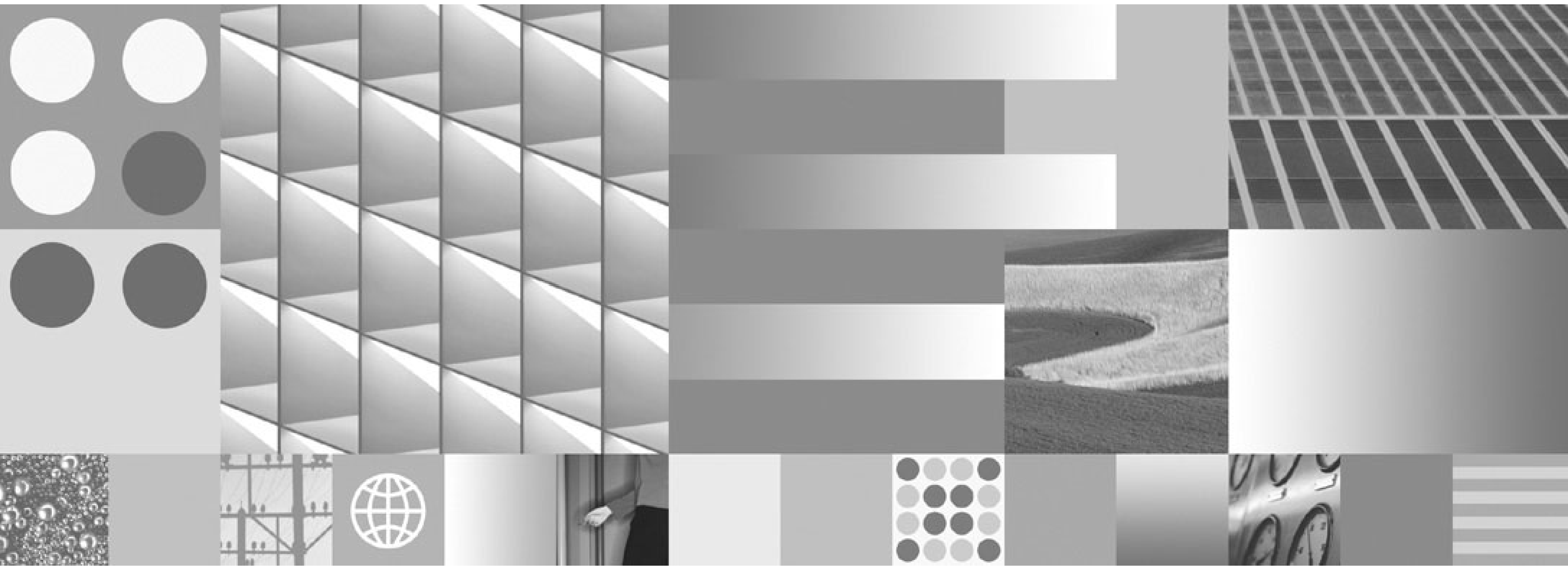
IBM WebSphere DataStage and QualityStage

Version 8 Release 1

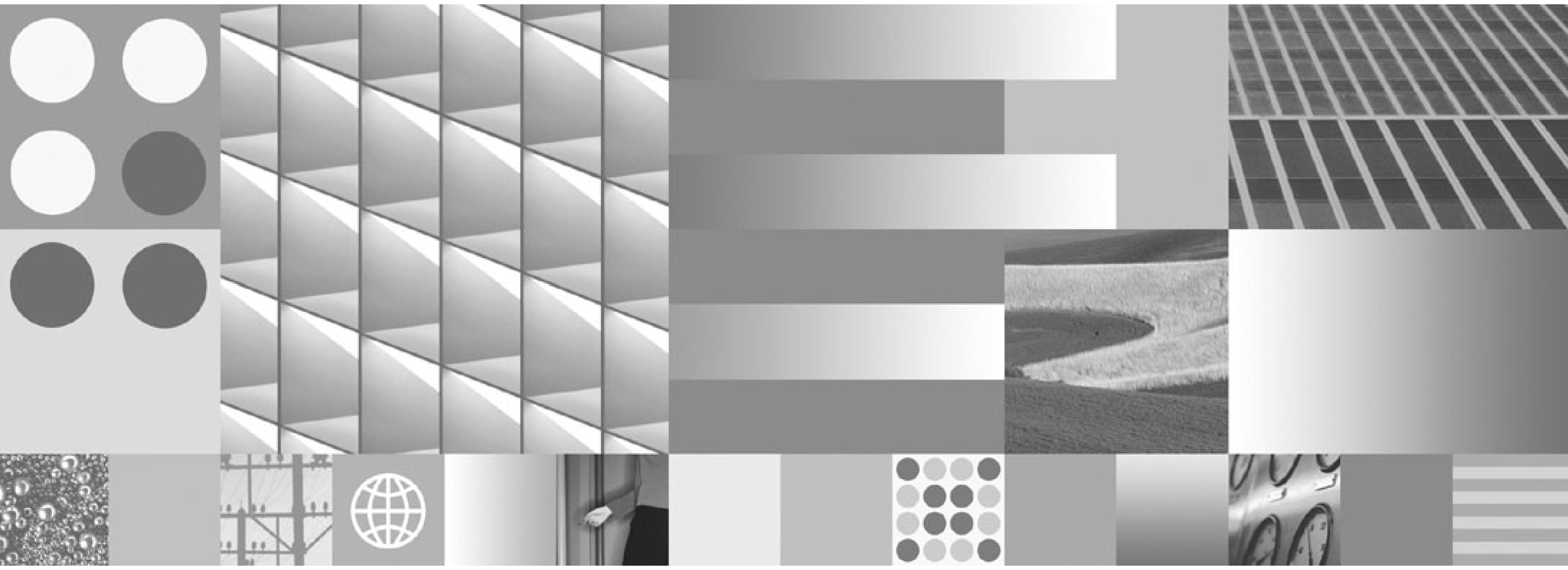


Connectivity Guide for Oracle Databases

LC18-9937-03

IBM WebSphere DataStage and QualityStage

Version 8 Release 1



Connectivity Guide for Oracle Databases

LC18-9937-03

Note

Before using this information and the product that it supports, read the information in "Notices" on page 153.

Contents

Chapter 1. Oracle enterprise stage

The Oracle enterprise stage is a database stage. It allows you 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 hosted by an Oracle database.

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 (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 using 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). See the *WebSphere DataStage Parallel Job Developer Guide* for more details about performing lookups.

When you edit a Oracle enterprise stage, the Oracle enterprise stage editor appears. This is based on the generic stage editor described in the *WebSphere DataStage Parallel Job Developer Guide*.

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.

Accessing Oracle databases

You need to be running Oracle 9 Enterprise Edition or higher in order to use the Oracle enterprise stage.

You must also perform the following tasks:

1. Create the user defined environment variable ORACLE_HOME and set this to the $ORACLE_HOME path (for example, /disk3/oracle9i).

2. Create the user defined environment variable ORACLE_SID and set this to the correct service name (for example, ODBCSOL).
3. Add *ORACLE_HOME/bin* to your PATH and *ORACLE_HOME/lib* to your LIBPATH, LD_LIBRARY_PATH, or SHLIB_PATH.
4. 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.
5. 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 IBM® WebSphere® DataStage® and QualityStage jobs, as follows:

```
GRANT DSXE to <oracle userid>;
```

Handling Special Characters (# and $)

The characters # and $ are reserved in WebSphere DataStage and special steps are needed to handle Oracle databases which use the characters # and $ in column names. WebSphere 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 tasks:

- In the WebSphere DataStage Administrator, open the Environment Variables dialog 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 column names. __035__ is the internal representation of # and __036__ is the internal representation of $.

When using this feature in your job, you should import meta data using the Plug-in Meta Data Import tool, and avoid hand-editing (this minimizes the risk of mistakes or confusion).

Once 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. The original names are used in generated SQL statements, however, and you should use them if entering SQL in the job yourself.

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

When using the Oracle stage as a target, you should 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 using the Oracle stage as a source, you should 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 using the Oracle stage in parallel jobs as a look-up, you should enter external or internal names as follows:

- For Lookups using 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 using 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 stage is attached.

Loading tables

There are some special points to note when using the Load method in this stage (which uses the Oracle fast loader) to load tables with indexes.

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

- DIRECT=TRUE
- PARALLEL = TRUE

This causes the load to run using parallel direct load mode. In order to use the parallel direct mode load, the table must not have indexes, or you must include one of the Index Mode properties, 'rebuild' or 'maintenance' (see the **Index Mode** section). If the only index on the table is from a primary key or unique key constraint, you can instead use the Disable Constraints property (see the **Disable Constraints** section) which will disable 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 would still run 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 (see the **Advanced tab** section).

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

Data type conversion - writing to Oracle

When writing or loading, the Oracle Enterprise stage automatically converts WebSphere DataStage data types to Oracle data types as shown in the following table:

Table 1. Data type conversion for writing data to an Oracle database

WebSphere DataStage SQL Data Type	Underlying Data Type	Oracle Data Type
Date	date	DATE
Time	time	DATE (does not support microsecond resolution)

Table 1. Data type conversion for writing data to an Oracle database (continued)

WebSphere DataStage SQL Data Type	Underlying Data Type	Oracle Data Type
Timestamp	timestamp	DATE (does not support microsecond resolution)
Decimal Numeric	decimal (*p*, *s*)	NUMBER (*p*, *s*)
TinyInt	int8/uint8	NUMBER (3, 0)
SmallInt	int16/uint16	NUMBER (3, 0)
Integer	int32/uint32	NUMBER (10, 0)
BigInt	int64	NUMBER (19, 0)
BigInt	uint64	NUMBER (20, 0)
Float Real	sfloat	NUMBER
Double	dfloat	NUMBER
Binary Bit LongVarBinary VarBinary	raw	not supported
Unknown Char	fixed-length string in the form string[*n*] and ustring[*n*]; length <= 255 bytes	CHAR(*n*) where *n* is the string length
LongVarChar VarChar	variable-length string, in the form string[max=*n*] and ustring[max=*n*]; maximum length <= 2^31-5 bytes	VARCHAR(*n*) where *n* is the maximum string length
LongVarChar VarChar	variable-length string in the form string and ustring	VARCHAR(32)*

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 - reading from Oracle

When reading, the Oracle enterprise stage automatically converts Oracle data types to WebSphere DataStage data types as shown in the following table:

Table 2. Data type conversion for reading data from an Oracle database

WebSphere 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*)

Table 2. Data type conversion for reading data from an Oracle database (continued)

WebSphere DataStage SQL Data Type	Underlying Data Type	Oracle Data Type
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	RAW(*n*)

Examples

Looking up an Oracle table

This example shows what happens when data is looked up in an Oracle table. The stage in this case will look up the interest rate for each customer based on the account type. Here is the data that arrives on the primary link:

Table 3. 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
Hooper	7176672	flexi	3456.99
Moore	7146789	gold	424.76

Here is the data in the Oracle lookup table:

Table 4. 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 5. 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.

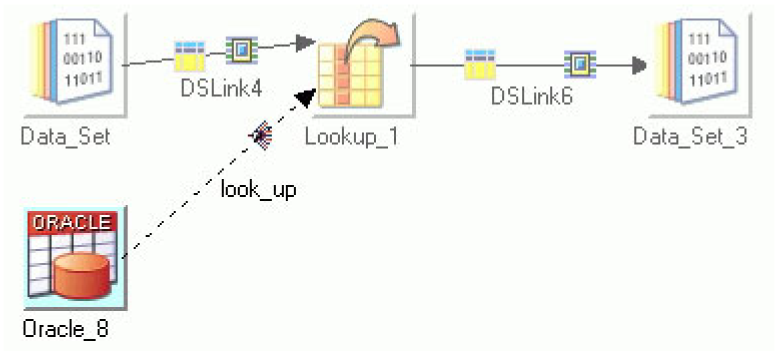


Figure 1. Example look up job

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

Table 6. Properties for Oracle stage

Property name	Setting
Lookup Type	Sparse
Read Method	Table
Table	interest

Updating an Oracle table

This example shows an Oracle table being updated with three new columns. The database records the horse health records of a large stud. Details of the worming records are being added to the main table and populated with the most recent data, using the existing column "name" as a key. The metadata for the new columns is as follows:

Table 7. Column metadata on the Properties tab

Column name	Key	SQL type	Extended	Nullable
name	✓	Char		No

Table 7. Column metadata on the Properties tab (continued)

Column name	Key	SQL type	Extended	Nullable
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 WebSphere 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 WebSphere 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

WebSphere DataStage has many defaults which means that it can be very easy to include Oracle enterprise stages in a job. This section specifies the minimum steps to take to get a Oracle enterprise stage functioning. WebSphere 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.

The steps required depend on what you are using an Oracle enterprise stage for.

Updating an Oracle database

To update an Oracle database:

- In the Input link **Properties** tab, under the Target category specify the update method as follows:
  - Specify a Write Method of Upsert.
  - Specify the Table you are writing.
  - 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 WebSphere DataStage or specify your own.

  - If you have chosen an Upsert Mode of User-defined Update and Insert, specify the Insert SQL statement to use. WebSphere DataStage provides the auto-generated statement as a basis, which you can edit as required.
  - 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. WebSphere 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 WebSphere DataStage generate one for you using a user name and password you supply. Either way you need to supply a valid user name and password. WebSphere DataStage encrypts the password when you use the auto-generate option.

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

    Under the Options category:
  - If you want to send rejected rows down a rejects link, set Output Rejects to True (it is false by default).
- Ensure column metadata has been specified for the write.

Deleting rows from an Oracle database

This is the same as writing an Oracle database, except you need to specify details of the SQL statements used to delete rows from the database:

- In the Input link **Properties** tab:
  - Select a Write Method of Delete Rows.
  - Select the Delete Rows Mode, this allows you to specify whether to use a statement automatically generated by WebSphere DataStage or specify your own.
  - If you select a Delete Rows Mode of User-defined delete, specify the Delete SQL statement to use. WebSphere DataStage provides the auto-generated statement as a basis, which you can edit as required.

Loading an Oracle Database

This is the default write method.

- In the Input link **Properties** tab, under the Target category:
  - Specify a Write Method of Load.
  - Specify the Table you are writing.
  - Specify the Write Mode (by default WebSphere 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 WebSphere DataStage generate one for you using a user name and password you supply. Either way you need to supply a valid user name and password. WebSphere DataStage encrypts the password when you use the auto-generate option.

    By default, WebSphere DataStage assumes Oracle resides on the local server, but you can specify a remote server if required.
- Ensure column metadata has been specified for the write.

Reading an Oracle database

To read an Oracle database:

- In the Output link **Properties** tab:
  - 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).
  - Specify the table to be read.
  - If using a Read Method of user-generated SQL, specify the SELECT SQL statement to use. WebSphere 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 WebSphere DataStage generate one for you using a user name and password you supply. Either way you need to supply a valid user name and password. DataStage encrypts the password when you use the auto-generate option.

    By default, WebSphere DataStage assumes Oracle resides on the local server, but you can specify a remote server if required.
- Ensure column metadata has been specified for the read.

Performing a direct lookup on an Oracle database table

To perform a direct lookup:

- Connect the Oracle enterprise stage to a Lookup stage using a reference link.
- In the Output link **Properties** tab:
  - Set the Lookup Type to Sparse.
  - 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.
  - Specify the table to be read for the lookup.
  - If using a Read Method of user-generated SQL, specify the SELECT SQL statement to use. WebSphere 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 WebSphere DataStage generate one for you using a user name and password you supply. Either way you need to supply a valid user name and password. WebSphere DataStage encrypts the password when you use the auto-generate option.

    By default, WebSphere DataStage assumes Oracle resides on the local server, but you can specify a remote server if required.
- Ensure column meta data has been specified for the lookup.

Performing an in memory lookup on an Oracle database table

This is the default method. It has the same requirements as a direct lookup, except:

- In the Output link **Properties** tab:
  - Set the Lookup Type to Normal.

Stage page

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

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 WebSphere 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

The **NLS Map** tab allows you to 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 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 a WebSphere DataStage map. To do this, add an entry to the file *oracle_cs*, located at *$APT_ORCHHOME/etc*, to associate the WebSphere 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 WebSphere DataStage map names and the second column the Oracle map names they are associated with.

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

Inputs page

The Inputs page allows you to 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 *WebSphere DataStage Parallel Job Developer Guide* for a general description of the other tabs.

Input Link Properties tab

The **Properties** tab allows you to specify properties for the input link. These dictate how incoming data is written and where. 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 8. 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
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

Table 8. Input link properties and values (continued)

Category/ Property	Values	Default	Required?	Dependent of
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
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

Table 8. Input link properties and values (continued)

Category/ Property	Values	Default	Required?	Dependent of
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

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**. WebSphere 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**. WebSphere 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**. WebSphere 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

These are the properties available in the Connection category.

DB Options

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

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

WebSphere 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, WebSphere 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. WebSphere 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

These are the properties available in the Options category.

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 WebSphere 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 8 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

The **Partitioning** tab allows you to specify details about how the incoming data is partitioned or collected before it is written to the Oracle database. It also allows you to specify that the data should be sorted before being written.

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 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)**. WebSphere 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 Auto mode, WebSphere 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

The Outputs page allows you to 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** drop-down 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.

