IBM InfoSphere Information Server
Version 11 Release 3

Connectivity Guide for Accessing JDBC Data Sources

SC19-4345-00

IBM InfoSphere Information Server
Version 11 Release 3

Connectivity Guide for Accessing JDBC Data Sources

IBM

SC19-4345-00

Note

Before using this information and the product that it supports, read the information in “Notices and trademarks” on page 55.

Contents

JDBC data sources

Use the JDBC connector to connect to supported JDBC data sources and perform data access and metadata import operations on them.

Use the JDBC connector to perform the following operations:

- Read data from or write data to JDBC data sources.
- Import metadata from JDBC data sources through InfoSphere Metadata Asset Manager (IMAM).

Related reference:

"Troubleshooting the JDBC Connector stage" on page 40
Several common errors are specific to the JDBC Connector stage.

Setting up the driver configuration file

Before you can use the JDBC connector, you must set up the driver configuration file. The connector uses this file to obtain information about the available JDBC drivers in your system.

Procedure

1. Create a driver configuration file named `isjdbc.config` with read permission enabled for all the users. The driver configuration file name is case sensitive.
2. Open a text editor and include the following two lines that specify the class path and driver Java classes:

   ```
   CLASSPATH=driver_classpath
   CLASS_NAMES=driver_class_names
   ```

   The value `driver_classpath` is the cumulative Java class path for the JDBC drivers that you plan to utilize through the connector. The value is specified as a semicolon separated list of fully-qualified directory paths, `.jar` file paths and `.zip` file paths. For example, if the driver that you plan to use is implemented as a `.jar` file, include the full path to that `.jar` file in the `driver_classpath` value. For more information about the class path requirements for your driver, see the driver documentation.

   The value `driver_class_names` is a semicolon separated list of fully-qualified driver class names implemented by the JDBC drivers that you plan to utilize through the connector. The driver classes are the Java classes in the drivers that implement the `java.sql.Driver` JDBC API interface. For more information about the driver class implemented by the driver, see your JDBC driver documentation. Note that you do not need to provide this information for the drivers implemented as JAR files and based on JDBC 4.0 or later JDBC specification because in those cases the connector is able to automatically determine the driver class name from the driver JAR file.
3. Save the `isjdbc.config` file on the InfoSphere® Information Server engine tier host under the directory `IS_HOME/Server/DSEngine` directory, where *IS_HOME* is the InfoSphere Information Server home directory. For example, the home directory might be `C:\IBM\InformationServer` on a Windows system or `/opt/IBM/InformationServer` on a Linux or UNIX system. If the engine tier in your InfoSphere Information Server installation consists of multiple hosts, this file must be available from the same location on all the hosts. You can make this file available from the same location on all the hosts, by configuring the `DSEngine` directory as a shared network directory.

Example

This example shows how you can set up the driver configuration file.

Assume the following details:

- You want to use the JDBC connector with the following two JDBC drivers: JDBC 4.0 driver, Driver A and JDBC 3.0 driver, Driver B.
- Driver A is implemented as the `/opt/productA/driverA.jar` file, and Driver B is implemented as the`/app/productB/driverBimpl.jar` file.
- For Driver A, you determine that the name of the driver Java class is `com.example.A.Driver`, and for Driver B, you determine that the driver Java class name is `com.example.DriverB`.

To set up the driver configuration file, complete the following actions:

1. Create the `isjdbc.config` file and enter the following two lines:

   ```
   CLASSPATH=/opt/productA/driverA.jar;/app/productB/driverBimpl.jar
   CLASS_NAMES=com.example.A.Driver;com.example.DriverB
   ```

   Note: The `com.example.A.Driver` value can be omitted from the `CLASS_NAMES` because the Driver A is a JDBC 4.0 driver, and for this driver the connector can automatically retrieve the driver class name from the driver JAR file.
2. Save the file on the InfoSphere Information Server engine tier host under the `IS_HOME/Server/DSEngine` directory, where *IS_HOME* is the InfoSphere Information Server home directory.

After making changes to the driver configuration file you do not need to restart the DataStage engine, ISF agents or WebSphere Application Server. The JDBC connector recognizes the changes that are made to this file the next time it is used to access JDBC data sources.

Designing jobs with JDBC Connector stages

You can use JDBC Connector stages in your jobs to read data from the JDBC data sources or write data to JDBC data sources in the contexts of those jobs.

Before you begin

Set up the driver configuration file.

Procedure

1. Define a job that includes a JDBC Connector stage.
2. Define a connection to a JDBC data source.
3. To set up the JDBC Connector stage as a source stage to read data from the JDBC data source, complete the following steps:
   a. Configure the JDBC Connector stage as a source.
   b. Set up column definitions on the output link.
4. To set up the JDBC Connector stage as a target stage to write data to the JDBC data source, complete the following steps:
   a. Set up column definitions on the input link.
   b. Configure the JDBC Connector stage as a target.
   c. Optional: Create reject links to manage input data with errors.
5. To set up the JDBC Connector stage to look up data in an JDBC data source, complete the following step:

a. Configure normal lookup operations or configure sparse lookup operations.

Importing JDBC metadata

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

About this task

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

- The host computer that contains the JDBC data source.
- The data source.
- Database schemas.
- Database tables, system tables, and views. All imported tables are stored in the metadata repository as database tables.
- Database columns.

When you import JDBC metadata using the InfoSphere Metadata Asset Manager,

- When you select **Include system objects**, the JDBC connector requests that the JDBC driver include objects that have the JDBC table type attribute set to SYSTEM TABLE. The JDBC driver and the data source determine which objects in the data source meet this criterion.
- When you specify the metadata repository host system and the database objects into which to import the metadata, if you do not specify a value for **Database name**, the connector uses the value that is between the first and second colon in the specified JDBC connection URL. To prevent the connector from importing the metadata under the automatically generated default database object, enter the **Database name** under which you want to import the metadata.

Procedure

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

Related tasks:

"Connecting to a JDBC data source" on page 4
To access JDBC data sources, you must define a connection by using the properties in the **Connection** section on the Properties page.

Related reference:

"Metadata import" on page 36
When you use IBM InfoSphere Metadata Asset Manager to import metadata, you might be able to improve performance by importing the whole schema rather than individual data objects. The import methods that are available depend on the implementation of the JDBC driver and the number of objects to import.

Defining a job that includes a JDBC Connector stage

Before you can read, write, or look up data from a JDBC data source, you must create a job that includes a JDBC Connector stage. Then, you add any additional stages that are required and create the necessary links.

