.
   b. Specify the Table you are writing.
   c. Select the Upsert Mode, this allows you to specify whether to insert and update, or update only, and whether to use a statement automatically generated by IBM InfoSphere DataStage or specify your own.
   d. If you have chosen an Upsert Mode of User-defined Update and Insert, specify the Insert SQL statement to use. InfoSphere DataStage provides the auto-generated statement as a basis, which you can edit as required.
   e. If you have chosen an Upsert Mode of User-defined Update and Insert or User-defined Update only, specify the Update SQL statement to use. InfoSphere DataStage provides the auto-generated statement as a basis, which you can edit as required.

      Under the Connection category, you can either manually specify a connection string, or have InfoSphere DataStage generate one for you by using a user name and password you supply. Either way you need to supply a valid user name and password. InfoSphere DataStage encrypts the password when you use the auto-generate option.

      By default, InfoSphere DataStage assumes Oracle resides on the local server, but you can specify a remote server if required.

      Under the Options category:

   f. If you want to send rejected rows down a rejects link, set Output Rejects to True (it is false by default).
2. Ensure that column metadata is specified for the write operation.

Deleting rows from an Oracle database

To delete rows from an Oracle database, you specify details of the SQL statements that are used to delete rows from the database.

Procedure

On the Properties page for the input link, configure the properties:

1. Set the **Write Method** property to **Delete Rows**.
2. Specify a value for the **Delete Rows Mode** property. You can use a statement that is automatically generated by IBM InfoSphere DataStage or specify your own.
3. If you select a Delete Rows Mode of User-defined delete, specify the Delete SQL statement to use. InfoSphere DataStage provides the auto-generated statement, which you can edit as required.

Loading an Oracle database

To load data to an Oracle database, you configure properties and ensure that column metadata is defined.

About this task

This method is the default write method.

Procedure

1. In the Input link **Properties** tab, under the Target category:
   a. Specify a Write Method of Load.
   b. Specify the Table you are writing.
   c. Specify the Write Mode (by default the IBM InfoSphere DataStage appends to existing tables, you can also decide to create a new table, replace an existing table, or keep existing table details but replace all the rows).

      Under the Connection category, you can either manually specify a connection string, or have the InfoSphere DataStage generate one for you by using a user name and password you supply. Either way you need to supply a valid user name and password. The InfoSphere DataStage encrypts the password when you use the auto-generate option.

      By default, the InfoSphere DataStage assumes Oracle resides on the local server, but you can specify a remote server if required.
2. Ensure that column metadata is specified for the write operation.

Reading data from an Oracle database

To read data from an Oracle database, you specify properties and ensure that column metadata is specified for the read operation.

Procedure

1. In the Output link **Properties** tab:
   a. Select a Read Method. This is Table by default, but you can also decide to read using auto-generated SQL or user-generated SQL. The read operates sequentially on a single node unless you specify a Partition Table property (which causes parallel execution on the processing nodes containing a partition derived from the named table).
   b. Specify the table to be read.
   c. If using a Read Method of user-generated SQL, specify the SELECT SQL statement to use. The IBM InfoSphere DataStage provides the auto-generated statement as a basis, which you can edit as required.

      Under the Connection category, you can either manually specify a connection string, or have the InfoSphere DataStage generate one for you by using a user name and password you supply. Either way you need to supply a valid user name and password. the InfoSphere DataStage encrypts the password when you use the auto-generate option.

      By default, the InfoSphere DataStage assumes Oracle resides on the local server, but you can specify a remote server if required.
2. Ensure that column metadata is specified for the read operation.

Performing a direct lookup on an Oracle database table

To perform a direct lookup on an Oracle database table, you set up the job, configure lookup properties, and ensure that column metadata is specified for the lookup operation.

Procedure

1. Connect the Oracle Enterprise stage to a Lookup stage by using a reference link.
2. In the Output link **Properties** tab:
   a. Set the Lookup Type to Sparse.
   b. Select a Read Method. This is Table by default (which reads directly from a table), but you can also decide to read using auto-generated SQL or user-generated SQL.
   c. Specify the table to be read for the lookup.
   d. If using a Read Method of user-generated SQL, specify the SELECT SQL statement to use. The IBM InfoSphere DataStage provides the auto-generated statement as a basis, which you can edit as required. You would use this if, for example, you wanted to perform a non-equality based lookup.

      Under the Connection category, you can either manually specify a connection string, or have the InfoSphere DataStage generate one for you by using a user name and password you supply. Either way you need to supply a valid user name and password. The InfoSphere DataStage encrypts the password when you use the auto-generate option.

      By default, the InfoSphere DataStage assumes Oracle resides on the local server, but you can specify a remote server if required.
3. Ensure that column metadata is specified for the lookup operation.

Performing an in-memory lookup on an Oracle database table

To perform an in-memory lookup operation on an Oracle database table, you set up the job, configure lookup properties, and ensure that column metadata is specified for the lookup operation.

About this task

This method is the default method.

Procedure

1. Connect the Oracle Enterprise stage to a Lookup stage by using a reference link.
2. In the Output link **Properties** tab:
   a. Set the Lookup Type to Normal.
   b. Select a Read Method. This is Table by default (which reads directly from a table), but you can also decide to read using auto-generated SQL or user-generated SQL.
   c. Specify the table to be read for the lookup.
   d. If using a Read Method of user-generated SQL, specify the SELECT SQL statement to use. The IBM InfoSphere DataStage provides the auto-generated statement as a basis, which you can edit as required. You would use this if, for example, you wanted to perform a non-equality based lookup.

Under the Connection category, you can either manually specify a connection string, or have the InfoSphere DataStage generate one for you by using a user name and password you supply. Either way you need to supply a valid user name and password. The InfoSphere DataStage encrypts the password when you use the auto-generate option.

By default, the InfoSphere DataStage assumes Oracle resides on the local server, but you can specify a remote server if required.

3. Ensure that column metadata is specified for the lookup operation.

Stage page

The Stage page includes the **General**, **Advanced**, and **NLS Map** tabs.

The **General** tab allows you to specify an optional description of the stage. The **Advanced** tab allows you to specify how the stage executes. The **NLS Map** tab appears if you have NLS enabled on your system, it allows you to specify a character set map for the stage.

Advanced tab

On the Advanced tab, you can configure properties that affect the execution mode, combinability mode, partitioning, node pools, and node maps.

This tab allows you to specify the following values:

- **Execution Mode**. The stage can run in parallel mode or sequential mode. In parallel mode the data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the **Advanced** tab. In Sequential mode the data is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows the IBM InfoSphere DataStage to combine the operators that underlie parallel stages. Then they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. You can select **Set** or **Clear**. If you select **Set** read operations will request that the next stage preserves the partitioning as is (it is ignored for write operations). Note that this field is only visible if the stage has output links.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or the resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

NLS Map tab

On the **NLS Map** tab, you can define a character set map for the Oracle Enterprise stage. You can set character set maps separately for NCHAR and NVARCHAR2 types and all other data types.

This setting overrides the default character set map set for the project or the job. You can specify that the map be supplied as a job parameter if required.

Load performance might be improved by specifying an Oracle map instead of an IBM InfoSphere DataStage map. To do this, add an entry to the file *oracle_cs*, located at *$APT_ORCHHOME/etc*, to associate the InfoSphere DataStage map with an Oracle map.

The *oracle_cs* file has the following format:

```
UTF-8              UTF8
ISO-8859-1         WE8ISO8859P1
EUC-JP             JA16EUC
```

The first column contains the InfoSphere DataStage map names and the second column the Oracle map names they are associated with.

By using the example file shown above, specifying the InfoSphere DataStage map EUC-JP in the Oracle stage will cause the data to be loaded using the Oracle map JA16EUC.

Inputs page

On the Inputs page, you can specify details about how the Oracle Enterprise stage writes data to a Oracle database. The Oracle Enterprise stage can have only one input link writing to one table.

The **General** tab allows you to specify an optional description of the input link. The **Properties** tab allows you to specify details of exactly what the link does. The **Partitioning** tab allows you to specify how incoming data is partitioned before being written to the database. The **Columns** tab specifies the column definitions of incoming data. The **Advanced** tab allows you to change the default buffering settings for the input link.

Details about Oracle enterprise stage properties, partitioning, and formatting are given in the following sections. See the *IBM InfoSphere DataStage and QualityStage Parallel Job Developer's Guide* for a general description of the other tabs.

Input Link Properties tab

On the **Properties** page, you can specify properties for the input link, which dictate how and where incoming data is written.

Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 28. Input link properties and values

Category/ Property	Values	Default	Required?	Dependent of
Target/Table	string	N/A	Y (if Write Method = Load)	N/A
Target/Delete Rows Mode	Auto-generated delete/user-defined delete	Auto-generated delete	Y if Write method = Delete Rows	N/A

Table 28. Input link properties and values (continued)

Category/ Property	Values	Default	Required?	Dependent of
Target/Delete SQL	String	N/A	Y if Write method = Delete Rows	N/A
Target/Upsert mode	Auto-generated Update & insert/Auto-generated Update Only/User-defined Update & Insert/User-defined Update Only	Auto-generated Update & insert	Y (if Write Method = Upsert)	N/A
Target/Upsert Order	Insert then update/Update then insert	Insert then update	Y (if Write Method = Upsert)	N/A
Target/Insert SQL	string	N/A	N	N/A
Target/Insert Array Size	number	500	N	Insert SQL
Target/Update SQL	string	N/A	Y (if Write Method = Upsert)	N/A
Target/Write Method	Delete Rows/Upsert/ Load	Load	Y	N/A
Target/Write Mode	Append/ Create/ Replace/ Truncate	Append	Y (if Write Method = Load)	N/A
Connection/DB Options	string	N/A	Y	N/A
Connection/DB Options Mode	Auto-generate/ User-defined	Auto-generate	Y	N/A
Connection/ User	string	N/A	Y (if DB Options Mode = Auto-generate)	DB Options Mode
Connection/ Password	string	N/A	Y (if DB Options Mode = Auto-generate)	DB Options Mode
Connection/ Additional Connection Options	string	N/A	N	DB Options Mode
Connection/ Remote Server	string	N/A	N	N/A
Options/Output Reject Records	True/False	False	Y (if Write Method = Upsert)	N/A

Table 28. Input link properties and values (continued)

Category/ Property	Values	Default	Required?	Dependent of
Options/Silently Drop Columns Not in Table	True/False	False	Y (if Write Method = Load)	N/A
Options/Table Organization	Heap/Index	Heap	Y (if Write Method = Load and Write Mode = Create or Replace)	N/A
Options/ Truncate Column Names	True/False	False	Y (if Write Method = Load)	N/A
Options/Close Command	string	N/A	N	N/A
Options/Default String Length	number	32	N	N/A
Options/Index Mode	Maintenance/ Rebuild	N/A	N	N/A
Options/Add NOLOGGING clause to Index rebuild	True/False	False	N	Index Mode
Options/Add COMPUTE STATISTICS clause to Index rebuild	True/False	False	N	Index Mode
Options/Open Command	string	N/A	N	N/A
Options/Oracle Partition	string	N/A	N	N/A
Options/Create Primary Keys	True/False	False	Y (if Write Mode = Create or Replace)	N/A
Options/Create Statement	string	N/A	N	N/A
Options/Disable Constraints	True/False	False	Y (if Write Method = Load)	N/A
Options/ Exceptions Table	string	N/A	N	Disable Constraints
Options/Table has NCHAR/ NVARCHAR	True/False	False	N	N/A

Target category

On the Input Link Properties tab, the Target category includes properties for the table to write to and how to write to the table.

These are the properties available in the Target category.

Table

Specify the name of the table to write to. You can specify a job parameter if required.

Delete Rows Mode

This only appears for the Delete Rows write method. Allows you to specify how the delete statement is to be derived. Select from:

- **Auto-generated Delete**. The IBM InfoSphere DataStage generates a delete statement for you, based on the values you have supplied for table name and column details. The statement can be viewed by selecting the **Delete SQL** property.
- **User-defined Delete**. Select this to enter your own delete statement. Then select the **Delete SQL** property and edit the statement proforma.

Delete SQL

Only appears for the Delete Rows write method. This property allows you to view an auto-generated Delete statement, or to specify your own (depending on the setting of the **Delete Rows Mode** property).

Upsert mode

This only appears for the Upsert write method. Allows you to specify how the insert and update statements are to be derived. Select from:

- **Auto-generated Update & Insert**. The InfoSphere DataStage generates update and insert statements for you, based on the values you have supplied for table name and on column details. The statements can be viewed by selecting the **Insert SQL** or **Update SQL** properties.
- **Auto-generated Update Only**. The InfoSphere DataStage generates an update statement for you, based on the values you have supplied for table name and on column details. The statement can be viewed by selecting the **Update SQL** properties.
- **User-defined Update & Insert**. Select this to enter your own update and insert statements. Then select the **Insert SQL** and **Update SQL** properties and edit the statement proformas.
- **User-defined Update Only**. Select this to enter your own update statement. Then select the **Update SQL** property and edit the statement proforma.

Upsert Order

This only appears for the Upsert write method. Allows you to decide between the following values:

- Insert and, if that fails, update (Insert then update)
- Update and, if that fails, insert (Update then insert)

Insert SQL

Only appears for the Upsert write method. This property allows you to view an auto-generated Insert statement, or to specify your own (depending on the setting of the **Update Mode** property). It has a dependent property:

- **Insert Array Size**

Specify the size of the insert host array. The default size is 500 records. If you want each insert statement to be executed individually, specify 1 for this property.

Update SQL

Only appears for the Upsert write method. This property allows you to view an auto-generated Update statement, or to specify your own (depending on the setting of the **Upsert Mode** property).

Write Method

Select from Delete Rows, Upsert or Load (the default value). Upsert allows you to provide the insert and update SQL statements and uses Oracle host-array processing to optimize the performance of inserting records. Load sets up a connection to Oracle and inserts records into a table, taking a single input data set. The Write Mode property determines how the records of a data set are inserted into the table.

Write Mode

This only appears for the Load Write Method. Select from the following values:

- **Append**. This is the default value. New records are appended to an existing table.
- Create. Create a new table. If the Oracle table already exists an error occurs and the job terminates. You must specify this mode if the Oracle table does not exist.
- **Replace**. The existing table is first dropped and an entirely new table is created in its place. Oracle uses the default partitioning method for the new table.
- **Truncate**. The existing table attributes (including schema) and the Oracle partitioning keys are retained, but any existing records are discarded. New records are then appended to the table.

Connection category

On the Input Link Properties tab, the Connection category includes properties for database options and the remote server.

DB Options

Specify a user name and password for connecting to Oracle in the form:

```
<user=< user >,password=< password >[,arraysize= < num_records >]
```

The IBM InfoSphere DataStage does not encrypt the password when you use this option. Arraysize is only relevant to the Upsert Write Method.

DB Options Mode

If you select Auto-generate for this property, the InfoSphere DataStage will create a DB Options string for you. If you select User-defined, you have to edit the DB Options property yourself. When Auto-generate is selected, there are three dependent properties:

- **User**

  The user name to use in the auto-generated DB options string.
- **Password**

The password to use in the auto-generated DB options string. The InfoSphere DataStage encrypts the password.

Note: If you have a password with special characters, enclose the password in quotation marks. For example: "passw#rd".

- **Additional Connection Options**

  Optionally allows you to specify additional options to add to the Oracle connection string.

Remote Server

This is an optional property. Allows you to specify a remote server name.

Options category

On the Input Link Properties tab, the properties in the Options category depend on the write method that you choose.

Create Primary Keys

This option is available with a Write Method of Load and Write Mode of Create or Replace. It is False by default, if you set it True, the columns marked as keys in the **Columns** tab will be marked as primary keys. You must set this true if you want to write index organized tables, and indicate which are the primary keys on the **Columns** tab. Note that, if you set it to True, the Index Mode option is not available.

Create Statement

This is an optional property available with a Write Method of Load and a Write Mode of Create. Contains an SQL statement to create the table (otherwise the IBM InfoSphere DataStage will auto-generate one).

Disable Constraints

This is False by default. Set True to disable all enabled constraints on a table when loading, then attempt to re-enable them at the end of the load. This option is not available when you select a Table Organization type of Index to use index organized tables. When set True, it has a dependent property:

- **Exceptions Table**

  This property enables you to specify an exceptions table, which is used to record ROWID information for rows that violate constraints when the constraints are re-enabled. The table must already exist.

Output Reject Records

This only appears for the Upsert write method. It is False by default, set to True to send rejected records to the reject link.

Silently Drop Columns Not in Table

This only appears for the Load Write Method. It is False by default. Set to True to silently drop all input columns that do not correspond to columns in an existing Oracle table. Otherwise the stage reports an error and terminates the job.

Table Organization

This appears only for the Load Write Method using the Create or Replace Write Mode. Allows you to specify Index (for index organized tables) or heap organized tables (the default value). When you select Index, you must also set Create Primary Keys to true. In index organized tables (IOTs) the rows of the table are held in the index created from the primary keys.

Truncate Column Names

This only appears for the Load Write Method. Set this property to True to truncate column names to 30 characters.

Close Command

This is an optional property and only appears for the Load Write Method. Use it to specify any command, in single quotes, to be parsed and executed by the Oracle database on all processing nodes after the stage finishes processing the Oracle table. You can specify a job parameter if required.

Default String Length

This is an optional property and only appears for the Load Write Method. It is set to 32 by default. Sets the default string length of variable-length strings written to a Oracle table. Variable-length strings longer than the set length cause an error.

The maximum length you can set is 2000 bytes. Note that the stage always allocates the specified number of bytes for a variable-length string. In this case, setting a value of 2000 allocates 2000 bytes for every string. Therefore, you should set the expected maximum length of your largest string and no larger.

Index Mode

This is an optional property and only appears for the Load Write Method. Lets you perform a direct parallel load on an indexed table without first dropping the index. You can select either Maintenance or Rebuild mode. The Index property only applies to append and truncate Write Modes.

Rebuild skips index updates during table load and instead rebuilds the indexes after the load is complete using the Oracle alter index rebuild command. The table must contain an index, and the indexes on the table must not be partitioned. The Rebuild option has two dependent properties:

- Add NOLOGGING clause to Index rebuild

  This is False by default. Set True to add a NOLOGGING clause.
- Add COMPUTE STATISTICS clause to Index rebuild

  This is False by default. Set True to add a COMPUTE STATISTICS clause.

