

# **Technical Bulletin**

**Part No. 74-0143**

## **DataStage Teradata API**

This technical bulletin describes Release 1.2 of the DataStage Teradata API plug-in stage. This stage includes a GUI to read and write data to and from any DataStage stage into a Teradata database. It also provides native data browsing and meta data import from the Teradata database to DataStage.

© 2000–2003 Ascential Software Corporation. All rights reserved. Ascential, Ascential Software, DataStage, MetaStage, MetaBroker, and Axielle are trademarks of Ascential Software Corporation or its affiliates and may be registered in the United States or other jurisdictions. Adobe Acrobat is a trademark of Adobe Systems, Inc. Microsoft, Windows, Windows NT, and Windows Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Teradata is a registered trademark of NCR International, Inc. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd. Other marks mentioned are the property of the owners of those marks.

This product may contain or utilize third party components subject to the user documentation previously provided by Ascential Software Corporation or contained herein.

## **Printing History**

First Edition (74-0143) for Release 1.0, February 2000  
Second Edition (74-0143) for Release 1.0 and 1.1, August 2002  
Updated for Release 1.0 and 1.1, October 2002  
Third Edition (74-0143) for Release 1.1, November 2002  
Fourth Edition for Release 1.2, August 2003  
Updated for Release 1.2, December 2003

## **How to Order Technical Documents**

To order copies of documents, contact your local Ascential subsidiary or distributor, or call our office at (508) 366-3888.

Documentation Team: Marie E. Hedin

## Introduction

This technical bulletin describes the following for Release 1.2 of the Teradata API stage for DataStage Release 7.0.1:

- [Functionality](#)
- [Installation](#)
- [Defining the Teradata connection](#)
- [Defining Teradata input data](#)
- [Defining Teradata output data](#)
- [Mapping data types](#)

Currently, you can only use the DataStage ODBC stage to access Teradata data. Teradata API enables DataStage to read and write data to and from Teradata databases using the Teradata CLI native programming interface. The Teradata API stage offers the following advantages over the DataStage ODBC stage:

- Increased processing speed
- Support for the Teradata client software for Windows NT and UNIX
- Simplified configuration on UNIX platforms
- Advanced support for target table DDL (Create and Drop Table)
- Native meta data import

This plug-in stage uses the Teradata CLI programming API for network-attached systems to let you connect and process SQL statements in the native Teradata environment.

In summary, Teradata API lets you do the following for a target Teradata database:

- Read and write data
- Create and drop tables
- Import table and column definitions
- Browse native data with the custom GUI

Each Teradata API stage is a passive stage that can have any number of the following links:

- **Input links.** Specify the data you are writing, which is a stream of rows to be loaded into a Teradata database. You can specify the data on an input link using an SQL statement generated by DataStage or constructed by the user.
- **Output links.** Specify the data you are extracting, which is a stream of rows to be read from a Teradata database. You can specify the data on an output link using an SQL SELECT statement generated by DataStage or constructed by the user.

- **Reference output links.** Each link represents rows that are key read from a Teradata database (using the key columns in a WHERE clause of the SELECT statement that is constructed by DataStage or specified by the user).

For more information on using a stage in a DataStage job, see *DataStage Server Job Developer's Guide*.

## Functionality

The Teradata API stage has the following functionality:

- Stream input, stream output, and reference output links.
- The ability to import table and column definitions from the target Teradata database and store them in the DataStage Repository. For more information about meta data import, see *DataStage Server Job Developer's Guide*.
- NLS (National Language Support). For more information, see *DataStage NLS Guide*.
- Reject row handling.
- File names to contain your SQL statements.
- Support of MetaStage. For more information, see *MetaStage User's Guide*.
- Data browsing, which is the ability to use the custom GUI for the stage to view sample native table data residing on the target Teradata database.

The following functionality is not supported:

- Bulk loading of Teradata tables. Use the Teradata Load stage for bulk loading into a Teradata database. For more information, see the technical bulletin *DataStage Teradata Load (74-0144)*.
- Replacing the ODBC stage. The Teradata API stage does not replace the ODBC stage. DataStage users who created jobs using the ODBC stage to access a Teradata database may continue to run these jobs.
- Stored procedures.
- Non-ANSI SQL statements in stage-generated SQL statements.
- Version-specific SQL statements in stage-generated SQL statements.
- Text and byte data types.

## Installing the Plug-In

For instructions and information supporting the installation, see *DataStage Plug-In Installation and Configuration Guide*.