Details about Oracle enterprise stage properties are given in the following sections. See the *WebSphere DataStage Parallel Job Developer Guide* for a general description of the other tabs.

Output Link Properties tab

The **Properties** tab allows you to specify properties for the output link. These dictate how incoming data is read from what 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 *WebSphere DataStage 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 9. 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
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

Table 9. Output link properties and values (continued)

Category/ Property	Values	Default	Required?	Dependent of
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

These are the properties available in the Source category.

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 WebSphere 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 *WebSphere DataStage 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

These are the properties available in the Connection category.

DB Options

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

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

WebSphere 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, WebSphere 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. WebSphere 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

These are the properties available in the Options category.

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 2. Oracle OCI Stages

Oracle OCI lets you rapidly and efficiently prepare and load streams of tabular data from any IBM WebSphere 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 Windows® or UNIX® uses SQL*Net to access an Oracle server on Windows or UNIX.

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 using an SQL statement constructed by WebSphere 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 using an SQL statement constructed by WebSphere 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 Version 9.*n* or 10*g* client, which uses SQL*Net to access the Oracle server.

Oracle OCI works with Oracle Version 9*i* and 10*g* servers, provided you install the appropriate Oracle 9*i* or 10*g* software. For information about exceptions to this, see Oracle documentation for the appropriate release.

With Oracle OCI, you can:

- Generate your SQL statement.
- Use a file name to contain your SQL statement.
- Clear a table before loading 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.

Oracle OCI 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 WebSphere DataStage jobs to run successfully using this stage.

Functionality of Oracle OCI Stages

Oracle OCI has the following functionality and benefits:

- 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 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 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 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.

Configuration Requirements of Oracle OCI Stages

Oracle OCI requires the following configuration:

- Version 9.*n* or 10*g* of the Oracle client software on the WebSphere DataStage **server**, which requires one of the following configurations:
  - **Version 9*i*:** Oracle9*i* Client (run time)
  - **Version 10g**: Oracle 10g Client (run time)

  Note: AIX® 5.1 requires version 9.2 or later of the Oracle client software.
- 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 10. 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 WebSphere DataStage library entries must be referenced before any branded-ODBC library entries at run time.

The Oracle Connection

When you use the client GUI to edit an Oracle OCI stage, the ORAOCI9Stage dialog box opens.

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. For details see the following section, ″Connecting 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.) For details, see ″Defining Character Set Mapping″
- **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

To edit an Oracle OCI stage from the ORAOCI9 Stage window:

1. Define the connection (see the following section).
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

Set the Oracle connection parameters on the **General** tab on the Stage page of the GUI. To connect to an Oracle database:

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 (see the **General** tab in ″Defining Input Data″ or ″Defining Output Data″). 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. Do this from the **NLS** tab on the Stage page. The **NLS** tab appears only if you have installed NLS.

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 WebSphere 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.

Converting an Oracle 8 Project to Oracle 9 or 10

The OCI8TO9.B utility allows you to migrate any Oracle OCI 8 stage in your jobs to an Oracle OCI stage. This utility can be run on WebSphere DataStage release 6.0 and later. The utility is located in the utilities/unsupported directory. Before beginning, create a backup of the project to be converted. Also ensure no other user is accessing the project at the time of conversion.

Use ftp in binary mode to move the utility to your WebSphere DataStage server, placing it into a temporary directory. Copy the utility into the DSU_BP.O directory of each project that you want to convert. In addition, on Unix you need to run

```
chmod 750 OCI8TO9.B
```

Start WebSphere DataStage Administrator from any WebSphere DataStage Client workstation. Select the **Project** tab, highlight the project you are converting, and click **Command**. At the **Command** prompt, type

```
RUN DSU_BP OCI8TO9.B
```

Click **Execute**. You are prompted to continue or exit. Reply with either 'Y' or 'N' and click **Respond**. If continuing, the output will pause at end-of-page by default; you can either click **Next** after each pause or clear **Pause...**.

The utility creates a report of the jobs that were converted in the &COMO& directory called OCI8TO9.CONV.

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 dialog box of the GUI.

About the Input Page

The Input page has an **Input name** field; the **General**, **Options**, **Columns**, **SQL**, and **Transaction Handling** tabs; and the **Columns...** and **View Data...** buttons:

- 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.

- 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.
- 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

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 WebSphere DataStage and 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 WebSphere 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 WebSphere 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

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

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

Oracle OCI supports transaction grouping, that is, the grouping of input links that come from a Transformer stage. This lets you control the group of input links for start, commit, or rollback in one transaction when writing to a single data source. You can use **On Fail** or **On Skip** values to specify whether the transaction is committed.

This 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

To specify transaction control information for a transaction group:

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 offending row to the log for the job. For the Oracle message detail, you must use the error messages returned by the Oracle database.

WebSphere DataStage provides additional reject row handling. To use this capability:
1. Set **Array Size** to 1.
2. Use a Transformer stage to redirect the rejected rows.

You can then 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 offending row values.

Writing Data to Oracle

The following sections describe the differences when you use generated or user-defined SQL INSERT, DELETE, or UPDATE statements to write data from WebSphere 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.

To access the SQL builder from a server stage:

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.

Using Generated SQL Statements

By default, WebSphere DataStage writes data to an Oracle table using an SQL INSERT, DELETE, or UPDATE statement that it constructs. The generated SQL statement is automatically constructed using the WebSphere DataStage table and column definitions that you specify in the input properties for this stage. The **SQL** tab displays the SQL statement used to write the data.

To use a generated SQL statement:

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.

Using User-Defined SQL Statements

Instead of writing data using an SQL statement constructed by WebSphere DataStage, you can enter your own SQL INSERT, DELETE, or UPDATE statement for each ORAOCI input link. (You can include other SQL statements such as CREATE TABLE only in a Before SQL statement.) Ensure that the SQL statement contains the table name, the type of update action you want to perform, and the columns you want to write.

To enter a user-defined SQL statement:

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 you are extracting from an Oracle database. You can also specify the data on an output link using an SQL statement constructed by IBM WebSphere 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.

About the Output Page

The Output page has one field and the **General**, **Options**, **Columns**, and **SQL tab**s.

- **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

This tab is displayed by default. It provides the type of query and, where appropriate, a button to open an associated dialog box. 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 WebSphere 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 dialog opens (see ″SQL Clauses Dialog Box″). Use this dialog box to refine the SQL statement.
  - **Enter custom SQL statement.** Specifies that a custom SQL statement is built using the **SQL** tab. See ″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 Dialog Box

Use this dialog box 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

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

This tab contains the column definitions for the data being output on the chosen link.

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 WebSphere 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. Oracle OCI key reads the data by using a WHERE clause in the SQL SELECT statement.

See the *WebSphere DataStage Designer Client Guide* for

- A description of how to enter and edit column definitions
- Details on how key fields are specified and used

SQL Tab

Use this tab page to build the SQL statements used to read data from Oracle. It contains the **Query**, **Before**, and **After** tab pages:

- **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

The following sections describe the differences when you use generated queries or user-defined queries to read data from an Oracle database into IBM WebSphere 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

WebSphere DataStage extracts data from an Oracle data source using a complete SQL SELECT statement that it constructs. The SQL statement is automatically constructed using the information that you entered in the stage output properties.

To use generated queries:

1. Select **Generate SELECT clause from column list; enter other clauses**. Data is extracted from an Oracle database using an SQL SELECT statement constructed by WebSphere DataStage. 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 a 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** text boxes in the SQL Clauses window.

For example,

- 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 constructed by WebSphere DataStage, you can enter your own SQL statement for each Oracle OCI output link.

To enter a user-defined SQL query:

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). IBM WebSphere DataStage maps the Oracle DATE data type to a Timestamp data type. This is the default WebSphere DataStage data type when you import the Oracle metadata type of DATE.

WebSphere DataStage uses a conversion of *YYYY-MM-DD HH*24:*MI*:*SS* when reading or writing an Oracle date. If the WebSphere DataStage data type is Timestamp, WebSphere DataStage uses the **to_date** function for this column when it generates the INSERT statement to write an Oracle date. If the WebSphere DataStage data type is Timestamp or Date, WebSphere 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 WebSphere 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 following tables document the support for Oracle data types. When creating WebSphere DataStage table definitions for an Oracle table, specify the SQL type, length, and scale attributes as noted.

Character Data Types

The following table summarizes character data types for Oracle, their WebSphere DataStage SQL type definitions, and the corresponding length attributes that you need to specify:

Table 11. Oracle's character data types and WebSphere DataStage's corresponding data types

Oracle Data Type	WebSphere 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 11. Oracle's character data types and WebSphere DataStage's corresponding data types (continued)

Oracle Data Type	WebSphere 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 following table summarizes the NUMBER data type for Oracle, its WebSphere DataStage SQL type definitions, and the corresponding length and scale attributes that you need to specify:

Table 12. Oracle's numeric data types and WebSphere DataStage's corresponding data types

Oracle Data Type	WebSphere DataStage SQL Type	Length	Scale	Notes®
NUMBER (*p*,*s*)	Decimal Double Float Numeric Integer Real	p *p*	s *s*	The WebSphere 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 10g

The following table summarizes the additional numerical data types for Oracle 10g and their WebSphere DataStage SQL type definitions:

Table 13. Oracle's additional numeric data types and WebSphere DataStagee's corresponding data types

Oracle Data Types	**WebSphere DataStage SQL Type**	**Notes**
BINARY_DOUBLE	Double	• When a table is read, WebSphere DataStage converts columns with a data type of BINARY_DOUBLE to SQL_DOUBLE. • When a table is updated, WebSphere 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 13. Oracle's additional numeric data types and WebSphere DataStagee's corresponding data types (continued)

Oracle Data Types	WebSphere DataStage SQL Type	Notes
BINARY_FLOAT	Float	• When a table is read, WebSphere 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, WebSphere 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, WebSphere DataStage converts SQL_FLOAT to BINARY_FLOAT. If **BINARY_FLOAT** is not present, WebSphere DataStage converts SQL_FLOAT to FLOAT (with precision).

Date Data Types

The following table summarizes the DATE data type for Oracle and its WebSphere DataStage SQL type definition:

Table 14. Oracle's date data types and WebSphere DataStage's corresponding data types

Oracle Data Type	WebSphere DataStage SQL Type	Notes
DATE	Timestamp	The default format for the default WebSphere DataStage data type Timestamp is *YYYY-MM-DD HH*24:*MI*:*SS*. If the WebSphere DataStage data type is Timestamp, WebSphere DataStage uses the **to_date** function for this column when it generates the INSERT statement to write an Oracle date. If the WebSphere DataStage data type is Timestamp or Date, WebSphere DataStage uses the **to_char** function for this column when it generates the SELECT statement to read an Oracle date. For more information, see ″DATE Data Type Considerations″

Miscellaneous Data Types

The following table summarizes miscellaneous data types for Oracle and its WebSphere DataStage SQL type definition:

Table 15. Oracle's miscellaneous data types and WebSphere DataStage's corresponding data types

Oracle Data Types	WebSphere 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 WebSphere DataStage's LONGVARCHAR as the column's data type and provide a Length of more than 4 KB in the **Columns** tab. The maximum size supported by WebSphere DataStage is 2 GB. A column with a data type of CLOB cannot be used as a key.

For a list of unsupported Oracle data types, see ″Functionality of Oracle OCI Stages″ .

Handling $ and # Characters

WebSphere DataStage has been modified to enable it to handle Oracle OCI 9*i* or 10*g* databases which use the WebSphere DataStage reserved characters # and $ in table names and column names. WebSphere DataStage converts these characters into an internal format and then converts them back as necessary.

To take advantage of this facility, perform the following tasks:

- In the WebSphere DataStage Administrator, 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 3. Oracle OCI Load Stages

Oracle OCI Load is a passive stage that loads data from external files into an Oracle table. The Oracle database can reside locally or remotely. 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 WebSphere 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 Oracle OCI Load Stages

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 Oracle OCI Load Stages

See the online readme.txt file for your platform for the latest information about the IBM WebSphere DataStage release. The Oracle OCI Load stage requires the following configuration for WebSphere DataStage and Oracle Enterprise Edition:

- WebSphere DataStage
  - For information about WebSphere DataStage configuration requirements, see *IBM Information Server Planning, Installation, and Configuration Guide*.
- Oracle Client
  - Use a version of the Oracle Client on the WebSphere DataStage server.

Platforms

Your Oracle client and server machines must have the same operating system type, such as UNIX to UNIX or Windows 2000 to Windows 2000, in order to run successfully. If you mix UNIX and Windows platforms for your Oracle client and

Oracle server machines, the WebSphere DataStage job will fail, for example, if the Oracle client is on a UNIX workstation and the Oracle server is on a Windows 2000 workstation.