Maintenance results in each table partition's being loaded sequentially. Because of the sequential load, the table index that exists before the table is loaded is maintained after the table is loaded. The table must contain an index and be partitioned, and the index on the table must be a local range-partitioned index that is partitioned according to the same range values that were used to partition the table. Note that in this case *sequential* means *sequential per partition*, that is, the degree of parallelism is equal to the number of partitions.

Open Command

This is an optional property and only appears for the Load Write Method. Use it to specify a command, in single quotes, to be parsed and executed by the Oracle database on all processing nodes before the Oracle table is opened. You can specify a job parameter if required.

Oracle Partition

This is an optional property and only appears for the Load Write Method. Name of the Oracle table partition that records will be written to. The stage assumes that the data provided is for the partition specified.

Table has NCHAR/NVARCHAR

This option applies to Create or Replace Write Modes. Set it True if the table being written contains NCHAR and NVARCHARS. The correct columns are created in the target table.

Partitioning tab

On the **Partitioning** tab, you can specify details about how the incoming data is partitioned or collected before it is written to the Oracle database. You can also specify that the data is sorted before it is written to the database.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Oracle enterprise stage is operating in sequential mode, it will first collect the data before writing it to the file by using the default Auto collection method.

The **Partitioning** tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Oracle enterprise stage is set to run in parallel or sequential mode.
- Whether the preceding stage in the job is set to run in parallel or sequential mode.

If the Oracle enterprise stage is set to run in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Oracle enterprise stage is set to run in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list.

The following partitioning methods are available:

- **(Auto)**. The IBM InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Oracle enterprise stage.
- **Entire**. Each file written to receives the entire data set.
- Hash. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.

- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place. This is the default value for Oracle enterprise stages.
- **DB2**. Replicates the partitioning method of the specified IBM DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. Divides a data set into approximately equal size partitions based on one or more partitioning keys. Range partitioning is often a preprocessing step to performing a total sort on a data set. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Oracle enterprise stages. Normally, when you are using the Auto mode, the InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and continuing on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and continuing on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The **Partitioning** tab also allows you to specify that data arriving on the input link should be sorted before being written to the file or files. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default value.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether

the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the pop-up menu.

Outputs page

On the Outputs page, you can specify details about how the Oracle Enterprise stage reads data from a Oracle database. The Oracle Enterprise stage can have only one output link.

Alternatively it can have a reference output link, which is used by the Lookup stage when referring to a Oracle lookup table. It can also have a reject link where rejected records are routed (used in conjunction with an input link). The **Output Name** list allows you to choose whether you are looking at details of the main output link or the reject link.

The **General** tab allows you to specify an optional description of the output link. The **Properties** tab allows you to specify details of exactly what the link does. The **Columns** tab specifies the column definitions of the data. The **Advanced** tab allows you to change the default buffering settings for the output link.

Output Link Properties tab

On the Outputs page, the **Properties** tab includes properties for the output link. These properties dictate how incoming data is read from a table.

Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The **Build SQL** button allows you to instantly open the SQL Builder to help you construct an SQL query to read data. See the *IBM InfoSphere DataStage and QualityStage Designer Client Guide* for guidance on using it.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 29. Output link properties and values

Category/ Property	Values	Default	Required?	Dependent of
Source/Lookup Type	Normal/ Sparse	Normal	Y (if output is reference link connected to Lookup stage)	N/A
Source/Read Method	Auto-generated SQL /Table/SQL builder generated SQL /User-defined SQL	SQL builder generated SQL	Y	N/A
Source/Table	string	N/A	N	N/A
Source/Where	string	N/A	N	Table
Source/Select List	string	N/A	N	Table

Table 29. Output link properties and values (continued)

Category/ Property	Values	Default	Required?	Dependent of
Source/SQL Query	string	N/A	N	N/A
Source/Partition Table	string	N/A	N	N/A
Connection/DB Options	string	N/A	Y	N/A
Connection/ DB Options Mode	Auto-generate/ User-defined	Auto-generate	Y	N/A
Connection/ User	string	N/A	Y (if DB Options Mode = Auto-generate)	DB Options Mode
Connection/ Password	string	N/A	Y (if DB Options Mode = Auto-generate)	DB Options Mode
Connection/ Additional Connection Options	string	N/A	N	DB Options Mode
Connection/ Remote Server	string	N/A	N	N/A
Options/Close Command	string	N/A	N	N/A
Options/Open Command	string	N/A	N	N/A
Options/Table has NCHAR/ NVARCHAR	True/False	False	N	N/A

Source category

On the **Output Link Properties** tab, the Source category includes properties for the lookup type, read method, SQL query, table, and partition table.

Lookup Type

Where the Oracle enterprise stage is connected to a Lookup stage using a reference link, this property specifies whether the Oracle enterprise stage will provide data for an in-memory look up (Lookup Type = Normal) or whether the lookup will access the database directly (Lookup Type = Sparse).

Read Method

This property specifies whether you are specifying a table or a query when reading the Oracle database, and how you are generating the query.

- Select the Table method in order to use the Table property to specify the read. This will read in parallel.

- Select Auto-generated SQL to have the IBM InfoSphere DataStage automatically generate an SQL query based on the columns you have defined and the table you specify in the Table property.
- Select User-defined SQL to define your own query. By default a user-defined or auto-generated SQL will read sequentially on one node. Read methods of Auto-generated SQL and User-defined SQL operate sequentially on a single node. You can have the User-defined SQL read operate in parallel if you specify the Partition Table property.
- Select SQL Builder Generated SQL to open the SQL Builder and define the query using its helpful interface. (See the *IBM InfoSphere DataStage and QualityStage Designer Client Guide*.)

By default, Read methods of SQL Builder Generated SQL, Auto-generated SQL, and User-defined SQL operate sequentially on a single node. You can have the User-defined SQL read operate in parallel if you specify the Partition Table property.

SQL Query

Optionally allows you to specify an SQL query to read a table. The query specifies the table and the processing that you want to perform on the table as it is read by the stage. This statement can contain joins, views, database links, synonyms, and other entities.

Table

Specifies the name of the Oracle table. The table must exist and you must have SELECT privileges on the table. If your Oracle user name does not correspond to the owner of the specified table, you can prefix it with a table owner in the form:

```
table_owner.table_name
```

Table has dependent properties:

- **Where**

  Stream links only. Specifies a WHERE clause of the SELECT statement to specify the rows of the table to include or exclude from the read operation. If you do not supply a WHERE clause, all rows are read.
- **Select List**

  Optionally specifies an SQL select list, enclosed in single quotes, that can be used to determine which columns are read. You must specify the columns in *list* in the same order as the columns are defined in the record schema of the input table.

Partition Table

Specifies execution of the SELECT in parallel on the processing nodes containing a partition derived from the named table. If you do not specify this, the stage executes the query sequentially on a single node.

Connection category

On the **Output Link Properties** tab, the Connection category includes properties for database options and the remote server.

DB Options

Specify a user name and password for connecting to Oracle in the form:
`<user=< user >,password=< password >[,arraysize=< num_records >]`

The IBM InfoSphere DataStage does not encrypt the password when you use this option. Arraysize only applies to stream links. The default arraysize is 1000.

DB Options Mode

If you select Auto-generate for this property, the InfoSphere DataStage will create a DB Options string for you. If you select User-defined, you have to edit the DB Options property yourself. When Auto-generate is selected, there are two dependent properties:

- **User**

  The user name to use in the auto-generated DB options string.
- **Password**

  The password to use in the auto-generated DB options string. The InfoSphere DataStage encrypts the password

  Note: If you have a password with special characters, enclose the password in quotation marks. For example: "passw#rd".
- **Additional Connection Options**

  Optionally allows you to specify additional options to add to the Oracle connection string.

Remote Server

This is an optional property. Allows you to specify a remote server name.

Options category

On the **Output Link Properties** tab, the Options category includes properties for the close and open commands. It also include a property to use if the table contains data that has the NCHAR or NVARCHARS data types.

Close Command

This is an optional property and only appears for stream links. Use it to specify any command to be parsed and executed by the Oracle database on all processing nodes after the stage finishes processing the Oracle table. You can specify a job parameter if required.

Open Command

This is an optional property only appears for stream links. Use it to specify any command to be parsed and executed by the Oracle database on all processing nodes before the Oracle table is opened. You can specify a job parameter if required

Table has NCHAR/NVARCHAR

Set this True if the table being read from contains NCHAR and NVARCHARS.

Chapter 5. Oracle OCI stage

Use the Oracle OCI stage to rapidly and efficiently prepare and load streams of tabular data from any IBM InfoSphere DataStage stage (for example, the ODBC stage or the Sequential File stage) to and from tables of the target Oracle database. The Oracle client on Microsoft Windows or UNIX uses SQL*Net to access an Oracle server on Windows or UNIX.

When you use IBM InfoSphere DataStage to access Oracle databases, you can choose from a collection of connectivity options. For most new jobs, use the Oracle Connector stage, which offers better functionality and performance than the Oracle OCI stage.

If you have jobs that use the Oracle OCI stage and want to use the connector, use the Connector Migration Tool to migrate jobs to use the connector.

Each Oracle OCI stage is a passive stage that can have any number of input, output, and reference output links:

- Input links specify the data you are writing, which is a stream of rows to be loaded into an Oracle database. You can specify the data on an input link by using an SQL statement constructed by InfoSphere DataStage or a user-defined SQL statement.
- Output links specify the data you are extracting, which is a stream of rows to be read from an Oracle database. You can also specify the data on an output link by using an SQL statement constructed by InfoSphere DataStage or a user-defined SQL statement.
- Each reference output link represents a row that is key read from an Oracle database (that is, it reads the record using the key field in the WHERE clause of the SQL SELECT statement).

Oracle offers a proprietary call interface for C and C++ programmers that allows manipulation of data in an Oracle database. The Oracle Call Interface (OCI) stage can connect and process SQL statements in the native Oracle environment without needing an external driver or driver manager. To use the Oracle OCI stage, you need only to install the Oracle client, which uses SQL*Net to access the Oracle server.

The Oracle OCI stage works with Oracle servers, provided you install the appropriate Oracle software. For information about exceptions to this, see Oracle documentation for the appropriate release.

With the Oracle OCI stage, you can:

- Generate your SQL statement.
- Use a file name to contain your SQL statement.
- Clear a table before loading by using a TRUNCATE statement. (Clear table)
- Select how often to commit rows to the database. (Transaction size)
- Input multiple rows of data in one call to the database. (Array size)
- Read multiple rows of data in one call from the database. (Array size)
- Specify transaction isolation levels for concurrency control and transaction performance tuning. (Transaction Isolation)

- Specify criteria that data must meet before being selected. (WHERE clause)
- Specify criteria to sort, summarize, and aggregate data. (Other clauses)
- Specify the behavior of parameter marks in SQL statements.

The Oracle OCI stage is dependent on the *libclntsh* shared library, which is created during the installation of the Oracle client software. You must include the location containing this shared library in the shared library search path for InfoSphere DataStage jobs to run successfully by using this stage.

Functionality of the Oracle OCI stage

The Oracle OCI stage supports features including transaction grouping, reject row handling, and create and drop table functionality before writing to a table.

- Support for transaction grouping to control a group of input links from a Transformer stage. This lets you write a set of data to a database in one transaction. Oracle OCI stage opens one database session per transaction group.
- Support for reject row handling. Link reject variables tell the Transformer stage the Oracle DBMS error code when an error occurs in the Oracle OCI stage for insert, update, and other actions, for control of job execution. The format of the error is DBMS.CODE=ORA-*xxxxx*.
- Support for create and drop table functionality before writing to a table.
- Support for before and after SQL statements to run user-defined SQL statements before or after the stage writes or reads into a database.
- Support of stream input, stream output, and reference output links.
- The ability to use the **Derivation** cell to specify fully-qualified column names used to construct an SQL SELECT statement for output and reference links.

  Note: When you select **Enable case sensitive table/column name**, it is your responsibility to use quotation marks for the owner/ table.column name in the **Derivation** cell to preserve any lowercase letters.
- Performance and scalability benefits by using Oracle OCI stage rather than the ODBC stage to access Oracle tables.
- Prefetching of SELECT statement result set rows when executing a query. This minimizes server round trips and enhances performance.
- Reduction of the number of network round trips (more processing is done on the client).
- Support of new transparent data structures and interfaces.
- Elimination of open and close cursor round trips.
- Improved error handling.
- Use of Oracle OCI stage as a supplement to existing jobs that already use the ODBC stage, rather than as a replacement for the ODBC stage.
- Importing of table definitions. Support of a file name to contain your SQL statement.
- Support for NLS (National Language Support).
- Support for foreign key metadata import.
- Support for the behavior of parameter marks for SQL statements.

The following functionality is not supported:

- Loading stream input links in bulk. Use the Oracle OCI Load stage to bulk load data into Oracle databases.
- Stored procedures.

- Support of Oracle data types such as BLOB, FILE, LOB, LONG, LONG RAW, MSLABEL, OBJECT, RAW, REF, ROWID, or a named data type.
- Running on the parallel canvas under either Windows or UNIX.
- SUBSTR2
- SUBSTR4
- NCHAR
- LENGTH2
- LENGTH4
- INSTR2
- INSTR4
- CAST
- NEW_TIME
- RPAD
- MONTHS_BETWEEN
- Functions having an OVER clause

Configuration requirements of the Oracle OCI stage

To use the Oracle OCI stage, ensure that the configuration requirements are met.

The Oracle OCI stage has the following requirements:

- Install the Oracle standard client on the engine tier. You cannot use the stage if only Oracle Instant Client is installed.
- Configuration of SQL*Net using a configuration program, for example, SQL*Net Easy Configuration, to set up and add database aliases.
- The following environment variables on the server in UNIX:
  - ORACLE_HOME
  - TWO_TASK
  - ORACLE_SID
  - LD_LIBRARY_PATH

The name of the environment variable LD_LIBRARY_PATH differs depending on the platform.

Table 30. Platform-specific names for LD_LIBRARY_PATH

PLATFORM	NAME OF ENVIRONMENT VARIABLE
AIX	LIBPATH
HP_UX	SHLIB_PATH
LINUX or Solaris	LD_LIBRARY_PATH

For the SHLIB_PATH environment variable, the InfoSphere DataStage library entries must be referenced before any branded-ODBC library entries at run time.

Note: You should have read and execute permissions to use libraries in the $ORACLE_HOME/lib and $ORACLE_HOME/bin directories and read permissions on all files in the $ORACLE_HOME directory. Otherwise, you might experience problems using Oracle OCI stage to connect to Oracle.

Oracle OCI stage editor

The editor for the Oracle OCI stage includes the Stage, Input, and Output pages.

This dialog box can have up to three pages (depending on whether there are inputs to and outputs from the stage):

- **Stage.** This page displays the name of the stage you are editing. The **General** tab defines the Oracle database source and logon information to connect to an Oracle database.

  The **NLS** tab defines a character set map to be used with the stage. (The **NLS tab** appears only if you have installed NLS.)
- **Input.** This page is displayed only if you have an input link to this stage. It specifies the SQL table to use and the associated column definitions for each data input link. This page also specifies the type of update action and transaction isolation level information for concurrency control and performance tuning. It also contains the SQL statement used to write the data and lets you enable case sensitivity for SQL statements.
- **Output.** This page is displayed only if you have an output link to this stage. It specifies the SQL tables to use and the associated column definitions for each data output link. This page also specifies the type of query and transaction isolation level information for concurrency control and performance tuning. It also contains the SQL SELECT statement used to extract the data, and lets you enable case sensitivity for SQL statements.

Defining the Oracle connection

You can configure the data and connection for the Oracle OCI stage in the ORAOCI9 Stage window.

Procedure

1. Connect to the Oracle database.
2. Optional. Define a character set map.
3. Define the data on the input links.
4. Define the data on the output links.

Connecting to an Oracle database

To connect to an Oracle database, you set the Oracle connection parameters on the **General** tab of the Stage page in the stage editor.

Procedure

1. Enter the name of the Oracle database alias to access in the **Database source name** field. (This is the name you created using the Oracle Configuration Assistant.) Unless the database has a guest account, **User ID** must be a valid user in the database, have an alias in the database, or be a system administrator or system security officer. There is no default value.
2. Enter the user name to use to connect to the Oracle database in the **User ID** field. This user must have sufficient privileges to access the specified database and source and target tables. This field is required except when **Use OS level authentication** is selected. There is no default value.
3. Enter the password that is associated with the specified user name to use in the **Password** field. This field is required except when **Use OS level authentication** is selected. There is no default value.

4. Select an appropriate transaction isolation level to use from the **Transaction Isolation** list on the **General** tab on the Input page or Output page. This level provides the necessary consistency and concurrency control between transactions in the job and other transactions for optimal performance. Because Oracle does not prevent other transactions from modifying the data read by a query, that data might be changed by other transactions between two executions of the query. Thus, a transaction that executes a given query twice might experience both nonrepeatable reads and phantoms. Use one of the following transaction isolation levels:

   Read Committed
   Takes exclusive locks on modified data and sharable locks on all other data. Read committed is the default ISO level for all transactions.

   Serializable
   Takes exclusive locks on modified data and sharable locks on all other data. Serializable transactions see only those changes that were committed at the time the transaction began.

   For more information about using these levels, see your Oracle documentation.
5. Enter an optional description of the Oracle OCI stage in the **Description** field.
6. Select **Use OS level authentication** to automatically log on using your operating system user name and password. The default value is cleared. For further details on Oracle login information, see your Oracle documentation.

Defining character set mapping

You can define a character set map for a stage on the **NLS** tab of the Stage page. The **NLS** tab appears only if you installed NLS.

Procedure

Specify information using the following fields:

Map name to use with stage
Defines the default character set map for the project or the job. You can change the map by selecting a map name from the list.

Show all maps
Lists all the maps that are shipped with the IBM InfoSphere DataStage.

Loaded maps only
Lists only the maps that are currently loaded.

Use Job Parameter...
Specifies parameter values for the job. Use the format #*Param*#, where *Param* is the name of the job parameter. The string #*Param*# is replaced by the job parameter when the job is run.

Defining input data

When you write data to a table in an Oracle database, the Oracle OCI stage has an input link. The properties of this link and the column definitions of the data are defined on the Input page in the ORAOCI Stage editor.