Procedure

1. In the InfoSphere DataStage® and QualityStage® Designer client, select **File** > **New** from the menu.
2. In the **New** window, select the **Parallel Job** icon, and then click **OK**.
3. Add the JDBC Connector stage to the job:
   a. In the palette, select the **Database** category.
   b. Locate **JDBC Connector** in the list of available stage types.
   c. Drag the JDBC Connector stage icon to the canvas.
   d. Optional: Rename the JDBC Connector stage. Choose a name that indicates the role of the stage in the job.
4. Create the necessary links and additional stages for the job:
   - For a job the reads data from a JDBC data source, create the next stage in the job, and then create an output link from the JDBC Connector stage to the next stage.
   - For a job that writes data to a JDBC data source, create one or more links from other stages in the job to the JDBC Connector stage. If the JDBC Connector stage has multiple input links, you can use the Link Ordering page in the stage editor to specify their logical order. If you want to manage rejected records, add a stage to hold the rejected records, and then add a reject link from the JDBC connector to that stage. If the stage has multiple input links, it can have multiple reject links, and each reject link can be associated with an input link.
   - For a job that looks up JDBC data source, create a job that includes a Lookup stage, and then create a reference link from the JDBC Connector stage to the Lookup stage.
5. Save the job.

Connecting to a JDBC data source

To access JDBC data sources, you must define a connection by using the properties in the **Connection** section on the Properties page.

Before you begin

- Create the `isjdbc.config` driver configuration file and specify information about the drivers in this file.
- Define a job that includes a JDBC Connector stage.

Procedure

1. To open the stage editor, on the job design canvas, double-click the **JDBC Connector** stage icon.
2. On the Properties page, specify values for the connection properties. In the **URL** property, specify the URL string for the JDBC data source in the driver-specific format.
3. In the **Username** and **Password** connection properties specify the credentials of the user to authenticate and authorize for the connection. Note that some drivers support including username and password values directly in the URL connection string. For more information, see your driver documentation.
4. In the **Attributes** property specify any additional driver-specific connection properties for your driver. For example, if the driver supports SSL connections, provide the SSL connection properties here. Each driver specific connection property should be entered as a separate line in *property_name=property_value*

format. Note that some drivers support including driver specific connection properties directly in the URL connection string. For more information, see your driver documentation.

5. Click **OK** to save the details.

Related tasks:

"Setting up the driver configuration file" on page 1
Before you can use the JDBC connector, you must set up the driver configuration file. The connector uses this file to obtain information about the available JDBC drivers in your system.

Reading data by using a JDBC Connector stage

You can configure a JDBC Connector stage to connect to a JDBC data source and read data from it.

Before you begin

- Define a job that contains a JDBC Connector stage.
- Define a connection to a JDBC data source.

About this task

The following figure shows an example of using the JDBC Connector stage to read data. In this example, the JDBC Connector stage reads data from a JDBC data source, and the Sequential File stage writes the data to a file. When you configure the JDBC Connector stage to read data, you create an output link, which is shown in the figure below transferring rows from the JDBC Connector stage to the Sequential File stage.

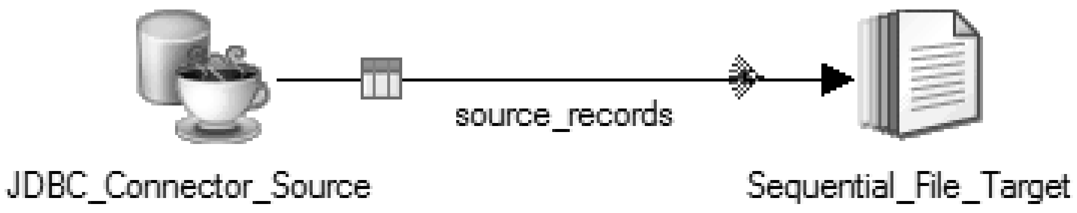


Figure 1. Example of using the JDBC Connector stage to read data from a data source

Configuring the JDBC connector as a source

To configure a JDBC Connector stage to read or look up rows in a JDBC table or view, you must specify the source table or view or define a complete SELECT statement.

Procedure

1. From the job design canvas, double-click the JDBC Connector stage.
2. Click the **Output** tab.
3. Click the **Properties** tab, and, in the **Usage** section specify the settings for the read operation.

4. Use one of these methods to specify the source of the data:
   - Set **Generate SQL at runtime** to **Yes**, and then enter the name of the table or view in the **Table name** property. Use the syntax *table_name* or *owner*.*table_name*, where *owner* is the owner of the table. If you do not specify an owner, the connector uses the schema that belongs to the user who is currently connected.
   - Set **Generate SQL at runtime** to **No**, and then specify the SELECT statement in the **Select statement** property.
   - Set **Generate SQL at runtime** to **No**, and then enter the fully qualified file name of the file that contains the SQL statement in the **Select statement** property. Note that this file must reside on the engine host. If you enter a file name, you must also set **Read select statement from file** to **Yes**.
5. Click **OK**, and then save the job.

Setting up column definitions on a link

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

Procedure

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

Configuring the JDBC connector for partitioned reads

If the JDBC Connector stage is configured to run on multiple processing nodes, each of the processing nodes reads data from the data source concurrently with other processing nodes. The partitions of records from all the processing nodes are combined to produce the complete record data set for the output link of the stage.

Before you begin

- Add the JDBC Connector stage to a parallel job.
- If the connector is configured to look up data, ensure that the **Lookup type** property is set to **Normal**.

About this task

When a JDBC Connector stage is configured to perform partitioned reads, each of the processing nodes of the stage reads a portion of data from the data source and the records retrieved by all the processing nodes are combined to produce the result set for the output link.

Procedure

1. On the job design canvas, double-click the JDBC Connector stage, and then click the **Stage** tab.
2. On the Advanced page, set **Execution mode** to **Parallel** or **Default(Parallel)**, and then click the **Output** tab.
3. Define the SELECT statement that the connector uses at run time:
   - Set **Generate SQL** to **No**, and then specify the SELECT statement in the **Select statement** property or set the **Read select statement from file** to **Yes**, specify the name of the file in **Select statement** property and include the SELECT statement in that file.
4. Set **Enable partitioned reads** to **Yes**. Use the [[node-number]], [[node-number-base-one]] and [[node-count]] placeholders in the SELECT statement to control which portions of data are fetched on individual processing nodes.
5. Click **OK**, and then save the job.

Related reference:

"Partitioned reads" on page 26
The JDBC Connector stage can be configured to run on multiple processing nodes and read data from the data source. In this scenario each of the processing nodes for the stage retrieves a set (partition) of records from the data source. The partitions are then combined to produce the entire result set for the output link of the stage.

Writing data by using the JDBC Connector stage

You can configure a JDBC Connector stage to connect to a JDBC data source and write data to it.

Before you begin

- Define a job that contains a JDBC Connector stage.
- Define a connection to a JDBC data source.

About this task

The following figure shows an example of using the JDBC connector to write data. In this example, the Sequential File stage reads data from a file and then the JDBC Connector stage writes data to the JDBC data source.