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:

```
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

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 generate a control file and a data file to populate the database later. The load modes are automatic and manual.

Automatic Load Mode

Automatic loading, which is the default value, loads the data during the WebSphere DataStage job. The stage populates the Oracle database immediately after loading the source data. Automatic data loading occurs when the WebSphere 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 WebSphere 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 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

Using the WebSphere DataStage Designer,

1. Add an Oracle OCI Load stage to your WebSphere DataStage job
2. Link the Oracle OCI Load stage to its data source
3. Specify column definitions using the **Columns** tab
4. Determine the appropriate load mode, as documented in ″Load Modes″

5. Add the appropriate property values on the **Stage** tab, as documented in ″Properties″
6. Compile the job
7. If the job compiles correctly, you can select one of the following actions:
   - Run the job from within the DataStage Designer
   - Run or schedule the job using the WebSphere DataStage Director
8. If the job does not compile correctly, correct the errors and recompile

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.
Schema Name	String		The name of the schema where the table being loaded resides. If unspecified, the schema name is ″User Name.″

Prompt	Type	Default	Description
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 WebSphere DataStage server 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 4. 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 16. 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 from various connectivity stages that WebSphere 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.

Starting SQL builder from a stage editor

You reach the SQL builder through the stage editors. Where the stage type supports the builder you will see a **Build SQL** button. Click this to open the SQL builder. For some stages you will have to select an appropriate access method for the button to be visible. See the documentation for the individual stage types for details.

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.

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.

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 WebSphere 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.

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 WebSphere 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.

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 WebSphere 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.

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 WebSphere 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 top half of the SQL builder are common to all the types of statement you can build. The bottom half comprises a series of tabbed pages. What pages are available depends on the type of query you are building.

Toolbar

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 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

This displays the table definitions that currently exist within the WebSphere DataStage repository. The easiest way to get a table definition into the repository is to import it directly from the database you want to query. You can do this via 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

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 Meta Data** 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`.

For details about joining tables, see Joining Tables.

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

The Selection page appears when you are using the SQL builder to define a Select statement. Use this page to specify details of your select query. It has the following components.

Column Selection Grid

This is where you specify which columns are to be included 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 drop-down list in the grid.
- double-click the column name in the table selection canvas.
- copy and paste from the table selection canvas.

The grid has the following fields:

Column expression

Identifies the column to be included in the query. You can 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)
- **Lookup Column**. You can directly select a column from one of the tables in the table selection canvas.

Table

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 drop-down list.

To specify the table name at runtime, choose a job parameter from the drop-down list.

Column Alias

This allows you to specify an alias for the column.

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.

Sort

Choose Ascending or Descending to have the query sort the returned rows by the value of this column. Selecting to sort results in an ORDER BY clause being added to the query.

Sort Order

Allows you to specify the order in which rows are sorted if you are ordering by more than one column.

Context Menu

A shortcut menu allows you to:
- Paste a column that you've copied from the table selection canvas.
- Insert a row in the grid.
- Show or hide the filter panel.
- Remove a row from the grid.

Filter Panel

The filter panel allows you to specify a WHERE clause for the SELECT statement you are building. It comprises a predicate list and an expression editor panel, the contents of which depends on the chosen predicate.

See Expression Editor for details on using the expression editor that the filter panel provides.

Filter Expression Panel

This panel, at the bottom of the SQL builder window, displays any filters that you have added to the query being built. You can edit the filter manually in this panel. Alternatively you can type a filter straight in, without using the filter expression editor.

Group Page

The Group page appears when you are using the SQL builder to define a select statement. Use the Group page to specify that the results of a select query are grouped by a column, or columns. Also, use it to aggregate the results in some of the columns, for example, you could specify COUNT to count the number of rows that contain a not-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

This is where you specify which columns are to be grouped by or aggregated 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 available vary according to the stage you have opened the SQL builder from. The following are the basic ones supported by all SQL syntax variants.

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

The filter panel allows you to specify a HAVING clause for the SELECT statement you are building. It comprises a predicate list and an expression editor panel, the contents of which depends on the chosen predicate.

See Expression Editor for details on using the expression editor that the filter panel provides.

Filter Expression Panel

This panel displays any filters that you have added to the query being built. You can edit the filter manually in this panel. Alternatively you can type a filter straight in, without using the filter panel.

Insert Page

The Insert page appears when you are using the SQL builder to define an insert statement. Use this page to specify details of your insert statement. The only component the page has is insert column grid.

Insert Columns Grid

This is where you specify which columns are to be included in your statement and what values they will take. The grid has the following fields:

Insert Column

Identifies the columns to be included in the statement. You can populate this in a number of ways:

- drag columns from the table in the table selection canvas.
- choose columns from a drop-down list in the grid.
- double-click the column name in the table selection canvas.
- copy and paste from the table selection canvas.

Insert Value

Identifies the values that you are setting the corresponding column to. You can specify one of the following in giving a value. You can also type a value directly into this field.

- **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

The Update page appears when you are using the SQL builder to define an update statement. Use this page to specify details of your update statement. It has the following components.

Update Column Grid

This is where you specify which columns are to be included in your statement and what values they will take. The grid has the following fields:

Update Column

Identifies the columns to be included in the statement. You can populate this in a number of ways:

- drag columns from the table in the table selection canvas.
- choose columns from a drop-down list in the grid.
- double-click the column name in the table selection canvas.
- copy and paste from the table selection canvas.

Update Value

Identifies the values that you are setting the corresponding column to. You can specify one of the following in giving a value. You can also type a value directly into this field.

- **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

The filter panel allows you to specify a WHERE clause for the update statement you are building. It comprises a predicate list and an expression editor panel, the contents of which depends on the chosen predicate.

See Expression Editor for details on using the expression editor that the filter panel provides.

Filter Expression Panel

This panel displays any filters that you have added to the query being built. You can edit the filter manually in this panel. Alternatively you can type a filter straight in, without using the filter panel.

Delete Page

The Delete page appears when you are using the SQL builder to define a delete statement. Use this page to specify details of your delete statement. It has the following components.

Filter Panel

The filter panel allows you to specify a WHERE clause for the delete statement you are building. It comprises a predicate list and an expression editor panel, the contents of which depends on the chosen predicate.

See ″Expression Editor″ for details on using the expression editor that the filter panel provides.

Filter Expression Panel

This panel displays any filters that you have added to the query being built. You can edit the filter manually in this panel. Alternatively you can type a filter straight in, without using the filter panel.

Sql Page

Click the Sql tab to view the generated statement. Using the shortcut menu, you can copy the statement for use in other environments.

For select queries, if the columns you have 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 you have loaded onto your stage editor (the loaded columns) do not match the columns generated by the SQL statement (the result columns) you have defined, the Resolve columns grid gives you the opportunity to reconcile them. Ideally the columns should match (and in normal circumstances usually would). A mismatch would cause the meta data in your job to become out of step with the meta data 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

The Expression Editor allows you to specify details of a WHERE clause that will be inserted in your select query or update or delete statement. You can also use it to specify WHERE clause for a Join condition where you are joining multiple tables, or for a HAVING clause. 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

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, or should not be, null.

If you are building a query for Oracle 8i, you can use Join predicate. The logic appears in the query as a WHERE statement. Oracle 8i does not support JOIN statements.

Between

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

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

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

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

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

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/Case Expression Editor

This version of the expression editor allows you to specify an expression within a WHERE or HAVING expression, or a join condition. Expression Editor dialogs are numbered to show how deeply you are nesting them. Fields in the Expression Editor panel vary according to the chosen predicate as follows:

Calculation

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

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

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.

- **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 or function.
- **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'.
- **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 or function. 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.
- **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 or function.
- **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

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 WebSphere 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 Dialog Box

This dialog box allows you to select a function for use within and 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 Dialog Box:

This dialog box allows you to select a function for use within and 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 Dialog Box

This dialog box lists the job parameters that are currently defined for the job within which you are working. It also gives the data type of the parameter. Note that the SQL builder does not check that the type of parameter you are inserting matches the type expected by the argument you are using it for.

Joining Tables

When you use the SQL builder to help you build select queries, you can specify table joins within the query.

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. It uses captured foreign key meta data where this is available. 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

There are three ways of altering the automatic join that the SQL builder inserts when you add more than one table to the table selection canvas:

- 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 Dialog Box

This dialog box allows you to change the type of an existing join and modify or specify the join condition.

The dialog box 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 Dialog Box

This dialog box displays all the foreign key relationships that have been defined between the target table and other tables that appear to the left of it in the table selection canvas. You can select the relationship that you want to appear as the join in your query by selecting it so that it appears in the list box, and clicking **OK**.

Properties Dialogs

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. This dialog is described in Join Properties Dialog Box.

Table Properties Dialog Box

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 (see Parameters 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 Dialog Box

This dialog box gives you details about the SQL grammar that the SQL builder uses. It 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 5. Oracle connector

The following topics describe how to configure and use the Oracle connector in a job.

Installation and configuration requirements

Before you create a job that uses the Oracle connector, install the connector and set the required environment variables.

Confirm that your system meets the system requirements and that you are using a supported version of the Oracle client and Oracle server. For system requirement information, go to http://www-01.ibm.com/software/data/infosphere/info-server/overview/

Follow the installation instructions that are provided for the connector. Then to ensure that the connector operates properly, you must set the library path. NLS_LANG, TNS_ADMIN, and ORACLE_HOME are Oracle environment variables that you might consider setting as part of the configuration process. For additional information about the Oracle environment variables, see the Oracle documentation.

Library path
: Edit the library path environment variable to include the directory where the Oracle client libraries are located. The actual name of this environment variable is PATH on Microsoft Windows, SHLIB_PATH on HP-UX for PA-RISC, LIBPATH on IBM AIX, and LD_LIBRARY_PATH on all other supported UNIX and Linux® operating systems.

NLS_LANG
: This Oracle environment variable specifies the language, territory and character set settings for the data that the Oracle connector and the Oracle client exchange. If you do not set this environment variable, the Oracle client uses the default value AMERICAN_AMERICA.US7ASCII. Note that on Microsoft Windows systems, you can specify the NLS_LANG value in the Microsoft Windows Registry. Then when the NLS_LANG environment variable is not set, the Oracle client uses the value from the registry rather than using the default value AMERICAN_AMERICA.US7ASCII.

TNS_ADMIN or ORACLE_HOME
: You can use either the Oracle TNS_ADMIN or the Oracle ORACLE_HOME environment variable to specify the location of the Oracle `tnsnames.ora` configuration file. The `tnsnames.ora` configuration file contains connect descriptors that describe the destination services and network route information that the connector requires to establish connections to Oracle databases.

  If you use the TNS_ADMIN environment variable, specify as its value the directory where the `tnsnames.ora` file is located. If you use the ORACLE_HOME environment variable, specify as its value the directory that contains the network subdirectory, which contains the admin subdirectory, which contains the `tnsnames.ora` file.

If you set both the TNS_ADMIN and ORACLE_HOME environment variables, the TNS_ADMIN environment variable takes precedence over the ORACLE_HOME environment variable for locating the `tnsnames.ora` configuration file.

The TNS_ADMIN and the ORACLE_HOME environment variables are not mandatory. However, if one or both are not specified, 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.

Using the Oracle connector

Follow these step-by-step directions for using the Oracle connector to read, write, or look up Oracle data.

Reading data from an Oracle database

Configure the Oracle connector to connect to an Oracle database and read data from it.

Prerequisites

Complete these prerequisite tasks:

- Verify that the Oracle connector is installed and configured properly.
- Verify that the user name that connects to the Oracle database has the correct authority and privileges to perform the actions of the job.

About this task

The following figure shows an example of using the Oracle connector to read data. In this case, the Oracle connector reads data from an Oracle database, passes the rows to a Transformer stage, which transforms the data and then loads the data into the ODBC connector. When you configure the Oracle connector to read data, you create only one output link, which is shown here transferring rows to the Transformer stage.

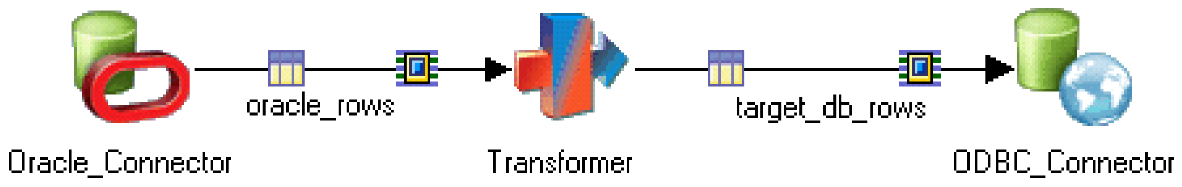


Figure 2. Example of reading data from an Oracle database

Procedure

To configure the connector to read data from an Oracle database, complete these tasks:

1. "Importing Oracle metadata" on page 82
2. "Creating a job that includes the Oracle connector and the required links" on page 83
3. "Defining a connection to an Oracle database" on page 96
4. "Setting up column definitions on a link" on page 97
5. "Specifying the read mode and the data source" on page 99
6. "Compiling and running a job" on page 104

Writing data to an Oracle database

Configure the Oracle connector to connect to an Oracle database and write data to it.

Prerequisites

Complete these prerequisite tasks:

- Verify that the Oracle connector is installed and configured properly.
- Verify that the user name that connects to the Oracle database has the correct authority and privileges to perform the actions of the job.

About this task

The following figure shows an example of using the Oracle connector to write data. In this case, 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. When you configure a reject link, you can choose to include the Oracle error code and associated message text with each rejected record. Then after you run the job, you can evaluate the rejected records and adjust the job and the data accordingly.

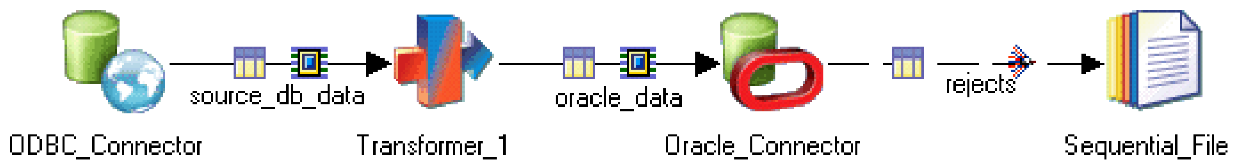


Figure 3. Example of writing data and using a reject link.

Procedure

To configure the connector to write data to an Oracle database, complete these tasks:

1. "Importing Oracle metadata" on page 82
2. "Creating a job that includes the Oracle connector and the required links" on page 83
3. "Defining a connection to an Oracle database" on page 96

4. “Setting up column definitions on a link” on page 97.
5. “Specifying the write mode and the target table” on page 100
6. Optional: “Rejecting records that contain errors” on page 102
7. “Compiling and running a job” on page 104

Looking up data in an Oracle database

Configure the connector to perform either a normal lookup or a sparse lookup on an Oracle database.

Prerequisites

Complete these prerequisite tasks:

- Verify that the Oracle connector is installed and configured properly.
- Create a job that includes an Oracle connector that is linked to a Lookup stage by a reference link. For the Lookup stage, define the columns and column mappings for the input link, output link, and input reference link.
- Verify that the user name that connects to the Oracle database has the correct authority and privileges to perform the actions of the job.

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. Even though 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 stage 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.

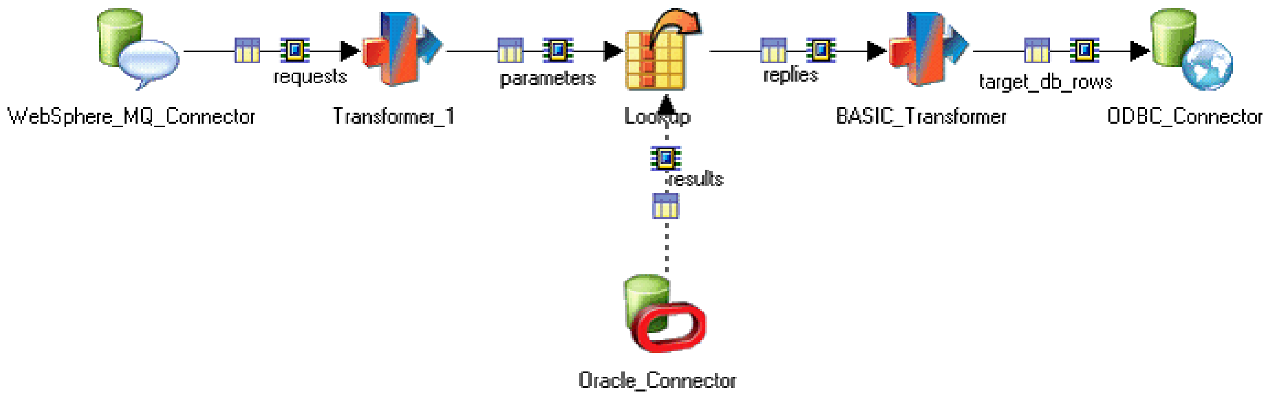


Figure 4. Example of using the Oracle connector with a Lookup stage.

In a normal lookup, the connector runs the specified SELECT statement or PL/SQL block only once; therefore, the SELECT statement or PL/SQL block cannot include any input parameters. The Lookup stage searches the retrieved result set data 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.

In a sparse lookup, the connector runs the specified SELECT statement or PL/SQL block once 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 that the connector sets on the bind variables in the SELECT statement or PL/SQL block, and then the connector runs the statement or block. The result of the lookup is routed as one or more records through the reference link from the 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.

Procedure

To configure the connector to look up data in an Oracle database:

1. Complete these steps:
   a. Add the Oracle connector to the job, create a reference link from the Oracle connector to the Lookup stage, and then double-click the connector to open the properties.
   b. In the **Lookup Type** field, choose **Normal** or **Sparse**.
2. Complete these tasks:
   a. "Defining a connection to an Oracle database" on page 96
   b. "Specifying the read mode and the data source" on page 99
   c. "Setting up column definitions on a link" on page 97
   d. "Compiling and running a job" on page 104

Common tasks for reading and writing data

All jobs require that you complete some of these common tasks.

Setting required user privileges

To run a job that uses the Oracle connector, the user name that the connector uses to connect to the database requires SELECT access to a set of Oracle dictionary views and must have access to the target Oracle database and table.

About this task

To perform some operations, the Oracle connector accesses Oracle dictionary views. All but one of the views are in the ALL_ or USR_ category, and permission to these views is enabled by default. Therefore, the user name that the connector uses to connect to the database typically has access to those views.

However, you must explicitly grant access to the DBA_EXTENTS dictionary view. Access to this view is required for the Rowid range partitioned read method. Rowid range is the default partitioned read method, so the connector attempts to use this method if you do not explicitly select a different partition 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 automatically switches the partitioned read method from Rowid range to Rowid hash.

Procedure

To grant SELECT access to an Oracle dictionary view:

1. To grant SELECT access to a single dictionary view, issue the following statement:

   ```
   GRANT SELECT ON dictionary_view TO user_name
   ```

   where *dictionary_view* is the name of the view and *user_name* is the user name with which the connector connects to the database.
2. To use a role to grant a user SELECT access to multiple dictionary views, use 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 that are specific to your configuration and issue one GRANT SELECT ON statement for each dictionary view.

   ```
   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
   ```

Importing Oracle metadata

Use the Connector Import Wizard to import data objects that represent tables and views in an Oracle database. The data objects are saved in the metadata repository.

Prerequisites

Verify the Oracle connector configuration requirements.

Procedure

To import Oracle metadata:

1. From the IBM WebSphere DataStage and QualityStage Designer client, choose **Import** → **Table Definitions** → **Start Connector Import Wizard**.
2. Select the variant of the Oracle connector that corresponds to the release of the Oracle client that you installed on the IBM WebSphere DataStage Server.
3. Click the down arrow in the **Server** field to obtain a list of Oracle services, and then do one of the following:
   - 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 or ORACLE_HOME environment variables.
   - Enter the complete content of the connect descriptor, as it would appear in the Oracle `tnsnames.ora` file; or enter the Easy Connect string that defines the connection to the Oracle database.
4. In the **Username** and **Password** fields, enter the user ID 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 values that you specify in the **Username** and **Password** fields match the credentials that are configured for the user in the Oracle database.
5. Optional: In the **Use OS authentication** field, select **Yes**. This form of authentication requires that the user be registered in Oracle and identified as a user who is authenticated by the operating system.

   Note: When the Connector Import Wizard or the connector stage dialog invokes the Oracle connector to perform a design-time operation such as importing metadata, viewing data, testing a connection, or enumerating services, the connector runs within the ASB agent process. This process runs on the computer where the IBM WebSphere DataStage Server is installed. On a computer that runs Microsoft Windows, the ASB agent runs under the Local System account; on a computer that runs Linux or UNIX, the ASB agent runs under the root system account. Therefore, choosing **Use OS Authentication** causes design-time operations to use the built-in system accounts to authenticate with the database, a scenario that you typically want to avoid.

6. Click **Test connection**, and then click **Save** to save the connection. If you do not save the connection definition in the repository, only IBM WebSphere DataStage can access the imported metadata; other components and tools of IBM Information Server might have no access to the metadata.
7. In the **Host name** and **Database name** fields, specify the names of the repository objects under which to import the metadata. If you choose the values that the connector provides as defaults, the objects are created in the metadata repository if they don't already exist. Alternatively, choose from the list of Host and Database objects that are already present in the repository. To define new host and database object, click **New location...** The names of the Host and Database objects do not need to match the actual names of the Oracle server host system and database. However, using matching names makes it easy to track the imported metadata. The Host object in the repository serves as a logical container of the Database objects, which in turn serve as containers for the imported metadata objects.
8. The wizard provides three levels of filtering so that you can narrow down the list of objects to import. Perform these steps to specify one or more filters to use:
   a. The **Schema** filter displays a list of all of the table owners in the database. Select a schema as the first filter.
   b. The table types filter displays a list of schema objects types to include in the results. By default, the options **Include views**, **Include tables**, **Include materialized views**, and **Include index-organized tables** are selected. You can also select **Include external tables** and **Include synonyms** in the results. The types filter is the second filter.
   c. In the **Name filter**, enter additional criteria that filters the list of objects by name. You can use the percent sign (%) as a wildcard character. For example, to obtain a list of objects that contain the word *blue* in their names, enter `%blue%` in the **Name filter**.
9. On the **Selection** panel, select one or more tables to import. To import definitions for the primary keys, foreign keys and indexes that are associated with the selected tables, check the related boxes. To view the current data in a table, select the table and then click **View data**. To select tables that have a primary key or foreign key relationship with the selected table, click **Related tables**.
10. Click **Import**, and then select the location in the metadata repository under which to import the table definitions.

Creating a job that includes the Oracle connector and the required links

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

To create a job that includes the Oracle connector and the required links:

1. From the IBM WebSphere DataStage and QualityStage Designer client, select **File** → **New** from the menu.
2. In the **New** window, select the Parallel Job icon, and click OK.
3. Follow these steps to add the Oracle connector to the job:
   a. In the Designer client 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 to the parallel canvas.
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 an input link from the previous stage in the job to the Oracle connector. 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 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.

Configuring a parallel job

When you configure the Oracle connector to run in parallel on multiple nodes, the data that the connector will process is distributed across all of the available nodes. Then working in parallel, each node processes a distinct subset of the data.

To configure a parallel job, configure the nodes, and then configure a parallel read or a parallel write:

Configuring nodes

To modify the number of nodes on which a job runs, edit the configuration file that specifies nodes, node pools, and constraints.

About this task

The default parallel configuration file, `default.apt`, is created when you install IBM WebSphere DataStage. The APT_CONFIG_FILE environment variable specifies the location of the configuration file. The default value of the variable is *IS_HOME*`/Server/Configurations/default.apt` where *IS_HOME* is the home directory of the IBM Information Server. On computers that run Microsoft Windows, the default installation directory is `C:\IBM\InformationServer`. On computers that run UNIX or Linux, the default installation directory is `/opt/IBM/InformationServer`.

You can specify the configuration file at the project level and at the job level. To specify the value of the APT_CONFIG_FILE environment variable as a project property, you use the IBM WebSphere DataStage and QualityStage Administrator client. The file that you specify at the project level controls all stages that are configured to run in parallel in all of the jobs that are included in the project. To specify a configuration file that applies to one specific job, you add the APT_CONFIG_FILE environment variable to the set of job parameters that you define for that job.

There are two ways to specify an environment variable as a job parameter. You can use the **Job Parameters** dialog box to specify the default value to use each time that the job runs, or you can manually specify the value each time that you run the job.

In addition, you can configure the connector to run on a subset of the nodes that are defined in the configuration file.

Procedure

To configure the connector to run on multiple processing nodes:

1. Create a new configuration file or edit an existing configuration file.
2. Set the APT_CONFIG_FILE environment variable to the full path of the configuration file that you want to use.
3. On the **Advanced** tab of the connector stage properties, perform these tasks:
   a. Verify that the **Execution mode** property is set to **Parallel**, which is the default setting.
   b. Optional: Use the **Node pool and resource constraints** and the **Node map constraint** fields to restrict the nodes on which the connector runs.

      Note: If you plan to use the **Oracle partitions** read method to read data from a partitioned Oracle table or if you plan to use the **Oracle connector** partition type to write data to a partitioned Oracle table, do not specify any constraints. As an alternative include the ORACLE resource constraint in the configuration file to specify the nodes on which you want to run the Oracle connector.

Examples: Constraining nodes in a parallel job:

These examples present a sample parallel configuration file and show how to use node pools to constrain the nodes on which the connector runs.

In the parallel configuration file, you specify nodes and node pools. Then you use one of the following methods to configure the connector to run on only a subset of the nodes that are specified in the parallel configuration file:

- Define a node pool in the parallel configuration file. Then on the **Advanced** tab of the connector properties, select that node pool. The connector runs on only the nodes that are members of that node pool.
- On the **Advanced** tab of the connector properties, select specific nodes.

The following example parallel configuration file defines four nodes: node1, node2, node3, and node4. Each node runs on MYHOST, which is the computer that runs the IBM WebSphere DataStage Server. The file defines four node pools: pool1, pool2, pool3, and pool4, as well as the default "" pool. The APT_CONFIG_FILE environment variable points to this parallel configuration file.