The input page

The Input page has an **Input name** list; the **General**, **Options**, **Columns**, **SQL**, and **Transaction Handling** tabs; and the **Columns** and **View Data** buttons.

Procedure

1. Choose the name of the input link you want to edit from the **Input name** list. This list displays all the input links to the Oracle OCI stage.
2. Click **Columns** to display a brief list of the columns designated on the input link. As you enter detailed metadata in the **Columns** tab, you can leave this list displayed.
3. Click **View Data** to invoke the Data Browser. This lets you look at the data associated with the input link in the database.

General tab of the Input page of the Oracle OCI stage

Use this tab to indicate how the SQL statements are created from an **Input** link on the Oracle OCI stage.

This tab is displayed by default. It contains the following fields:

- **Query Type.** Determines how the SQL statements are created. Options are
  - **Use SQL Builder tool.** Causes the **SQL Builder** button and the **Update action** property to appear. This is the default value for new jobs.
  - **Generate Update action from Options and Columns tabs.** Causes the **Update action** property to appear. Uses values from the **Options** and **Columns** tabs and from **Update action** to generate the SQL.
  - **Enter custom SQL statement.** Writes the data using a user-defined SQL statement, which overrides the default SQL statement generated by the stage. If you select this option, you enter the SQL statement on the SQL tab.
  - **Load SQL from a file at run time.** Uses the contents of the specified file to write the data.
- **SQL Builder.** Causes SQL Builder to open.
- **Update action.** Specifies which SQL statements are used to update the target table. Some update actions require key columns to update or delete rows. There is no default value. Select the option you want from the list:
  - **Clear table then insert rows.** Deletes the contents of the table and adds the new rows, with slower performance because of transaction logging. When you click **SQL Button**, the Insert page opens.
  - **Truncate table then insert rows.** Truncates the table with no transaction logging and faster performance. When you click **SQL Button**, the Insert page opens.
  - **Insert rows without clearing.** Inserts the new rows in the table.
  - **Delete existing rows only.** Deletes existing rows in the target table that have identical keys in the source files. When you click **SQL Button**, the Delete page opens.
  - **Replace existing rows completely.** Deletes the existing rows, then adds the new rows to the table. When you click **SQL Button**, the **Delete** page opens. However, you must also complete an Insert page to accomplish the replace.
  - **Update existing rows only.** Updates the existing data rows. Any rows in the data that do not exist in the table are ignored. When you click **SQL Button**, the Update page opens.
  - **Update existing rows or insert new rows.** Updates the existing data rows before adding new rows. It is faster to update first when you have a large number of records. When you click **SQL Button**, the Update page opens. However, you must also complete an Insert page to accomplish the replace.
  - **Insert new rows or update existing rows.** Inserts the new rows before updating existing rows. It is faster to insert first if you have only a few

records. When you click **SQL Button**, the Insert page opens. However you must also complete an Update page to accomplish the update.

- **Description.** Contains an optional description of the input link.

Options tab of the Input page of the Oracle OCI stage

Use the **Options** tab to create or drop tables and to specify miscellaneous Oracle link options.

- **Table name.** Names the target Oracle table to which the data is written. The table must exist or be created by choosing **generate DDL** from the **Create table action** list. Depending on the operations performed, you must be granted the appropriate permissions or privileges on the table. There is no default value.

  Click **...** (Browse button) to browse the Repository to select the table.

- **Create table action.** Creates the target table in the specified database if **Generate DDL** is selected. It uses the column definitions in the **Columns** tab and the table name and the TABLESPACE and STORAGE properties for the target table. The generated Create Table statement includes the TABLESPACE and STORAGE keywords, which indicate the location where the table is created and the storage expression for the Oracle storage-clause. You must have CREATE TABLE privileges on your schema.

  You can also specify your own CREATE TABLE SQL statement. You must enter the storage clause in Oracle format. (Use the **User-defined DDL** tab on the **SQL** tab for a complex statement.)

  Select one of the following options to create the table:

  – **Do not create target table.** Specifies that the target table is not created, and the Drop table action field and the Create Table Properties button on the right of the dialog are disabled.

  – **Generate DDL.** Specifies that the stage generates the CREATE TABLE statement using information from Table name, the column definitions grid, and the values in the Create Table Properties dialog.

  – **User-defined DDL.** Specifies that you enter the appropriate CREATE TABLE statement.

    Click the button to open the Create Table Properties dialog to display the table space and storage expression values for generating the DDL.

- **Drop table action.** Drops the target table before it is created by the stage if **Generate DDL** is selected. This field is disabled if you decide not to create the target table. The list displays the same items as the **Create table action** list except that they apply to the DROP TABLE statement. You must have DROP TABLE privileges on your schema.

- **Array size.** Specifies the number of rows to be transferred in one call between the IBM InfoSphere DataStage and the Oracle before they are written. Enter a positive integer to indicate how often Oracle performs writes to the database. The default value is 1, that is, each row is written in a separate statement.

  Larger numbers use more memory on the client to cache the rows. This minimizes server round trips and maximizes performance by executing fewer statements. If this number is too large, the client might run out of memory.

  Array size has implications for the InfoSphere DataStage's handling of reject rows.

- **Transaction size.** This field exists for backward compatibility, but it is ignored for version 3.0 and later of the stage. The transaction size for new jobs is now handled by **Rows per transaction** on the **Transaction Handling** tab.

- **Transaction Isolation.** Provides the necessary concurrency control between transactions in the job and other transactions. Use one of the following transaction isolation levels:
  - **Read committed.** Takes exclusive locks on modified data and sharable locks on all other data. Each query executed by a transaction sees only data that was committed before the query (not the transaction) began. Oracle queries never read dirty (uncommitted) data. This is the default value.
  - **Serializable.** Takes exclusive locks on modified data and sharable locks on all other data. Serializable transactions see only the changes that were committed at the time the transaction began.

    Note: If **Enable transaction grouping** is selected on the **Transaction Handling** tab, only the **Transaction Isolation** value for the first link is used for the entire group.
- **Treat warning message as fatal error.** Determines the behavior of the stage when an error is encountered while writing data to a table. If the check box is selected, a warning message is logged as fatal, and the job aborts. The format of the error message is:

  ```
  ORA-xxxxx Oracle error text message and row value
  ```

  If the check box is cleared (the default), three warning messages are logged in the InfoSphere DataStage Director log file, and the job continues. The format of the error message is:

  ```
  value of the row causing the error
  ORA-xxxxx Oracle error text message
  DBMS.CODE=ORA-xxxxx
  ```

  The last warning message is used for Reject Link Variables. If you want to use the Reject Link Variables functionality, you must clear the check box.
- **Enable case sensitive table/column name.** Enables the use of case-sensitive table and column names. Select to enclose table and column names in SQL statements in double quotation marks (" "). It is cleared by default.

Columns tab of the Input page of the Oracle OCI stage

On the Columns tab, you can view and modify column metadata for the input link.

Use the Save button to save any modifications that you make in the column metadata. Use the Load button to load an existing source table. From the Table Definitions window, select the appropriate table to load and click OK. The Select Column dialog is displayed. To ensure appropriate conversion of data types, clear the Ensure all Char columns use Unicode check box.

SQL tab of the Input page of the Oracle OCI stage

The SQL tab contains the **Query**, **Before**, **After**, **Generated DDL**, and **User-defined DDL** tabs. Use these tabs to display the stage-generated SQL statement and the SQL statement that you can enter.

- **Query.** This tab is displayed by default. It is similar to the **General** tab, but it contains the SQL statements that are used to write data to Oracle. It is based on the current values of the stage and link properties. You cannot edit these statements unless **Query type** is set to **Enter custom SQL statement** or **Load SQL from a file at run time**.
- **Before.** Contains the SQL statements executed before the stage processes any job data rows. The parameter on the **Before** tab corresponds to the Before SQL and Continue if Before SQL fails grid properties. The Continue if Before SQL fails

property is represented by the **Treat errors as non-fatal** check box, and the SQL statement is entered in a resizable edit box. The **Before** and **After** tabs look alike.

If the property value begins with FILE=, the remaining text is interpreted as a path name, and the contents of the file supplies the property value.

The Before SQL is the first SQL statement to be run. Depending on your choice, the job can continue or terminate after failing to execute a Before statement. It does not affect the transaction grouping scheme. The commit or rollback is performed on a per-link basis.

Each SQL statement is executed as a separate transaction if the statement separator is a double semi-colon (;;). All SQL statements are executed in a single transaction if a semi-colon (;) is the separator.

Treat errors as non-fatal. If selected, errors caused by Before SQL are logged as warnings, and processing continues with the next command batch. Each separate execution is treated as a separate transaction. If cleared, errors are treated as fatal to the job, and result in a transaction rollback. The transaction is committed only if all statements successfully run.

- **After.** Contains the SQL statements executed after the stage processes the job data rows. The parameters on this tab correspond to the After SQL and Continue if After SQL fails grid properties. The Continue if After SQL fails property is represented by the **Treat errors as non-fatal** check box, and the SQL statement is entered in a resizable edit box. The Before and After tabs look alike.

  If the property value begins with FILE=, the remaining text is interpreted as a path name, and the contents of the file supplies the property value.

  The After SQL statement is the last SQL statement to be run. Depending on your choice, the job can continue or terminate after failing to execute an After SQL statement. It does not affect the transaction grouping scheme. The commit or rollback is performed on a per-link basis.

  Each SQL statement is executed as a separate transaction if the statement separator is a double semi-colon (;;). All SQL statements are executed in a single transaction if a semi-colon (;) is the separator.

  The behavior of Treat errors as non-fatal is the same as for Before.

- **Generated DDL**. Select **Generate DDL** or **User-defined DDL** from the **Create table action** field on the **Options** tab to enable this tab.

  The **CREATE TABLE statement** field displays the CREATE TABLE statement that is generated from the column metadata definitions and the information provided on the Create Table Properties dialog box. If you select an option other than **Do not drop target table** from the **Drop table action** list, the **DROP statement** field displays the generated DROP TABLE statement for dropping the target table.

- **User-defined DDL.** Select **User-defined DDL** from the **Create table action** or **Drop table action** field on the **Options** tab to enable this tab. The generated DDL statement is displayed as a starting point to define a CREATE TABLE and a DROP TABLE statement. If the property value begins with FILE=, the remaining text is interpreted as a path name, and the contents of the file supplies the property value.

  The **DROP TABLE statement** field is disabled if **User-defined DDL** is not selected from the **Drop table action** field. If **Do not drop target** is selected, the **DROP statement** field is empty in the **Generated DDL** and **User-defined DDL** tabs.

  Note: Once you modify the user-defined DDL statement from the original generated DDL statement, changes made to other table-related properties do not

affect the user-defined DDL statement. If, for example, you add a new column in the column grid after modifying the user-defined DDL statement, the new column appears in the generated DDL statement but does not appear in the user-defined DDL statement.

Transaction handling tab

The Oracle OCI stage supports transaction grouping, which is the grouping of input links that come from a Transformer stage. You can use transaction handling to control the group of input links for start, commit, or rollback in one transaction when writing to a single data source.

You can use the **On Fail** or **On Skip** values to specify whether the transaction is committed.

The **Transaction Handling** tab lets you view the transaction handling features of the stage as it writes to the data source. You can select an isolation level.

If you have a single link, the **Transaction Handling** tab contains the following parameter:

- **Rows per transaction.** If **Enable transaction grouping** is cleared, you can set **Rows per transaction** to specify the number of rows written before the data is committed to the table. The default value is 0, that is, *all* the rows are written before being committed to the table.

  If you are upgrading an existing job that has a value in the **Transaction size** field on the **General** tab page, that value determines the number of rows per transaction, provided that the **Rows per transaction** field contains a value of 0.

  If the **Rows per transaction** field contains a value greater than zero, this value determines the number of rows per transaction**,** and any value in the **Transaction size** field is ignored.

  When creating a new job, use the **Rows per transaction** field to set the number of rows per transaction. Do not use the **Transaction size** field.

  Note: In previous releases of Oracle OCI, if you manually stopped a job, pending transactions were written to the database. Now pending transactions, that is, transactions that have not been committed, are rolled back.

If you have two or more links from a single Transformer stage, the **Transaction Handling** tab contains the following parameters:

- **Enable transaction grouping.** If selected, displays the grid with details of the transaction group to which the currently selected input link belongs. The check box is cleared by default.

  If **Enable transaction grouping** is selected, a transaction group can use only a value of 1 for **Rows per transaction**.
- **Input name.** The non-editable name of the input link.
- On Skip. Specifies whether to continue or to roll back the transaction if a link is skipped because of an unsatisfied constraint on it.
- **On Fail.** Specifies whether to continue or roll back if the SQL statement fails to run.

Handling transactions

You can specify transaction control information for a transaction group.

Procedure

1. Click the **Transaction Handling** tab.
2. Select **Enable transaction grouping**.
3. For transaction groups, **Rows per transaction** is automatically set to 1, and you cannot change this setting.
4. Supply necessary details about the transaction group in the grid. The grid has a line for every link in the transaction group. The links are shown in transaction processing order, which is set in the preceding Transformer stage. Each line contains the following information:
   - **Input name.** The non-editable name of the input link.
   - **On Skip.** Specifies whether to continue or to roll back the transaction if a link is skipped because of an unsatisfied constraint on it. Rows arriving at its link are skipped until the controlling link starts another transaction. Choose **Continue** or **Rollback** from the list.
   - **On Fail.** Specifies whether to continue or rollback if the SQL statement fails to execute. Choose **Continue** or **Rollback** from the list.

Reject row handling

During input link processing, rows of data might be rejected by the database for various reasons, such as unique constraint violations or data type mismatches. The Oracle OCI stage writes the row that is rejected to the log for the job.

About this task

For the Oracle message detail, you must use the error messages returned by the Oracle database.

IBM InfoSphere DataStage provides additional reject row handling.

Procedure

1. Set **Array Size** to 1.
2. Use a Transformer stage to redirect the rejected rows.

What to do next

You can design your job by selecting an appropriate target for the rejected rows, such as a Sequential stage. Reuse this target as an input source once you resolve the issues with the rejected row values.

Writing data to Oracle

You can use generated or user-defined INSERT, DELETE, or UPDATE SQL statements to write data from IBM InfoSphere DataStage to an Oracle database.

SQL statements and the Oracle OCI stage

You can create SQL statements in the Oracle OCI stage from input and output links.

From an input link, you can create INSERT statements, UPDATE statements, and DELETE statements. From an output link, you can create SELECT statements.

You have four options for creating SQL statements:

- Using the SQL builder.
- Generating statements based on the values provided to the OCI stage.
- Entering user-defined SQL statements.
- Loading SQL statements from a file at run time.

Accessing the SQL builder from a server stage

You use the SQL builder to create SQL statements by using a graphical interface.

Procedure

1. Select **Use SQL Builder tool** as the **Query Type** from the **General** tab of the input or output link or from the **SQL** tab.
2. Click the **SQL Builder** button. The SQL Builder window opens.

Writing data with generated SQL statements

By default, the IBM InfoSphere DataStage writes data to an Oracle table by using an INSERT, DELETE, or UPDATE SQLstatement that it constructs. The generated SQL statement is automatically constructed by using the InfoSphere DataStage table and column definitions that you specify in the input properties for the stage. The **SQL** tab displays the SQL statement used to write the data.

Procedure

1. Select **Generate Update actions from Options and Columns tabs** from the **Query Type** list.
2. Specify how you want the data to be written by choosing a suitable option from the **Update action** list. Select one of these options for a generated statement:
   - **Clear table then insert rows**
   - **Truncate table then insert rows**
   - **Insert rows without clearing**
   - **Delete existing rows only**
   - **Replace existing rows completely**
   - **Update existing rows only**
   - **Update existing rows or insert new rows**
   - **Insert new rows or update existing rows**
   - **User-defined SQL**
   - **User-defined SQL file**

   See "Defining Input Data" for a description of each update action.
3. Enter an optional description of the input link in the **Description** field.
4. Enter a table name in the **Table name** field on the Options page.
5. Click the **Columns** tab on the Input page. The **Columns** tab appears.
6. Edit the Columns grid to specify column definitions for the columns you want to write.

   The SQL statement is automatically constructed using your chosen update action and the columns you have specified.
7. Click the **SQL** tab on the Input page, then the **Generated** tab to view this SQL statement. You cannot edit the statement here, but you can click this tab at any time to select and copy parts of the generated statement to paste into the user-defined SQL statement.

8. Click **OK** to close the ORAOCI9 Stage dialog box. Changes are saved when you save your job design.

Writing data with user-defined SQL statements

Instead of writing data by using a SQL statement that is constructed by IBM InfoSphere DataStage, you can enter your own INSERT, DELETE, or UPDATE statement for each ORAOCI input link. Ensure that the SQL statement contains the table name, the type of update action to perform, and the columns to write to.

Procedure

1. Select **Enter custom SQL statement** from the **Query Type** list.
2. Click the **User-defined** tab on the **SQL** tab.
3. Enter the SQL statement you want to use to write data to the target Oracle tables. This statement must contain the table name, the type of update action you want to perform, and the columns you want to write. Only two SQL statements are supported for input links.

   When writing data, the INSERT statements must contain a VALUES clause with a colon (:) used as a parameter marker for each stage input column. UPDATE statements must contain SET clauses with parameter markers for each stage input column. UPDATE and DELETE statements must contain a WHERE clause with parameter markers for the primary key columns. The parameter markers must be in the same order as the associated columns listed in the stage properties. For example:

   ```
   insert emp (emp_no, emp_name) values (:1, :2)
   ```

   If you specify two SQL statements, they are executed as one transaction. Do not use a trailing semicolon.

   You cannot call stored procedures as there is no facility for parsing the row values as parameters.

   Unless you specify a user-defined SQL statement, the stage automatically generates an SQL statement.
4. Click **OK** to close the ORAOCI9 Stage dialog box. Changes are saved when you save your job design.

Defining output data

Output links specify the data that you are reading from an Oracle database. You can also specify the data on an output link by using an SQL statement that is constructed by IBM InfoSphere DataStage or a user-defined SQL statement.