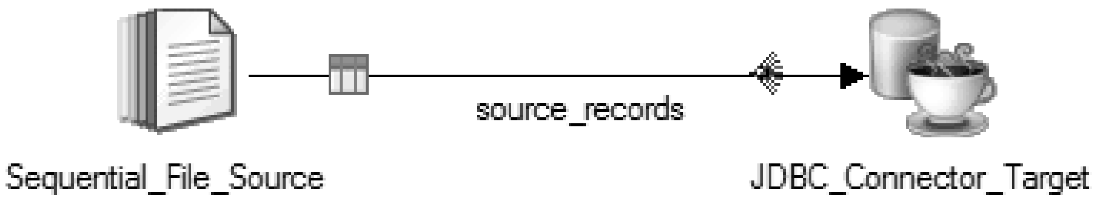


Figure 2. Example of using the JDBC Connector stage to write data to a data source

Related reference:

"Supported write modes" on page 29
When you configure the JDBC connector as a target, you can use the **Write mode** property to specify the mode to use to write rows to the JDBC data source.

Configuring the JDBC connector as a target

To configure a JDBC Connector stage to write rows to a JDBC table, you must specify the target table or view or define the SQL statements.

Procedure

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

Table 1. Methods for writing data to a JDBC table

Method	Procedure
Automatically generate the SQL at run time	1. Set **Generate SQL at runtime** to **Yes**. 2. Set **Write mode** to **Insert**, **Update**, **Delete**, **Insert then update**, **Update then insert**, **Delete then insert**, or **Insert new rows only**. 3. Enter the name of the target table in the **Table name** field. Use the syntax *table_name* or *schema_name.table_name*, where *schema_name* is the owning schema of the table. When *schema_name* is not specified, the connector uses the default schema of the currently connected user.

Table 1. Methods for writing data to a JDBC table (continued)

Method	Procedure
Enter the SQL manually	1. Set **Generate SQL at runtime** to **No**. 2. Set **Write mode** to **Insert**, **Update**, **Delete**, **Insert then update**, **Update then insert**, **Delete then insert**, **Insert new rows only** or **Custom**. 3. Enter SQL statements in the fields that correspond to the write mode that you selected.
Read the SQL statement from a file	1. Set **Generate SQL at runtime** to **No**. 2. Set **Write mode** to **Insert**, **Update**, **Delete**, **Insert then update**, **Update then insert**, **Delete then insert**, **Insert new rows only** or **Custom**. 3. Enter the fully qualified names of the file that contain the SQL statements in the fields that correspond to the write mode that you selected. 4. Set **Read insert statement from file**, **Read update statement from file**, **Read delete statement from file**, or **Read custom statements from file** to **Yes**.

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

Setting up column definitions on a link

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

Procedure

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

c. Select the folder, and then click **Save**.

Rejecting records that contain errors

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

About this task

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

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

Procedure

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

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

9. Optional: From the **Add to reject row** list, select **ERRORCODE**, **ERRORTEXT**, or both. The **ERRORCODE** and **ERRORTEXT** fields in each rejected record are set to the values that depend on the reject condition under which the record was rejected:

Reject condition	ERRORCODE value	ERRORTEXT value
SQL error	Set to the error code provided by the driver.	Set to a text message, which includes information that is provided by the driver about the error.
Row not inserted	1	Set to a text message which indicates that zero rows were inserted for the current record.
Row not updated	2	Set to a text message which indicates that zero rows were inserted for the current record.
Row not deleted	3	Set to a text message which indicates that zero rows were inserted for the current record.

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

Related reference:

"Reject conditions" on page 26
When you use the JDBC connector to write data to a JDBC data source, you can add a reject link to the JDBC Connector stage and send rejected records to the stage attached to the other end of the link. Reject conditions determine when a record is rejected.

"Supported write modes" on page 29
When you configure the JDBC connector as a target, you can use the **Write mode** property to specify the mode to use to write rows to the JDBC data source.

Looking up data in a JDBC data source

You can configure the connector to complete a normal lookup or a sparse lookup on a JDBC data source.

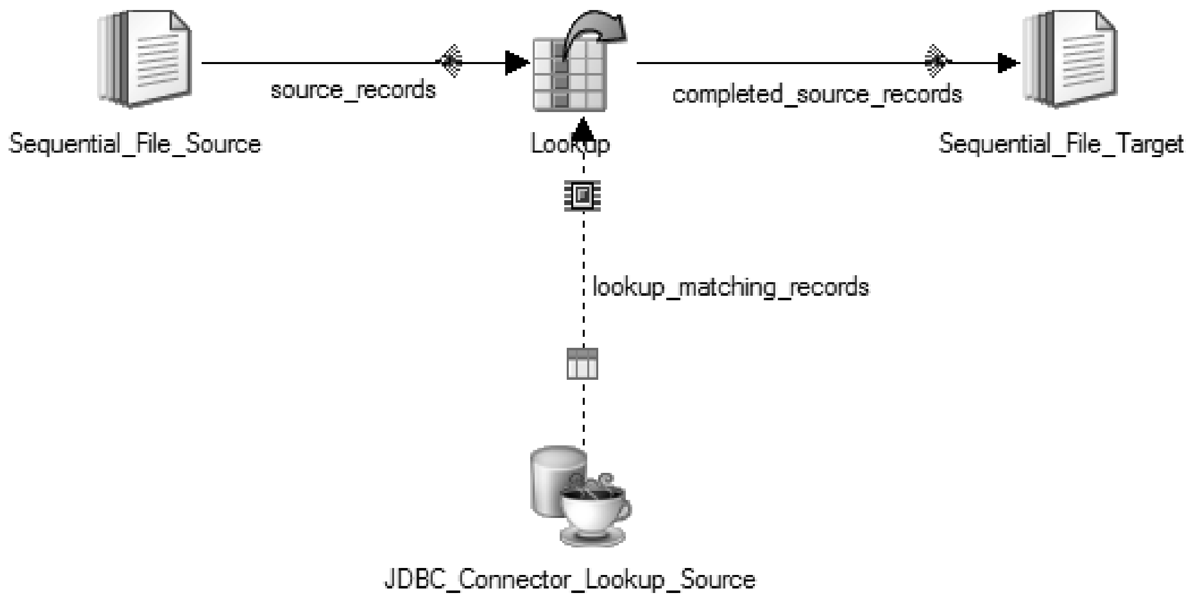
Before you begin

- Define a job that contains the JDBC Connector stage.
- Define a connection to a JDBC data source.

About this task

In the following figure, a Lookup stage extracts data from a JDBC data source based on the input parameter values that the Lookup stage provides. Although the reference link appears to go from the JDBC Connector stage to the Lookup stage, the link transfers data both to and from the JDBC Connector stage. Input parameters are transferred from the input link on the Lookup stage to the reference link, and output values that the JDBC Connector stage provides are transferred from the JDBC Connector 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 3. Example of using the JDBC Connector stage with a Lookup stage.