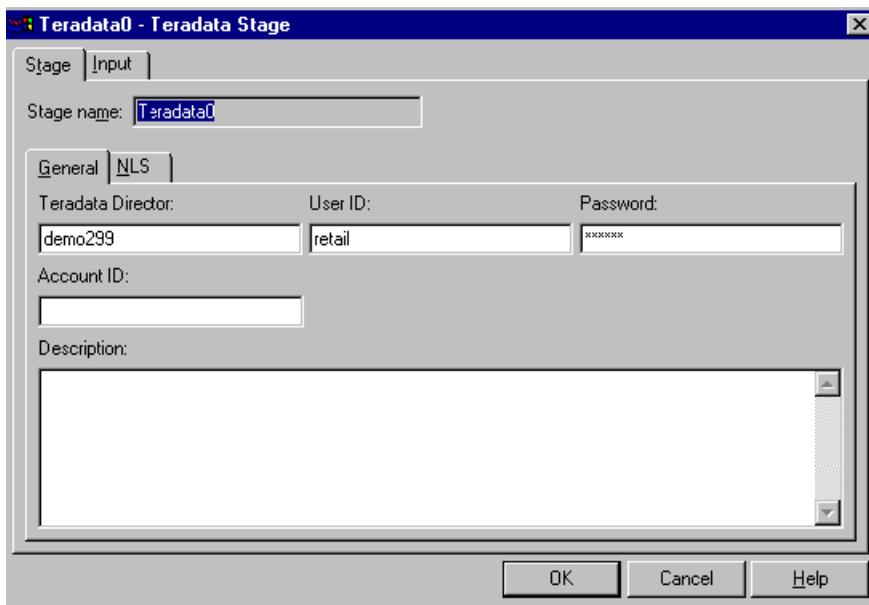
Before installing the Plug-in, consult Teradata documentation for any specific configuration requirements.

**Note:** Very slow performance can occur as data is written if you run the Teradata server software on the same NT machine where you run Teradata API or ODBC (to Teradata).

To improve the performance, use two different machines for the Teradata client and Teradata server. This balances the load since the DataStage server represents the Teradata client and the Teradata RDBMS represents the server.

## Defining the Teradata Stage

Using the GUI is easier than using grids to edit the values they contain. When you use the GUI to edit a Teradata API stage, the **Teradata Stage** dialog box appears:



This dialog box has the **Stage**, **Input**, and **Output** 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** page defines the Teradata data source and login information. You can enter text to describe the purpose of the stage in the **Description** field. The properties on this page define the connection to the Teradata data source. For details, see [“Connecting to a Teradata Data Source”](#) on page 5. The **NLS** page defines a character set map to use with the stage. This page appears only if you have installed NLS for DataStage. For details, see [“Defining Character Set Mapping”](#) on page 6.

**Note:** You cannot change the name of the stage from this dialog box. For details on changing stage names, see *DataStage Server Job Developer’s Guide*.

- **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. It also specifies how data is written and contains the SQL statement or call syntax used to write data to a Teradata table. It also specifies how to create the target table if desired and how to drop it if necessary.
- **Output.** This page is displayed only if you have an output or reference link to this stage. It specifies the SQL tables to use and the associated column definitions for each data output link. It contains the SQL SELECT statement or call syntax used to read data from one or more Teradata tables or views.

The main phases in defining a Teradata CLI stage from the **Teradata Stage** dialog box are as follows:

1. Connect to a Teradata data source (see [page 5](#)).
2. Optional. Define a character set map (see [page 6](#)).
3. Define the data on the input links (see [page 7](#)).
4. Define the data on the output links (see [page 15](#)).
5. Click **OK** to close this dialog box.

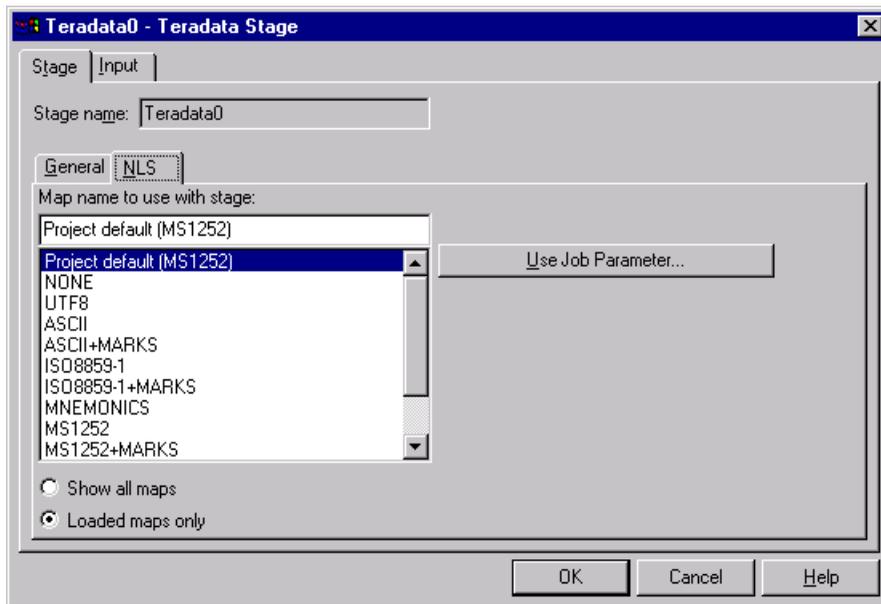
## Connecting to a Teradata Data Source

The Teradata API connection parameters are set on the **General** page of the **Stage** page. Enter the appropriate information for the following fields:

- **Teradata Director.** The Teradata Director Program ID (tdpid) that is associated with a particular Teradata server. The Teradata server has a unique tdpid. (See your system administrator for the identifier associated with the Teradata RDBMS that you plan to use.) If no value is given, the value in the **dbcname** field in the clispb.dat file is used.
- **User ID.** The name to use to connect to the Teradata server. This user must have sufficient privileges to access the specified database and source and target tables.
- **Password.** The password associated with the specified user name. For security, it displays asterisks instead of the value you enter.
- **Account ID.** Your individual user account that is associated with **User ID**.
- **Description.** Optionally, describe the purpose of the Teradata API stage.