These SQL statements can be:

- Fully generated, using **Use SQL Builder tool** as the **Query Type**
- Column-generated, using **Generate SELECT clause from column list; enter other clauses** as the **Query Type**
- Entered or edited entirely as text, using **Enter custom SQL statement** as the **Query Type**
- Entered from a file, using **Load SQL from a file at run time** as the **Query Type**

The SQL Builder option of fully generated SQL statements provides the most convenient method of generating SQL text. It is activated when you select **Use SQL Builder tool** as the **Query Type** (see "General Tab"). The SQL Builder dialog box contains all the information necessary to generate the SQL to extract data from an Oracle database.

The following sections describe the differences when you use SQL SELECT statements for generated or user-defined queries that you define on the Output page in the ORAOCI9 Stage window of the GUI.

The output page

The Output page has one field and the **General**, **Options**, **Columns**, and **SQL** tabs.

- **Output name.** The name of the output link. Choose the link you want to edit from the **Output name** list. This list displays all the output links from the Oracle OCI stage.
- The **Columns...** and the **View Data...** buttons function like those on the Input page.

General tab of the Output page of the Oracle OCI stage

The **General** provides the type of query and, where appropriate, a button to open an associated window.

The **General** tab contains the following fields:

- **Query type.** Displays the following options.
  - **Use SQL Builder tool.** Specifies that the SQL statement is built using the SQL Builder graphical interface. When this option is selected, the **SQL Builder** button appears. If you click **SQL Builder**, the SQL Builder opens. This is the default setting.
  - **Generate SELECT clause from column list; enter other clauses.** Specifies that InfoSphere DataStage generates the SELECT clause based on the columns you select on the **Columns** tab. When this option is selected, the **SQL Clauses** button appears. If you click **SQL Clauses**, the SQL Clauses window opens. Use this window to refine the SQL statement.
  - **Enter custom SQL statement.** Specifies that a custom SQL statement is built using the **SQL** tab..
  - **Load SQL from a file at run time.** Specifies that the data is extracted using the SQL query in the path name of the designated file that exists on the server. Enter the path name for this file instead of the text for the query. With this choice, you can edit the SQL statements.
- **Description.** Lets you enter an optional description of the output link.

SQL Clauses window

Use this window to enter FROM, WHERE, or any other SQL clauses. It contains the **Clauses** and **SQL** tabs.

- **Clauses tab.** Use this tab to build column-generated SQL queries. It contains optional SQL clauses for the conditional extraction of data. The Clauses tab is divided into three panes.
  - **FROM clause (table name):.** Allows you to name the table against which the SQL statement runs. To access **Table Definitions**, click **...** (ellipsis).
  - **WHERE clause.** Allows you to insert an SQL WHERE clause to specify criteria that the data must meet before being selected.
  - **Other clauses.** Allows you to insert a GROUP BY, HAVING, or ORDER BY clause to sort, summarize, and aggregate data.
- **SQL Tab.** Use this tab to display the SQL statements that read data from Oracle. You cannot edit these statements, but you can use **Copy** to copy them to the Clipboard for use elsewhere.

Options tab of the Output page of the Oracle OCI stage

Use this tab to specify transaction isolation, array size, prefetch memory size, and case-sensitivity.

The **Options** tab contains the following parameters:

- **Transaction Isolation.** Specifies the transaction isolation levels that provide the necessary consistency and concurrency control between transactions in the job and other transactions for optimal performance. Because Oracle does not prevent other transactions from modifying the data read by a query, that data might be changed by other transactions between two executions of the query. Thus, a transaction that executes a given query twice might experience both non-repeatable reads and phantoms. Use one for the following transaction isolation levels:
  - **Read Committed.** Takes exclusive locks on modified data and sharable locks on all other data. Each query executed by a transaction sees only data that was committed before the query (not the transaction) began. Oracle queries never read dirty, that is, uncommitted data. This is the default value.
  - **Serializable.** Takes exclusive locks on modified data and sharable locks on all other data. It sees only those changes committed when the transaction began plus those made by the transaction itself through INSERT, UPDATE, and DELETE statements. Serializable transactions do not experience non-repeatable reads or phantoms.
  - **Read-only.** Sees only those changes that were committed when the transaction began. This level does not allow INSERT, UPDATE, and DELETE statements.
- **Array size.** Specifies the number of rows read from the database at a time. Enter a positive integer to indicate the number of rows to prefetch in one call. This value is used both for prefetching rows and for array fetch. Larger numbers use more memory on the client to cache the rows. This minimizes server round trips and maximizes performance by executing fewer statements. If this number is too large, the client might run out of memory.
- **Prefetch memory setting.** Sets the memory level for top-level rows to be prefetched. See Oracle documentation for further information. Express® the value in number of bytes.
- **Disable array fetch.** Enables or disables Oracle array fetch. Array fetch is enabled by default. The value in **Array size** is used for array fetch size.
- **Enable case sensitive table/column name.** Enables the use of case-sensitive table and column names. Select to automatically enclose table and column names in SQL statements in double quotation marks (" "). It is cleared by default.

  Note: If **Enable case sensitive table/column name** is selected, when qualified column names are specified in the **Derivation** cell on the **Columns** tab, you must enclose these table and column names in double quotation marks (" ").

Columns tab of the Output page of the Oracle OCI stage

This tab contains the column definitions for the data that is output for the link that is selected.

The column tab page behaves the same way as the **Columns** tab in the ODBC stage, and it specifies which columns are aggregated.

The column definitions for output links contain a key field. Key fields are used to join primary and reference inputs to a Transformer stage. For a reference output link, the Oracle OCI key reads the data by using a WHERE clause in the SQL SELECT statement.

The **Derivation** cell on the **Columns** tab contains fully-qualified column names when table definitions are loaded from the IBM InfoSphere DataStage Repository. If the **Derivation** cell has no value, Oracle OCI uses only the column names to generate the SELECT statement displayed in the **Generated** tab of the **SQL** tab. Otherwise, it uses the content of the **Derivation** cell. Depending on the format used in the Repository, the format is *owner.table.name.columnname* or *tablename.columnname*.

The column definitions for reference links require a key field. Key fields join reference inputs to a Transformer stage. The Oracle OCI key reads the data by using a WHERE clause in the SQL SELECT statement.

SQL tab of the Output page of the Oracle OCI stage

Use this tab to build the SQL statements that are used to read data from Oracle. It contains the **Query**, **Before**, and **After** tabs.

- **Query.** This tab is read-only if you select **Use SQL Builder tool** or **Generate SELECT clause from column list; enter other clauses** for **Query Type**. If **Query Type** is **Enter Custom SQL statement**, this tab contains the SQL statements executed to read data from Oracle. The GUI displays the stage-generated SQL statement on this tab as a starting point. However, you can enter any valid, appropriate SQL statement. If **Query Type** is **Load SQL from a file at run time**, enter the path name of the file.
- **Before.** Contains the SQL statements executed before the stage processes any job data rows. The Before is the first SQL statement to be executed, and you can specify whether the job continues or aborts after failing to run a Before SQL statement. It does not affect the transaction grouping scheme. The commit/rollback is performed on a per-link basis.

  If the property value begins with FILE=, the remaining text is interpreted as a path name, and the contents of the file supplies the property value.
- **After.** Contains the After SQL statement executed after the stage processes any job data rows. It is the last SQL statement to be executed, and you can specify whether the job continues or aborts after failing to run an After SQL statement. It does not affect the transaction grouping scheme. The commit/rollback is performed on a per-link basis.

  If the property value begins with FILE=, the remaining text is interpreted as a path name, and the contents of the file supplies the property value.

Reading data from Oracle

You can use generated queries or user-defined queries to read data from an Oracle database into IBM InfoSphere DataStage.

The column definitions for reference links must contain a key field. You use key fields to join primary and reference inputs to a Transformer stage.

Oracle OCI key reads the data by using a WHERE clause in SQL SELECT statements.

Using generated queries

The Oracle OCI stage extracts data from an Oracle data source by using a complete SQL SELECT statement that it generates. The SQL statement is automatically constructed by using the information that you enter in the stage output properties.

Procedure

1. Select **Generate SELECT clause from column list; enter other clauses**. Data is extracted from an Oracle database by using an SQL SELECT statement that is generated by the Oracle OCI stage. Also, the **SQL Clauses** button appears.
2. Click **SQL Clauses**. The SQL Clauses window opens.

   SQL SELECT statements have the following syntax:

```
SELECT clause FROM clause
[WHERE clause]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause];
```

Example of an SQL SELECT statement

The SQL SELECT statement includes other appropriate clauses based on your entries in the **FROM clause (table name)**, **WHERE clause**, and **Other clauses** fields in the SQL Clauses window.

For example, you can specify values to complete the following tasks:

- Select the columns **Name**, **Address**, **City**, **State**, **AreaCode**, and **Telephone Number** from a table called Table1
- Specify the value of **AreaCode** to be 617 in the **Where clause** text box
- Specify **City** as the column to order by in the **Other clauses** text box

  The SQL statement displayed on the **SQL** tab is:

```
SELECT Name, Address, City, State, AreaCode, Telephone
FROM Table1 WHERE AreaCode = 617 ORDER BY City;
```

Using user-defined queries

Instead of using the SQL statement that is generated by the Oracle OCI stage, you can enter your own SQL statement for each Oracle OCI output link.

Procedure

1. Select **Enter custom SQL statement** from the **Query type** list on the **General** tab on the Output page. The **SQL** tab is enabled.
2. You can edit or drag the selected columns into your user-defined SQL statement. Only one SQL statement is supported for an output link. You must ensure that the table definitions for the output link are correct and represent the columns that are expected.
3. If your entry begins with {FILE}, the remaining text is interpreted as a path name, and the contents of the file supplies the text for the query.
4. Click **OK** to close this window. Changes are saved when you save your job design.

DATE data type considerations

An Oracle DATE data type contains date and time information (there is no TIME data type in Oracle). When you import Oracle metadata that has the DATE data type, the Oracle OCI stage maps the Oracle DATE data type to a Timestamp data type by default.

InfoSphere DataStage uses a conversion of *YYYY-MM-DD HH*24:*MI*:*SS* when reading or writing an Oracle date. If the InfoSphere DataStage data type is Timestamp, InfoSphere DataStage uses the **to_timestamp** function for this column when it generates the INSERT statement to write an Oracle date. If the InfoSphere DataStage data type is Timestamp or Date, the InfoSphere DataStage uses the **to_char** function for this column when it generates the SELECT statement to read an Oracle date.

The following example creates a table with a DATE data type on an Oracle server. The imported InfoSphere DataStage data type is Timestamp.

```
create table dsdate (one date);
```

The results vary, depending on whether the Oracle OCI stage is used as an input or an output link:

- **Input link.** The stage generates the following SQL statement:

  ```
  insert into dsdate(one) values(TO_DATE(:1, 'yyyy-mm-dd hh24:mi:ss'))
  ```

- **Output link.** The stage generates the following SQL statement:

  ```
  select TO_CHAR(one, 'YYYY-MM-DD HH24:MI:SS') FROM dsdate
  ```

Oracle data type support

The Oracle OCI stage supports character, numeric, date, and miscellaneous data types. When you create the IBM InfoSphere DataStage table definitions for an Oracle table, specify the SQL type, length, and scale attributes that are appropriate for the data type.

Character data types

The Oracle OCI stage supports the CHAR and VARCHAR2 Oracle data types.

The following table summarizes character data types for Oracle, their IBM InfoSphere DataStage SQL type definitions, and the corresponding length attributes that you need to specify:

Table 31. Oracle character data types and InfoSphere DataStage corresponding data types

Oracle data type	InfoSphere DataStage SQL type	Length	Notes®
CHAR (*size*)	Char (*size)*	size	Fixed length character data of length *size*. Fixed for every row in the table (with trailing spaces). Maximum size is 255 bytes per row, default size is 1 byte per row.

Table 31. Oracle character data types and InfoSphere DataStage corresponding data types (continued)

Oracle data type	InfoSphere DataStage SQL type	Length	Notes®
VARCHAR2 (*size*)	VarChar (*size)*	size	Variable length character data. A maximum *size* must be specified. VarChar is variable for each row, up to 2000 bytes per row.

Numeric data types

The Oracle OCI steage supports the NUMBER Oracle data type.

The following table summarizes the NUMBER data type for Oracle, the IBM InfoSphere DataStage SQL type definitions, and the corresponding length and scale attributes that you need to specify:

Table 32. Oracle numeric data types and InfoSphere DataStage corresponding data types

Oracle data type	InfoSphere DataStage SQL Ttype	Length	Scale	Notes
NUMBER (*p*,*s*)	Decimal Double Float Numeric Integer Real	p *p*	s *s*	The InfoSphere DataStage SQL type definition used depends on the application of the column in the table, that is, how the column is used. Decimal values have a maximum precision of 38 digits. Decimal and Numeric are synonyms. The full range of Oracle *NUMBER* values are supported without loss of precision.

Additional numeric data types for Oracle

The Oracle OCI stage supports the BINARY_DOUBLE and BINARY_FLOAT Oracle data types.

The following table summarizes the additional numerical data types for Oracle and their IBM InfoSphere DataStage SQL type definitions:

Table 33. Additional numeric data types and the corresponding data type in InfoSphere DataStage

Oracle data type	InfoSphere DataStage SQL type	Notes
BINARY_DOUBLE	Double	• When a table is read, the InfoSphere DataStage converts columns with a data type of BINARY_DOUBLE to SQL_DOUBLE. • When a table is updated, InfoSphere DataStage converts columns with a data type of SQL_DOUBLE to BINARY_DOUBLE. **Note:** Perform the following steps to determine the data type of the source column. When importing metadata definitions, select **Import** > **Table Definitions** > **Plug-in Meta Data Definitions**. Select **ORAOCI9**. If you select **Include Column Description**, the metadata import includes the description column on the **Columns** tab.

Table 33. Additional numeric data types and the corresponding data type in InfoSphere DataStage (continued)

Oracle data type	InfoSphere DataStage SQL type	Notes
BINARY_FLOAT	Float	• When a table is read, InfoSphere DataStage converts columns with a data type of either BINARY_FLOAT or FLOAT to SQL_FLOAT. **Note:** Perform the following steps to determine the data type of the source column. When importing metadata definitions, select **Import** > **Table Definitions** > **Plug-in Meta Data Definitions**. Select **ORAOCI9**. If you select **Include Column Description**, the metadata import includes the description column on the **Columns** tab. • When a table is updated, InfoSphere DataStage converts SQL_FLOAT to either BINARY_FLOAT or FLOAT. To indicate BINARY_FLOAT, place the keyword **BINARY_FLOAT** anywhere in the column description field on the **Columns** tab. If **BINARY_FLOAT** is present, InfoSphere DataStage converts SQL_FLOAT to BINARY_FLOAT. If **BINARY_FLOAT** is not present, InfoSphere DataStage converts SQL_FLOAT to FLOAT (with precision).

Date data types

The Oracle OCI stage supports the DATE Oracle data type.

The following table summarizes the DATE data type for Oracle and the IBM InfoSphere DataStage SQL type definition:

Table 34. Oracle date data types and the InfoSphere DataStage corresponding data types

Oracle data type	InfoSphere DataStage SQL type	Notes
DATE	Timestamp	The default format for the default InfoSphere DataStage data type Timestamp is *YYYY-MM-DD HH*24:*MI*:*SS*. If theInfoSphere DataStage data type is Timestamp, InfoSphere DataStage uses the **to_date** function for this column when it generates the INSERT statement to write an Oracle date. If the InfoSphere DataStage data type is Timestamp or Date,InfoSphere DataStage uses the **to_char** function for this column when it generates the SELECT statement to read an Oracle date.

Miscellaneous data types

The Oracle OCI stage supports the CLOB Oracle data type.

The following table summarizes miscellaneous data types for Oracle and IBM InfoSphere DataStage:

Table 35. Oracle miscellaneous data types and the InfoSphere DataStage's corresponding data types

Oracle data type	InfoSphere DataStage SQL type	Notes
CLOB	SQL_LONGVARCHAR	The Oracle OCI stage supports the CLOB data type by mapping the LONGVARCHAR data type with a precision greater than 4 KB to Oracle's CLOB data type. To work with a CLOB column definition, select the InfoSphere DataStage LONGVARCHAR data type as the column's data type and provide a Length of more than 4 KB in the **Columns** tab. The maximum size supported by InfoSphere DataStage is 2 GB. A column with a data type of CLOB cannot be used as a key.

Handling $ and # characters

In jobs that use the Oracle OCI stage, you can connect to Oracle OCI databases that use the number sign (#) and dollar sign ($) characters in table names and column names. InfoSphere DataStage converts these characters into an internal format and then converts them back as necessary.

About this task

To take advantage of this facility, perform the following tasks:

- In the IBM InfoSphere DataStage and QualityStage Administrator client, open the Environment Variables dialog box for the project in question, and set the environment variable DS_ENABLE_RESERVED_CHAR_CONVERT to true (this can be found in the General\Customize branch).
- Avoid using the strings __035__ and __036__ in your Oracle table or column names. __035__ is the internal representation of # and __036__ is the internal representation of $.

Import metadata using the stage Meta Data Import tool; avoid hand-editing (this minimizes the risk of mistakes or confusion).

Once the table definition is loaded, the internal table and column names are displayed rather than the original Oracle names both in table definitions and in the Data Browser. They are also used in derivations and expressions. The original names (that is, those containing the $ or #) are used in generated SQL statements, however, and you should use them if entering SQL in the job yourself.

When using an Oracle OCI stage in a server job, you should use the external names when entering SQL statements that contain Oracle columns. The columns within the stage are represented by :1, ;2, and onward. (parameter markers) and bound to the Oracle columns by order, so you do not need to worry about entering names for them. This applies to:

- Query
- Update
- Insert
- Key
- Select
- Where clause

For example, for an update you might enter:

```
UPDATE tablename SET ##B$ = :1 WHERE  $A# = :2
```

Particularly note the key in this statement ($A#) is specified using the external name.

Chapter 6. Oracle OCI Load stage

The Oracle OCI Load stage is a passive stage that loads data from external files into an Oracle table. The Oracle database can reside locally or remotely.

When you use IBM InfoSphere DataStage to access Oracle databases, you can choose from a collection of connectivity options. For most new jobs, use the Oracle Connector stage, which offers better functionality and performance than the Oracle OCI Load stage.

If you have jobs that use the Oracle OCI Load stage and want to use the connector, use the Connector Migration Tool to migrate jobs to use the connector.