Configuring normal lookup operations

You can configure a JDBC Connector stage to retrieve a set of records from a JDBC data source and provide them to a Lookup stage in the same job, which then performs a normal (in-memory) lookup operation on those records.

Before you begin

- To specify the format of the records that the JDBC Connector stage reads from a JDBC data source, set up column definitions on a link.
- Configure the JDBC Connector stage as a source for the reference data.

About this task

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

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

Procedure

1. Add a Lookup stage to the job design canvas, and then create a reference link from the JDBC Connector stage to the Lookup stage.
2. Double-click the JDBC Connector stage.
3. From the **Lookup type** list, select **Normal**.

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

Related tasks:

"Setting up column definitions on a link" on page 9
Column definitions, which you set on a link, specify the format of the data records that the connector reads from a database or writes to a database.

"Configuring the JDBC connector as a source" on page 5
To configure a JDBC Connector stage to read or look up rows in a JDBC table or view, you must specify the source table or view or define a complete SELECT statement.

Configuring sparse lookup operations

You can configure a JDBC Connector stage to perform sparse (direct) lookup operation on a JDBC data source.

Before you begin

- To specify the format of the records that the JDBC Connector stage reads from a JDBC data source, set up column definitions on a link.
- Configure the JDBC Connector stage as a source for the reference data.

About this task

In a sparse lookup, the connector runs the specified SELECT statement one time for each parameter set that arrives in the form of a record on the input link to the Lookup stage. The specified input parameters in the statement must have corresponding columns defined on the reference link. Each input record includes a set of parameter values that are represented by key columns. The JDBC Connector stage sets the parameter values on the bind variables in the SELECT statement, and then the JDBC Connector stage runs the statement. The result of the lookup is routed as one or more records through the reference link from the JDBC Connector stage 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 data source.

Typically, you use a sparse lookup when the target table is too large to fit in memory.

Procedure

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

Related tasks:

"Setting up column definitions on a link" on page 6
Column definitions, which you set on a link, specify the format of the data records that the connector reads from a database or writes to a database.

"Configuring the JDBC connector as a source" on page 5
To configure a JDBC Connector stage to read or look up rows in a JDBC table or view, you must specify the source table or view or define a complete SELECT statement.

Reference

The following sections cover in more detail specific functional features and usage scenarios of the JDBC connector.

Properties for the JDBC connector

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

Enable quoted identifiers property

To configure the connector to enclose in quotation marks the object names in the generated SQL statements, set the **Enable quoted identifiers** property to **Yes**.

Usage

When the connector is configured to automatically generate SQL statements, it can be configured to enclose the column, schema, and table object names in the statements with the quotation marks that are supported by the driver. Enclosing the object names in quotation marks in the statements might be required to preserve the case or to allow the use of spaces or special characters in names.

The connector queries the driver to determine the quotation mark that is used by the data source. If the connector fails to obtain this information, it uses a double quote as the quotation mark by default.

Example

Assume that the data source is using a double quote as the quotation mark and that it automatically changes the object names in statements to uppercase when they are not explicitly enclosed in double quotation mark . The following table shows examples of the SELECT statement that is generated for combinations of column names, and table names and **Enable quoted identifiers** property values.

Input link columns	Table name	Value that is set for the Enable quoted identifiers property	Generated statement
c1, c2	t1	No	SELECT c1, c2 FROM t1
c1, c2	t1	Yes	SELECT "c1","c2" FROM "t1"
c1, c2	s1.t1	No	SELECT c1, c2 FROM s1.t1
c1, c2	s1.t1	Yes	SELECT "c1", "c2" FROM "s1"."t1"

Related reference:

"Column derivations" on page 24
When the connector uses a SELECT statement to read rows from the data source, the connector must map the column names that are returned by the SELECT statement to the columns that are defined on the output link.

Properties for transaction management

Use the **Isolation level** property to specify the isolation level for the transactions on the connection that the connector establishes to the data source.

Usage

The default value for this property is **Default**. When this value is selected, the connector does not set any isolation level for the transactions and the default isolation level for the driver is used. For more information about the isolation levels that your driver supports and how it maps them to the isolation levels supported by the backend data source, see the product documentation of your driver.

If a value other than **Default** is selected for **Isolation level** property and the connector fails to set the specified isolation level, the connector stops the job and logs an error message. The error message includes the error information from the driver. If the connector sets a particular isolation level but determines that the driver automatically changed the isolation level, the connector stops the job and logs an error message.

If the driver does not support isolation levels, the **Isolation level** property must be set to **Default**.

The connector also supports enabling **Auto-commit mode** for the connection. When this mode is selected, operations that are performed by the connector on the connection to the data source are committed automatically after they complete.

Record ordering property

Use the **Record ordering** property to specify the processing order of records from multiple input links.

Usage

You can set the **Record ordering** property to the values that are listed in the following table.

Table 2. Values of the **Record ordering** *property*

Value	Description
All records	All records from the first link are processed, then all records from the second link are processed, and so on.
First record	One record from each link is processed until all records from all links are processed.
Ordered	Records are selected from the input links based on the order that you specify for the **Key column**, **Column name**, **Sort order**, **Null order**, and **Case sensitive** properties.

Example

This example describes how changes in **Record ordering** property values affect the order in which the records are processed by the stage.

Assume that the JDBC Connector stage has three input links, Link1, Link2 and Link3. Each of the links has two columns defined for it: C1 of VarChar(20) type and C2 of Date type.

Records arrive on these links in the order that is shown in the following table:

Record number	Link1	Link2	Link3
1	value 2,2001-01-01	NULL,2002-02-01	Value 3,2003-03-01
2	NULL,2001-01-02	value 2,2002-02-02	value 2,2003-03-02
3	Value 1,2001-01-03	Value 3,2002-02-03	Value 2,2003-03-03
4	value 3,2001-01-04		value 4,2003-03-04

Assume the following settings:

- **Record ordering** is set to **Ordered**.
- Column C1 is used as the key column for sorting.
- **Sort order** is set to **Ascending**.
- **Null order** is set to **Before**.
- **Case sensitive** is set to **No**.

In this scenario, the connector sorts the records across the input links based on the sorting options that are specified and the records are processed in the order that is shown in the following table:

Order	Link	Record
1	Link1	NULL,2001-01-02
2	Link2	NULL,2002-02-01
3	Link1	Value1,2001-01-03
4	Link1	value2,2001-01-01
5	Link2	value2,2002-02-02
6	Link3	value2,2003-03-02
7	Link3	Value2,2003-03-03
8	Link1	value3,2001-01-04
9	Link2	Value3,2002-02-03
10	Link3	Value3,2003-03-01
11	Link3	value4,2003-03-04

Table actions property

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

Usage

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

Table 3. Values of the **Table action** *property*

Value	Description
Append	No action is completed on the table. This option is the default.