```
{
 node "node1"
  {
  fastname "MYHOST"
  pools "" "group1" "group2" "group3"
  resource disk "/opt/IBM/InformationServer/Server/Datasets" {pools ""}
  resource scratchdisk "/opt/IBM/InformationServer/Server/Scratch" {pools ""}
  }
 node "node2"
  {
  fastname "MYHOST"
  pools "" "group1" "group2"
  resource disk "/opt/IBM/InformationServer/Server/Datasets" {pools ""}
  resource scratchdisk "/opt/IBM/InformationServer/Server/Scratch" {pools ""}
  }
 node "node3"
  {
  fastname "MYHOST"
  pools "" "group1" "group3"
  resource disk "/opt/IBM/InformationServer/Server/Datasets" {pools ""}
  resource scratchdisk "/opt/IBM/InformationServer/Server/Scratch" {pools ""}
  }

 node "node4"
  {
  fastname "MYHOST"
  pools "" "group1" "group2"
  resource disk "/opt/IBM/InformationServer/Server/Datasets" {pools ""}
  resource scratchdisk "/opt/IBM/InformationServer/Server/Scratch" {pools ""}
  }
}
```

For the first example, assume that you perform these steps:
1. On the **Advanced** tab of the stage properties, select **Node pool and resource constraints**.
2. In the **Constraint** field, select **Node pool**.
3. In the **Name** field, select **group3**.

The connector is restricted to running on only node1 and node3 because only these two nodes belong to the group3 node pool.

For the second example, assume that you perform these steps:
1. On the **Advanced** tab of the stage properties, select **Node map constraint**.
2. Select **node1** and **node2**.

The connector is restricted to running on only node1 and node2 because those nodes are explicitly specified.

Configuring a parallel read

In a parallel read, each node that is specified for the stage reads a distinct subset of data from the source table.

Prerequisites

The connector must be used in a parallel job.

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

If the connector is configured to look up data, the **Lookup type** must be **Normal**.

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

To configure a parallel read:

1. On the **Advanced** tab, set **Execution mode** to **Parallel**.
2. On the **Properties** tab, set **Enable partitioned reads** to **Yes**.
3. Set **Read mode** to **Select**.
4. Use one of these methods to define the SELECT statement that the connector will use at runtime:
   - Set **Generate SQL at runtime** to **Yes**, and then enter the name of the table 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 schema that belongs to currently connected user is used. The connector automatically generates and runs the SELECT * FROM *schema_name.table_name* statement.

     Note: 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 generated SELECT statement. 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. On the **Partitioning** tab, set the **Partitioned reads method** property to the partitioning method that you want to use. The default partitioning method is **Rowid range**.
6. To provide the input values that the partitioned read method uses, complete these steps:
   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.

      Note: 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 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.

Support for partitioned read methods:

The 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

Every Oracle table includes the ROWID pseudo column that contains a rowid value that uniquely identifies each row in the table. When you use the Rowid range method, the connector performs 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 runtime, 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:

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

In these cases, the connector logs a Warning message and automatically uses the Rowid hash method, which does not have these restrictions.

Rowid round robin

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 in which the row resides and 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.

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. In general, the Rowid range method requires that the connector perform fewer read operations on the source table.

Rowid hash

The Rowid hash method is similar to the Rowid round robin method except that 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.

Modulus

To use this method, you must specify a column name from the input table in the **Column name for partitioned reads** property. The specified column must be of the data type NUMBER(*p*), where *p* is a value between 1 and 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, which is used only if you do not explicitly specify the table name in one of the other two properties.

For each node, the connector reads the rows that satisfy the following condition: MOD(*column_value*, *number_of_nodes*) = *node_number*, where MOD is the modulus function, *column_value* is the value for the column specified in **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, first node has index 0; the second node has index 1; and so on.

Minimum and maximum range

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

The connector calculates the minimum and maximum value for the specified column and then divides the calculated range into an equal number of subranges. 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 for which the specified column has the values that are within the subrange that is associated with that node.

Oracle partitions

The Oracle partitions method can be used with partitioned tables. 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 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 associated with the current node. Consequently, when you specify a value for the **Select statement** property, do not include a PARTITION or SUBPARTITION clause.

Note that 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 dialog 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.

Examples: Using partitioned read methods:

To understand how each partitioned read method works, review these examples of using the Rowid range, Rowid round robin, Rowid hash, Modulus, Minimum and maximum range, and Oracle partitions methods.

Rowid range

This the configuration for this example:
- 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

This is the configuration for this example:
- 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

This is the configuration for this example:
- 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

This is the configuration for this example:
- 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

This is the configuration for this example:
- 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

This is the configuration for this example:

- 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
```

Configuring a parallel write

In a parallel write, 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.

Prerequisites

The connector must be used in a parallel job.

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 specific partition type based on your knowledge about the actual data and the target table that the connector will work with at runtime. In particular, if the target table is range partitioned or list partitioned, selecting **Oracle connector** might be a good choice because then the connector partitions the input records so that each node writes rows to the partition that is associated with that node.

Procedure

To configure a parallel write:

1. From the parallel canvas, double-click the connector icon, and then select the input link.
2. On the **Partitioning** tab, select a partition type.

Oracle connector partition type:

For 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.

The Oracle connector supports the use of the built-in partition types such as Random and Modulus. In addition, the connector provides one additional partition type: Oracle connector. The following information describes how the connector works when you select Oracle connector as the partition type

To partition the input records across nodes when Oracle connector partition type is selected, the connector first looks at the partitioning information for the table. In most cases, the name of the table matches the name of the table to which the connector writes the data; therefore, the table name 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.

The connector logs an Informational message that contains the name of the table from which it collects partitioning information. If the connector lacks sufficient information to determine the name of the table, the connector logs a Warning message and forces sequential execution.

After determining 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. There are three cases that result in a mismatch between the number of nodes and the number of partitions. In the first case, the configuration of the parallel processing nodes specifies a node pool, a resource constraint, or a node map. If the configuration specifies a constraint, the connector cannot dynamically modify the set of processing nodes, reports a Fatal error, and stops the operation. In the second case, 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. In the third case, 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 added node matches the definition of the last node in the original list.

Next, the connector determines the node to which to send each input record. 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 that number of partitions and the partition type that the table will have after it is created at runtime. The connector uses this information to determine the number of nodes that the stage will run on. Note that if the table uses a supported partition type, for example range, list, or interval, 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 actually 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 message 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 because it 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** is set to **Single partition** or **Single subpartition**, the connector runs the stage in sequential mode and logs a Warning message. 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.

Oracle partition types

The following list describes how the Oracle connector partition type supports specific Oracle partition types.

Range, Composite range-range, Composite range-list, Composite range-hash,
: The Oracle connector partition type supports writing to range-partitioned tables. 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
: The Oracle connector partition type supports writing to list-partitioned tables. 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 The Oracle connector partition type does not support writing to hash-partitioned tables. In this case, the connector runs the stage in sequential mode and logs a Warning message.

Interval, Composite interval-range, Composite interval-list, Composite interval-hash
The Oracle connector partition type supports writing to interval-partitioned tables. 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 new, dynamically created partitions.

Reference
The Oracle connector partition type does not support writing to reference-partitioned tables. In this case, the connector runs the stage in sequential mode and logs a Warning message.

Virtual
The Oracle connector partition type does not support writing to a table in which the partition key includes a virtual column. In this case, the connector runs the stage in sequential mode and logs a Warning message.

System
The Oracle connector partition type does not support writing to system-partitioned tables. In this case, the connector runs the stage in sequential mode and logs a Warning message.

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.

Prerequisites

Complete these prerequisite tasks:
- Verify that the Oracle connector is installed and configured properly.
- Create a job that reads or writes data, add the Oracle connector to the job, and create any required links.
- Verify that the user name that connects to the Oracle database has the correct 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 correctly set: TNS_ADMIN, ORACLE_HOME, ORACLE_SID, TWO_TASK, and LOCAL.

Procedure

To define a connection to an Oracle database:
1. Double-click the connector stage icon to open the connector properties.
2. In the **Server** field, do one of the following:

- 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, as it would appear in the Oracle `tnsnames.ora` file.
- Use the following syntax to enter an Oracle Easy Connect string: *host*[:*port*][/*service_name*]
- Leave the property blank to connect to the default local Oracle service. 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.

  Note: Selecting an Oracle service is preferable to using the TWO_PHASE or LOCAL environment variables.

3. In the **Username** and **Password** fields, enter the user ID 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 specified name and password match the credentials that are configured for the user in the Oracle database.
4. Optional: In the **Use OS authentication** field, select **Yes**. This form of authentication requires that the user be registered in Oracle and identified as a user who is authenticated by the operating system.

Setting up column definitions on a link

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

Procedure

To set up column definitions:

1. From the parallel canvas, double-click the Oracle connector icon.
2. In the top left corner of the stage editor, select the link to edit. Note that you cannot directly edit columns on a reject link. The columns on the reject link are copies of the columns that are defined on the input link and any reject-specific columns that you select on the **Reject** tab of the stage dialog.
3. Use one of the following methods to set up the column definitions:
   a. Drag and drop a table definition from the repository view to the link on the job canvas. Then use the arrow keys to move the columns back and forth between the **Available columns** and **Selected columns** lists.
   b. From the **Columns** tab, click **Load** and select a table definition from the metadata repository. Then to choose which columns from the table definition to apply to the link, move the columns from the list of **Available columns** to the list of **Selected columns**.
4. Right-click within the columns grid, and select **Properties** from the menu. Select the properties to display, specify the order in which to display them, then click **OK**.
5. Modify the column definitions. You can change the column names, data types, and other attributes. In addition, you can manually add or insert new columns or remove existing columns.

6. To save the new table definition in the metadata repository, complete these steps:
   a. From the **Columns** tab, 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**.

Runtime mappings between DataStage columns and SQL statement parameters

When exchanging 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 data type name, length, scale, nullable, and optional 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 takes the appropriate action, for example by failing the job or rejecting the row of data.

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. For example, if the output link defines columns COL1, COL2, and COL3 in that order; and the specified SQL SELECT statement retrieves a result set that is described by the set of columns COL2, COL3 and COL1, the connector matches all of the columns based on their names, even though the column orders do not match. If you set the **Enable quoted identifiers** property to **Yes**, the connector performs case-sensitive name matching. Otherwise, the connector performs case-insensitive name matching, which is the default.

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 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: DataStage syntax and Oracle syntax. The following list describes how the connector performs matching, based on the format that you use to specify the parameters:

DataStage syntax
The DataStage syntax is ORCHESTRATE.*parameter_name*. If you use 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, a message is logged and the operation stops.

Note: To use a keyword other than ORCHESTRATE in the DataStage syntax, define the CC_ORA_BIND_KEYWORD environment variable, and set its value to the keyword that you want to use.

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, the connector reports a Fatal error and the operation stops. If some of all of the name values are not integers, the connector performs matching based on column order.

Note: For PL/SQL blocks, you must use the Oracle syntax. If you use DataStage syntax, the connector logs an error and the operation stops. If you use integer values for parameter names, you must specify the integers in increasing order; otherwise, the connector logs a Fatal message, and the operation stops.

Both DataStage syntax and Oracle syntax
If you use both DataStage syntax and Oracle syntax to specify parameters, the connector logs a Fatal error, and the operation stops. To avoid this problem, you must consistently use the same format to specify parameters.

Specifying the read mode and the data source

To configure the connector to read or 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.

Prerequisites

Complete these prerequisite tasks:
- Create a job that includes the Oracle connector and required links.
- Define a connection to the Oracle database.
- Set up column definitions on the links.

About this task

When you set the **Read mode** property to **Select**, there are four ways to define the source of the data:
- Specify the table or view name.
- Enter the SELECT statement manually.
- Enter the name of a file that contains the SELECT statement.
- Invoke the Oracle SQL Builder to build the SELECT statement for you.

When you set the **Read mode** property to **PL/SQL**, there are two ways to define the source of the data:

- Enter the PL/SQL block manually.
- Enter the name of a file that contains the PL/SQL block.

When you run the job, the connector runs the specified PL/SQL block only once and returns the output bind variables that are specified in the PL/SQL block. 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

To configure the connector to read data:

1. From the parallel canvas, double-click the Oracle connector icon, and then select the output link to edit.
2. Set **Read mode** to **Select** or **PL/SQL**.
3. If you set **Read mode** to **Select**, use one of these methods to specify 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 schema that belongs to currently connected user is used.
   - 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 and drop table and column definitions that are stored in the repository and choose options for configuring clauses in the SQL statement.
4. If you set **Read mode** to **PL/SQL**, use one of these methods to specify the source of the data:
   - Manually enter the PL/SQL block 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**.

   Note: The specified PL/SQL block 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.

Specifying the write mode and the target table

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.

Prerequisites

Complete these prerequisite tasks:

- Create a job that includes the Oracle connector and required links.
- Define a connection to the Oracle database.
- Set up column definitions on the links.

About this task

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

Table 17. 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.
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 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.