## Defining Character Set Mapping

You can define a character set map for a stage. Do this from the **NLS** tab that appears on the **Stage** page. The **NLS** page appears only if you have installed NLS.



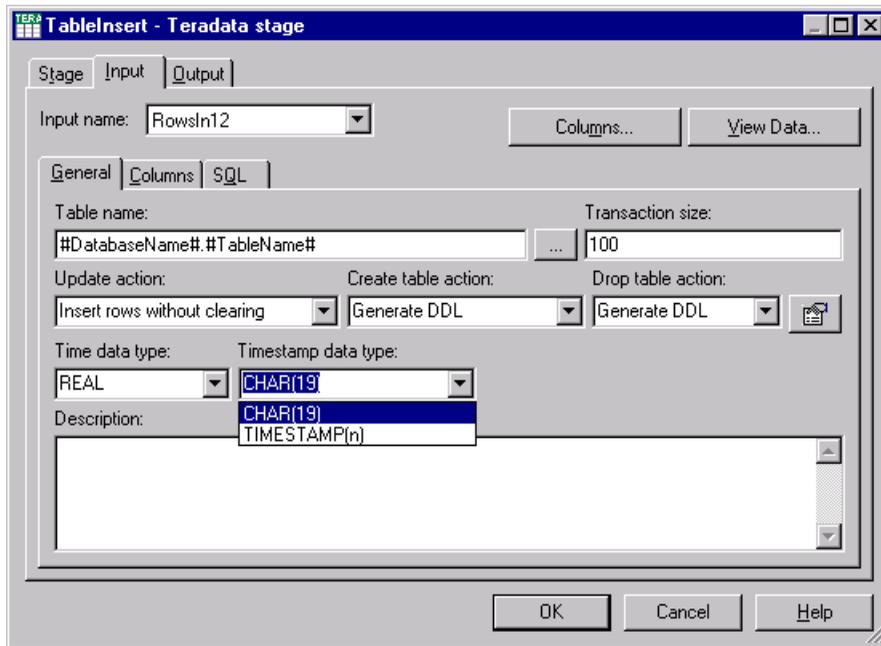
Enter information for the following button and fields, if appropriate:

- **Map name to use with stage.** The default character set map is defined for the project or the job. You can change the map by selecting a map name from the list.
- **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.
- **Show all maps.** Lists all the maps that are shipped with DataStage.
- **Loaded maps only.** Lists only the maps that are currently loaded.

For additional information about NLS, see *DataStage NLS Guide*, or for additional information about job parameters, see *DataStage Server Job Developer's Guide*.

## Defining Teradata Input Data

When you write data to a table in a Teradata database, the Teradata API stage has an input link. Define the properties of this link and the column definitions of the data on the **Input** page in the **Teradata Stage** dialog box.



## About the Input Page

The **Input** page has an **Input name** field, the **General**, **Columns**, and **SQL** pages, and the **Table Properties** (at the right of the **Drop table action** list box), **Columns...**, and **View Data...** buttons.

- **Input name.** The name of the input link. Choose the link you want to edit from the **Input name** drop-down list box. This list displays all the input links to the Teradata API stage.
- Click the **Columns...** button to display a brief list of the columns designated on the input link. As you enter detailed meta data in the **Columns** page, you can leave this list displayed.

- Click the **View Data...** button to start the Data Browser. This lets you look at the data associated with the input link. For a description of the Data Browser, see *DataStage Server Job Developer's Guide*.

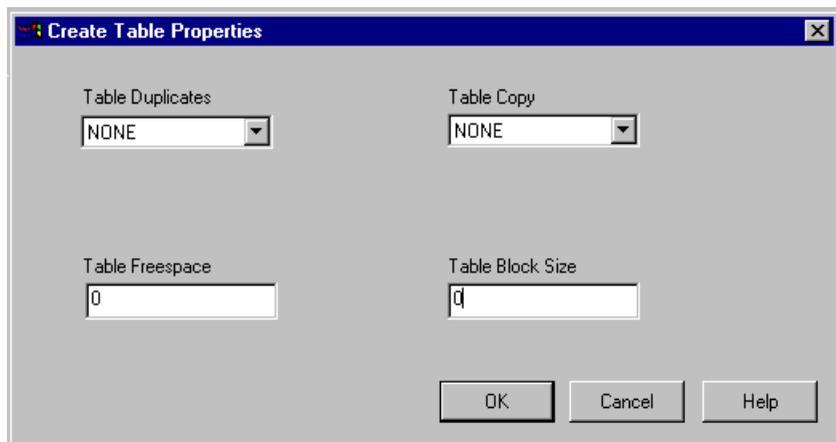
## General Page

This page is displayed by default. Enter the appropriate information for the following fields:

- **Table name.** This field is editable when the update action is *not* **User-defined SQL** (otherwise, it is read-only). It is the name of the target table to update. You must specify **Table name** if you do not specify **User-defined SQL**. There is no default. You can also click the ... button at the right of the **Table name** field to browse the Repository to select the table.
- **Transaction size.** The number of rows that the stage processes before committing a transaction to the database. The default is 100. This field is ignored for nonlogging databases.
- **Update action.** Specifies which stage-generated SQL statements are used to update the target table. Some update actions require key columns to update or delete rows. The default is insert rows without clearing. Choose one of the following options:
  - **Insert rows without clearing.** Inserts the new rows in the table.
  - **Clear the table, then insert rows.** Deletes the contents of the table before inserting the new rows. Transaction logging causes slower performance.
  - **Delete existing rows only.** Deletes existing rows in the target file that have identical keys in the input rows.
  - **Replace existing rows completely.** Deletes the existing rows, then adds the new rows to the table.
  - **Update existing rows only.** Updates the existing data rows. Any rows in the data that do not exist in the table are ignored.
  - **Update existing or insert new rows.** Updates the existing data rows before inserting new rows. Performance depends on the contents of the target table and the rows being processed in the job. If most rows exist in the target table, it is faster to update first.
  - **Insert new or update existing rows.** Inserts the new rows before updating existing rows. Performance depends on the contents of the

target table and the rows being processed in the job. If most rows do not exist in the target table, it is faster to insert first.

- **User-defined SQL.** Writes the data using a user-defined SQL statement. When you select this option, it overrides the default SQL statement generated by the stage. If you choose this option, you enter data on the SQL page. See [“Using User-Defined SQL Statements”](#) on page 14 for details on how to do this.
- **Create table action.** Choose one of the following options to create the target table in the specified database:
  - **Do not create target table.** Specifies that the target table is not created, and disables the **Drop table action** field and the **Table Properties** button (at the right of the field).
  - **Generate DDL.** Specifies that the stage generates the CREATE TABLE statement using information obtained from the “Target Table” property, the column definitions grid, and the advanced table properties (see the **Table Properties** button later in this section).
  - **User-defined DDL.** Specifies that you enter the appropriate CREATE TABLE statement on the **SQL** page as described on page 11.
- **Drop table action.** Controls the dropping of the target table before it is created by the stage. If you choose not to create the target table, this field is disabled. The list box displays the same items as the **Create table action** list box except that they apply to the DROP TABLE statement.
- **Table Properties button.** Click the button at the right of the **Drop table action** list box to display the **Create Table Properties** dialog box.



You can then specify the following advanced table properties from this dialog box:

- **Table Duplicates.** Controls duplicate row control. Use one of the following values:
  - **NONE.** Duplicate rows are not allowed in Teradata mode and but are allowed in ANSI mode. This is the default.
  - **SET.** Duplicate rows are not allowed. This is compatible with Teradata RDBMS tables from prior releases.
  - **MULTISET.** Duplicate rows are allowed. This is compliant with the ANSI standard.
- **Table Copy.** Specifies whether to choose duplicate copy protection for the table. Use one of the following values:
  - **NONE.** This option is established by a CREATE DATABASE statement for the database in which the table is to be created. Do not use this clause in the CREATE TABLE statement.
  - **FALLBACK.** Duplicate copies of rows in the table are created and stored.
  - **NOFALLBACK.** Duplicate copies of rows in the table are not created and stored.
- **Table Freespace.** Sets the percent of free space that remains on a cylinder during loading or update operations. The default value of 0 means that this clause is not used in the CREATE TABLE statement. Enter a number from 0 to 75.
- **Table Block Size.** Sets the value of the data block size attribute to the unit specified in bytes. The default value of 0 means this clause is not used in the CREATE TABLE statement. Enter a number from 6144 to 32256.
- **Time data type.** If you select **REAL** (the default value), the Plug-in defines time columns with a data type of REAL. Time values are encoded as (hour\*10000 + minute\*100 + second), where second can include a fractional value. If you select **TIME(n)**, the Plug-in defines time columns as TIME(n), where n is the Scale value, in the range 0 through 6, representing the fractional seconds precision. The Length value for the time column must be 8 if Scale equals 0, or it must be 9+ if Scale is greater than 0.
- **Timestamp data type.** If you select **CHAR(19)** (the default value), the Plug-in defines timestamp columns as CHAR(19). If you select **TIMESTAMP(n)**, the Plug-in defines timestamp columns as TIMESTAMP(n), where n is the

Scale value, in the range 0 through 6, representing the fractional seconds precision. The Length value for the timestamp column must be 19 if Scale equals 0, or it must be 20+ if Scale is greater than 0.