Table 3. Values of the **Table action** *property (continued)*

Value	Description
Create	Create a table at run time. Use one of these methods to specify the CREATE TABLE statement: • Set **Generate create table statement at runtime** to **Yes** and enter the name of the table to create in the **Table name** property. In this case, the connector automatically generates the CREATE TABLE statement from the column definitions on the input link. The column names in the new table match the column names on the link. The data types of columns in the new table are generated from the column definitions on the link. • Set **Generate create table statement at runtime** to **No**, and enter the CREATE TABLE statement in the **Create table statement** property.
Replace	Replace a table at run time. Use one of these methods to specify the DROP TABLE statement: • Set **Generate drop table statement at runtime** to **Yes**, and enter the name of the table to drop in the **Table name** property. • Set **Generate drop table statement at runtime** to **No**, and enter the DROP TABLE statement in the **Drop table statement** property. Use one of these methods to specify the CREATE TABLE statement: • Set **Generate create table statement at runtime** to **Yes**, and enter the name of the table to create in the **Table name** property. • Set **Generate create table statement at runtime** to **No**, and enter the CREATE TABLE statement in the **Create table statement** property.
Truncate	Truncate a table at run time. Use one of these methods to specify the TRUNCATE TABLE statement: • Set **Generate truncate table statement at runtime** to **Yes**, and enter the name of the table to truncate in the **Table name** property. • Set **Generate truncate table statement at runtime** to **No**, and enter the TRUNCATE TABLE statement in the **Truncate table statement** property.

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

- **Stop the job when create table statement fails**
- **Stop the job when drop table statement fails**
- **Stop the job when truncate table statement fails**

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

Report schema mismatch property

Use the **Report schema mismatch** property to compare the column definitions on the link with the column definitions in the data source.

Usage

Set the **Report schema mismatch** property to **Yes** to compare the column definitions on the link with the column definitions in the data source. The connector logs messages for any data type discrepancies that it detects.

Depending on the driver and the usage scenario, the connector might not be able to detect all discrepancies. The connector performs the schema mismatch reporting on a best-effort basis. Irrespective of the schema mismatch reporting, the connector logs an error message if it detects a data type conversion or data truncation error.

When the connector is configured to write input link records to the data source, the **Report schema mismatch** property is enabled for the input links only if the connector is configured to generate SQL statements automatically. If you specify the SQL statements manually, the connector does not attempt to perform early detection of schema discrepancies and the **Report schema mismatch** property is disabled.

Drop unmatched fields property

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

Usage

When you create a job that writes data from the input link to the data source, you can use the **Drop unmatched fields** property to control how to handle any unused columns (fields) on the input link. Unused columns on the input link are the columns that the connector did not pair with any parameter in the target SQL statement

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

Table 4. Values of the **Drop unmatched fields** *property*

Value	Description
Yes	The connector drops any unused columns on the input link. For each dropped column, the connector writes an informational message in the job log to indicate that the column and its associated values are ignored.

Table 4. Values of the **Drop unmatched fields** *property (continued)*

Value	Description
No	When the connector encounters an unused column on the input link, the connector logs an error message and stops the job.

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

Related reference:

"Enable quoted identifiers property" on page 15
To configure the connector to enclose in quotation marks the object names in the generated SQL statements, set the **Enable quoted identifiers** property to **Yes**.

Character set for non-Unicode columns property

You can use the character set properties to specify the character set that is used for non-Unicode columns on the links attached to the stage.

Usage

Use the **Character set for non-Unicode columns** property to select the character set that is used for values in Char, VarChar and LongVarChar link columns for which the **Extended** property is not set to Unicode.

If you set the property to **Default**, the character set of the engine host system locale is used.

If you set the property to **Custom**, you must provide the character set name to be used. Set the **Character set name** property to the name of the character set encoding for the values of Char, VarChar and LongVarChar link columns for which the **Extended** property is not set to Unicode.

Click the **Select** button in this property to obtain the list of all the available character sets.

Run before and after SQL statements property

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

Usage

Running SQL statements before or after processing data is useful when you need to perform operations that prepare data source 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 statements that you specify are performed once for the whole job, before any data is processed.

After the connector runs the statements that are 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, or DELETE in the **Before SQL statement** property, the results of the DML statement are visible to individual nodes.

To run SQL statements 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 statements once before any data is processed on each node or once after any data is processed on each node. Then, the connector explicitly commits the current transaction.

When you specify the statement to run before or after processing, enter the SQL statements, or enter the fully qualified path to the file that contains the SQL statement. Do not include input bind variables or output bind variables in the SQL statement. If the statement contains these types of variables, the connector logs an error message, and the operation stops. If you specify a file name, the file must reside on the engine tier host, and you must set the **Read Before SQL statement from file**, **Read Before SQL (node) statement from file**, **Read After SQL (node) statement from file** or **Read After SQL statement from file** property to **Yes**.

When the connector runs a set of statements that are specified in any of the **Before SQL statement**, **Before SQL (node) statement**, **After SQL (node) statement**, or **After SQL statement** properties, it reports an error and stops the job if any of the statements in the set fail. You can configure the connector to report a warning message and continue processing the remaining statements in the set. To configure the connector to report a warning message, set the **Stop the job when Before SQL statement fails**, **Stop the job when Before SQL (node) statement fails**, **Stop the job when After SQL (node) statement fails**, or **Stop the job when After SQL statement fails** property to **No**.

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

Run begin and end SQL statements property

Use the **Run begin and end SQL statements** property to run SQL statements in each transaction, immediately after the transaction starts and immediately before the transaction ends successfully.

Usage

Statements that are specified in the **Begin SQL statement** property are run every time that the connector starts a transaction and before any records are processed in that transaction.

Statements that are specified in the **End SQL statement** property are run after all the records are processed in a transaction and immediately before the transaction ends successfully.

Informational messages are written to the job log at the end of the job to indicate the number of times **Begin SQL statement** and **End SQL statement** were completed on individual processing nodes for the stage.

In the JDBC connector, transactions are used to group records on the links of the stage so that they can be processed as a single unit of work. The number of records in each transaction is defined by using the **Record count** connector property, end of wave marker records, or a combination of both. Transactions in the JDBC connector correspond to transactions in the JDBC data source to which the stage is connected when the following conditions are met:

- The data source is a transactional data source.
- In the JDBC connector, the **Auto-commit mode** property is set to **Disable**.
- In the JDBC connector, the **Isolation level** property is set to a value other than **Default**. Or, the value is set to **Default** and the default isolation level in the data source implies the use of transactions in the data source.

The **Begin SQL statement** and **End SQL statement** properties are applicable irrespective of whether or not the transactions in the JDBC connector stage correspond to JDBC data source transactions.

When the job runs, all occurrences of `[[node-number]]`, `[[node-number-base-one]]` and `[[node-count]]` expressions in the **Begin SQL statement** and **End SQL statement** properties are replaced with the zero-based index of the current processing node, one-based index of the current processing node, and total number of processing nodes for the stage, respectively.