Procedure

To configure the connector to write data:

1. From the parallel canvas, double-click the Oracle connector icon and then select the input link to edit.
2. To automatically generate the SQL at runtime, perform these steps:
   a. Set **Generate SQL at runtime** to **Yes**.

   b. Set **Write mode** to **Insert**, **Update**, **Delete**, **Insert then update**, **Update then insert**, or **Delete then insert**.
   c. Enter the name of the target table in the **Table name** property.
3. To manually enter the SQL, perform these steps:
   a. Set **Generate SQL at runtime** to **No**.
   b. Set **Write mode** to **Insert**, **Update**, **Delete**, **Insert then update**, **Update then insert**, or **Delete then insert**.
   c. Enter SQL statements in the properties the correspond to the selected Write mode: **Insert statement**, **Update statement**, **Delete statement**. As an alternative, click **Build** beside each property to start the SQL Builder. Then to build the statement, drag and drop column definitions that are stored in the repository, and choose options for configuring clauses in the statement.
4. To read the SQL statement from a file, perform these steps:
   a. Set **Generate SQL at runtime** to **No**.
   b. Enter the fully-qualified name of the file that contains the SQL statement in the **Insert**, **Update**, **Insert then update**, **Update then insert**, or **Delete then insert** property.
   c. Set **Read insert statement from file**, **Read update statement from file**, or **Read delete statement from file** to **Yes**.
5. To specify a PL/SQL block, perform these steps:
   a. Set the **Write mode** to **PL/SQL**.
   b. Enter the PL/SQL block in the **PL/SQL block** property.

      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.
6. To bulk load data, perform these steps:
   a. Set **Write mode** to **Bulk load**.
   b. Enter the name of the table 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 schema that belongs to currently connected user is used.

Rejecting records that contain errors

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

Prerequisites

Complete these prerequisite tasks:
- Create a job that includes the Oracle connector and required links.
- Define a connection to the Oracle database.
- Set up column definitions on the links.
- Specify the write mode and the target table.

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 criteria, the connector reports a Fatal error and stops the job.

The following list describes the reject conditions that you can set:

Row not updated
This condition occurs when the connector attempts to update a row in the target table and the operation succeeds, but no data is updated. These situations result in a row not being updated:

- The key field values in the input record do not match the key column values of any row in the target table
- The key field values in the input record match the key column values in some rows in the target table, and the remaining column values in the input record match the corresponding column values in those same rows.

The connector checks for this condition only when the **Write mode** property is set to **Update**. Note that this condition does not have a corresponding Oracle error code and error message.

Row not deleted
This condition occurs when the connector attempts to delete a row in the target table and the operation succeeds, but no data is deleted. This situation occurs when the key field values in the input record do not match the key column values any row in the target table.

The connector checks for this condition only when the **Write mode** property is set to **Delete**. Note that this condition does not have a corresponding Oracle error code and message.

SQL error – constraint check
This condition occurs when an operation cannot be completed because of a constraint check. Note that there are some situations when 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 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 there is a loss of data because of a size mismatch.

SQL error – character set conversion
This condition occurs when the record contains Unicode data for some of its NChar, NVarChar or LongNVarChar columns, and conversion errors happen when that data is converted to the database character set specified in 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 conditions listed above.

Procedure

To manage rejected data:

1. 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, and then in the navigator, highlight the reject link, which is represented by a line of wide dashes.
5. Click the **Reject** tab.
6. In the **Filter rejected rows based on selected conditions** list, select one or more conditions to use to reject records. If you do not choose any conditions, none of the rows are rejected. In this case, any error that occurs while the records are being written to the target table results in job failure.
7. Use one of the following methods to specify when to stop a job because of too many rejected rows:
   - In the **Abort when** field, select **Percent**. Then in the **Abort when (%)** field, enter the percentage of rejected rows that will cause the job to stop. In the **Start count after (rows)** field, specify the number of input rows to process before calculating the percentage of rejected rows.
   - In the **Abort when** field, select **Rows**. Then in the **Abort after (rows) field**, specify the maximum number of reject rows allowed before the job stops.
8. Optional: In the **Add to reject row** list, select **ERRORCODE** or **ERRORMESSAGE** or select both. Then 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.

Compiling and running a job

When you finish designing a job, compile and run it. Then use the log file to gather information that might help you adjust the job configuration or correct error conditions.

About this task

After you compile a job, you can use either the IBM WebSphere DataStage and QualityStage Designer client or Director client to run the job.

Procedure

To compile and run a job:

1. In the Designer client, open the job that you want to compile.
2. Click **Compile**.
3. If the Compilation Status area displays errors, edit the job to resolve the errors. After resolving the errors, click **Compile** again.
4. When the job compiles successfully, click **Run**, and specify the job run options:
   a. Enter the job parameters, as required.
   b. Click the **Validate** button to check the job configuration without actually reading or writing any data.
   c. Click the **Run** button to read, write, or look up data.
5. View the status of the job:
   a. Open the Director client.
   b. In the Status column, verify that the job was validated and completed successfully. If the job or the validation failed, choose **View** → **Log** to view messages that describe runtime problems.
6. If the job has runtime problems, fix the problems, recompile, validate, and run the job until it completes successfully.

Options for reading and writing data

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

Controlling the isolation level of transactions

You can configure the isolation level of transactions.

About this task

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. 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. All of the operations that the connector performs on the database are part of the current transaction. Note that 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.

Procedure

To configure transactions, set the **Isolation level** property to one of the following:

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

Option	Description
Read only	Read only isolation works the same way that serializable isolation works, except that 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, be aware that 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**.

Configuring row prefetching

If you configure row prefetching, when a SELECT statement runs, the connector fetches the number of rows specified by the **Array size** property plus the number of rows specified in the **Prefetch row count** property.

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 first tries to prefetch the number of rows specified for the **Prefetch row count** property. If the number of rows cannot fit in the memory size specified for the **Prefetch buffer size** property, the Oracle client prefetches as many rows as can fit into the buffer.

The Oracle client immediately provides the rows that are fetched towards **Array size** to the connector. The Oracle client caches the rows fetched towards **Prefetch row count** so that as the connector continues requesting data for the currently running SELECT statement, the fetch requests are optimized.

By default, the connector is configured to prefetch one row.

Procedure

To configure Oracle row prefetching, set one or both of the following properties:
- Set **Prefetch row count** to the number of rows to prefetch for each fetch request that results in a roundtrip to the Oracle server.
- Set **Prefetch buffer size** to the size in KB to use as the buffer for the prefetched rows.

Setting the array size

To control the number of records to fetch from a database or write to a database at one time, increase or decrease the array size.

About this task

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 you configure row prefetching, when a SELECT statement runs, the connector fetches the number of rows specified by the **Array size** property plus the number of rows specified by the **Prefetch row count** property.

Procedure

To set the array size:

1. Set **Array size** to a number between 1 and 999,999,999. The default is 2,000.
2. Set **Record count** to the number of records to process before the connector commits the current transaction. The default is 2,000. The value that you specify must be a number between 0 and 999,999,999 and be a multiple of the value that you specify for the **Array size** property. Enter 0, and the connector processes all records before it commits the transaction.
3. Optional: Use the **Mark end of wave** property to specify whether or not to insert an end-of-wave marker after the number of records that are specified in the **Record count** property are processed. By default, end-of-wave markers are not inserted.

   Note: 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.

Performing actions on the table before writing

As an option, you can configure the connector to perform create, replace, and truncate actions on a table at runtime. These actions are performed before any data is written to the table.

About this task

The **Table action** property controls the actions that the connector performs at runtime before data is written to a table. By default, the property is set to **Append**, and no action is performed on the table.

Procedure

To perform actions on a table before writing data to it:

1. To create a table at runtime, perform these steps:
   a. Set **Table action** to **Create**.
   b. 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.
      - Enter the CREATE TABLE statement in the **Create table statement** property.
2. To replace a table at runtime, perform these steps:
   a. Set **Table action** to **Replace**.
   b. 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.
      - Enter the DROP TABLE statement in the **Drop table statement** property.
   c. 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.

    - Enter the CREATE TABLE statement in the **Create table statement** property.
3. To truncate a table at runtime, perform these steps:
   a. Set **Table action** to **Truncate**.
   b. 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.
      - Enter the TRUNCATE TABLE statement in the **Truncate table statement** property.
4. To cause the job to fail when a statement fails, set **Fail on error for [create, truncate, drop] statement** to **Yes**. Then when the statement fails, the job stops. Otherwise, when the statement fails, the connector logs a Warning message and the job continues.

Running an SQL statement before or after processing data

Configure the connector to run an SQL statement once 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.

About this task

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 running 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. For example, to set the data format to use for the client session on a node, you specify the ALTER SESSION statement in **Before SQL (node)** property.

After running the statement that is specified in the **Before SQL (node) statement** property or **After SQL (node) statement** property, the connector explicitly commits the current transaction.

Procedure

You use the same basic procedure to configure the **Before SQL statement**, **After SQL statement**, **Before SQL (node) statement**, and **After SQL (node) statement** properties. The following steps describe how to configure the **Before SQL statement** property.

To run an SQL statement before data processing, complete these steps:

1. Set **Run before and after SQL statements** to **Yes**.
2. In the **Before SQL statement** property, enter the SQL or PL/SQL statement, or enter the fully-qualified path to the file that contains the SQL or PL/SQL statement.

   Note: 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 IBM WebSphere DataStage Server is installed.
3. If you specify a file name, set **Read Before SQL statement from file** to **Yes**.
4. Set **Fail on error for before SQL statement** to **Yes** (default) or **No**. If this property is set to **Yes** and the SQL or PL/SQL statement fails, the connector logs a Fatal message, and the job stops. Otherwise, the connector logs a Warning message, and the job continues.

Substituting node placeholders with node numbers

To configure the connector to reference node numbers in user-defined SQL and PL/SQL blocks, define the environment variables CC_ORA_NODE_USE_PLACEHOLDER and CC_ORA_NODE_PLACEHOLDER_NAME.

About this task

After you define these two environment variables and make them job parameters, each time that the connector detects a placeholder in a user-defined SQL statement or PL/SQL block, the connector substitutes the current node number for the placeholder. The connector assigns 0 to the first node, and assigns 1 to the second processing node, and so on.

After you set up node number substitution, you can configure the connector to run a different statement on each node. Note that substitution applies only to SQL statements that you specify for these properties:
- **Select statement**
- **Insert statement**
- **Update statement**
- **Delete statement**
- **PL/SQL block**
- **Before SQL (node) statement**
- **After SQL (node) statement**

Procedure

To configure the connector replace node placeholders with the actual node numbers at runtime:
1. Set the CC_ORA_NODE_USE_PLACEHOLDER environment variable to TRUE.
2. Set the CC_ORA_NODE_PLACEHOLDER_NAME environment variable to the node placeholder name that you will use in user-defined SQL statements or PL/SQL blocks.
3. Include the CC_ORA_NODE_PLACEHOLDER_NAME and CC_ORA_NODE_USE_PLACEHOLDER environment variables as parameters of the job.

In this example, two nodes insert data into two different tables. The following are the assumptions for this example:

- The connector is configured to write data to a database table.
- The **Write mode** property is set to **PL/SQL**.
- The connector is configured to run on two nodes.
- The **Partition type** property is set to **Entire** so that all input records are sent to all of the nodes and no partitioning takes place.
- The CC_ORA_NODE_USE_PLACEHOLDER environment variable is set to TRUE, and the CC_ORA_NODE_PLACEHOLDER_NAME environment variable is set to DSNODENUM so that the connector substitutes the current node number for each occurrence of DSNODENUM.

The **PL/SQL** property contains this value:

```
BEGIN
 IF DSNODENUM = 0 THEN
  INSERT INTO TABLE1 VALUES (:COL1, :COL2);
 ELSE
  IF DSNODENUM = 1 THEN
   INSERT INTO TABLE2 VALUES (:COL1, :COL2);
 END IF;
END;
```

On the first node, which is node 0, this PL/SQL block runs:

```
BEGIN
 IF 0 = 0 THEN
  INSERT INTO TABLE1 VALUES (:COL1, :COL2);
 ELSE
  IF 0 = 1 THEN
   INSERT INTO TABLE2 VALUES (:COL1, :COL2);
 END IF;
END;
```

This PL/SQL block yields the same results as this statement:

```
INSERT INTO TABLE1 VALUES (:COL1, :COL2);
```

On the processing node, which is node 1, this PL/SQL block runs:

```
BEGIN
 IF 1 = 0 THEN
  INSERT INTO TABLE1 VALUES (:COL1, :COL2);
 ELSE
  IF 1 = 1 THEN
   INSERT INTO TABLE2 VALUES (:COL1, :COL2);
 END IF;
END;
```

This PL/SQL block yields the same results as this statement:

```
INSERT INTO TABLE2 VALUES (:COL1, :COL2);
```

Options for bulk loading data

Set options for managing constraints, triggers, and indexes; and configure the use of the date cache and the redo log.

Managing table constraints and triggers for a bulk load

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.

Prerequisites

Perform these prerequisite tasks:
- Configure the Oracle connector to perform a bulk load.
- To use an exceptions table that already exists in the database, obtain the name of the table.

When you configure the connector to disable constraints before loading the data, the connector disables the constraints and logs a message about this action. If disabling some of the constraints fails, the connector logs an error message, and the job stops.

When you configure the connector to enable constraints after loading the data, the connector enables the constraints and logs a message about this action. The connector stores the ROWID values for any rows that violate the constraints in the exceptions table that you specify.

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 under the directory `%ORACLE_HOME%\RDBMS\ADMIN`. 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 specified in one of the two scripts that correspond to the type of the target table; otherwise, the connector reports a fatal error about the table format, and the job stops. If the database does not already have an exceptions table, the connector uses the correct format to create one.

When you configure the connector to disable triggers before loading the data, the connector disables the triggers and logs a message about this action. If disabling some of the triggers fails, the connector logs an error message, and the job fails. The connector uses a similar process to enable triggers after loading the data.

Procedure

To disable and enable constraints and triggers:
1. To disable and enable constraints, complete these steps:
   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. Enter the name of the exceptions table in the **Exceptions table name** property. 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 some rows fail a constraint check, the job stops.

2. To disable and enable triggers, complete these steps:
   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**.

Configuring the Oracle date cache for a bulk load

When bulk loading data into tables that contain DATE or TIMESTAMP columns, configuring the Oracle date cache might improve load performance.

About this task

If the date cache is enabled, the connector logs date cache usage statistics when the bulk load completes. You can use the statistics to determine whether changing the date cache size might improve the future load performance of similar input data sets. The connector logs these statistics:

- Size of cache.
- Number of elements that are stored in the cache.
- Total number of hits, which occur when matches are found in the cache.
- Total number of misses, which occur when matches are not found in the cache. Note that the connector does not count the misses until the cache is full.
- Information about whether the cache was disabled during the bulk load.

Procedure

To configure the date cache:

1. Set **Use Oracle date cache** to **Yes**.
2. In the **Cache size** property, enter the maximum number of entries that the cache stores. The default is 1,000.
3. Set **Disable cache when full** to **Yes**. When the number of entries in the cache reaches the number specified in the **Cache size** property and the next lookup in the cache results in a miss, the cache is disabled.

Managing indexes

Specify how to control table indexes during a bulk load and how to rebuild indexes after a bulk load completes.

About this task

You can configure the connector to maintain indexes during the bulk load or to rebuild them after the bulk load. When you configure the connector to rebuild indexes, if you set the **Fail on error for index rebuilding** property to **Yes** and some of the indexes cannot be rebuilt, the connector logs an error message, and the job stops. Otherwise, the connector logs a Warning message, and the job continues.

When rebuilding indexes, you can specify settings for parallelism and for the generation of the Oracle redo log and the Oracle invalidation redo log. For more information about these concepts, see the Oracle documentation.

Procedure

To manage indexes:

1. To control how to handle table indexes during a bulk load, set the **Index maintenance option** property to one of the following:

Option	Description
Do not skip unusable	While loading rows into the table, the connector tries to maintain indexes. If an index on the table is in an unusable state, the bulk load fails.
Skip unusable	The connector skips indexes that are in an unusable state and maintains indexes that are in a usable state. **Note:** When performing a bulk load 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.

2. To rebuild indexes after a bulk load, complete these steps:
   a. Set **Perform operations after bulk** load to **Yes**.
   b. Set **Rebuild indexes** to **Yes**.
   c. Optional: To include a parallel clause in the ALTER INDEX statement when the index is rebuilt, select one of the following for the **Parallel clause** property:
      - Select **Do not include** to include no parallel clause and to use the existing setting for the index.
      - Select **NOPARALLEL** to disable parallelism. In this case, access to the index segment is serialized.
      - Select **PARALLEL** to enable parallelism for rebuilding the index and for all subsequent queries and DML statements that are performed on the index segment. As an option, for the **Degree of parallelism** property, enter a number that represents the degree of parallelism to use in the parallel clause. Leave the property blank, and the Oracle client automatically calculates the optimal parallelism degree.
   d. Optional: To include a logging clause in the ALTER INDEX statement when the index is rebuilt, select one of the following for the **Logging clause** property:
      - Select **Do not include** to include no logging clause and to use the existing setting for the index.
      - Select **NOLOGGING** to disable logging to the redo log.
      - Select **LOGGING** to enable logging to the redo log.
   e. Optional: To stop rebuilding an index if the index rebuild statement fails, set **Fail on error for index rebuilding** to **Yes**. If an index rebuild fails, the connector logs a Fatal message.