This stage has one stream input link and no output or output reference links. The input link provides a stream of data rows to load into the Oracle table using Oracle direct path loading. This input link corresponds to one bulk loading session in an IBM InfoSphere DataStage job. You have the option to use different loading modes.

Oracle Call Interface (OCI) supports direct path loading calls that access the direct block formatter of the Oracle server. These calls perform the functions of the Oracle SQL*Loader utility. This lets you load data immediately from an external file into an Oracle database object, which is a table or a partition of a partitioned table, in automatic mode.

Functionality of the Oracle OCI Load stage

You can use the Oracle OCI Load stage to load data to a target table. The stage also has national language support.

The Oracle OCI Load stage has the following functionality:

- Bulk loading from a stream input link to provide rows of data into the target table residing locally or remotely.
- Immediate and delayed loading.
- Load actions to specify how data is loaded to the target table.
- Partition or table loading.
- NLS (National Language Support).

The following functionality is not supported:

- Output or output reference links.
- Importing of table definitions.
- Use of the TIMESTAMP data type with fractions of seconds, for example, *hh:mm:ss:ff*. Use the CHAR data type instead.

Configuration requirements of the Oracle OCI Load stage

To use the Oracle OCI Load stage, ensure that the configuration requirements are met.

See the online readme.txt file for your platform for the latest information about the IBM InfoSphere DataStage release.

Before you use the Oracle OCI Load stage, you must install the Oracle standard client on the engine tier. You cannot use the stage if only Oracle Instant Client is installed.

Note: For Oracle direct path load, the client version must be the same as or earlier than the server version. You should have read and execute permissions to use libraries in the $ORACLE_HOME/lib and $ORACLE_HOME/bin directories and read permissions on all files in the $ORACLE_HOME directory. Otherwise, you might experience problems using Oracle OCI Load stage to connect to Oracle.

Operating system requirement

For the Oracle OCI Load stage to run successfully, the Oracle client and server computers must have the same operating system type, such as UNIX or Windows 2000. For example, if the Oracle client is on a UNIX computer and the Oracle server is on a Windows 2000 computer, jobs fail.

Oracle Enterprise Manager

If you install Oracle Enterprise Manager on the same workstation as Oracle Client, the Oracle server home directory must precede the Oracle Enterprise Manager home directory. You must ensure that the PATH system environment variable has the correct setting.

For example, the following setting is correct:

```
d:\ oraclehome \bin;d:\ oraclemanager \bin
```

oraclehome is the location where your Oracle software is installed.

oraclemanager is the name of the Oracle Enterprise Manager home directory.

Any changes to system environment variables might require a system reboot before the values of the variables take effect.

The configuration of SQL*Net using a configuration program, for example, SQL*Net Easy Configuration, to set up and add database aliases is also required.

Load modes

The load mode specifies whether to load the data into the target file in automatic or manual mode. The **Load Mode** property specifies whether to populate the Oracle database immediately or to generate a control file and a data file to populate the database later.

Automatic load mode

Automatic loading, which is the default load mode, loads the data during the IBM InfoSphere DataStage job. The stage populates the Oracle database immediately after reading the source data.

Automatic data loading can occur only when the InfoSphere DataStage server resides on the same system as the Oracle server or when the Oracle server is remote and has the same operating system as the InfoSphere DataStage server.

Manual load mode

Use manual loading to modify and move the data file, the control file, or both to a different system before the actual loading process.

Use manual mode to delay loading the data, which causes the data and control files that are required to load the data to be written to an ASCII file. The data and control files are used to load the data later.

Loading an Oracle database

You can use the Oracle OCI Load stage to load data to an Oracle database.

Procedure

1. Add an Oracle OCI Load stage to an InfoSphere DataStage job
2. Link the Oracle OCI Load stage to its data source
3. Specify column definitions using the **Columns** tab
4. Choose the load mode.
5. On the **Stage** tab, configure the properties for your job.
6. Compile the job. If the job does not compile correctly, correct the errors and recompile.
7. Run the job in the InfoSphere DataStage and QualityStage Designer client or run or schedule the job in the InfoSphere DataStage and QualityStage Director client.

Properties

Use the **Properties** tab to specify the load operation.

Each stage property is described in the order in which it appears.

Prompt	Type	Default	Description
Service Name	String		The name of the Oracle service. It is the logical representation of the database, which is the way the database is presented to clients. The service name is a string that is the global database name, a name consists of the database name and domain name, which is entered during installation or database creation.
User Name	String		The user name for connecting to the service.
Password	String		The password for "User Name."
Table Name	String		The name of the target Oracle table to load the files into.

Prompt	Type	Default	Description
Schema Name	String		The name of the schema where the table being loaded resides. If unspecified, the schema name is "User Name."
Partition Name	String		The name of the partition or subpartition that belongs to the table to be loaded. If not specified, the entire table is loaded. The name must be a valid partition or subpartition name.
Date Format	String List	DD-*MON-YYYY*	The date format to be used. Use one of the following values: *DD.MM.YYYY* *YYYY-MM-DD* *DD-MON-YYYY* *MM/DD/YYYY*
Time Format	String List	hh24:mi:ss	The time format to be used. Use one of the following values: *hh24:mi:ss* *hh:mi:ss am*
Max Record Number	Long	100	Specifies the maximum number of input records in a batch. This property is used only if "Load Mode" is set to Automatic.

Prompt	Type	Default	Description
Load Mode	String List	Automatic	The method used to load the data into the target file. This property specifies whether to populate the Oracle database or generate a control file and a data file to populate the database. Use one of the following values: **Automatic (immediate mode).** The stage populates an Oracle database immediately after loading the source data. Automatic data loading can occur only when the IBM InfoSphere DataStageserver resides on the same system as an Oracle server. **Manual (delayed mode).** The stage generates a control file and a data file that you can edit and run on any Oracle host system. The stage does not establish a connection with the Oracle server.
Directory Path	String		The path name of the directory where the Oracle SQL*Loader files are generated. This property is used only when "Load Mode" is set to Manual.

Prompt	Type	Default	Description
Control File Name	String	*servicename_ tablename.ctl*	The name of the Oracle SQL*Loader control file generated when "Load Mode" is set to Manual. This text file contains the sequence of commands telling where to find the data, how to parse and interpret the data, and where to insert the data. You can modify and execute this file on any Oracle host system. This file has a *.ctl* extension.
Data File Name	String	*servicename_ tablename.dat*	The name of the Oracle SQL*Loader sequential data file created when "Load Mode" is set to Manual. This file has a *.dat* extension.
Delimiter	String	, (comma)	The character used to delimit fields in the loader input data.
Preserve Blanks	String List	No	The indicator specifying whether SQL*Loader should preserve blanks in the data file. If **No**, SQL*Loader treats blanks as nulls.
Column Name Case-sensitivity	String List	No	The indicator specifying whether both uppercase and lowercase characters can be used in column names. If **No**, all column names are handled as uppercase. If **Yes**, a combination of uppercase and lowercase characters is acceptable.

Chapter 7. Building SQL statements

Use the graphical interface of SQL builder to construct SQL statements that run against databases.

You can construct the following types of SQL statements.

Table 36. SQL statement types

SQL statement	Description
SELECT	Selects rows of data from a database table. The query can perform joins between multiple tables and aggregations of values in columns.
INSERT	Inserts rows in a database table.
UPDATE	Updates existing rows in a database table.
DELETE	Deletes rows from a database table.

You can use the SQL builder from various connectivity stages that IBM InfoSphere DataStage supports.

Different databases have slightly different SQL syntax (particularly when it comes to more complex operations such as joins). The exact form of the SQL statements that the SQL builder produces depends on which stage you invoke it from.

You do not have to be an SQL expert to use the SQL builder, but it helps to have some familiarity with the basic structure of SQL statements in this documentation.

Avoid using column names that are SQL reserved words as their use might result in unexpected results when the SQL is built or run.

Starting SQL builder from a stage editor

If a stage supports the SQL builder, you can open the SQL builder by clicking **Build SQL** in the stage editor. For some stages, you can use the SQL builder only for some access methods.

The SQL builder is available to help you build select statements where you are using a stage to read a database (that is, a stage with an output link).

The SQL builder is available to help you build insert, update, and delete statements where you are using the stage to write to database (that is, a stage with an input link).

Starting SQL builder

Use the graphical interface of SQL builder to construct SQL queries that run against federated databases.

Procedure

1. In the Reference Provider pane, click **Browse**. The Browse Providers dialog box opens.
2. In the **Select a Reference Provider** type list, select **Federation Server**. In the Select a Federated Datasource tree, the list of database aliases opens.
3. Click a database alias. The list of schemas opens as nodes beneath each database alias.
4. In the **SQL Type** list, select the type of SQL query that you want to construct.
5. Click the **SQL builder** button. The SQL Builder - DB2 / UDB 8.2 window opens. In the Select Tables pane, the database alias appears as a node.

Building SELECT statements

Build SELECT statements to query database tables and views.

Procedure

1. Click the **Selection** tab.
2. Drag any tables you want to include in your query from the repository tree to the canvas. You can drag multiple tables onto the canvas to enable you to specify complex queries such as joins. You must have previously placed the table definitions in the IBM InfoSphere DataStage repository. The easiest way to do this is to import the definitions directly from your relational database.
3. Specify the columns that you want to select from the table or tables on the column selection grid.
4. If you want to refine the selection you are performing, choose a predicate from the **Predicate** list in the filter panel. Then use the expression editor to specify the actual filter (the fields displayed depend on the predicate you choose). For example, use the Comparison predicate to specify that a column should match a particular value, or the Between predicate to specify that a column falls within a particular range. The filter appears as a WHERE clause in the finished query.
5. Click the **Add** button in the filter panel. The filter that you specify appears in the filter expression panel and is added to the SQL statement that you are building.
6. If you are joining multiple tables, and the automatic joins inserted by the SQL builder are not what is required, manually alter the joins.
7. If you want to group your results according to the values in certain columns, select the Group page. Select the Grouping check box in the column grouping and aggregation grid for the column or columns that you want to group the results by.
8. If you want to aggregate the values in the columns, you should also select the Group page. Select the aggregation that you want to perform on a column from the **Aggregation** drop-down list in the column grouping and aggregation grid.
9. Click on the **Sql** tab to view the finished query, and to resolve the columns generated by the SQL statement with the columns loaded on the stage (if necessary).

Building INSERT statements

Build INSERT statements to insert rows in a database table.

Procedure

1. Click the **Insert** tab.
2. Drag the table you want to insert rows into from the repository tree to the canvas. You must have previously placed the table definitions in the IBM InfoSphere DataStage repository. The easiest way to do this is to import the definitions directly from your relational database.
3. Specify the columns that you want to insert on the column selection grid. You can drag selected columns from the table, double-click a column, or drag all columns.
4. For each column in the column selection grid, specify how values are derived. You can type a value or select a derivation method from the drop-down list.
   - **Job Parameters**. The Parameter dialog box appears. Select from the job parameters that are defined for this job.
   - **Lookup Columns**. The Lookup Columns dialog box appears. Select a column from the input columns to the stage that you are using the SQL builder in.
   - **Expression Editor**. The Expression Editor opens. Build an expression that derives the value.
5. Click on the **Sql** tab to view the finished query.

Building UPDATE statements

Build UPDATE statements to update existing rows in a database table.

Procedure

1. Click the **Update** tab.
2. Drag the table whose rows you want to update from the repository tree to the canvas. You must have previously placed the table definitions in the IBM InfoSphere DataStage repository. The easiest way to do this is to import the definitions directly from your relational database.
3. Specify the columns that you want to update on the column selection grid. You can drag selected columns from the table, double-click a column, or drag all columns.
4. For each column in the column selection grid, specify how values are derived. You can type a value or select a derivation method from the drop-down list. Enclose strings in single quotation marks.
   - **Job Parameters**. The Parameter dialog box appears. Select from the job parameters that are defined for this job.
   - **Lookup Columns**. The Lookup Columns dialog box appears. Select a column from the input columns to the stage that you are using the SQL builder in.
   - **Expression Editor**. The Expression Editor opens. Build an expression that derives the value.
5. If you want to refine the update you are performing, choose a predicate from the **Predicate** list in the filter panel. Then use the expression editor to specify the actual filter (the fields displayed depend on the predicate you choose). For example, use the Comparison predicate to specify that a column should match a particular value, or the Between predicate to specify that a column falls within a particular range. The filter appears as a WHERE clause in the finished statement.
6. Click the **Add** button in the filter panel. The filter that you specify appears in the filter expression panel and is added to the update statement that you are building.

7. Click on the **Sql** tab to view the finished query.

Building DELETE statements

Build DELETE statements to delete rows from a database table.

Procedure

1. Click the **Delete** tab.
2. Drag the table from which you want to delete rows from the repository tree to the canvas. You must have previously placed the table definitions in the IBM InfoSphere DataStage repository. The easiest way to do this is to import the definitions directly from your relational database.
3. You must choose an expression which defines the rows to be deleted. Choose a predicate from the **Predicate** list in the filter panel. Then use the expression editor to specify the actual filter (the fields displayed depend on the predicate you choose). For example, use the Comparison predicate to specify that a column should match a particular value, or the Between predicate to specify that a column falls within a particular range. The filter appears as a WHERE clause in the finished statement.
4. Click the **Add** button in the filter panel. The filter that you specify appears in the filter expression panel and is added to the update statement that you are building.
5. Click on the **Sql** tab to view the finished query.

The SQL builder interface

The components in the upper half of the SQL builder are common to all types of SQL statement that you can build. The pages that are available in the lower half depend on the type of query that you build.

Toolbar

The toolbar for the SQL builder contains tools for actions such as clearing the current query, viewing data, and validating the statement.

The SQL builder toolbar contains the following tools.

- **Clear Query** removes the field entries for the current SQL query.
- **Cut** removes items and placed them on the Microsoft Windows clipboard so they can be pasted elsewhere.
- **Copy** copies items and place them on the Windows clipboard so they can be pasted elsewhere.
- **Paste** pastes items from the Windows clipboard to certain places in the SQL builder.
- **SQL properties** opens the Properties dialog box.
- **Quoting** toggles quotation marks in table and column names in the generated SQL statements.
- **Validation** toggles the validation feature. Validation automatically occurs when you click OK to exit the SQL builder.
- **View Data** is available when you invoke the SQL builder from stages that support the viewing of data. It causes the calling stage to run the SQL as currently built and return the results for you to view.
- **Refresh** refreshes the contents of all the panels on the SQL builder.

- **Window View** allows you to select which panels are shown in the SQL builder window.
- **Help** opens the online help.

Tree panel

The tree panel shows the table definitions in the IBM InfoSphere DataStage repository. You can import a table definition from the database that you want to query.

You can import the table definition by using the Designer client, or you can do it directly from the shortcut menu in the tree panel. You can also manually define a table definition from within the SQL builder by selecting **New Table...** from the tree panel shortcut menu.

To select a table to query, select it in the tree panel and drag it to the table selection canvas. A window appears in the canvas representing the table and listing all its individual columns.

A shortcut menu allows you to:

- Refresh the repository view
- Define a new table definition (the Table Definition dialog box opens)
- Import metadata directly from a data source (a sub menu offers a list of source types)
- Copy a table definition (you can paste it in the table selection canvas)
- View the properties of the table definition (the Table Definition dialog box opens)

You can also view the properties of a table definition by double-clicking on it in the repository tree.

Table selection canvas

The table selection canvas shows a list of columns and column types for the table that the SQL statement accesses.

You can drag a table from the tree panel to the table selection canvas. If the desired table does not exist in the repository, you can import it from the database you are querying by choosing **Import Metadata** from the tree panel shortcut menu.

The table appears in a window on the canvas, with a list of the columns and their types. For insert, update, and delete statements you can only place one table on the canvas. For select queries you can place multiple tables on the canvas.

Wherever you try to place the table on the canvas, the first table you drag will always be placed in the top left hand corner. If you are building a select query, subsequent tables can be dragged before or after the initial, or on a new row underneath. Eligible areas are highlighted on the canvas as you drag the table, and you can only drop a table in one of the highlighted areas. When you place tables on the same row, the SQL builder will automatically join the tables (you can alter the join if it's not what you want).

When you place tables on a separate row, no join is added. An old-style Cartesian product of the table rows on the different rows is produced: `FROM FirstTable, SecondTable`.

Click the **Select All** button underneath the table title bar to select all the columns in the table. Alternatively you can double-click on or drag individual columns from the table to the grid in the **Select**, **Insert**, or Update page to use just those columns in your query.

With a table selected in the canvas, a shortcut menu allows you to:

- Add a related table (select queries only). A submenu shows you tables that have a foreign key relationship with the currently selected one. Select a table to insert it in the canvas, together with the join expression inferred by the foreign key relationship.
- Remove the selected table.
- Select all the columns in the table (so that you could, for example, drag them all to the column selection grid).
- Open a Select Table dialog box to allow you to bind an alternative table for the currently selected table (select queries only).
- Open the **Table Properties** dialog box for the currently selected table.

With a join selected in the canvas (select queries only), a shortcut menu allows you to:

- Open the Alternate Relation dialog box to specify that the join should be based on a different foreign key relationship.
- Open the Join Properties dialog box to modify the type of join and associated join expression.

From the canvas background, a shortcut menu allows you to:

- Refresh the view of the table selection canvas.
- Paste a table that you have copied from the tree panel.
- View data - this is available when you invoke the SQL builder from stages that support the viewing of data. It causes the calling stage to run the SQL as currently built and return the results for you to view.
- Open the Properties dialog box to view details of the SQL syntax that the SQL builder is currently building a query for.

Selection page

Use the Selection page to specify details for a SELECT statement.

Column selection grid

Use the column selection grid to specify the columns to include in your query.

You can populate the grid in a number of ways:

- Drag columns from the tables in the table selection canvas
- Choose columns from a list in the grid
- Double-click the column name in the table selection canvas
- Copy and paste from the table selection canvas

Column expression

The column expression identifies the columns to include in the SELECT statement.

You can specify the following parts:

- **Job parameter**. A dialog box appears offering you a choice of available job parameters. This allows you to specify the value to be used in the query at run time (the stage you are using the SQL builder from must allow job parameters for this to appear).
- **Expression**. An expression editor dialog box appears, allowing you to specify an expression that represents the value to be used in the query.
- **Data flow variable**. A dialog box appears offering you a choice of available data flow variables (the stage you are using the SQL builder from must support data flow variables for this to appear)
- **Lookup Column**. You can directly select a column from one of the tables in the table selection canvas.