For example, assume that the JDBC connector stage is configured to run on three processing nodes and that the **Begin SQL statement** property has the following value:

```
INSERT INTO T1 VALUES ([[node-number]], 'Node [[node-number-base-one]] of
[[node-count]]')
```

When the job runs, the stage runs the following **Begin SQL statement** on the individual processing nodes:

- On the first node: `INSERT INTO T1 VALUES (0, 'Node 1 of 3')`
- On the second node: `INSERT INTO T1 VALUES (1, 'Node 2 of 3')`
- On the third node: `INSERT INTO T1 VALUES (2, 'Node 3 of 3')`

The statements that are specified in the **Begin SQL statement** and **End SQL statement** properties run while the main statement for processing data records is open. For example, when the connector stage has an output link, the SELECT statement that reads records from the data source is open when the statements in the **Begin SQL statement** and **End SQL statement** properties are created and run. As another example, suppose that the connector stage has an input link and is configured to insert records into the data source. The INSERT statement that is used to insert records into the data source is open when the statements in the **Begin SQL statement** and **End SQL statement** properties are created and run.

For some JDBC drivers, only one statement can be open on the connection at a time. If you use the **Begin SQL statement** and **End SQL statement** properties with these drivers, the driver reports an error, and the job fails.

Runtime column propagation

You can configure a JDBC Connector stage to automatically add columns to its output link at run time.

Usage

Before you can enable runtime column propagation in a stage, runtime column propagation must be enabled for parallel jobs at the project level from the IBM InfoSphere DataStage and QualityStage Administrator.

When the runtime column propagation property is enabled for the JDBC Connector stage, the connector inspects the columns for the result set of the statement that it is configured to run. Then the connector compares this set of columns to the set of columns that are defined on the output link. If any columns in the result set are not defined on the output links, the connector adds them to the output link.

To display and enable the **Runtime column propagation** property, you need to first enable the **Enable Runtime Column Propagation for Parallel Jobs** option for the current DataStage project in the DataStage and QualityStage Administrator. Then select the **Runtime column propagation** check box on the Columns page for the output link in the stage editor.

To check whether the link column exists for a particular result set column, the connector performs the following steps:

1. Checks if Derivation properties for any link column matches the result set column name. This matching is case-sensitive. If the Derivation property value starts and ends with the quotation marks, the quoted identifiers are removed before the property is checked. If a matching column is found, the column is assumed to represent the result set column and the connector does not add a column to the link.
2. If a matching column is not found, the connector checks if a Name property for any link column matches the result set column. This matching is case-sensitive only if the **Enable quoted identifiers property** is set to **Yes** for the link. If a matching column is found, the column is assumed to represent the result set column and the connector does not add a column to the link. If a matching column is not found, the connector adds a column to the link. The connector adds a column to the output link that has the same name as the result set column except for the following modifications:
   - The dollar sign ($) is replaced with the __036__ value.
   - The pound sign (#) is replaced with the __035__ value.
   - Every non-alphanumeric character other than underscore sign (_), dollar sign ($) and pound sign (#) is replaced with the pair of underscores __ value.
   - If any replacement with a pair of underscores is performed, or if the first character in the column name is not a letter character, underscore sign (_), dollar sign ($) or pound sign (#), the prefix CC_*n* is appended to the column name, where *n* is the index of the SQL expression column in the SELECT statement list.
   - If the driver reported that the result set column name was an empty string or null, the column index (1, 2, and so on) is used as the column name.
   - White space characters are ignored.
   - If the resulting column name is the same as column name was already added to the link, the suffix *m* is added to it, where *m* is the smallest integer value that is greater than or equal to 1 that results in a unique column name.

In the columns that the connector adds, the Derivation property is set to a value that matches the result set column name.

Example

The following examples illustrate how runtime column propagation works.

Assume that the JDBC Connector stage is configured to fetch data from the data source and provide records on the output link. The examples further assume that the **Runtime column propagation** check box is selected for the output link.

Suppose there are no columns defined for the output link, and the connector is configured to automatically generate SELECT statement to read from table TABLE1. TABLE1 contains the C1, C2 and C3 columns. The connector generates and runs the statement SELECT * FROM TABLE1 and adds the C1, C2, and C3 columns to the output link.

Suppose there are no columns defined for the output link, and the connector is configured to automatically generate a SELECT statement to read from table TABLE1. TABLE1 contains the C$1, C 2 and C!3# columns. The connector generates and runs the statement SELECT * FROM TABLE1 and adds the C__036__1, C2 and CC_3_C__3__035__ columns to the output link. The connector generates the column names in the following way:

C$1 The connector replaces $ with __036__.

C 2 The connector removes the space character between C and 2.

C!3# The connector replaces ! with __ and it replaces # with __035__ and it adds prefix CC_3_ because this column is the third column in the result set.

Column derivations

When the connector uses a SELECT statement to read rows from the data source, the connector must map the column names that are returned by the SELECT statement to the columns that are defined on the output link.

The connector uses the following process to map columns: The connector looks for the output link column that has the Derivation property set to a value that matches the column name that is returned by the SELECT statement. If the connector fails to locate such a column on the link, it looks for the output link column that has the Name property set to a value that matches the column name that is returned by the SELECT statement. If the connector again fails to locate such a column, it adds the column to the output link if runtime column propagation is enabled for the link. Otherwise, it ignores the values in the column that is returned by the SELECT statement when it reads rows from the data source.

When the connector maps columns, it compares the column name that is returned by the SELECT statement and the value of the Derivation property for the output link columns in a case-sensitive fashion, irrespective of the **Enable quoted identifiers** property value. If the Derivation value is surrounded by quotation marks, the quotation marks are removed before performing the comparison. However, the way that it compares the statement column names and output link column Name values is based on the value of the **Enable quoted identifiers** property. If this property is set to **Yes**, the comparison is case-sensitive, and if this property is set to **No**, the comparison is not case-sensitive.

When the connector is configured to automatically generate the SELECT statement at run time, it assembles the statement based on the specified **Table name** property and the column definitions on the output link. If runtime column propagation is enabled for the output link, the connector includes all the table columns in the statement by generating the following statement:

```
SELECT * FROM table_name
```

If runtime columns propagation is not enabled for the output link, the connector includes in the statement only the columns on the output link. For each column on the output link, it includes the value that is set for the Derivation property of the column. If no value is set for the Derivation property, the connector includes the value that is set for the Name property in the statement.

When the connector includes Derivation values in the SELECT statement that is generated, the connector always includes them in the exact same form in which they are specified for the link columns. This includes any quotation marks specified in the Derivation values. On the other hand, when it includes the column Name values in the generated SELECT statement text, if **Enable quoted identifiers** property is set to **Yes**, the connector encloses the values in quotation marks in the statement text. If **Enable quoted identifiers** property is set to **No**, the connector includes the values in the same form that they are specified for the link columns.