- **Description.** Optionally enter text to describe the purpose of the link.

## Columns Page

This page contains the column definitions for the data written to the table or file. The **Columns** page behaves the same way as the **Columns** page in the ODBC stage. For a description of how to enter and edit column definitions, see *DataStage Server Job Developer's Guide*.

## SQL Page

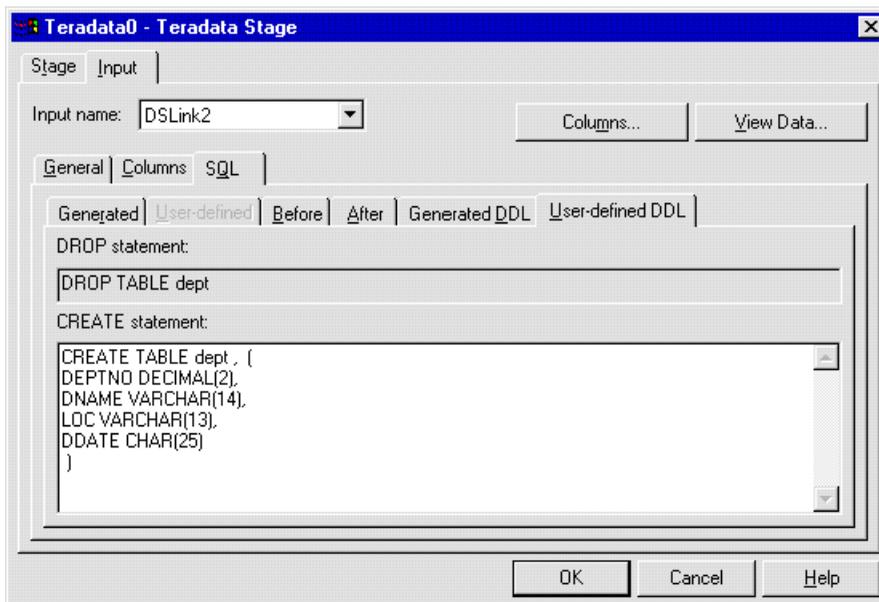
This page contains the **Generated**, **User-defined**, **Before**, **After**, **Generated DDL**, and **User-defined DDL** pages. Use these pages to display the stage-generated SQL statement and the SQL statement that you can enter.

- **Generated.** This page displays the SQL statements constructed by DataStage that are used to write data to Teradata. The statements represent the uneditable result of the selection made in the **Update action** field on the **General** page. You can use **Copy** to copy them to the Clipboard for use elsewhere. This page is displayed by default.
- **User-defined.** Select **User-defined SQL** from the **Update action** field on the **General** page to enable this page. The GUI displays the stage-generated SQL statement on this page as a starting point. However, you can enter any valid, appropriate SQL statement. The box size changes proportionately when you resize the main window to display long SQL statements.
- **Before.** This page contains the SQL statements executed before the stage processes any job data rows. The **Before** and **After** pages look alike. The SQL statement is entered in a resizable edit box. It includes the **Treat errors as non-fatal** check box. If selected (the default), errors caused by Before SQL are logged as warnings, and processing continues with the next command batch, if any. Each successful execution is treated as a separate transaction. If not selected, errors are treated as fatal to the job and result in a transaction rollback. The transaction is committed only if all statements successfully execute.
- **After.** This page contains the SQL statements executed after the stage processes job data rows. The **Before** and **After** pages look alike. The SQL statement is entered in a resizable edit box. It includes the **Treat errors as non-fatal** check box. If selected (the default), errors caused by After SQL

are logged as warnings, and processing continues with the next command batch, if any. Each successful execution is treated as a separate transaction. If not selected, errors are treated as fatal to the job and result in a transaction rollback. The transaction is committed only if all statements successfully execute.

- **Generated DDL.** Select **Generate DDL** or **User-defined DDL** from the **Create table action** field on the **General** page to enable this page. The **CREATE statement** field displays the CREATE TABLE statement that is generated from the column meta data 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 **General** page to enable this page. The generated DDL statement is displayed as a starting point to define a CREATE TABLE and a DROP TABLE statement.

The **DROP 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** pages.



**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.

## Writing Data to Teradata

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

### Using Generated SQL Statements

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

To use a generated statement:

1. Enter a table name in the **Table name** field on the **General** page of the **Input** page.
2. Specify how you want the data to be written by choosing an option from the **Update action** drop-down list box. See [“General Page”](#) on page 7 for a description of the update actions.
3. Optional. Enter a description of the input link in the **Description** field.
4. Click the **Columns** tab on the **Input** page. The **Columns** page appears.
5. 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. You can now optionally view this SQL statement.

6. 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 always

access this tab to select and copy parts of the generated statement to paste into the user-defined SQL statement.

7. Click **OK** to close this 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 DataStage, you can enter your own SQL INSERT, DELETE, or UPDATE statement for each Teradata input link. 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 use your own SQL statement:

1. Choose **User-defined SQL** from the **Update action** drop-down list box on the **General** page of the **Input** page.
2. Click the **SQL** tab, then the **User-defined** tab. The **User-defined** page appears.