Table

This property identifies the table that the column belongs to.

If you populate the column grid by dragging, copying or double-clicking on a column from the table selection canvas, the table name is filled in automatically. You can also choose a table from the list.

To specify the table name at run time, choose a job parameter from the list.

Column alias

Use this property to specify an alias for the column.

Output

Select this property to indicate that the column is part of the query output. The property is selected automatically when you add a column to the grid.

Sort

Choose **Ascending** or **Descending** to have the query sort the returned rows by the value of this column. Selecting to sort adds an ORDER BY clause to the query.

Sort order

You can specify the order in which rows are sorted if you order by more than one column.

Shortcut menu

Use the shortcut menu to paste a column that you copied from the table selection canvas, insert or remove a row, and show or hide the filter panel.

Filter panel

In the filter panel, you specify a WHERE clause for the SELECT statement that you are building. The filter panel includes a predicate list and an expression editor panel, the contents of which depends on the chosen predicate.

Filter expression panel

The filter expression panel shows the filters that you added to the query. You can edit a filter that you added by using the filter expression editor or you can enter a filter manually.

Group page

Use the Group page, which appears when you build SELECT statements, to specify that the results of the query are grouped by a column or columns.

Also, you can use the page to aggregate the results in some of the columns. For example, you can specify COUNT to count the number of rows that contain a non-null value in a column.

The **Group** tab gives access to the toolbar, tree panel, and the table selection canvas, in exactly the same way as the Selection page.

Grouping grid

In the grouping grid, you can specify the columns to group by or aggregate on.

The grid is populated with the columns that you selected on the Selection page. You can change the selected columns or select new ones, which will be reflected in the selection your query makes.

The grid has the following fields:

- **Column expression**. Identifies the column to be included in the query. You can modify the selections from the Selection page, or build a column expression.
  - Job parameter. A dialog box appears offering you a choice of available job parameters. This allows you to specify the value to be used in the query at run time (the stage you are using the SQL builder from must allow job parameters for this to appear).
  - Expression Editor. An expression editor dialog box appears, allowing you to specify an expression that represents the value to be used in the query.
  - Data flow variable. A dialog box appears offering you a choice of available data flow variables (the stage you are using the SQL builder from must support data flow variables for this to appear).
  - Lookup Column. You can directly select a column from one of the tables in the table selection canvas.
- **Column Alias**. This allows you to specify an alias for the column. If you select an aggregation operation for a column, SQL builder will automatically insert an alias of the form Alison; you can edit this if required.
- **Output**. This is selected to indicate that the column will be output by the query. This is automatically selected when you add a column to the grid.
- **Distinct**. Select this check box if you want to add the DISTINCT qualifier to an aggregation. For example, a COUNT aggregation with the distinct qualifier will count the number of rows with distinct values in a field (as opposed to just the not-null values). For more information about the DISTINCT qualifier, see SQL Properties Dialog Box.
- **Aggregation**. Allows you to select an aggregation function to apply to the column (note that this is mutually exclusive with the Group By option). See Aggregation Functions for details about the available functions.
- **Group By**. Select the check box to specify that query results should be grouped by the results in this column.

Aggregation functions

The aggregation functions that are available depend on the stage that you opened the SQL builder from. All SQL syntax variants include the AVG, COUNT, MAX, MIN, STDDEV, and VARIANCE aggregation functions.

The following aggregation functions are supported.

- AVG. Returns the mean average of the values in a column. For example, if you had six rows with a column containing a price, the six rows would be added together and divided by six to yield the mean average. If you specify the

DISTINCT qualifier, only distinct values will be averaged; if the six rows only contained four distinct prices then these four would be added together and divided by four to produce a mean average.

- COUNT. Counts the number of rows that contain a not-null value in a column. If you specify the DISTINCT qualifier, only distinct values will be counted.
- MAX. Returns the maximum value that the rows hold in a particular column. The DISTINCT qualifier can be selected, but has no effect on this function.
- MIN. Returns the minimum value that the rows hold in a particular column. The DISTINCT qualifier can be selected, but has no effect on this function.
- STDDEV. Returns the standard deviation for a set of numbers.
- VARIANCE. Returns the variance for a set of numbers.

Filter panel

In the Filter panel, you can specify a HAVING clause for the SELECT statement. The Filter panel includes a predicate list and an expression editor panel, the contents of which depends on the chosen predicate.

Filter Expression panel

The Filter Expression panel shows the filters that you added to the query. You can edit a filter that you added by using the filter expression editor, or you can enter a filter manually.

Insert page

Use the Insert page to specify the details of an INSERT statement. The page includes the insert columns grid.

Insert Columns grid

In the Insert Columns grid, you specify the columns to include in the INSERT statement and the values that they will take.

Insert column

This property identifies the columns to include in the INSERT statement.

You can populate this in a number of ways:

- Drag columns from the table in the table selection canvas
- Choose columns from a list in the grid
- Double-click the column name in the table selection canvas
- Copy and paste from the table selection canvas

Insert value

This property identifies the values that you are setting the corresponding column to. You can enter a value manually or specify a job parameter, expression, data flow variable, or lookup column.

When you specify a value, you can use the following objects:

- **Job parameter**. A dialog box appears offering you a choice of available job parameters. This allows you to specify the value to be used in the query at run time (the stage you are using the SQL builder from must allow job parameters for this to appear).
- **Expression**. An expression editor dialog box appears, allowing you to specify an expression that represents the value to be used in the query.

- **Data flow variable**. A dialog box appears offering you a choice of available data flow variables (the stage you are using the SQL builder from must support data flow variables for this to appear)
- **Lookup column**. You can directly select a column from one of the tables in the table selection canvas.

Update page

Use the Update page to specify details of an UPDATE statement.

Update Column grid

In the Update Column grid, you specify the columns to include in the UPDATE statement and the values that they will take.

Update column

This property identifies the columns to include in the UPDATE statement.

You can populate this in the following ways:

- Drag columns from the table in the table selection canvas.
- Choose columns from a list in the grid.
- Double-click the column name in the table selection canvas.
- Copy and paste from the table selection canvas.

Update value

This property identifies the value that you are setting the corresponding column to. You can enter a value in the field manually, or you can specify a job parameter, expression, data flow variable, or lookup column.

You can specify the following objects:

- **Job parameter**. A dialog box appears offering you a choice of available job parameters. This allows you to specify the value to be used in the query at run time (the stage you are using the SQL builder from must allow job parameters for this to appear).
- **Expression**. An expression editor dialog box appears, allowing you to specify an expression that represents the value to be used in the query.
- **Data flow variable**. A dialog box appears offering you a choice of available data flow variables (the stage you are using the SQL builder from must support data flow variables for this to appear)
- **Lookup column**. You can directly select a column from one of the tables in the table selection canvas.

Filter panel

In the filter panel, you can specify a WHERE clause for the UPDATE statement that you build. The filter panel includes a predicate list and an expression editor panel, the contents of which depends on the chosen predicate.

Filter expression panel

The filter expression panel shows the filters that you added for the query. You can edit the filter in the panel or enter a filter manually.

Delete page

On the Delete page, you specify details for the DELETE statement that you build.

Filter panel

On the filter panel, you can specify a WHERE clause for the DELETE statement that you build. The filter panel includes a predicate list and an expression editor panel, the contents of which depend on the chosen predicate.

Filter expression panel

The filter expression panel shows the filters that you add to the query. You can edit a filter in the panel or enter a filter manually.

SQL page

On the SQL page, you view the SQL statement that you build.

For SELECT queries, if the columns that you defined as output columns for your stage do not match the columns that the SQL statement is generating, use the Resolve columns grid to reconcile them. In most cases, the columns match.

Resolve columns grid

If the columns that you loaded in a stage do not match the columns that are generated by the SQL statement that you built, you can reconcile the differences in the Resolve columns grid.

Ideally the columns should match (and in normal circumstances usually would). A mismatch would cause the metadata in your job to become out of step with the metadata as loaded from your source database (which could cause a problem if you are performing usage analysis based on that table).

If there is a mismatch, the grid displays a warning message. Click the Auto Match button to resolve the mismatch. You are offered the choice of matching by name, by order, or by both. When matching, the SQL builder seeks to alter the columns generated by the SQL statement to match the columns loaded onto the stage.

If you choose Name matching, and a column of the same name with a compatible data type is found, the SQL builder:

- Moves the result column to the equivalent position in the grid to the loaded column (this will change the position of the named column in the SQL).
- Modifies all the attributes of the result column to match those of the loaded column.

If you choose Order matching, the builder works through comparing each results column to the loaded column in the equivalent position. If a mismatch is found, and the data type of the two columns is compatible, the SQL builder:

- Changes the alias name of the result column to match the loaded column (provided the results set does not already include a column of that name).
- Modifies all the attributes of the result column to match those of the loaded column.

If you choose Both, the SQL builder applies Name matching and then Order matching.

If auto matching fails to reconcile the columns as described above, any mismatched results column that represents a single column in a table is overwritten with the details of the loaded column in the equivalent position.

When you click **OK** in the Sql tab, the SQL builder checks to see if the results columns match the loaded columns. If they don't, a warning message is displayed allowing you to proceed or cancel. Proceeding causes the loaded columns to be merged with the results columns:

- Any matched columns are not affected.
- Any extra columns in the results columns are added to the loaded columns.
- Any columns in the loaded set that do not appear in the results set are removed.
- For columns that don't match, if data types are compatible the loaded column is overwritten with the results column. If data types are not compatible, the existing loaded column is removed and replaced with the results column.

You can also edit the columns in the Results part of the grid in order to reconcile mismatches manually.

Expression editor

In the expression editor, you can specify a WHERE clause to add to your SQL statement. If you are joining tables, you can also specify a WHERE or HAVING clause for a join condition.

A variant of the expression editor allows you to specify a calculation, function, or a case statement within an expression. The expression editor can be opened from various places in the SQL builder.

Main expression editor

In the expression editor, you can specify a filter that uses the Between, Comparison, In, Like, or Null predicates.

To specify an expression:

- Choose the type of filter by choosing a predicate from the list.
- Fill in the information required by the Expression Editor fields that appear.
- Click the **Add** button to add the filter to the query you are building. This clears the expression editor so that you can add another filter if required.

The contents of the expression editor vary according to which predicate you have selected. The following predicates are available:

- **Between**. Allows you to specify that the value in a column should lay within a certain range.
- **Comparison**. Allows you to specify that the value in a column should be equal to, or greater than or less than, a certain value.
- **In**. Allows you to specify that the value in a column should match one of a list of values.
- **Like**. Allows you to specify that the value in a column should contain, start with, end with, or match a certain value.
- **Null**. Allows you to specify that a column should be null or should not be null.

Between predicate

When you specify a Between predicate in the expression editor, you choose a column, specify a range, and specify whether a value must be in the range or not in the range.

The expression editor when you have selected the Between predicate contains:

- **Column**. Choose the column on which you are filtering from the drop-down list. You can also specify:
  - **Job parameter**. A dialog box appears offering you a choice of available job parameters. This allows you to specify the value to be used in the query at run time (the stage you are using the SQL builder from must allow job parameters for this to appear).
  - **Expression**. An expression editor dialog box appears, allowing you to specify an expression that represents the value to be used in the query.
  - **Data flow variable**. A dialog box appears offering you a choice of available data flow variables (the stage you are using the SQL builder from must support data flow variables for this to appear)
  - **Column**. You can directly select a column from one of the tables in the table selection canvas.
- **Between/Not Between**. Choose Between or Not Between from the drop-down list to specify whether the value you are testing should be inside or outside your specified range.
- **Start of range**. Use this field to specify the start of your range. Click the menu button to the right of the field and specify details about the argument you are using to specify the start of the range, then specify the value itself in the field.
- **End of range**. Use this field to specify the end of your range. Click the menu button to the right of the field and specify details about the argument you are using to specify the end of the range, then specify the value itself in the field.

Comparison predicate

When you specify a Comparison predicate in the expression editor, you choose a column, a comparison operator, and a comparison value.

The expression editor when you have selected the Comparison predicate contains:

- **Column**. Choose the column on which you are filtering from the drop-down list. You can specify one of the following in identifying a column:
  - **Job parameter**. A dialog box appears offering you a choice of available job parameters. This allows you to specify the value to be used in the query at run time (the stage you are using the SQL builder from must allow job parameters for this to appear).
  - **Expression**. An expression editor dialog box appears, allowing you to specify an expression that represents the value to be used in the query.
  - **Data flow variable**. A dialog box appears offering you a choice of available data flow variables (the stage you are using the SQL builder from must support data flow variables for this to appear)
  - **Column**. You can directly select a column from one of the tables in the table selection canvas.
- **Comparison operator**. Choose the comparison operator from the drop-down list. The available operators are:
  - = equals
  - <> not equal to
  - < less than

- <= less than or equal to
- > greater than
- >= greater than or equal to

- **Comparison value**. Use this field to specify the value you are comparing to. Click the menu button to the right of the field and choose the data type for the value from the menu, then specify the value itself in the field.

In predicate

When you specify an In predicate in the expression editor, you choose a column, select items to include in the query, and specify whether selected values are in the list or not in the list.

The expression editor when you have selected the In predicate contains:

- **Column**. Choose the column on which you are filtering from the drop-down list. You can specify one of the following in identifying a column:
  - **Job parameter**. A dialog box appears offering you a choice of available job parameters. This allows you to specify the value to be used in the query at run time (the stage you are using the SQL builder from must allow job parameters for this to appear).
  - **Expression**. An expression editor dialog box appears, allowing you to specify an expression that represents the value to be used in the query.
  - **Data flow variable**. A dialog box appears offering you a choice of available data flow variables (the stage you are using the SQL builder from must support data flow variables for this to appear)
  - **Column**. You can directly select a column from one of the tables in the table selection canvas.
- **In/Not In**. Choose IN or NOT IN from the drop-down list to specify whether the value should be in the specified list or not in it.
- **Selection**. These fields allows you to specify the list used by the query. Use the menu button to the right of the single field to specify details about the argument you are using to specify a list item, then enter a value. Click the double right arrow to add the value to the list.

  To remove an item from the list, select it then click the double left arrow.

Like predicate

When you specify a Like predicate in the expression editor, you choose a column, an operator, and a value. You then specify whether values are included or excluded by the comparison.

The expression editor when you have selected the Like predicate is as follows. The fields it contains are:

- **Column**. Choose the column on which you are filtering from the drop-down list. You can specify one of the following in identifying a column:
  - **Job parameter**. A dialog box appears offering you a choice of available job parameters. This allows you to specify the value to be used in the query at run time (the stage you are using the SQL builder from must allow job parameters for this to appear).
  - **Expression**. An expression editor dialog box appears, allowing you to specify an expression that represents the value to be used in the query.
  - **Data flow variable**. A dialog box appears offering you a choice of available data flow variables (the stage you are using the SQL builder from must support data flow variables for this to appear)

  - **Column**. You can directly select a column from one of the tables in the table selection canvas.
- **Like/Not Like**. Choose LIKE or NOT LIKE from the drop-down list to specify whether you are including or excluding a value in your comparison.
- **Like Operator.** Choose the type of Like or Not Like comparison you want to perform from the drop-down list. Available operators are:
  - Match Exactly. Your query will ask for an exact match to the value you specify.
  - Starts With. Your query will match rows that start with the value you specify.
  - Ends With. Your query will match rows that end with the value you specify.
  - Contains. Your query will match rows that contain the value you specify anywhere within them.
- **Like Value**. Specify the value that your LIKE predicate will attempt to match.

Null predicate

When you specify a Null predicate in the expression editor, you choose a column and specify whether your query must match a NULL or NOT NULL condition in the column.

The expression editor when you have selected the Null predicate is as follows. The fields it contains are:

- **Column**. Choose the column on which you are filtering from the drop-down list. You can specify one of the following in identifying a column:
  - **Job parameter**. A dialog box appears offering you a choice of available job parameters. This allows you to specify the value to be used in the query at run time (the stage you are using the SQL builder from must allow job parameters for this to appear).
  - **Expression**. An expression editor dialog box appears, allowing you to specify an expression that represents the value to be used in the query.
  - **Data flow variable**. A dialog box appears offering you a choice of available data flow variables (the stage you are using the SQL builder from must support data flow variables for this to appear)
  - **Column**. You can directly select a column from one of the tables in the table selection canvas.
- **Is Null/Is Not Null**. Choose whether your query will match a NULL or NOT NULL condition in the column.

Join predicate

When you specify a Join predicate in the expression editor, you choose the columns to join and a join type.

This predicate is only available when you are building an Oracle 8i query with an `old style' join expression. The Expression Editor is as follows.

- **Left column**. Choose the column to be on the left of your join from the drop-down list.
- **Join type**. Choose the type of join from the drop-down list.
- **Right column**. Choose the column to be on the right of your query from the drop-down list.

Calculation, function, and case expression editor

In this version of the expression editor, you can specify an expression in a WHERE expression, a HAVING expression, or a join condition. The expression editor windows are numbered to show how deeply they are nested.

Calculation predicate

When you use the Calculation predicate, you specify the left value, right value, and calculation operator in the expression editor.

The expression editor when you have selected the Calculation predicate contains these fields:

- **Left Value**. Enter the argument you want on the left of your calculation. You can choose the type of argument by clicking the menu button on the right and choosing a type from the menu.
- **Calculation Operator**. Choose the operator for your calculation from the drop-down list.
- **Right Value**. Enter the argument you want on the right of your calculation. You can choose the type of argument by clicking the menu button on the right and choosing a type from the menu.

Functions predicate

When you use the functions predicate, you can specify the function, description, and function parameters in the expression editor.

The expression editor when you have selected the Functions predicate contains these fields:

- **Function**. Choose a function from the drop-down list.

  The list of available functions depends on the database you are building the query for.
- **Description**. Gives a description of the function you have selected.
- **Parameters**. Enter the parameters required by the function you have selected. The parameters that are required vary according to the selected function.

Case predicate

When you use the case predicate, you can include case statements in the SQL that you build in the expression editor.