Example

Assume that the table T1 has the following columns:

C1 A

C1 B

C2

To read rows from this table, you can define the columns on the output link with the Name property set to C1A, C1B, and C2 and Derivation property set to empty strings. You can then configure the connector to run the following SELECT statement (assume that the data source supports column aliases) :

```
SELECT "C1 A" AS C1A, "C1 B" AS C1B, C2 FROM T1
```

In this case, the columns that are returned by the statement are C1A, C1B and C2. The connector does not find any columns on the output link that have the Derivation property set one of these values. However, because it finds columns that have the Name property set to each of these values, it maps the statement columns to those output link columns successfully.

Alternatively, you can utilize the Derivation property values of the output link columns. You can define the C1A, C1B, and C2 columns on the output link, set the Derivation property of the C1A columns to `C1 A`, set the Derivation property of the C1B columns to `C1 B`, and leave the Derivation property of the C2 column empty. You can then configure the stage to automatically generate the SELECT statement.

The connector uses the Derivation property of the C1A and C1B columns and generates the following SELECT statement:

```
SELECT "C1 A", "C1 B", C2 FROM T1
```

In this case, the columns that are returned by the statement are C1 A, C1 B and C2. The connector maps the statement column C1 A to the output link column C1A because the output link column has the Derivation property set to `C1 A`, which corresponds to the C1 A statement column name. Likewise, the connector maps the statement column C1 B to the output link column C1B because this output link column has the Derivation property set to `C1 B` which again corresponds to the C1 B statement column name. Finally, the connector maps the statement column C2 to the output link column C2.

Related reference:

"Enable quoted identifiers property" on page 15
To configure the connector to enclose in quotation marks the object names in the generated SQL statements, set the **Enable quoted identifiers** property to **Yes**.

"Runtime column propagation" on page 22
You can configure a JDBC Connector stage to automatically add columns to its

output link at run time.

Reject conditions

When you use the JDBC connector to write data to a JDBC data source, you can add a reject link to the JDBC Connector stage and send rejected records to the stage attached to the other end of the link. Reject conditions determine when a record is rejected.

You can set the following reject conditions:

SQL error
: The record driver reports an error when it tries to perform the write operation that is specified by the connector. This condition is applicable to all write modes.

Row not inserted
: The specified INSERT statement was performed on the record, and the statement did not result in an error. However, no rows were inserted in the target table. This condition is applicable to the following write modes: Insert, Insert then update, Update then insert, Delete then insert, Insert new rows only.

Row not updated
: The specified UPDATE statement was performed on the record, and the statement did not result in an error. However, no rows were updated in the target table. This condition is applicable to the following write modes: Update, Insert then update.

Row not deleted
: The specified DELETE statement was performed on the record, and the statement did not result in an error. However, no rows were deleted from the target table. This condition is only applicable to the Delete write mode.

If a reject condition is selected with a write mode for which it is not applicable, the connector ignores the reject condition setting.

Related tasks:

"Rejecting records that contain errors" on page 10
When a JDBC Connector stage includes a reject link, records that meet specified criteria are automatically routed to the target stage on the reject link. Processing continues for the remaining records.

Related reference:

"Supported write modes" on page 29
When you configure the JDBC connector as a target, you can use the **Write mode** property to specify the mode to use to write rows to the JDBC data source.

Partitioned reads

The JDBC Connector stage can be configured to run on multiple processing nodes and read data from the data source. In this scenario each of the processing nodes for the stage retrieves a set (partition) of records from the data source. The partitions are then combined to produce the entire result set for the output link of the stage.

To enable partitioned reads set the **Enable partitioned reads** property to **Yes**.

When the stage is configured for partitioned reads, it runs the statement that is specified in the **Select statement** property on each processing node. You can utilize special placeholders in the statements to ensure that each of the processing nodes retrieves a distinct partition of records from the data source.

The following placeholders can be used in the statements to specify distinct partitions of records to retrieve in individual processing nodes:

- [[node-number]] - replaced at run time with the index of the current processing node. The indexes are zero-based. The placeholder is replaced with the value 0 on the first processing node, value 1 on the second processing node, value 2 on the third processing node, and so forth.
- [[node-number-base-one]] - replaced at run time with the index of the current processing node. The indexes are one-based. The placeholder is replaced with the value 1 on the first processing node, value 2 on the second processing node, value 3 on the third processing node, and so forth.
- [[node-count]] - the total number of processing nodes for the stage. By default this number is the number of nodes in the parallel configuration file. The location of the parallel configuration file is specified in the *APT_CONFIG_FILE* environment variable. The stage can be configured to run on a subset of nodes from the parallel configuration file, by defining node constraints on the Advanced page under the Stage page in the stage editor.

When partitioned reads are enabled and **Read select statement from file** property is set to **Yes**, the connector resolves any [[node-number]], [[node-number-base-one]], and [[node-count]] placeholders in the **Select statement** property value before accessing the file location specified through this property. Once it retrieves the SELECT statement text from the file, it again resolves any [[node-number]], [[node-number-base-one]], and [[node-count]] placeholders in the statement text before executing the statement.

The placeholder support also applies to the **Before SQL (node)** and **After SQL (node)** properties. When partitioned reads are enabled, any [[node-number]], [[node-number-base-one]] and [[node-count]] placeholders in the Before SQL (node) and After SQL (node) statements are resolved on each processing node before running the statements. When the **Read Before SQL (node) from file** or **Read After SQL (node) from file** properties are set to **Yes**, any [[node-number]], [[node-number-base-one]], and [[node-count]] placeholders in the file names that are specified in **Before SQL (node)** and **After SQL (node)** properties are resolved before opening the files. Also, any [[node-number]], [[node-number-base-one]], and [[node-count]] placeholders in the Before SQL (node) and After SQL (node) statements that are retrieved from the files are resolved on each processing node before running the statements on that node.

Ensure that the records returned by statements on individual processing nodes have matching field definitions. For example, if the statement on one of the processing nodes returns records with the columns C1 INTEGER, C2 VARCHAR(20), and C3 DATE, ensure that the statements on all the remaining processing nodes return records with the same column definitions.

Example

Assume that the stage was configured to read records from the data source table TABLE1, which has the following columns: C1 INTEGER, C2 VARCHAR(10) and C3 DATE. Also, assume that the stage was configured to run in parallel on four processing nodes and perform partitioned reads. The **Select statement** property was set to the following value:

```
SELECT C1, C2, C3 FROM TABLE1 WHERE MOD(C1, [[node-count]]) = [[node-number]]
```

The connector runs the following statements on each of the processing nodes:

Table 5.

Node	Statement
1	SELECT C1, C2, C3 FROM TABLE1 WHERE MOD(C1, 4) = 0
2	SELECT C1, C2, C3 FROM TABLE1 WHERE MOD(C1, 4) = 1
3	SELECT C1, C2, C3 FROM TABLE1 WHERE MOD(C1, 4) = 2
4	SELECT C1, C2, C3 FROM TABLE1 WHERE MOD(C1, 4) = 3

Assume that the source table contains the following rows:

Table 6.