Controlling bulk record loading

To control when the connector bulk loads buffered records into a target table, set an array size and a buffer size.

About this task

The connector always tries to load data in chunks, where each chunk contains the number of rows that are specified in the **Array size** property. The **Buffer size** property controls the maximum size of the buffer that holds each chunk of records. By default, the buffer size is 1,024 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.

When an upstream stage provides records to the Oracle connector in the form of waves, each wave includes an end-of-wave marker, which is a special record that signifies the end of the wave. In this case, the array size applies to each separate wave of records. If there are not enough records 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 applies to that single wave.

Procedure

To control bulk record loading:

1. Set **Array size** to a value between 1 and 999,999,999. The default is 2,000.
2. Set **Buffer size** to a value between 4 and 100,240, which represents the buffer size in KB. The default is 1,024 KB.

Case-sensitivity

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 set to **Yes**.

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 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**.

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 the 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 use enclosed the table and column names in 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; that the input link contains columns Col1, Col2, and Col3, all of which are of VarChar(10) data type; and 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 **No**, 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));
```

White space characters, NULL values, and empty string values

To understand the results that you get when you use the connector to read or write data, review how the connector and the Oracle database treat these characters and values.

When reading data from a database or writing data to a database, the Oracle connector always preserves white space characters such as SPACE, TAB, CR (carriage return), and LF (line feed). In addition, the connector treats text values as-is and does not trim leading or trailing white space characters.

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 columns COL1 and COL2 and the corresponding values that will be stored in TABLE1. In the table, the dash (-) represents a space character.

Table 18. 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

Using the Oracle connector in a Distributed Transaction stage job

When you define the connection between the Oracle connector and the Oracle database, you specify the Oracle resource manager to write to.

To use the Oracle connector in a Distributed Transaction stage (DTS) job, you perform these tasks:
1. Configure IBM WebSphere MQ queue manager as the transaction manager.
2. Configure the Oracle database as the resource manager.
3. Define the connection between the Oracle connector and the Oracle database.

When you add resource manager configuration information to the WebSphere MQ queue manager, you generally define the DB field and the SqlNet field in each XAOpenString entry. The DB field is required only if a job writes to more than one Oracle resource manager. In that case, the field is used to differentiate among multiple Oracle resource managers.

Record the value for the SqlNet parameter. If you define more than one Oracle resource manager with the MQ queue manager, record value for the DB parameter. You must provide these values when you configure the Oracle connector to run a DTS job.

When you define the connection between the Oracle connector and the Oracle database, you must complete these fields:

XA database name
Enter the value from the DB field of the XAOpenString entry. This field is required only if you register more than one Oracle resource manager with the MQ queue manager that the DTS stage references.

Server Enter the value of the SqlNet field of the XAOpenString entry.

Error diagnosis and recovery

Log Oracle environment messages, enable support for Transparent Application Failover (TAF), set properties that control job failure, take corrective actions to fix error conditions that are reported to the log file, and configure reject links.

Configuring the Oracle connector for transparent application failover

Configure the Oracle connector to receive messages that describe when failover starts and how failover progresses.

When a database connection is enabled for transparent application failover (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, while the reconnection is taking place, it might seem that the connector unexpectedly stops running and hangs. 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** property 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.

There are two types of TAF: SESSION and SELECT. In order for the connector to continue fetching data for the SELECT statement that is interrupted when failover occurs, enable the SELECT failover type.

These are the steps that the connector takes when TAF starts:

1. The connector logs a Warning message that indicates that TAF has been initiated. This message includes the type of TAF, SESSION or SELECT, that is taking place.
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 is successful, the connector logs a Warning message to indicate the successful completion of TAF.
4. If the **Before SQL statement** property is set to **Yes**, the connector reruns the statement that is specified in **Before SQL statement** property. If the **Replay Before SQL (node) statement** property is set to **Yes**, the connector reruns the statement specified in **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 there is no valid connection to the database.

Examples: Transparent application failover

These examples illustrate how the connector performs when the **Manage application failover** property is set to **No** and when it is set to **Yes**.

Multiple database connections are configured, and Manage application failover is set to No

This is the configuration for this example:

- The connector is configure 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 database instance *orcl1*.
- 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. The alternative value for the METHOD option is BASIC. When BASIC is specified, the connection to the backup instance happens when the failover actually 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. If the failover TYPE was set to SESSION, the next fetch request that the connector issued would fail. Then the connector would log a Fatal message, and the job would stop.

If 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 would fail. The connector would log an error message, and the job would stop.

A single database connection is configured, and Manage application failover is set to Yes

In this example, there is only one database instance, and failover occurs only after the Oracle administrator restarts the instance. This is the configuration for this example:

- 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 database instance *orcl*.
- 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.

If **Manage application failover** is set to **No**, the Oracle client tries up to 20 times, the value of the RETRIES option, to reestablish the connection and waits 5 seconds, the value of the DELAY option, between failover attempts. During that time, the connector appears to stop and might seem to fail, when, in fact, the delay occurs because the Oracle client is attempting to failover.

Oracle environment logging

The Oracle connector can log Debug messages that contain information about the current Oracle environment settings. These messages are useful for performing problem diagnostics.

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 may 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 any 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.

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 mid-process is useful when you want to receive prompt notification that something you expected to work has instead failed. By design, a job stops when a Fatal message is reported. The following list contains the properties that control job failure:

- **Fail on error for create table statement**
- **Fail on error for drop table statement**
- **Fail on error for truncate table statement**
- **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 an error occurs, the message is reported to the log file, and the job continues. If a property is set to **No**, when an error occurs, the corresponding message is reported to the log file, and the job stops.

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.

Messages

Identify an error or problem and resolve the problem by using the appropriate recovery action.

Messages have the following severity levels: Fatal, Error, Warning, Informational, Debug, and Trace. The environment variable CC_MSG_LEVEL controls which messages are reported to the log file. By default, Informational level messages and higher are reported.

To change the level of messages that are reported to the log, edit the CC_MSG_LEVEL environment variable, and set it to one of the following values:

- 1 - Trace
- 2 - Debug
- 3 - Informational
- 4 - Warning
- 5 - Error
- 6 - Fatal

For example, when you perform problem diagnostics, you might want to set CC_MSG_LEVEL to 2 so that you can view Debug messages, as well as higher level messages.

You can set properties that control when to stop a job. For example, if the **Process warning messages as fatal errors** property is set to **Yes**, a job stops when the connector reports the first Warning message. The following topics list the messages by severity. Each message is documented along with corrective actions that might fix the error condition.

Fatal messages

Read the text of each fatal message, along with a description of the cause of the error and recommendations for corrective actions to take.

When a fatal message occurs, the job stops.

IIS-CONN-ORA-001001 The variable {0} has value {1} which is not valid in the current context.

Explanation: Fatal error. This is a generic error message that the connector reports when it cannot choose a more specific message.

User response: Use the specified variable and value to try to determine the source of the problem. For example, the values may be related to a property name and a value that was configured for the connector.

IIS-CONN-ORA-001002 The OCI function {0} returned status {1}: OCI_INVALID_HANDLE.

Explanation: Fatal. This is an internal error that occurs in the communication between the connector and the Oracle client. The problem might be related to an external problem.

User response: Check the log for additional Informational, Warning, and Fatal messages that might describe the situation that resulted in this Fatal error.

IIS-CONN-ORA-001003 The OCI function {0} returned status {1}, Error code {2}, Error message: {3}.

Explanation: Fatal. This is a generic message that indicates that the Oracle client returned an error after the connector called the specified function.

User response: Evaluate the reported error status, error code, and error message. Based on that information, try to deduce the reason for the error.

IIS-CONN-ORA-001004 The connector could not establish connection to Oracle server {0}. Method: {1}, Error code: {2}, Error message {3}.

Explanation: Fatal

User response: Ensure that the **Server**, **Username**, and **Password** properties are correctly specified, and verify that the Oracle service and the listener are running. If you are using OS authentication, ensure that OS authentication is correctly configured in the Oracle database.

IIS-CONN-ORA-001010 Unsupported data type: {0}.

Explanation: Fatal. This is an internal error that indicates that the connector encountered an Oracle data type that is not supported for the current context. The error might be related to an external problem.

IIS-CONN-ORA-001011 Unsupported type code: {0}.

Explanation: Fatal. This is an internal error that indicates that the connector encountered an Oracle data type that is not supported for the current context. The error might be related to an external problem.

IIS-CONN-ORA-001012 Memory allocation failed for {0} bytes.

Explanation: Fatal. The system ran out of available memory, and the connector could not allocate free memory for the operation that it was performing.

User response: Close some applications to free up the memory.

IIS-CONN-ORA-001013 The connector could not initialize XA environment by calling Oracle function {0}.

Explanation: Fatal. There was a problem initializing the distributed transaction environment at runtime.

User response: Ensure that the system is correctly configured for use with the Distributed Transaction stage (DTS) and the transaction manager.

IIS-CONN-ORA-001014 The statement failed with status {0}: {1} for input row {2}.

Explanation: Fatal.

User response: Look at the reported status and error message and the input data to try to determine what caused the error.

IIS-CONN-ORA-001015 The connector could not create table {0} because the data type information for the column {1} could not be obtained.

Explanation: Fatal. This is an internal error that might be related to an external problem.

IIS-CONN-ORA-001016 The array size must be set to 1 so that the connector can read LOB values.

Explanation: Fatal.

User response: Set the **Array size** property to 1.

IIS-CONN-ORA-001017 The connector could not determine the ROWID value for LOB column {0} in table {1}.

Explanation: Fatal. The connector could not obtain the row identifier for the row that contains the LOB value that is passed by reference (locator).

User response: Ensure that the column and the table that the error message references are available and accessible to the current user.

IIS-CONN-ORA-001018 The connector could not obtain the table name for the LOB column {0}. The LOB reference was not created.

Explanation: Fatal. The connector could not obtain the row identifier for the row that contains the LOB value that is passed by reference (locator).

User response: Ensure that the table that contains the LOB column is available and accessible to the current user.

IIS-CONN-ORA-001019 The connector could not find the tnsnames.ora file. Verify that ORACLE_HOME or TNS_ADMIN environment variables are set. Alternatively, use Oracle Easy Connect naming method or specify a full connect descriptor.

Explanation: Fatal.

IIS-CONN-ORA-001020 The connector could not open Oracle network configuration file {0}.

Explanation: Fatal. The connector failed to open the `tnsnames.ora` configuration file. The file might be in the correct location but might not have read-access granted, or a system-level error prevented the connector from opening the file.

User response: Verify that the file exists and that its contents can be viewed. To work around this problem, specify a full connect descriptor or an Easy Connect string in the **Server** property.

IIS-CONN-ORA-001021 The connector could not read Oracle network configuration file {0}.

Explanation: Fatal. The connector failed to read the contents of the `tnsnames.ora` configuration file.

User response: Verify that the file exists and that it is not empty. To override the `tnsnames.ora` configuration file, specify a full connect descriptor or an Easy Connect string for the **Server** property.

IIS-CONN-ORA-001022 The following SQL statement failed: {0}.

Explanation: Fatal.

User response: Verify that the syntax of the statement is correct. Look at the log file for additional warning and error messages that might contain information that identifies why the statement failed.

IIS-CONN-ORA-001023 The connector could not find a column in the input schema to match parameter {0}.

Explanation: Fatal.

User response: Ensure that the statement parameters match the column names on the input link. Look at the number and names of the parameters and columns, and ensure that an unambiguous mapping exists between each pair.

IIS-CONN-ORA-001024 While reading data for column {1}, the connector received Oracle error code ORA-{0}.

Explanation: Fatal.

User response: See the Oracle documentation for more information about the Oracle error. Compare the input column definition with the database column definition. Evaluate whether the Oracle error is a result of a mismatch between the two column definitions.

IIS-CONN-ORA-001025 The connector could not automatically generate the UPDATE statement. Specify at least one non-key column in the input schema.

Explanation: Fatal.

IIS-CONN-ORA-001026 The connector could not automatically generate the WHERE clause for the {0} statement. Specify at least one key column in the input schema.

Explanation: Fatal. For the connector to generate the UPDATE or DELETE statement automatically at runtime, the input link must have at least one key column.

IIS-CONN-ORA-001027 The connector is constrained to run on {0} processing nodes, but the Oracle partitioning scheme that was specified for table {1} requires that the total number of processing nodes be {2}.

Explanation: Fatal.

User response: Modify the constraint rules so that the total number of nodes matches the number of nodes that the connector requires, or remove the constraints so that the connector can dynamically specify the number of nodes that it needs.

IIS-CONN-ORA-001028 The connector could not find the specified column {0} in table {1}.

Explanation: Fatal.

User response: Ensure that the specified column name and table name are correct and that the column exists in the table. Also ensure that the current user owns the table or that the owner name is included with the table name.

IIS-CONN-ORA-001029 The data type of column {0} in table {1} is {2}, and the scale is {3}. Specify the data type NUMBER with the scale of 0.

Explanation: Fatal. When the connector reads data in parallel by using the Modulus method or the Minimum and maximum range method, the specified column must be a NUMBER column with the scale set to 0 or not specified.

IIS-CONN-ORA-001030 The connector could not match name {0} with any partition or subpartition name in table {1}.

Explanation: Fatal.

User response: Ensure that the specified values are correct and that the specified name matches a partition or subpartition name in the specified table.

IIS-CONN-ORA-001031 The connector could not find the specified table or view {0}.

Explanation: Fatal.

User response: Ensure that the specified table or view name matches an existing table or view name. Note that if the schema name is not specified with the table name or view name, the connector assumes that the table or view is owned by the currently connected user.

IIS-CONN-ORA-001032 The connector could not match the partition key column {0} with any column in the input schema.

Explanation: Fatal. The connector could not determine which column on the input link to use for comparing values against the partition key boundary value.

User response: Verify the number and names of the columns on the input link. Ensure that one of the columns matches the partition key column.

IIS-CONN-ORA-001033 The input schema column {0} is not compatible with the partition key column {1} of type {2}.

Explanation: Fatal. The data type of the selected column is not compatible with the data type of the partition key column.

User response: Ensure that the data types of the two columns are compatible.

IIS-CONN-ORA-001034 The connector could not validate the input schema. Specify at least one column in the input schema.

Explanation: Fatal.

User response: Define at least one column on the link of the stage.

IIS-CONN-ORA-001035 The property {0} requires a value, but no value was specified.

Explanation: Fatal.

IIS-CONN-ORA-001036 The index {0} is out of boundary for property {1}.

Explanation: Fatal. This is an internal error that might be related to an external problem. The message does not refer to the table index, but instead refers to the internal connector property index.

IIS-CONN-ORA-001038 The connector could not find any tables to include in the SELECT statement for the view data operation.

Explanation: Fatal.

User response: Look at the specified SQL statement in the connector properties and ensure that it correctly specifies table names. If the **Generate SQL at runtime** property is set to **Yes**, ensure that the **Table name** property specifies a table name.

IIS-CONN-ORA-001039 While parsing parameter {0}, the connector detected an unmatched double quote character at position {1}.

Explanation: Fatal.

User response: Ensure that the specified SQL statement has a valid value and that all double quote characters in the statement are properly matched.

IIS-CONN-ORA-001040 While parsing parameter {0}, the connector detected an unmatched single quote character at position {1}.

Explanation: Fatal.

User response: Ensure that the specified SQL statement has a valid value and that all single quote characters are properly matched.

IIS-CONN-ORA-001041 While parsing parameter {0}, the connector detected an unexpected character at position {1}.

Explanation: Fatal.

User response: Ensure that the specified SQL statement has a valid value.

IIS-CONN-ORA-001042 While parsing parameter {0}, the connector expected an identifier at position {1}.

Explanation: Fatal

User response: Ensure that the specified SQL statement has a valid value.

IIS-CONN-ORA-001043 While parsing table name {0}, the connector detected an unmatched double quote character at position {1}.

Explanation: Fatal.