The case option on the expression editor enables you to include case statements in the SQL you are building. You can build case statements with the following syntax.

```
CASE WHEN condition THEN value
CASE WHEN...
ELSE value
```

or

```
CASE subject
WHEN match_value THEN value
WHEN...
ELSE value
```

The expression editor when you have selected the Case predicate contains these fields:

- **Case Expression**. This is the subject of the case statement. Specify this if you are using the second syntax described above (CASE *subject* WHEN). By default, the field offers a choice of the columns from the table or tables you have dragged to

the table selection canvas. To choose an alternative, click the browse button next to the field. This gives you a choice of data types, or of specifying another expression, a function, or a job parameter.

- **When**. This allows you to specify a condition or match value for your case statement. By default, the field offers a choice of the columns from the table or tables you have dragged to the table selection canvas. To choose an alternative, click the browse button next to the field. This gives you a choice of data types, or of specifying another expression, a function, or a job parameter. You can access the main expression editor by choose case expression editor from the menu. This allows you to specify expressions such as comparisons. You would typically use this in the first syntax example. For example, you would specify grade=3 as the condition in the expression WHEN grade=3 THEN 'first class'.
- **Then**. Use this to specify the value part of the case expression. By default, the field offers a choice of the columns from the table or tables you have dragged to the table selection canvas. To choose an alternative, click the browse button next to the field. This gives you a choice of data types, or of specifying another expression, a function, or a job parameter.
- **Add**. Click this to add a case expression to the query. This clears the When and Then fields so that you can specify another case expression.
- **Else Expression**. Use this to specify the value for the optional ELSE part of the case expression.

Expression editor menus

From the expression editor, you can open a menu where you can specify details about an argument in the expression.

A button appears to the right of many of the fields in the expression editor and related dialogs. Where it appears you can click it to open a menu that allows you to specify more details about an argument being given in an expression.

- **Bit**. Specifies that the argument is of type bit. The argument field offers a choice of 0 or 1 in a drop-down list.
- **Column**. Specifies that the argument is a column name. The argument field offer a choice of available columns in a drop-down list.
- **Date**. Specifies that the argument is a date. The SQL builder enters today's date in the format expected by the database you are building the query for. You can edit this date as required or click the drop-down button and select from a calendar.
- **Date Time**. Specifies that the argument is a date time. The SQL builder inserts the current date and time in the format that the database the query is being built for expects. You can edit the date time as required.
- **Plaintext**. Allows you to select the default value of an argument (if one is defined).
- **Expression Editor.** You can specify a function or calculation expression as an argument of an expression. Selecting this causes the Calculation/Function version of the expression editor to open.
- **Function**. You can specify a function as an argument to an expression.

  Selecting this causes the Functions Form dialog box to open. The functions available depend on the database that the query you are building is intended for.

  Selecting this causes the Function dialog box to open.

- **Job Parameter**. You can specify that the argument is a job parameter, the value for which is supplied when you actually run the IBM InfoSphere DataStage job. Selecting this opens the Parameters dialog box.
- **Integer**. Choose this to specify that the argument is of integer type.
- **String**. Select this to specify that the argument is of string type.
- **Time**. Specifies that the argument is the current local time. You can edit the value.
- **Timestamp**. Specifies that the argument is a timestamp. You can edit the value.

  The SQL builder inserts the current date and time in the format that the database that the query is being built for expects.

Functions Form window

In the Functions Form window, you select a function to use in an expression and specify parameters for the function.

The fields are as follows:

- **Function**. Choose a function from the drop-down list.

  The available functions depend on the database that you are building the query for.
- **Format**. Gives the format of the selected function as a guide.
- **Description**. Gives a description of the function you have selected.
- **Result**. Shows the actual function that will be included in the query as specified in this dialog box.
- **Parameters**. Enter the parameters required by the function you have selected. The parameters that are required vary according to the selected function.

Function window:

In the Function window, you can select a function to use in an expression and specify parameters for the function.

The fields are as follows:

- **Function**. Choose a function from the drop-down list.

  The available functions depend on the database that you are building the query for.
- **Format**. Gives the format of the selected function as a guide.
- **Description**. Gives a description of the function you have selected.
- **Result**. Shows the actual function that will be included in the query as specified in this dialog box.
- **Parameters**. Enter the parameters required by the function you have selected. The parameters that are required vary according to the selected function.

Parameters window

This window lists the job parameters that are currently defined for the job and the data type of each parameter. The SQL builder does not check that the type of parameter that you insert matches the type that is expected by the argument that you use it for.

Joining tables

When you use the SQL builder to build SELECT statements, you can specify table joins in a statement.

When you drag multiple tables onto the table selection canvas, the SQL builder attempts to create a join between the table added and the one already on the canvas to its left. If foreign key metadata is available for the tables, the SQL builder uses it. The join is represented by a line joining the columns the SQL builder has decided to join on. After the SQL builder automatically inserts a join, you can amend it.

When you add a table to the canvas, SQL builder determines how to join the table with tables that are on the canvas. The process depends on whether the added table is positioned to the right or left of the tables on the canvas.

To construct a join between the added table and the tables to its left:

1. SQL builder starts with the added table.
2. Determine if there is a foreign key between the added table and the subject table.
   - If a foreign key is present, continue to Step 3.
   - If a foreign key is not present, skip to Step 4.
3. Choose between alternatives for joining the tables that is based on the following precedence.
   - Relations that apply to the key fields of the added tables
   - Any other foreign key relation

   Construct an INNER JOIN between the two tables with the chosen relationship dictating the join criteria.
4. Take the subject as the next table to the left, and try again from step 2 until either a suitable join condition has been found or all tables, to the left, have been exhausted.
5. If no join condition is found among the tables, construct a default join.

   If the SQL grammar does not support a CROSS JOIN, an INNER JOIN is used with no join condition. Because this produces an invalid statement, you must set a suitable condition, either through the Join Properties dialog box, or by dragging columns between tables.

   An INNER JOIN is used with no join condition. Because this produces an invalid statement, you must set a suitable condition, either through the Join Properties dialog box, or by dragging columns between tables.

To construct a join between the added table and tables to its right:

1. SQL builder starts with the added table.
2. Determine if foreign key information exists between the added table and the subject table.
   - If a foreign key is present, continue to Step 3.
   - If a foreign key is not present, skip to Step 4.
3. Choose between alternatives based on the following precedence:
   - Relations that apply to the key fields of the added tables
   - Any other joins

   Construct an INNER JOIN between the two tables with the chosen relationship dictating the join criteria.
4. Take the subject as the next table to the right and try again from step 2.
5. If no join condition is found among the tables, construct a default join.

If the SQL grammar does not support a CROSS JOIN, an INNER JOIN is used with no join condition. Because this produces an invalid statement, you must set a suitable condition, either through the Join Properties dialog box, or by dragging columns between tables.

An INNER JOIN is used with no join condition. Because this produces an invalid statement, you must set a suitable condition, either through the Join Properties dialog box, or by dragging columns between tables.

Specifying joins

When you add more than one table to the table selection canvas, the SQL builder inserts a join automatically. To change the join, you can use the Join Properties window, use the Alternate Relation window, or drag a column from one table to a column in another table.

You can change the join in the following ways:

- Using the Join Properties dialog box. Open this by selecting the link in the table selection canvas, right clicking and choosing **Properties** from the shortcut menu. This dialog allows you to choose a different type of join, choose alternative conditions for the join, or choose a natural join.
- Using the Alternate Relation dialog box. Open this by selecting the link in the table selection canvas, right clicking and choosing **Alternate Relation** from the shortcut menu. This dialog allows you to change foreign key relationships that have been specified for the joined tables.
- By dragging a column from one table to another column in any table to its right on the canvas. This replaces the existing automatic join and specifies an equijoin between the source and target column. If the join being replaced is currently specified as an inner or outer join, then the type is preserved, otherwise the new join will be an inner join.

Yet another approach is specify the join using a WHERE clause rather than an explicit join operation (although this is not recommended where your database supports explicit join statements). In this case you would:

1. Specify the join as a Cartesian product. (SQL builder does this automatically if it cannot determine the type of join required).
2. Specify a filter in the **Selection** tab filter panel. This specifies a WHERE clause that selects rows from within the Cartesian product.

If you are using the SQL builder to build Oracle 8i, Microsoft SQL Server, IBM Informix®, or Sybase queries, you can use the Expression Editor to specify a join condition, which will be implemented as a WHERE statement. Oracle 8i does not support JOIN statements.

Join Properties window

Use the Join Properties window to change the type of an existing join and modify or specify the join condition.

The window contains the following fields:

- **Cartesian product**. The Cartesian product is the result that is returned from two or more tables that are selected from, but not joined; that is, no join condition is specified. The output is all possible rows from all the tables selected from. For example, if you selected from two tables, the database would pair every row in the first table with every row in the second table. If each table had 6 rows, the Cartesian product would return 36 rows.

If the SQL builder cannot insert an explicit join based on available information, it will default to a Cartesian product that is formed with the CROSS JOIN syntax in the FROM clause of the resulting SQL statement: `FROM FirstTable CROSS JOIN SecondTable`. You can also specify a Cartesian product by selecting the Cartesian product option in the Join Properties dialog box. The cross join icon is shown on the join.

- **Table join**. Select the **Table Join** option to specify that your query will contain join condition for the two tables being joined. The **Join Condition** panel is enabled, allowing you to specify further details about the join.
- **Join Condition panel**. This shows the expression that the join condition will contain. You can enter or edit the expression manually or you can use the menu button to the right of the panel to specify a natural join, open the Expression Editor, or open the Alternate relation dialog box.
- **Include**. These fields allow you to specify that the join should be an outer join, where the result of the query should include the rows as specified by one of the following:
  - Select **All rows from left table name** to specify a left outer join
  - Select **All rows from right table name** to specify a right outer join
  - Select both **All rows from left table name** and **All rows from right table name** to specify a full outer join
- **Join Icon**. This tells you the type of join you have specified.

Alternate Relation window

The Alternate Relation window shows the foreign key relationships that are defined between the target table and tables that appear to the left of it on the table selection canvas. Select the relationship that you want to appear as the join in your query so that it appears in the list box, and then click **OK**.

Properties windows

The Properties windows contain properties for tables, SQL, and joins.

Depending where you are in the SQL builder, choosing **Properties** from the shortcut menu opens a dialog box as follows:

- The Table Properties dialog box opens when you select a table in the table selection canvas and choose **Properties** from the shortcut menu.
- The SQL Properties dialog box opens when you select the **Properties** icon in the toolbox or **Properties** from the table selection canvas background.
- The Join Properties dialog box opens when you select a join in the table selection canvas and choose **Properties** from the shortcut menu.

Table Properties window

In the Table Properties window, you can view the table name and view or edit the table alias.

The **Table Properties** dialog box contains the following fields:

- **Table name**. The name of the table whose properties you are viewing.

  You can click the menu button and choose **Job Parameter** to open the Parameter dialog box. This allows you to specify a job parameter to replace the table name if required, but note that the SQL builder will always refer to this table using its alias.

- **Alias**. The alias that the SQL builder uses to refer to this table. You can edit the alias if required. If the table alias is used in the selection grid or filters, changing the alias in this dialog box will update the alias there.

SQL Properties window

The SQL Properties window shows the SQL grammar that the SQL builder uses.

The SQL Properties window contains the following fields:

- **Description**. The name and version of the SQL grammar.
  The SQL grammar depends on the stage that you invoke the SQL builder from.
- **DISTINCT**. Specify whether the SQL builder supports the DISTINCT qualifier.
  If the stage supports it, the DISTINCT option is selected.

Chapter 8. Environment variables: Oracle connector

The Oracle Connector stage uses these environment variables.

CC_GUARDIUM_EVENTS

Set this environment variable to specify whether connectors report the InfoSphere DataStage context information to the InfoSphere Guardium Database Activity monitor.

When the value of this environment variable is set, the connectors report the InfoSphere DataStage context information such as host, project, job names, stage name and node ID that the stage is running on to the InfoSphere Guardium Database Activity monitor. When this environment variable is defined and set to any value, the connectors report context information to the Guardium server after the initial connection is established.

When this environment variable is undefined, the connectors do not attempt to report context information to Guardium servers. The setting of this environment variable applies to all database connectors in the job.

CC_IGNORE_TIME_LENGTH_AND_SCALE

Set this environment variable to change the behavior of the connector on the parallel canvas.

When this environment variable is set to 1, the connector running with the parallel engine ignores the specified length and scale for the timestamp column. For example, when the value of this environment variable is not set and if the length of the timestamp column is 26 and the scale is 6, the connector on the parallel canvas considers that the timestamp has a microsecond resolution. When the value of this environment variable is set to 1, the connector on the parallel canvas does not consider that the timestamp has a microsecond resolution unless the microseconds extended property is set even if the length of the timestamp column is 26 and the scale is 6.

CC_ORA_BIND_DATETIME_AS_CHAR

Set this environment variable to specify whether to bind Date and Timestamp values as character values.

When this environment variable is set to TRUE, the Oracle Connector stage uses character representation for Date and Timestamp values that are exchanged with the Oracle database. The stage uses the same date and time formats that are used by the Dynamic RDBMS stage

Use this environment variable only when the date and time formats that the Oracle connector uses must be compatible with the Dynamic RDBMS stage. If you use this environment variable, performance might be affected negatively.

CC_ORA_BIND_FOR_NCHARS

Set this connector environment variable to specify whether to bind a list of the character columns as national character columns with the Oracle database.

Set this environment variable to a comma-delimited list of InfoSphere DataStage column names that are national character columns in the database. When this environment variable is set, the columns that are defined in the comma-delimited list are bound as national character columns regardless of their definitions in the columns grid. In addition, you can set this environment variable to the following values:

-(none)
Bind no national character columns and bind all character columns as implicit.

-(all) Bind all national character columns.

When this environment variable is undefined, the connector binds based on the definitions in the columns grid.

CC_ORA_BIND_KEYWORD

Set this environment variable to specify the identifier that indicates a bind parameter in a user-defined SQL statement.

The default identifier is ORCHESTRATE. For example, you can use this environment variable to specify a different identifier when SQL statements require the use of the literal ORCHESTRATE in the name of a schema, table, or column.

CC_ORA_CHECK_CONVERSION

Set this environment variable to specify whether exceptions are thrown when data loss occurs because of a conversion from the Unicode character set to the native character set of the database.

The default value is FALSE. When the value of this variable is TRUE, an exception is thrown when data loss occurs. The values for this environment variable are not case sensitive.

CC_ORACLECONNECTOR_DEFAULT_CONNECTION_VERSION

Set this environment variable to specify the default value for the **Oracle client version** property in the Oracle connector stages.

The allowed values for this environment variable are the same as the ones specified for the **Oracle client version** property in the stage editor. For example, set this environment variable to `11g` for the default value of the property to be `11g`. The default value will be set for this property when the stage is placed on the job canvas and is opened for the first time.

CC_ORA_DEFAULT_DATETIME_TIME

Set this environment variable to specify the values for hours, minutes, and seconds when the connector writes the InfoSphere DataStage Date type to an Oracle DATE or TIMESTAMP column.

The format is *HH:MI:SS* where *HH* represents hours in 24-hour notation, *MI* represents minutes and *SS* represents seconds. When the environment variable is set, the stage uses the value that is specified for the default hour, minute and second portion of the target values.

When the connector writes to Oracle TIMESTAMP, the environment variable does not provide an option to specify default fractional seconds . To specify fractional seconds, you must use the InfoSphere DataStage Time or Timestamp column on the link. When this environment variable is not set, the hour, minute, and second portions in the target value are set to midnight.

CC_ORA_DEFAULT_DATETIME_DATE

Set this environment variable to specify the default values for the month, day, and year when the connector writes from a InfoSphere DataStage Time type to an Oracle DATE or TIMESTAMP column.

The format is *YYYY-MM-DD* where *YYYY* represents years, *MM* represents months and *DD* represents days. When the environment variable is set, the stage uses the value that is specified for the default year, month and day portion of the target values.

When the environment variable is not set, the month, day, and year to the current date in most scenarios. If the DRS Connector stage is used and the write mode is not bulk load, the month, day, and year are shown as *0000-00-00*.

CC_ORA_DROP_UNMATCHED_FIELDS_DEFAULT

Set this environment variable to specify the **Drop unmatched fields** property when the property is not set correctly in an Oracle Connector job generated by the connector migration tool.

When this environment variable is set to TRUE, the Oracle connector stages that do not have the property act as if the property was set to `Yes` and drop any unused fields from the design schema. When the environment variable is set to FALSE or undefined, the connector end the job if any fields from the design schema are unused and the **Drop unmatched fields** property does not exist.

CC_ORA_INDEX_MAINT_SINGLE_ROW

Set this environment variable to specify how index rows are inserted during bulk load.

When this environment variable is set to TRUE, the connector inserts index rows individually. When this environment variable is set to FALSE or undefined, the connector uses default bulk load behavior. If you use this environment variable, performance might be affected negatively.

CC_ORA_INVALID_DATETIME_ACTION

Set this environment variable to insert a NULL value into the database for invalid Date, Time or Timestamp fields.

When the value of this environment variable is set to NULL, the connector inserts a NULL value into the database for invalid Date, Time or Timestamp fields on its input link. If this environment variable is set to another value or if it is undefined, the connector stops the job for invalid Date, Time and Timestamp fields and in this situation and logs a fatal error message. The fatal error message indicates that the internal variable *bInvalidDateTime* is set to 1 which means that an invalid date or time field arrived on the input link of the stage. The values for this environment variable are not case sensitive.

CC_ORA_LOB_LOCATOR_COLUMNS

Set this environment variable so specify whether the connector uses OCI LOB locators when the connector writes data into LOB columns.

Set this environment variable to a comma-delimited list of InfoSphere DataStage LongVarchar, LongNVarchar, and LongVarBinary data types that you want to use OCI LOB locators to write data into their respective CLOB, NCLOB, or BLOB columns.

To use OCI LOB locators for all LongVarchar, LongNVarchar, and LongVarBinary columns, set this environment variable to `all`. Use this environment variable when you want to support SDO_GEOMETRY and XMLTYPE columns and functions or process LONG or LONG RAW columns in the same statement as CLOB, NCLOB, or BLOB columns.

When this environment variable is set to FALSE or undefined, the connector uses OCI LOB locators based on the definitions in the columns grid.

CC_ORA_MAX_ERRORS_REPORT

Set this environment variable to specify the maximum number of errors to report to the log file when an operation writes an array or bulk loads data.