By default you see the stage-generated SQL statement. You can edit this statement or enter your own SQL statement to write data to the target Teradata tables. This statement must contain the table name, the type of update action you want to perform, and the columns you want to write.

If the property value begins with {FILE}, the remaining text is interpreted as a pathname, and the contents of the file supplies the property value.

When writing data, the INSERT statements must contain a VALUES clause with a parameter marker ( ? ) for each stage input column. UPDATE statements must contain a SET clause 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. If you specify multiple SQL statements, each is executed as a separate transaction. Terminate individual SQL statements with a semicolon ( ; ). Use a double semicolon ( ;; ) to indicate the end of the command batch. You cannot combine multiple INSERT, UPDATE, and DELETE statements in one batch. You must execute each statement in a separate command batch.

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 (?, ?)
```

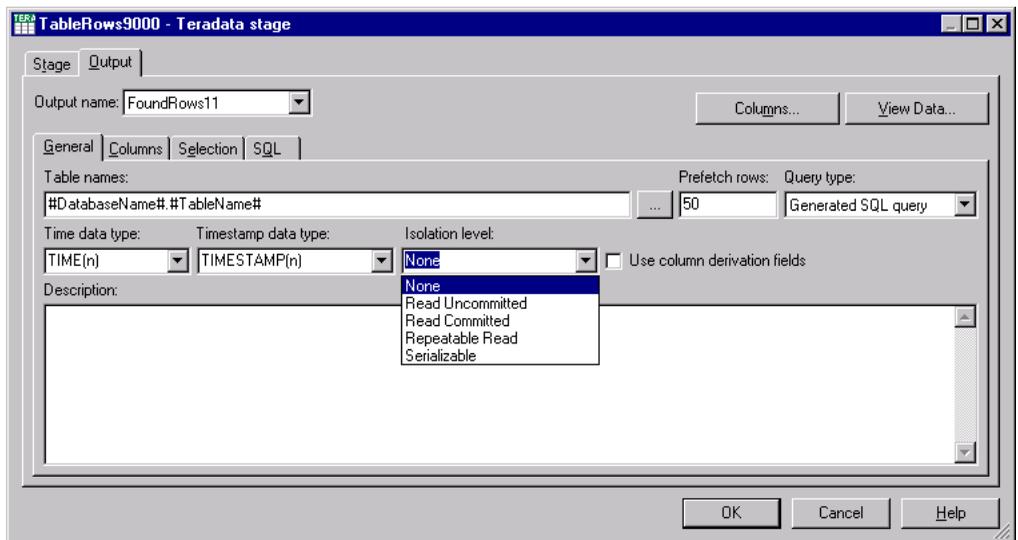
The size of this box changes proportionately when the main window is resized to conveniently display very long and/or complex SQL statements.

Unless you specify a user-defined SQL statement, the stage automatically generates an SQL statement.

3. Click **OK** to close this dialog box. Changes are saved when you save your job design.

## Defining Teradata Output Data

When you read data from a Teradata data source, the Teradata API stage has an output link. The properties of this link and the column definitions of the data are defined on the **Output** page in the **Teradata Stage** dialog box.



## About the Output Page

The **Output** page has an **Output name** field, the **General**, **Columns**, **Selection**, and **SQL** pages, and the **Columns...** and **View Data...** buttons. The pages displayed depend on how you specify the SQL statement to output the data.

- **Output name.** The name of the output link. Choose the link you want to edit from the **Output name** drop-down list box. This list displays all the output links.
- Click the **Columns...** button to display a brief list of the columns designated on the output link. As you enter detailed meta data in the **Columns** page, you can leave this list displayed.

- Click the **View Data...** button to start the Data Browser. This lets you look at the data associated with the output link. For a description of the Data Browser, see *DataStage Server Job Developer's Guide*.

## General Page

This page is displayed by default. It provides the interface for entering table names, the number of prefetch rows, and the type of query. Enter the appropriate information for the following fields:

- **Table names.** This field appears only when you select **Generated SQL query**. It contains the names of the Teradata source tables or files being accessed. These tables must exist or be created and populated by Before SQL statements.

Separate multiple table names by a comma ( , ). You must have select privileges on each table. There is no default. If you specify **User-defined SQL query**, **Table names** is ignored. You must specify a table name in **Table names** if you do not define **User-defined SQL query**.

Additionally, you can use a job parameter to specify the table name. For details on how to use define and use job parameters, see *DataStage Server Job Developer's Guide*.

You can also click the ... button at the right of the **Table names** field to browse the Repository to select the table.

- **Prefetch rows.** The number of rows that Teradata returns when DataStage fetches data from the source tables. Specifying a value greater than 1 improves performance (memory usage increases to accommodate buffering multiple rows).
- **Query type.** Displays the **Generated SQL query** and **User-defined SQL query** options. Choose one of the following options:
  - **Generated SQL query.** This is the default setting, which specifies that the data is extracted using an SQL statement constructed by DataStage. When this option is selected, the Generated page appears. You cannot edit this statement.
  - **User-defined SQL query.** Specifies that the data is extracted using a user-defined SQL query. When this option is selected, the User-defined page appears allowing you to edit SQL statements.
- **Time data type.** If you select **REAL** (the default value), the Plug-in expects time columns to be defined with a data type of REAL. If you select **TIME(n)**, the Plug-in expects time columns to be defined as TIME(n),

where *n* is the Scale value, in the range 0 through 6, representing the fractional seconds precision. The Length value for the time column must be 8 if Scale equals 0, or it must be 9 + Scale.

- **Timestamp data type.** If you select **CHAR(19)** (the default value), the Plug-in expects timestamp columns to be defined as **CHAR(19)**. If you select **TIMESTAMP(*n*)**, the Plug-in expects timestamp columns to be defined as **TIMESTAMP(*n*)**, where *n* is the Scale value, in the range 0 through 6, representing the fractional seconds precision. The Length value for the timestamp column must be 19 if Scale equals 0, or it must be 20 + Scale.
- **Isolation Level.** Sets the isolation level. The values are:
  - **None.** Uses the default isolation level for the database. See Teradata documentation.
  - **Read Uncommitted.** Allows the reading of changes before they are committed (dirty reads). When a job updates a table (inserts, updates or deletes rows), those changes are not final until the job commits the transaction. If the job aborts and rolls back the transaction, then those changes are backed out, and the table is restored to the state it was in when the transaction was initiated. Selecting **Read Uncommitted** allows the stage to read uncommitted changes while another transaction is updating the table.