User response: Ensure that the specified SQL statement has a valid value.

IIS-CONN-ORA-001044 While parsing table name {0}, the connector detected an unmatched single quote character at position {1}.

Explanation: Fatal.

User response: Ensure that the specified SQL statement contains a valid value.

IIS-CONN-ORA-001045 While parsing table name {0}, the connector detected an unexpected character at position {1}.

Explanation: Fatal.

User response: Ensure that the specified SQL statement contains a valid value.

IIS-CONN-ORA-001046 While parsing table name {0}, the connector expected an identifier at position {1}.

Explanation: Fatal.

User response: Ensure that the specified SQL statement contains a valid value.

IIS-CONN-ORA-001047 The connector could not find the specified file {0}; or the current user does not have read permission on the file; or the file is empty.

Explanation: Fatal.

User response: Ensure that the file location is specified correctly and that the specified file exists, has read permissions granted, and is not empty.

IIS-CONN-ORA-001049 The connector encountered parameter {0} in DataStage format and parameter {1} in Oracle format. A single format must be used consistently for all parameters.

Explanation: Fatal

User response: Use a single syntax consistently when specifying statement parameters. DataStage syntax is ORCHESTRATE:*parameter_name*. Oracle syntax is : *name*, where *name* is the parameter name or parameter number.

IIS-CONN-ORA-001050 A warning message was issued, and the connector is configured to stop when this type of message occurs.

Explanation: Fatal.

User response: Resolve the reported issue so that the connector no longer reports it as a Warning message, or change the connector properties to allow it to proceed when it reports a Warning message.

IIS-CONN-ORA-001051 An unsupported data type {0} was encountered during a bulk load.

Explanation: Fatal.

User response: Change the type of the column in the target table to one of the supported types, load data from a different table, or change the value of the **Write mode** property from **Bulk load** to **Insert statement**.

IIS-CONN-ORA-001052 While loading data, the connector received Oracle error code {0}.

Explanation: Fatal.

User response: Consult the Oracle documentation for information about the error code. Verify that the definitions for the input columns match the definitions for the columns in the target table.

IIS-CONN-ORA-001053 An index rebuild operation failed, and the connector is configured to stop when a rebuild fails.

Explanation: Fatal.

User response: Resolve the problem that caused the index rebuild to fail, or change the properties that configure how to proceed when an index rebuild fails.

IIS-CONN-ORA-001054 System call {0} failed with OS error {1} ({2}).

Explanation: Fatal. This is a generic error that the connector receives after it invokes an operating system function.

IIS-CONN-ORA-001055 The specified statement: {0} is of incorrect type. The required statement type is: {1}.

Explanation: Fatal.

User response: Ensure that the statement type is appropriate for the property for which it was specified.

IIS-CONN-ORA-001056 The schema column {0} must have the length specified because it is used to access database column {1} of data type {2}.

Explanation: Fatal.

User response: Enter a length value for the specified column. Choose a length that is appropriate for the data that will be transferred through the column.

IIS-CONN-ORA-001057 The connector encountered a parameter that uses DataStage syntax (ORCHESTRATE.parameter_name), which is not allowed in PL/SQL blocks. Use Oracle syntax (:name, where name is the parameter name or the parameter number) to specify the parameter.

Explanation: Fatal.

User response: Use only Oracle syntax for specifying bind parameters in the PL/SQL block.

Warning messages

Read the text of each Warning message along with a description of the cause of the error and recommendations for corrective actions to take.

If you set the **Process warning messages as fatal errors** property to **Yes**, a job stops when the connector reports the first Warning message.

IIS-CONN-ORA-003001 While dropping table {0}, the connector encountered an error.

Explanation: Warning. The connector receives this error when it tries to run a DROP TABLE statement. This error typically indicates that the table does not exist.

User response: Look in the log file for additional messages that indicate the actual reason for this failure at the SQL level.

IIS-CONN-ORA-003002 While creating table {0}, the connector encountered an error.

Explanation: Warning. The connector receives this message when it tries to run the CREATE TABLE statement. For example, this message is returned when the database already contains a table that has the specified name and that is owned by the specified table owner.

User response: Look at the log file for additional messages that indicate the actual reason for this failure at the SQL level.

IIS-CONN-ORA-003003 While truncating table {0}, the connector encountered an error.

Explanation: Warning. The connector received an error when it tried to delete rows from the table.

User response: Look in the log file for additional messages that indicate the actual reason for the failure at the SQL level.

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.

Explanation: Warning. The connector can send records down the reject link only on a processing node and not on the conductor node. The constraint checking in bulk load mode must be performed only once. To ensure that the constraint checking is done only once for the stage and is done on the processing node, set the **Execution mode** property to **Sequential**. Then there is exactly one node.

User response: To eliminate the warning, perform one of the following tasks:

- Change the **Execution mode** to **Sequential**.
- Change the **Write mode** to **Insert**.
- Deselect the **SQL error-constraint check** condition for the reject link.

IIS-CONN-ORA-003005 A data conversion error was encountered in bulk load mode for row {0}, column {1}.

Explanation: Warning.

User response: Look at the input value for the specified row and column. Ensure that the value is valid for the corresponding output table column.

IIS-CONN-ORA-003006 A data load error was encountered in bulk load mode for row {0}.

Explanation: Warning.

User response: Look at the input data for the specified row. Ensure that the column values are valid for the corresponding output table definition.

IIS-CONN-ORA-003007 The connector could not enable constraint {0} on table {1} because some row in the table violated the constraint. The ROWID values of those rows are stored in exception table {2}.

Explanation: Warning. Some rows in the loaded data violate the constraints that the connector disabled during the load and tried to re-enable after the load.

User response: If the connector is configured to process rejected records, look at the rejected records to determine the cause of the problem. If the connector is not configured to process reject records, look at the target table rows that have ROWID values that match the ROWID values that are stored in the specified exceptions table to determine how the rows violated the specified constraint.

IIS-CONN-ORA-003008 The connector could not rebuild index {0} on table {1}. Error code: {2}, Error message: {3}.

Explanation: Warning. The index rebuild operation failed to complete. An Oracle error code and error message were returned.

User response: Look at the returned Oracle error code and error message to determine why the operation failed.

IIS-CONN-ORA-003009 The connector was configured to use the Oracle partitions method to perform partitioned reads on {0} processing nodes, but table {1} is not partitioned. The connector will run in sequential mode.

Explanation: Warning.

User response: Change the **Execution mode** property from **Parallel** to **Sequential**, or change the **Partitioned reads method** property from **Oracle partitions** to another value.

IIS-CONN-ORA-003010 The connector was configured to perform partitioned writes on {0} processing nodes, but the table {1} is not partitioned. The connector will run in sequential mode.

Explanation: Warning.

User response: Change the **Execution mode** property from **Parallel** to **Sequential**, or change the **Partition type** property from **Oracle connector** to another value.

IIS-CONN-ORA-003011 The connector was configured to perform partitioned writes on table {0}, but this table uses partitioning scheme {1}, for which the connector does not support partitioned writes. The connector will run in sequential mode.

Explanation: Warning.

User response: Change the setting for the **Execution mode** property from **Parallel** to **Sequential**, or change the setting for the **Partition type** property from **Oracle connector** to another value.

IIS-CONN-ORA-003012 The connector was configured to perform partitioned reads on table {0} using the Oracle partitions method, but a single partition or subpartition {1} was specified. The connector will run in sequential mode.

Explanation: Warning.

User response: Change the setting for the **Execution mode** property from **Parallel** to **Sequential**, or change the setting for the **Partitioned reads method** property from **Oracle partitions** to another value.

IIS-CONN-ORA-003013 The connector was configured to perform partitioned writes on table {0}, but a single partition or subpartition {1} was specified. The connector will run in sequential mode.

Explanation: Warning.

User response: Change the setting for the **Execution mode** property from **Parallel** to **Sequential**, or change the setting for the **Partition type** property from **Oracle connector** to another value.

IIS-CONN-ORA-003014 The connector was configured to perform partitioned reads using the Oracle partitions method, but the specified SELECT statement already contains PARTITION or SUBPARTITION clauses. The connector will run in sequential mode.

Explanation: Warning.

User response: Change the setting for the **Execution mode** property from **Parallel** to **Sequential**, or change the setting for the **Partitioned reads method** property from **Oracle partitions** to another value.

IIS-CONN-ORA-003015 The connector could not obtain access to the {0} system view. Access to that system view is required for the Rowid range read method. The connector will use the Rowid hash read method instead.

Explanation: Warning.

User response: Ensure that the current user has read access to the specified Oracle static dictionary view, or change the setting for the **Partitioned read method** property from **Rowid range** to another value.

IIS-CONN-ORA-003016 Transparent application failover is not enabled for the current service.

Explanation: Warning.

User response: Enable transparent application failover for the current service, or set the **Manage application failover property** to **No**.

IIS-CONN-ORA-003017 Transparent application failover was initiated. The type of failover is {0}.

Explanation: Warning. The connection to the currently connected database instance failed, and the Oracle client initiated transparent application failover (TAF) for the connector.

User response: Try these three solutions:

- Wait for the failover to finish so that the job can continue running. Note that in some cases, even after TAF finishes, the job might still fail.
- Investigate why the database instance failed and correct the problem. Then run the job again.
- Wait until the instance is back up. Then run the job again.

IIS-CONN-ORA-003018 The connector will wait {0} seconds for transparent application failover to complete; attempt {1} of {2}.

Explanation: Warning.

IIS-CONN-ORA-003019 Transparent application failover completed. The connector will attempt to resume data processing.

Explanation: Warning. Note that even after transparent application failover (TAF) competes, the job might still fail. Job failure occurs when the operation that was interrupted when TAF started cannot be configured on the newly established client session.

IIS-CONN-ORA-003020 Transparent application failover failed.

Explanation: Warning. This message indicates that transparent application failover did not complete within the specified number of attempts. In most cases, the job fails because the connection to the database is invalid.

IIS-CONN-ORA-003021 Transparent application failover did not complete within the specified time and number of attempts.

Explanation: Warning. The Oracle client determined that it cannot complete the transparent application failover for the connector. In most cases, the job subsequently fails because the connection to the database is invalid.

IIS-CONN-ORA-003022 The connector was configured to perform partitioned writes, but the connector failed determine the name of the table to use as input. The connector will run in sequential mode.

Explanation: Warning.

User response: Enter a value in the **Table name** property or in the **Table name for partitioned writes** property.

IIS-CONN-ORA-003023 The connector was configured to perform partitioned reads, but the connector could not determine the name of the table to use as input. The connector will run in sequential mode.

Explanation: Warning.

User response: Enter a value in the **Table name** main property or enter a value in **Table name** subproperty of the **Enable partitioned writes** property.

IIS-CONN-ORA-003024 While running the {0} statement {1}, the connector encountered an error.

Explanation: Warning.

User response: Check the syntax of the specified statement and look for any errors. Check the log file for other messages that might contain more information about the failure.

IIS-CONN-ORA-003025 Number of records rejected the current on node {0}.

Explanation: This message reports the number of messages that were rejected on the current processing node. The total number of records that were rejected is the sum of all of the rejected records from all of the processing nodes. If the stage is running on a single node, the reported number matches the total number of records that were rejected by the stage.

User response: Inspect the rejected records, which contain the data from the original records. If you included the ERRORCODE and ERRORMESSAGE columns on the reject link, each rejected record includes information about the error that caused the record to be rejected.

Informational, debug, and trace messages

Informational messages report the current status of connector processing. Debug and trace messages provide detailed information that you can use to troubleshoot problems.

Informational messages

Table 19. Informational message numbers and corresponding message text

Message number	Message text
IIS-CONN-ORA-004001	The connector connected to Oracle server {0}.
IIS-CONN-ORA-004002	The connector is configured to use OS authentication.

Table 19. Informational message numbers and corresponding message text (continued)

Message number	Message text
IIS-CONN-ORA-004003	The connector is configured to participate in distributed transaction environment.
IIS-CONN-ORA-004004	The connector will run in sequential mode.
IIS-CONN-ORA-004005	The connector will run in parallel on {0} processing nodes.
IIS-CONN-ORA-004006	The connector generated the following {0} statement at runtime: {1}.
IIS-CONN-ORA-004007	The connector created the table {0}.
IIS-CONN-ORA-004008	The connector dropped the table {0}.
IIS-CONN-ORA-004009	The connector truncated the table {0}.
IIS-CONN-ORA-004010	The connector ran the specified Before SQL statement.
IIS-CONN-ORA-004011	The connector ran the specified After SQL statement.
IIS-CONN-ORA-004012	The connector ran the specified Before SQL (node) statement.
IIS-CONN-ORA-004013	The connector ran the specified After SQL (node) statement.
IIS-CONN-ORA-004014	Number of rows fetched on the current node: {0}.
IIS-CONN-ORA-004015	Number of rows inserted on the current node: {0}.
IIS-CONN-ORA-004016	Number of rows updated on the current node: {0}.
IIS-CONN-ORA-004017	Number of rows deleted on the current node: {0}.
IIS-CONN-ORA-004018	Number of rows processed by the PL/SQL block on the current node: {0}.
IIS-CONN-ORA-004019	Number of records processed by the lookup select statement on the current node: {0}.
IIS-CONN-ORA-004021	The connector was configured to run on {0} processing nodes, but the Oracle partitioning scheme used for table {1} requires a total of {2} processing nodes. The state will run on {3} processing nodes.
IIS-CONN-ORA-004022	The connector was configured to run in parallel mode on {0} nodes, but the partitioned reads were not enabled. The connector will run in sequential mode.
IIS-CONN-ORA-004023	The connector has matched partition key column {0} with input schema field {1}.
IIS-CONN-ORA-004024	Date cache statistics: cache size: {0}, number of elements in the cache: {1}, number of hits: {2}, number of misses: {3}, cache was disabled (1 - Yes, 0 - No): {4}.
IIS-CONN-ORA-004025	The connector disabled constraint {0} on table {1}.

Table 19. Informational message numbers and corresponding message text (continued)

Message number	Message text
IIS-CONN-ORA-004026	The connector disabled all triggers on table {0}.
IIS-CONN-ORA-004027	The connector enabled constraint {0} on table {1}.
IIS-CONN-ORA-004028	The connector deleted the rows in table {0} that violated constraint {1}.
IIS-CONN-ORA-004029	The connector enabled all triggers on table {0}.
IIS-CONN-ORA-004030	The connector will load {0} rows at a time.
IIS-CONN-ORA-004031	The connector rebuilt index {0} on table {1}.
IIS-CONN-ORA-004032	Transparent application failover is enabled for the current service.
IIS-CONN-ORA-004033	Number of rows loaded on the current node: {0}.
IIS-CONN-ORA-004034	The connector will use table {0} as input for the partitioned reads method.
IIS-CONN-ORA-004035	The connector will use table {0} as input for the partitioned writes method.
IIS-CONN-ORA-004036	The connector ran the {0} statement: {1}.

Debug messages

There is 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 performing problem diagnostics on a job.

Trace messages

There are 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 20. Trace message numbers and the corresponding message text

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

Reference

These reference topics provide detailed information about data type mappings, dictionary views, environment variables, and environment logging.

Data type mapping and Oracle data types

These topics provide detailed information about data type mapping and Oracle data types.

When the Oracle connector imports a table definition, the connector converts Oracle data types to 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 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.

How the connector handles Oracle datetime data types depends on whether the design-time schema specifies datetime 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.

Writing data to Oracle datetime 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 timestamps:

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

There are two ways to specify the session parameters. First, you can 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. Second, you can alter the current session by including ALTER SESSION SET *parameter* = *value* statements in the **Before SQL statement (node)** property.

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 displayed in the log file. To view Debug messages in the log file, set the CC_MSG_LEVEL environment variable to 2.

Reading data from Oracle datetime columns

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 Oracle LOB data types BFILE, BLOB, CLOB, NCLOB, LONG RAW, RAW, and XMLType.

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.

Using the inline form, the connector produces the actual values. The inline form is generally effective for transferring small LOB values that are under 100 KB. To configure the connector to use the inline form, you set **Enable LOB references** to **No**.

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. Then the stage 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.

There are advantages and disadvantages to using the reference form. The main advantage is that the reference form can transfer large LOB values from the source stage to the target stage. The main disadvantage is that 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. The reference form is generally effective for transferring large LOB values that are 1 MB or more.

Be aware of these 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 runtime.
- When you configure the Oracle connector to read data from a BFILE column, you can transfer the actual file contents, or you can transfer a reference to the file location. If you are transferring the file contents of a BFILE, set the **Transfer**

BFILE contents property to **Yes**. By default, **Transfer BFILE contents** is set to **No** and will transfer 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, and 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 passing 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, 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.

To configure the Oracle connector to use the reference form, you set **Enable LOB references** to **Yes** and 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.

Examples: Transferring XMLType data:

These examples illustrate reading XMLType data from a standard table, object table, and object view.

Writing to an XMLType column

The following 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));
```