This variable is relevant only when a reject link is not defined. The default value is `-1`, which reports all errors.

CC_MSG_LEVEL

Set this environment variable to specify the minimum severity of the messages that the connector reports in the log file.

At the default value of 3, informational messages and messages of a higher severity are reported to the log file.

The following list contains the valid values:

- 1 - Trace
- 2 - Debug
- 3 - Informational
- 4 - Warning

- 5 - Error
- 6 - Fatal

CC_ORA_NLS_LANG_ENV

Set this environment variable to specify whether the NLS_LANG character set is used when the connector initializes the Oracle client environment.

The default value is FALSE. When the value of this variable is TRUE, the NLS_LANG character set is used; otherwise, the UTF-16 character set is used. The values for this environment variable are not case sensitive.

CC_ORA_NODE_PLACEHOLDER_NAME

Set this environment variable to specify the case-sensitive value for the processing node numbers in SQL statements.

This environment variable is used as a placeholder in the WHERE clause of user defined SQL statements to enable the user to run a different statement on each node. The value of this environment variable will be replaced with the node the statement is currently running on.

CC_ORA_NODE_USE_PLACEHOLDER

Set this environment variable to specify whether the connector replaces the placeholder for the processing node number with the current processing node number in SQL statements that run on processing nodes.

When the value of this variable is TRUE, the connector replaces the placeholder. The values for this environment variable are not case sensitive.

CC_ORA_NULL_CHAR_ACTION

Set this environment variable to define behavior when the input data contains NULL characters.

This environment variable applies only when the Oracle Connector stage runs on the parallel canvas, and the variable applies only to fields of Char, VarChar, LongVarChar, NChar, NVarChar and LongNVarChar InfoSphere DataStage types.

You can set this environment variable to the following values:

TRUNCATE
The connector treats the NULL character as a value terminator in the character data that is retrieved on the input link. If the truncated value has a length of zero, NULL is inserted in the target.

FAIL When the connector encounters NULL characters in the input data, the connector logs a fatal error message and stops the job. The error message indicates the field that contained the NULL character or characters.

When the value of this environment variable is undefined or set to another value, the NULL character is treated the same as any other character. The value is passed to Oracle along with any NULL characters. This behavior is the default behavior for the connector. When this environment variable is set to `TRUNCATE` or `FAIL`, the

columns with LongVarChar and LongNVarChar data types are treated as columns with VarChar and NVarChar data types, respectively.

Use the **CC_ORA_NULL_CHAR_ACTION** environment variable only in jobs that were migrated from the Oracle Enterprise stage to the Oracle Connector stage to provide consistent behavior with the Oracle Enterprise stage. Alternatively, you can update the migrated jobs that rely on this truncation behavior so that they work correctly with the default connector behavior. The default connector behavior is to pass character data from the input link to the database, including any NULL characters. Set this environment variable to `FAIL` to help detect jobs in which the input data contains NULL characters.

CC_ORA_OPTIMIZE_CONNECTIONS

Set this environment variable to disconnect the conductor node's SQL sessions from the Oracle server during the job setup phase after completing any **Table action** or **Before SQL** operations.

At the end of the job, the connector connects to Oracle server again, to complete any **After SQL** operation or operation that occurs after a bulk load. When this environment variable is set to a value other than TRUE, the connector keeps the Oracle connections connected when the job runs. The values for this environment variable are not case sensitive.

CC_ORA_PRESERVE_DATE_TYPE_NAME

Set this environment variable to specify whether Oracle DATE data types are imported as InfoSphere DataStage Date data types.

When this environment variable is set to TRUE, Oracle DATE data types are imported as Date data types. The default value is FALSE, and Oracle DATE data types are imported as Timestamp data types.

CC_ORA_ROWS_REJECTED_MSG_INFO

Set this environment variable to specify the severity of the message that reports the number of records that were sent to a reject link.

When this environment variable is set to TRUE, the Oracle Connector message that reports the number of rejected records is logged as an informational message. When this environment is set to FALSE or undefined, the connector logs the message as a warning.

CC_ORA_UNBOUNDED_BINARY_LENGTH

Set this environment variable to override the default length that the connector uses for InfoSphere DataStage Binary and VarBinary columns for which a length is not defined in the design schema.

When this environment variable is set to a positive integer value, the connector uses that value as the length, in bytes, for Binary and VarBinary columns for which a length is not defined in the design schema. This environment variable applies to source, target, and request contexts, and it also applies when the connector generates DDL statements.

When the environment variable is not defined, the connector uses the default value of 4000 bytes as the length. This environment variable is typically used with migrated jobs, because the legacy Oracle stages used a different default value for columns when a length was not defined.

CC_ORA_UNBOUNDED_STRING_LENGTH

Set this environment variable to override the default length that the connector uses for InfoSphere DataStage Char, VarChar, NChar, and NVarChar columns for which a length is not defined in the design schema.

When this environment variable is set to a positive integer value, the connector uses that value as the length, in bytes, for Char, VarChar, NChar, and NVarChar columns for which a length is not defined in the design schema. This environment variable applies to source, target, and request contexts, and it also applies when the connector generates DDL statements.

When the environment variable is not defined, the connector uses the default value of 4000 bytes as the length. This environment variable is typically used with migrated jobs, because the legacy Oracle stages used a different default value for columns when a length was not defined.

CC_ORA_XMLTYPE_CSID_BLOB

Set this environment variable to specify the character set ID that is used when creating XMLType as BLOB data type and the **Enable LOB References** property is set to `Yes`.

This environment variable should be set to a valid Oracle character set ID. The default value of this environment variable is the character set that is defined by the NLS_LANG environment variable.

CC_SE_TIMESTAMP_FF

Set this environment variable to specify whether decimal point and fractional digits are included in the timestamp values, when the connector runs in server jobs.

When the environment variable is set to a value other than NONE, MICROSECONDS or SCALE, the behavior is the same as if the environment variable was not set. The environment variable values are case sensitive. When the environment variable is not set, the timestamp values that are produced by the job include a trailing decimal point and six fractional digits.

You can set the environment variable to the following values:

NONE
The trailing decimal point and the fractional digits are both omitted.

MICROSECONDS
The trailing decimal point and six fractional digits are included.

SCALE
The trailing decimal point and *S* fractional digits are included, where *S* represents the value of the Scale attribute in the timestamp column definition. When the Scale attribute value is not defined for the column, the Scale attribute value of zero is assumed.

CC_TRUNCATE_STRING_WITH_NULL

Set this environment variable to truncate string data that includes the string 0x00.

When the value of this environment variable is set and when the input data contains a null character, the input data is truncated with 0x00 and the rest of the string is dropped. This environment variable applies to fields of Char, VarChar, and LongVarChar InfoSphere DataStage types.

CC_TRUNCATE_NSTRING_WITH_NULL

Set this environment variable to truncate string data that includes the string 0x00.

When the value of this environment variable is set and when the input data contains a null character, the input data is truncated with 0x00 and the rest of the string is dropped.

CC_USE_EXTERNAL_SCHEMA_ON_MISMATCH

Set this environment variable to use an external schema rather than a design schema when the schemas do not match.

This schema is used for schema reconciliation. When the value of this environment variable is set, the behavior remains the same and is not changed from the old version.

Appendix A. Product accessibility

You can get information about the accessibility status of IBM products.

The IBM InfoSphere Information Server product modules and user interfaces are not fully accessible.

For information about the accessibility status of IBM products, see the IBM product accessibility information at http://www.ibm.com/able/product_accessibility/index.html.

Accessible documentation

Accessible documentation for InfoSphere Information Server products is provided in an information center. The information center presents the documentation in XHTML 1.0 format, which is viewable in most web browsers. Because the information center uses XHTML, you can set display preferences in your browser. This also allows you to use screen readers and other assistive technologies to access the documentation.

The documentation that is in the information center is also provided in PDF files, which are not fully accessible.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.

Appendix B. Reading command-line syntax

This documentation uses special characters to define the command-line syntax.

The following special characters define the command-line syntax:

[ ] Identifies an optional argument. Arguments that are not enclosed in brackets are required.

... Indicates that you can specify multiple values for the previous argument.

| Indicates mutually exclusive information. You can use the argument to the left of the separator or the argument to the right of the separator. You cannot use both arguments in a single use of the command.

{ } Delimits a set of mutually exclusive arguments when one of the arguments is required. If the arguments are optional, they are enclosed in brackets ([ ]).

Note:

- The maximum number of characters in an argument is 256.
- Enclose argument values that have embedded spaces with either single or double quotation marks.

For example:

wsetsrc*[-S server] [-l label] [-n name] source*

The *source* argument is the only required argument for the **wsetsrc** command. The brackets around the other arguments indicate that these arguments are optional.

wlsac *[-l | -f format] [key...] profile*

In this example, the -l and -f format arguments are mutually exclusive and optional. The *profile* argument is required. The *key* argument is optional. The ellipsis (...) that follows the *key* argument indicates that you can specify multiple key names.

wrb -import *{rule_pack | rule_set}...*

In this example, the rule_pack and rule_set arguments are mutually exclusive, but one of the arguments must be specified. Also, the ellipsis marks (...) indicate that you can specify multiple rule packs or rule sets.

Appendix C. How to read syntax diagrams

The following rules apply to the syntax diagrams that are used in this information:

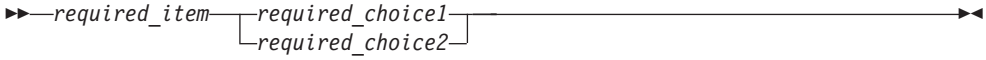
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol indicates that a syntax diagram is continued from the previous line.
  - The --->< symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).

- Optional items appear below the main path.

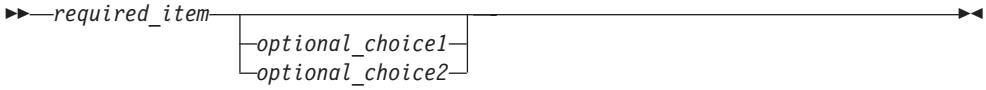
  If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.

- If you can choose from two or more items, they appear vertically, in a stack.

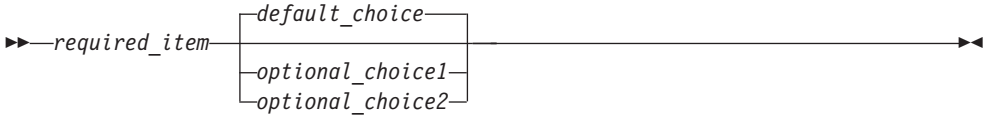
  If you must choose one of the items, one item of the stack appears on the main path.



  If choosing one of the items is optional, the entire stack appears below the main path.

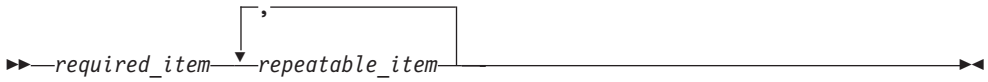


  If one of the items is the default, it appears above the main path, and the remaining choices are shown below.



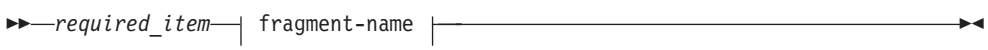
- An arrow returning to the left, above the main line, indicates an item that can be repeated.

If the repeat arrow contains a comma, you must separate repeated items with a comma.



A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



Fragment-name:

- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown.
- Variables appear in all lowercase italic letters (for example, **column-name**). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).

Appendix D. Contacting IBM

You can contact IBM for customer support, software services, product information, and general information. You also can provide feedback to IBM about products and documentation.

The following table lists resources for customer support, software services, training, and product and solutions information.

Table 37. IBM resources

Resource	Description and location
IBM Support Portal	You can customize support information by choosing the products and the topics that interest you at www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server
Software services	You can find information about software, IT, and business consulting services, on the solutions site at www.ibm.com/businesssolutions/
My IBM	You can manage links to IBM Web sites and information that meet your specific technical support needs by creating an account on the My IBM site at www.ibm.com/account/
Training and certification	You can learn about technical training and education services designed for individuals, companies, and public organizations to acquire, maintain, and optimize their IT skills at http://www.ibm.com/training
IBM representatives	You can contact an IBM representative to learn about solutions at www.ibm.com/connect/ibm/us/en/

Appendix E. Accessing the product documentation

Documentation is provided in a variety of formats: in the online IBM Knowledge Center, in an optional locally installed information center, and as PDF books. You can access the online or locally installed help directly from the product client interfaces.

IBM Knowledge Center is the best place to find the most up-to-date information for InfoSphere Information Server. IBM Knowledge Center contains help for most of the product interfaces, as well as complete documentation for all the product modules in the suite. You can open IBM Knowledge Center from the installed product or from a web browser.

Accessing IBM Knowledge Center

There are various ways to access the online documentation:

- Click the **Help** link in the upper right of the client interface.
- Press the F1 key. The F1 key typically opens the topic that describes the current context of the client interface.

  Note: The F1 key does not work in web clients.
- Type the address in a web browser, for example, when you are not logged in to the product.

  Enter the following address to access all versions of InfoSphere Information Server documentation:

  `http://www.ibm.com/support/knowledgecenter/SSZJPZ/`

  If you want to access a particular topic, specify the version number with the product identifier, the documentation plug-in name, and the topic path in the URL. For example, the URL for the 11.3 version of this topic is as follows. (The ⇒ symbol indicates a line continuation):

  ```
  http://www.ibm.com/support/knowledgecenter/SSZJPZ_11.3.0/⇒
  com.ibm.swg.im.iis.common.doc/common/accessingiidoc.html
  ```

  Tip:

  The knowledge center has a short URL as well:

  `http://ibm.biz/knowctr`

  To specify a short URL to a specific product page, version, or topic, use a hash character (#) between the short URL and the product identifier. For example, the short URL to all the InfoSphere Information Server documentation is the following URL:

  `http://ibm.biz/knowctr#SSZJPZ/`

  And, the short URL to the topic above to create a slightly shorter URL is the following URL (The ⇒ symbol indicates a line continuation):

  ```
  http://ibm.biz/knowctr#SSZJPZ_11.3.0/com.ibm.swg.im.iis.common.doc/⇒
  common/accessingiidoc.html
  ```

Changing help links to refer to locally installed documentation

IBM Knowledge Center contains the most up-to-date version of the documentation. However, you can install a local version of the documentation as an information center and configure your help links to point to it. A local information center is useful if your enterprise does not provide access to the internet.

Use the installation instructions that come with the information center installation package to install it on the computer of your choice. After you install and start the information center, you can use the **iisAdmin** command on the services tier computer to change the documentation location that the product F1 and help links refer to. (The ⇒ symbol indicates a line continuation):

Windows

```
IS_install_path\ASBServer\bin\iisAdmin.bat -set -key ⇒
com.ibm.iis.infocenter.url -value http://<host>:<port>/help/topic/
```

AIX Linux

```
IS_install_path/ASBServer/bin/iisAdmin.sh -set -key ⇒
com.ibm.iis.infocenter.url -value http://<host>:<port>/help/topic/
```

Where <host> is the name of the computer where the information center is installed and <port> is the port number for the information center. The default port number is 8888. For example, on a computer named server1.example.com that uses the default port, the URL value would be http://server1.example.com:8888/help/topic/.

Obtaining PDF and hardcopy documentation

- The PDF file books are available online and can be accessed from this support document: https://www.ibm.com/support/docview.wss?uid=swg27008803&wv=1.
- You can also order IBM publications in hardcopy format online or through your local IBM representative. To order publications online, go to the IBM Publications Center at http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss.

Appendix F. Providing feedback on the product documentation

You can provide helpful feedback regarding IBM documentation.

Your feedback helps IBM to provide quality information. You can use any of the following methods to provide comments:

- To provide a comment about a topic in IBM Knowledge Center that is hosted on the IBM website, sign in and add a comment by clicking **Add Comment** button at the bottom of the topic. Comments submitted this way are viewable by the public.
- To send a comment about the topic in IBM Knowledge Center to IBM that is not viewable by anyone else, sign in and click the **Feedback** link at the bottom of IBM Knowledge Center.
- Send your comments by using the online readers' comment form at www.ibm.com/software/awdtools/rcf/.
- Send your comments by e-mail to comments@us.ibm.com. Include the name of the product, the version number of the product, and the name and part number of the information (if applicable). If you are commenting on specific text, include the location of the text (for example, a title, a table number, or a page number).

Notices and trademarks

This information was developed for products and services offered in the U.S.A. This material may be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

Notices

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session or persistent cookies. If a product or component is not listed, that product or component does not use cookies.

Table 38. Use of cookies by InfoSphere Information Server products and components

Product module	Component or feature	Type of cookie that is used	Collect this data	Purpose of data	Disabling the cookies
Any (part of InfoSphere Information Server installation)	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
Any (part of InfoSphere Information Server installation)	InfoSphere Metadata Asset Manager	• Session • Persistent	No personally identifiable information	• Session management • Authentication • Enhanced user usability • Single sign-on configuration	Cannot be disabled

Table 38. Use of cookies by InfoSphere Information Server products and components (continued)

Product module	Component or feature	Type of cookie that is used	Collect this data	Purpose of data	Disabling the cookies
InfoSphere DataStage	Big Data File stage	• Session • Persistent	• User name • Digital signature • Session ID	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere DataStage	XML stage	Session	Internal identifiers	• Session management • Authentication	Cannot be disabled
InfoSphere DataStage	IBM InfoSphere DataStage and QualityStage Operations Console	Session	No personally identifiable information	• Session management • Authentication	Cannot be disabled
InfoSphere Data Click	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
InfoSphere Data Quality Console		Session	No personally identifiable information	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere QualityStage Standardization Rules Designer	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
InfoSphere Information Governance Catalog		• Session • Persistent	• User name • Internal identifiers • State of the tree	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere Information Analyzer	Data Rules stage in the InfoSphere DataStage and QualityStage Designer client	Session	Session ID	Session management	Cannot be disabled

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The United States Postal Service owns the following trademarks: CASS, CASS Certified, DPV, LACS^{Link}, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACS^{Link} licensee of the United States Postal Service.

Other company, product or service names may be trademarks or service marks of others.

Index

P

R

S

T

U

W

Printed in USA

SC19-4266-00

Spine information:

IBM InfoSphere DataStage and QualityStage **Version 11 Release 3** **Connectivity Guide for Oracle Databases** IBM