  - **Read Committed.** Prevents the reading of changes before they are committed but allows other transactions to modify rows that have been read. Normally when a job performs a lookup in a table, the rows that are read are locked until the end of the job; thus if the job rereads the same row it will get the same result (see “Repeatable Read” below). Selecting **Read Committed** causes the stage immediately to release the lock on a row that has been read so that other transactions can later update the row while the lookup is still in progress.
  - **Repeatable Read.** Prevents any transactions from modifying any data that has been read but allows the reading of phantom rows. Selecting **Repeatable Read** causes the stage to hold a read lock on any row that is read until the end of the job. This guarantees that rows that are reread during the execution of the job will return the same result. Other transactions cannot update or delete rows that have been read, but they can insert new rows that the lookup can then subsequently read. These are known as phantom rows.
  - **Serializable.** Prevents any transactions from modifying any data that has been read and prevents the reading of phantom rows. Selecting

**Serializable** causes the stage to use a table-level read lock until the end of the job. Thus, transactions cannot insert, update or delete rows while the table is being read. Reads are repeatable, and it is not possible to read phantom rows. Other transactions cannot update any rows in the table while the lookup is in progress.

- **Use column derivation fields.** If not selected (the default), the Derivation field on the Columns tab is ignored. If selected, any SQL expressions in the Derivation field on the Columns tab appear in the select list of the generated SQL SELECT statement. If an expression is not specified in the Derivation field, the column name appears in the select list of the generated SQL SELECT statement.
- **Description.** Optionally enter text to describe the purpose of the link.

## Columns Page

This page contains the column definitions for the data being output on the chosen link. For a description of how to enter and edit column definitions, see *DataStage Server Job Developer's Guide*.

The column definitions for reference links require a key field. Key fields join reference inputs to a Transformer stage. Teradata API key reads the data by using a WHERE clause in the SQL SELECT statement.

## Selection Page

This page is used primarily with generated SQL queries. It contains optional SQL SELECT clauses for the conditional extraction of data. These clauses are appended to the generated SQL statements.

## SQL Page

This page displays the stage-generated or user-defined SQL statements used to read data from Teradata. It contains the **Generated**, **User-defined**, **Before**, and **After** pages. These pages are identical to those under the SQL page for the **Input** page except there are no **Generated DDL** and **User-defined DDL** pages.

- **Generated.** This page is displayed by default. It contains the SQL statements constructed by DataStage as a result of the **Update action** from the **General** page of the **Output** page. You cannot edit these statements, but you can use **Copy** to copy them to the Clipboard for use elsewhere.
- **User-defined.** This page contains the SQL statements executed to read data from Teradata. This page is enabled when you select **User-defined SQL query** from the **Query type** field on the **General** page. The GUI displays

the stage-generated SQL statement on this page as a starting point. However, you can enter any valid, appropriate SQL statement. The box size changes proportionately when you resize the main window to display long SQL statements.

- **Before.** This page contains the SQL statements executed before the stage processes any job data rows. It includes the **Treat errors as non-fatal** check box. If selected (the default), errors caused by Before SQL are logged as warnings, and processing continues with the next command batch, if any. Each successful execution is treated as a separate transaction. If not selected, errors are treated as fatal to the job and result in a transaction roll-back. The transaction is committed only if all statements successfully execute.
- **After.** This page contains the SQL statements executed after the stage processes all job data rows. It includes the **Treat errors as non-fatal** check box. If selected (the default), errors caused by After SQL are logged as warnings, and processing continues with the next command batch, if any. Each successful execution is treated as a separate transaction. If not selected, errors are treated as fatal to the job and result in a transaction roll-back. The transaction is committed only if all statements successfully execute.

## Reading Data from Teradata

The following sections describe the differences when you use generated queries or user-defined queries to read data from a Teradata database into DataStage.

### Using Generated Queries

By default, DataStage extracts data from a Teradata data source using an SQL SELECT statement that it constructs. The SQL statement is automatically constructed using the table and column definitions that you entered in the stage output properties.

When you select **Generated SQL query**, data is extracted from a Teradata database using an SQL SELECT statement constructed by DataStage. SQL SELECT statements have the following syntax:

```
SELECT clause FROM clause
    [WHERE clause]
    [GROUP BY clause]
    [HAVING clause]
    [ORDER BY clause];
```

When you specify the tables to use and the columns to be output from the Teradata API stage, the SQL SELECT statement is automatically constructed and can be viewed by clicking the **SQL** tab on the **Output** page.

For example, if you extract the Name, Address, and Phone columns from a table called Table1, the SQL statement displayed on the **SQL** page is:

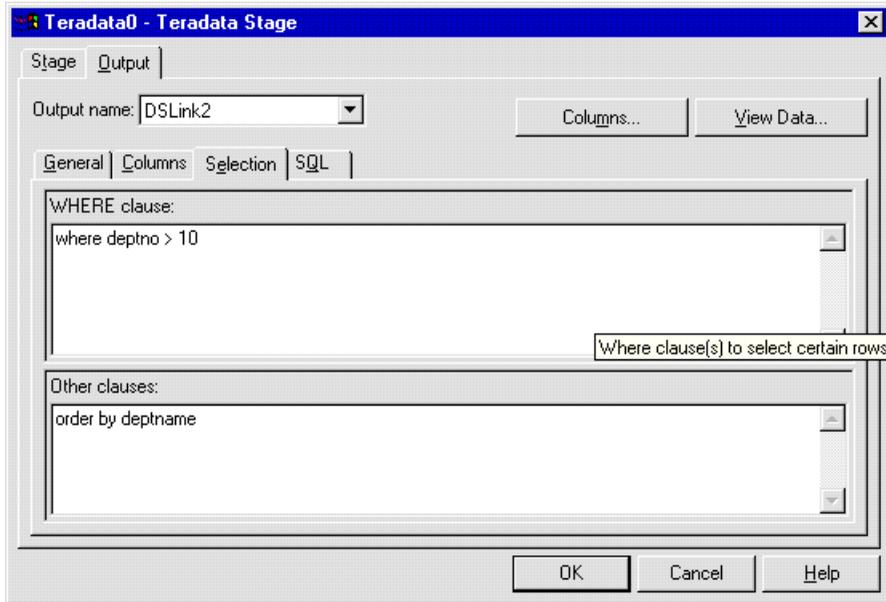
```
SELECT Name, Address, Phone FROM Table1;
```

The SELECT and FROM clauses are the minimum required and are automatically generated by DataStage. However, you can use any of these SQL SELECT clauses:

SELECT clause	Specifies the columns to select from the database.
FROM clause	Specifies the tables containing the selected columns.
WHERE clause	Specifies the criteria that rows must meet to be selected.
GROUP BY clause	Groups rows to summarize results.
HAVING clause	Specifies the criteria that grouped rows must meet to be selected.
ORDER BY clause	Sorts selected rows.

For more information about these clauses, see *DataStage Server Job Developer's Guide*.

**Using Additional SQL Select Clauses.** If you want to use the additional SQL SELECT clauses, you must enter them on the **Selection** page of the **Output** page. These clauses are appended to the SQL statement that is generated by the stage. If this link is a reference link, only the WHERE clause is enabled.



The **Selection** page is divided into two areas (panes). You can resize an area by dragging the split bar for displaying long SQL clauses.

- **WHERE clause.** This text box allows you to insert an SQL WHERE clause to specify criteria that the data must meet before being selected.
- **Other clauses.** This text box allows you to insert a GROUP BY, HAVING, or ORDER BY clause.

For more information about these clauses, see *DataStage Server Job Developer's Guide*.

## Using User-Defined Queries

Instead of using the SQL statement constructed by DataStage, you can enter your own SQL statement for each Teradata output link.

1. Select **User-defined SQL query** from the **Query type** drop-down list box on the **General** page of the **Output** page. The **User-defined** page on the **SQL** page is enabled. It looks like the **User-defined** page for the input link.
2. You can edit the statements or drag and drop the selected columns into your user-defined SQL statement. You must ensure that the table definitions for the output link are correct and represent the columns that are expected.

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.

3. Click **OK** to close this dialog box. Changes are saved when you save your job design.

## Mapping Data Types

You can map DataStage data types to Teradata data types. When “Create Table” is set to Yes for input links, the target table is created using the input link column definitions and the input link properties that define the properties for the target table.