Note: 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 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.

Reading XMLType data from a standard table or view

The following 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;
```

Note: 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.

Reading XMLType data from an object table

The following 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;
```

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

Reading XMLType data from an object view

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

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

To retrieve the XML value from VIEW1 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 mapping from Oracle to DataStage

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

The following table shows the mapping between Oracle data types 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, for example WE8MSWIN1252. For a multibyte character set, the NLS_CHARACTERSET database parameter is set to a multibyte character set, for example AL32UTF8.

The CHAR and VARCHAR2 Oracle data types in the table have their length specified as *n* BYTE or *n* CHAR. The length *n* BYTE is in byte units, and the length *n* CHAR is in character units. If neither the BYTE nor CHAR is explicitly specified when the column is defined, the NLS_LENGTH_SEMANTICS database parameter determines the unit of measure. For example, the column definition CHAR(5) is equivalent to CHAR(5 BYTE) when the NLS_LENGTH_SEMANTICS parameter is set to BYTE, and the column definition is equivalent to CHAR(5 CHAR) when the NLS_LENGTH_SEMANTICS database parameter is set to CHAR.

Table 21. Oracle data types and corresponding DataStage data types

Oracle data type	**DataStage data type**
CHAR(*n* BYTE)	SQL type: CHAR Length: *n* Scale: unset Extended: unset
CHAR(*n* CHAR) single-byte	SQL type: CHAR Length: *n* Scale: unset Extended: unset
CHAR(*n* CHAR) multibyte	SQL type: NCHAR Length: *n* Scale: unset Extended: unset
CHAR single-byte	If the NLS_LENGTH_SEMANTICS database parameter is set to CHAR, then see CHAR(*n* CHAR). Otherwise, see CHAR(*n* BYTE) single-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). Otherwise, see CHAR(*n* BYTE) multibyte. In both cases, assume that *n* = 1
VARCHAR2(*n* BYTE)	SQL type: VARCHAR Length: *n* Scale: unset Extended: unset
VARCHAR2(*n* CHAR) single-byte	SQL type: VARCHAR Length: *n* Scale: unset Extended: unset
VARCHAR2(*n* CHAR) multibyte	SQL type: NVARCHAR Length: *n* Scale: unset Extended: unset
CLOB single-byte	SQL type: LONGVARCHAR Length: unset Scale: unset Extended: unset
CLOB multibyte	SQL type: LONGNVARCHAR Length: unset Scale: unset Extended: unset

Table 21. Oracle data types and corresponding DataStage data types (continued)

Oracle data type	**DataStage data type**
LONG single-byte	SQL type: LONGVARCHAR Length: unset Scale: unset Extended: unset
LONG multibyte	SQL type: LONGNVARCHAR Length: unset Scale: unset Extended: unset
NCHAR(*n*)	SQL type: NCHAR Length: *n* Scale: unset Extended: unset
NCHAR	See NCHAR(*n*) and assume that *n* = 1.
NVARCHAR2(*n*)	SQL type: NVARCHAR Length: *n* Scale: unset Extended: unset
NCLOB	SQL type: LONGNVARCHAR Length: unset Scale: unset Extended: unset
NUMBER	SQL type: DOUBLE Length: unset Scale: unset Extended: unset
NUMBER (*p*, *s*) {*p*>=*s*} {*s*>=0}	SQL type: DECIMAL Length: *p* Scale: *s* Extended: unset
NUMBER(*p*, *s*) {*p*<*s*} {*s*>=0}	SQL type: DECIMAL Length: *s* Scale: *s* Extended: unset
NUMBER(*p*, *s*) {*s*<0}	SQL type: DECIMAL Length: *p-s* Scale: unset Extended: unset
FLOAT(*p*) {1 <=*p* <=63}	SQL type: FLOAT Length: unset Scale: unset Extended: unset
FLOAT(*p*) {64 <=*p* <= 127}	SQL type: DOUBLE Length: unset Scale: unset Extended: unset
BINARY_FLOAT	SQL type: FLOAT Length: unset Scale: unset Extended: unset

Table 21. Oracle data types and corresponding DataStage data types (continued)

Oracle data type	**DataStage data type**
BINARY_DOUBLE	SQL type: DOUBLE Length: unset Scale: unset Extended: unset
LONG RAW	SQL type: LONGVARBINARY Length: unset Scale: unset Extended: unset
RAW(*n*)	SQL type: VARBINARY Length: *n* Scale: unset Extended: unset
BLOB	SQL type: LONGVARBINARY Length: unset Scale: unset Extended: unset
BFILE	SQL type: VARCHAR Length: 285 Scale: unset Extended: unset
DATE	SQL type: DATE Length: unset Scale: unset Extended: unset
TIMESTAMP(*fsp*)	SQL type: TIMESTAMP Length: unset Scale: *fsp* Extended: Microseconds
TIMESTAMP(*fsp*) WITH TIME ZONE	SQL type: TIMESTAMP Length: unset Scale: *fsp* Extended: Microseconds
TIMESTAMP(*fsp*) WITH LOCAL TIME ZONE	SQL type: TIMESTAMP Length: unset Scale: *fsp* Extended: Microseconds
TIMESTAMP	See TIMESTAMP(*fsp*) and assume that *fsp*=6.
TIMESTAMP WITH TIME ZONE	See TIMESTAMP(*fsp*) WITH TIME ZONE and assume that *fsp*=6.
TIMESTAMP WITH LOCAL TIME ZONE	See TIMESTAMP(*fsp*) WITH LOCAL TIME ZONE and assume that *fsp*=6.
INTERVAL YEAR (*yp*) TO MONTH	SQL type: VARCHAR Length: *yp*+4 Scale: unset Extended: unset
INTERVAL DAY TO SECOND (*sp*)	SQL type: VARCHAR Length: *sp*+13 Scale: unset Extended: unset

Table 21. Oracle data types and corresponding DataStage data types (continued)

Oracle data type	DataStage data type
INTERVAL DAY (*dp*) TO SECOND	SQL type: VARCHAR Length: *dp*+17 Scale: unset Extended: unset
INTERVAL DAY (*dp*) TO SECOND (*sp*)	SQL type: VARCHAR Length: *dp*+*sp*+11 Scale: unset Extended: 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 that *dp*=2 and that *sp*=6.
ROWID	SQL type: CHAR Length: 18 Scale: 18 Extended: unset
UROWID(*n*)	SQL type: VARCHAR Length: *n* Scale: unset Extended: unset
UROWID	See UROWID(*n*) and assume that *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.
XMLType stored as BINARY XML	See BLOB.
Other	SQL type: UNKNOWN Length: unset Scale: unset Extended: unset

Data type mappings for creating a table

When you use **Table action** property to create a table, the connector maps 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

Note: For a DataStage column definition, if the Length, Scale, or Extended entry has no value, the attribute is not applicable to the data type.

Table 22. DataStage column definitions and corresponding Oracle column definitions

DataStage column definition	Oracle column definition
Data type: Bit Length: any Scale: any Extended:	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)
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:	NCHAR(1000)
Data type: NChar Length: *n* Scale: any Extended:	NCHAR(*n*)

Table 22. DataStage column definitions and corresponding Oracle column definitions (continued)

DataStage column definition	Oracle column definition
Data type: NVarChar Length: unset Scale: any Extended:	NVARCHAR2(2000)
Data type: NVarChar Length: *n* Scale: any Extended:	NVARCHAR2(*n*)
Data type: LongNVarChar Length: any Scale: any Extended:	NCLOB
Data type: Binary Length: unset Scale: any Extended:	RAW(2000)
Data type: Binary Length: *n* Scale: any Extended:	RAW(*n*)
Data type: VarBinary Length: unset Scale: any Extended:	RAW(2000)
Data type: VarBinary Length: *n* Scale: any Extended:	RAW(*n*)
Data type: LongVarBinary Length: any Scale: any Extended:	BLOB
Data type: Decimal Length: *p* Scale: unset Extended:	NUMBER(*p*)
Data type: Decimal Length: *p* Scale: *s* Extended:	NUMBER(*p*,*s*)
Data type: Double Length: any Scale: any Extended:	BINARY_DOUBLE
Data type: Float Length: any Scale: any Extended:	BINARY_FLOAT

Table 22. DataStage column definitions and corresponding Oracle column definitions (continued)

DataStage column definition	Oracle column definition
Data type: Real Length: any Scale: any Extended:	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)
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(19,0)
Data type: Numeric Length: *p* Scale: unset Extended:	NUMBER(*p*)
Data type: Numeric Length: *p* Scale: *s* Extended:	NUMBER(*p*,*s*)
Data type: Date Length: any Scale: any Extended:	DATE

Table 22. DataStage column definitions and corresponding Oracle column definitions (continued)

DataStage column definition	Oracle column definition
Data type: Time Length: any Scale: unset Extended: unset	INTERVAL DAY(2) TO SECOND(3)
Data type: Time Length: any Scale: *sp* Extended: unset	INTERVAL DAY(2) TO SECOND(sp)
Data type: Timestamp Length: any Scale: unset Extended: unset	TIMESTAMP(3)
Data type: Timestamp Length: any Scale: *sp* Extended: unset	TIMESTAMP(*sp*)
Data type: Time Length: any Scale: unset Extended: Microseconds	INTERVAL DAY(2) TO SECOND(6)
Data type: Time Length: any Scale: *sp* Extended: Microseconds	INTERVAL DAY(2) TO SECOND(*sp*)
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

Dictionary views

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

The following list describes how the Oracle connector uses each view.

ALL_CONSTRAINTS
: The Oracle connector accesses this view to obtain the list of constraints for a table. Obtaining a list of constraints is required for these tasks: importing a table definition, disabling constraints, and enabling constraints.

ALL_INDEXES
: The Oracle connector accesses this view to obtain the list of indexes for a table. Obtaining this information is required for these tasks: importing a

table definition, determining the list of indexes to rebuild, and determining how a table is organized, either by heap or by index.

ALL_OBJECTS
The Oracle connector accesses this view to obtain additional metadata, such as table names and view names, for the objects that the user specifies. 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
The Oracle connector accesses this view to determine the boundary (high) value for each partition in a table. This information is used in a partitioned write.

ALL_PART_KEY_COLUMNS
The Oracle connector accesses this view to determine the list of columns that are in the partition key for a table. This information is used in a partitioned write.

ALL_PART_TABLES
The Oracle connector accesses this view to determine partitioning method that the table uses. When the **Oracle connector** value is specified for the **Partition type** property, 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.

ALL_TAB_COLS
The Oracle connector accesses this view to determine column metadata, for example data type, length, precision, and scale; to determine if a column is a virtual column; and to 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.

ALL_TAB_PARTITIONS
The Oracle connector accesses this view to determine the number and names of the partitions in a partitioned table. This information is required for read and write operations.

ALL_TAB_SUBPARTITIONS
The Oracle connector accesses this view to determine the number and names of all subpartitions in a composite-partitioned table. This information is required for read and write operations.

ALL_TABLES
The Oracle connector accesses this view to determine the list of tables that are accessible by the current user. Obtaining this information is required for these tasks: importing a table definition, determining which users have tables with SYSTEM or SYSAUX table space as their default table space, and determining if a specified table is partitioned.

ALL_VIEWS
The Oracle connector accesses this view to determine the list of views that are accessible by the current user. This information is used to present the user with a list of views that can be imported.

DBA_EXTENTS
The Oracle connector accesses this view to gather information about the table storage organization. The connector uses the information when the

Rowid range partitioned read method is selected. If select access is not granted to this view, the connector automatically switches to the Rowid hash partitioned read method.

DUAL To obtain and calculate various intermediate values that the connector needs for its operation. The connector issues SELECT statements on this table.

USER_TAB_PRIVS
The Oracle connector accesses this table to determine if the current user was granted select privilege on a particular dictionary view, for example the DBA_EXTENTS view. If the current user was not granted select privilege, the connector takes corrective action.

Environment variables

The Oracle connector queries and uses these environment variables.

CC_MSG_LEVEL
This connector environment variable specifies the minimum severity of the messages that the connector reports in the log file. The default value is 3; Informational messages, as well as messages of a higher severity, are reported to the log file. The Oracle connector does not have any messages that are of the Error severity. The following list contains the valid values:

- 1 - Trace
- 2 - Debug
- 3 - Informational
- 4 - Warning
- 5 - Error
- 6 - Fatal

CC_ORA_BIND_KEYWORD
This connector environment variable specifies the identifier that indicates a bind parameter in a user-defined SQL statement. The default identifier is ORCHESTRATE. Use this environment variable to specify a different identifier in cases when SQL statements require the use of the literal ORCHESTRATE in the name of a schema, table, or column, for example.

CC_ORA_CHECK_CONVERSION
This connector environment variable controls 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 (case- insensitive), an exception is thrown when data loss occurs.

CC_ORA_MAX_ERRORS_REPORT
This connector environment variable specifies the maximum number of errors to report to the log file when an operation involves writing an array or bulk loading data. This variable is relevant only when a reject link is not defined. The default value is -1, which reports all errors.

CC_ORA_NLS_LANG_ENV
This connector environment variable controls 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 (case-insensitive), the NLS_LANG character set is used; otherwise, the UTF-16 character set is used.

CC_ORA_NODE_USE_PLACEHOLDER
This connector environment variable controls whether the connector replaces occurrences of the processing node number placeholder with the current processing node number in all SQL statements that run on processing nodes. When the value of this variable is TRUE (case-insensitive), the connector replaces the occurrences.

CC_ORA_NODE_PLACEHOLDER_NAME
This connector environment variable specifies the case-sensitive value for the processing node numbers in SQL statements.

Library path
This variable must include the directory where the Oracle client libraries are stored. The follow list contains the name of the library path variable for each operating system:

- 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 going through 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 under 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 without going through an Oracle listener. On Microsoft Windows, you can specify this environment variable in the Windows registry.

Note: 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 going through 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.

Note: If both ORACLE_SID and TWO_TASK are defined, TWO_TASK takes precedence. 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.

Product documentation

Documentation is provided in a variety of locations and formats, including in help that is opened directly from the product interface, in a suite-wide information center, and in PDF file books.

The information center is installed as a common service with IBM Information Server. The information center contains help for most of the product interfaces, as well as complete documentation for all product modules in the suite.

A subset of the product documentation is also available online from the product documentation library at publib.boulder.ibm.com/infocenter/iisinfsv/v8r1/index.jsp.

PDF file books are available through the IBM Information Server software installer and the distribution media. A subset of the information center is also available online and periodically refreshed at www.ibm.com/support/docview.wss?rs=14&uid=swg27008803.

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 www.ibm.com/shop/publications/order.

You can send your comments about documentation in the following ways:
- Online reader comment form: www.ibm.com/software/data/rcf/
- E-mail: comments@us.ibm.com

Contacting IBM

You can contact IBM for customer support, software services, product information, and general information. You can also provide feedback on products and documentation.

Customer support

For customer support for IBM products and for product download information, go to the support and downloads site at www.ibm.com/support/us/.

You can open a support request by going to the software support service request site at www.ibm.com/software/support/probsub.html.

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/us/.

Software services

For information about software, IT, and business consulting services, go to the solutions site at www.ibm.com/businesssolutions/us/en.

Information Management product support

For Information Management product support, news, and other product information, go to the Information Management support site at www.ibm.com/software/data/support/.

IBM InfoSphere Information Server support

For IBM InfoSphere Information Server support go towww.ibm.com/software/data/integration/support/info_server/.

General information

To find general information about IBM, go to www.ibm.com.

Product feedback

You can provide general product feedback through the Consumability Survey at www.ibm.com/software/data/info/consumability-survey.

Documentation feedback

You can click the feedback link in any topic in the information center to comment on the information center.

You can also send your comments about PDF file books, the information center, or any other documentation in the following ways:

- Online reader comment form: www.ibm.com/software/data/rcf/
- E-mail: comments@us.ibm.com

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).

```
►►─required_item──────────────────────────────────────────────►◄
```

- Optional items appear below the main path.

```
►►─required_item──┬───────────────┬───────────────────────────►◄
                  └─optional_item─┘
```

  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.

```
                  ┌─optional_item─┐
►►─required_item──┴───────────────┴───────────────────────────►◄
```

- 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.

```
►►─required_item──┬─required_choice1─┬────────────────────────►◄
                  └─required_choice2─┘
```

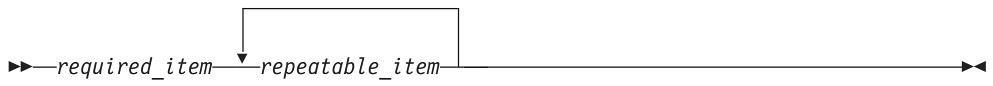
  If choosing one of the items is optional, the entire stack appears below the main path.

```
►►─required_item──┬──────────────────┬────────────────────────►◄
                  ├─optional_choice1─┤
                  └─optional_choice2─┘
```

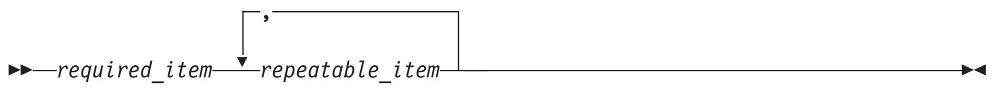
  If one of the items is the default, it appears above the main path, and the remaining choices are shown below.

```
                  ┌─default_choice───┐
►►─required_item──┼──────────────────┼────────────────────────►◄
                  ├─optional_choice1─┤
                  └─optional_choice2─┘
```

- An arrow returning to the left, above the main line, indicates an item that can be repeated.

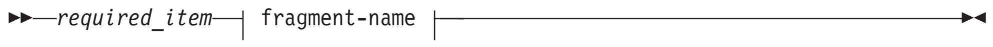


If the repeat arrow contains a comma, you must separate repeated items with a comma.

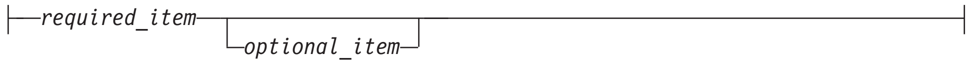


A repeat arrow above a stack indicates that you can repeat the items in the stack.

- 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).

Product accessibility

You can get information about the accessibility status of IBM products.

The IBM Information Server product modules and user interfaces are not fully accessible. The installation program installs the following product modules and components:

- WebSphere Business Glossary
- Information Server Business Glossary Anywhere
- IBM WebSphere DataStage and QualityStage
- IBM Information Server FastTrack
- IBM WebSphere Information Analyzer
- IBM WebSphere Information Services Director
- IBM InfoSphere™ Metadata Workbench

For more information about IBM product accessibility status, go to http://www.ibm.com/able/product_accessibility/index.html.

Accessible documentation

Accessible documentation for IBM 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. XHTML allows you to set display preferences in your browser. It also allows you to use screen readers and other assistive technologies to access the documentation.

Notices

This information was developed for products and services offered in the U.S.A.

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.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 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.

Trademarks

Index

Printed in USA

LC18-9937-03