C1	C2	C3
1	Value one	2013-01-01
2	Value two	2013-01-02
3	Value three	2013-01-03
4	Value four	2013-01-04
5	Value five	2013-01-05
6	Value six	2013-01-06
7	Value seven	2013-01-07
8	Value eight	2013-01-08
9	Value nine	2013-01-09
10	Value ten	2013-01-10

The following rows are read by the first processing node (node index 0):

Table 7.

C1	C2	C3
4	Value four	2013-01-04
8	Value eight	2013-01-08

The following rows are fetched by the second processing node (node index 1):

Table 8.

C1	C2	C3
1	Value one	2013-01-01
5	Value five	2013-01-05
9	Value nine	2013-01-09

The following rows are fetched by the third processing node (node index 2):

Table 9.

C1	C2	C3
2	Value two	2013-01-02
6	Value six	2013-01-06
10	Value ten	2013-01-10

The following rows are fetched by the fourth processing node (node index 3):

Table 10.

C1	C2	C3
3	Value three	2013-01-03
7	Value seven	2013-01-07

Related tasks:

"Configuring the JDBC connector for partitioned reads" on page 6
If the JDBC Connector stage is configured to run on multiple processing nodes, each of the processing nodes reads data from the data source concurrently with other processing nodes. The partitions of records from all the processing nodes are combined to produce the complete record data set for the output link of the stage.

Supported write modes

When you configure the JDBC connector as a target, you can use the **Write mode** property to specify the mode to use to write rows to the JDBC data source.

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

Table 11. 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.
Delete	The connector attempts to delete rows in the target table that correspond to the records that arrive on the input link.
Insert new rows only	The behaviour of this write mode is very similar to the Insert write mode. However, when this write mode is selected, records that cannot be written to the database because of an integrity constraint violation are ignored, and the connector processes the remaining records. When any error other than an integrity constraint violation occurs, the connector still logs an error and stops the job.
Insert then update	For each input record, the connector first tries to insert the record as a new row in the target table. If the insert operation fails because of an integrity constraint violation, the connector updates the existing row in the target table with the new values from the input record.

Table 11. Write modes and descriptions (continued)

Write mode	Description
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 update operation results in zero updated rows, 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 any 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.
Custom	The connector runs a set of statements on the input records. The set can have zero or more statements that are separated by semicolons.

If the connector generates the INSERT statement at run time, it utilizes all input link columns in the VALUES clause of the statement. If it generates the UPDATE statement at run time, it utilizes all input link key columns in the WHERE clause and all input link non-key columns in the SET clause of the statement. If it generates the DELETE statement at run time, it utilizes all input link key columns in the WHERE clause of the statement.

The connector can run the statements in batch mode. In this mode, the connector runs the statements on a batch of records instead of on individual records.

The connector runs the statements in batch mode only if all of the following conditions are satisfied:
- The **Write mode** property is set to either **Insert**, **Update**, **Delete** or **Custom**.
- No reject links are defined for the stage.
- The **Batch size** property is set to a value that is greater than 1.
- The driver supports the batch mode of operation.

In all other cases, the connector runs the statements on individual records.

Example

The following is an example of a user-defined INSERT statement, that uses the DataStage bind parameter syntax (**`ORCHESTRATE.param_name`**).

```
INSERT INTO TABLE1 VALUES (ORCHESTRATE.C1, ORCHESTRATE.C2)
```

In this case, the connector binds the statement parameters to input link columns by name. The statement parameter **`ORCHESTRATE.C1`** is mapped to input link column C1 and the statement parameter **`ORCHESTRATE.C2`** is mapped to input link column C2.

The following is an example of a user-defined UPDATE statement that mixes the DataStage (**`ORCHESTRATE.param_name`**) and JDBC (question marks) bind parameter syntax.

```
UPDATE TABLE_TEST SET C2 = ORCHESTRATE.C2, C3 = ?
                WHERE C1 = ? + ORCHESTRATE.C4 AND C4 = ?
```

In this case, the connector binds the ORCHESTRATE statement parameters to input link columns by name, and question mark statement parameters by position. In the example above the bind parameters **ORCHESTRATE.C2** and **ORCHESTRATE.C4** are bound to input link columns C2 and C4, respectively, and the first, second and third question marks are bound to the first, second and third input link columns, respectively.

Related tasks:

"Rejecting records that contain errors" on page 10
When a JDBC Connector stage includes a reject link, records that meet specified criteria are automatically routed to the target stage on the reject link. Processing continues for the remaining records.

"Writing data by using the JDBC Connector stage" on page 7
You can configure a JDBC Connector stage to connect to a JDBC data source and write data to it.

Related reference:

"Reject conditions" on page 26
When you use the JDBC connector to write data to a JDBC data source, you can add a reject link to the JDBC Connector stage and send rejected records to the stage attached to the other end of the link. Reject conditions determine when a record is rejected.

Data type mappings

InfoSphere DataStage supports data types that are different from Java and JDBC data types. To complete its operations, the connector must map the data types depending on the scenario in which the connector is used.

Data type mappings from InfoSphere DataStage data types to Java data types

The connector maps the InfoSphere DataStage data types to Java data types when used as a stage in DataStage jobs to read records from the JDBC data source and provide them on the output link, or to read records from the input link and write them to the JDBC data source.

The connector maps the DataStage data types to Java data types under the following scenarios:

- When the connector reads records from the JDBC data source, the connector retrieves the field values from the JDBC driver as Java type values. To determine the Java data type to use for the value that it retrieves from the driver, the connector checks the DataStage data type of the corresponding column on the link.
- When the connector writes records to the JDBC data source, the connector provides the field values to the JDBC driver as Java type values. To determine the Java data type to use for the value that it provides to the driver, the connector checks the DataStage data type of the corresponding column on the link.

The following table shows the mapping rules between InfoSphere DataStage data types and Java data types.

Note: For some InfoSphere DataStage data types the table shows more than one Java data type. In those cases the connector chooses the Java data type which it determines to most closely describe the corresponding JDBC data source column.

Table 12. InfoSphere DataStage data types and their corresponding Java data types

InfoSphere DataStage data types	Java data types
BigInt	long
BigInt (Unsigned)	java.lang.String
Binary	byte[]
Bit	boolean
Char	java.lang.String
Char (Unicode)	java.lang.String
Date	java.sql.Date
Decimal	java.math.BigDecimal
Double	double
Float	float
Integer	int
Integer (Unsigned)	long
LongNVarChar	java.sql.NClob, java.sql.Clob, java.lang.String
LongVarBinary	java.sql.Blob, byte[]
LongVarChar	java.sql.NClob, java.sql.Clob, java.lang.String
LongVarChar (Unicode)	java.sql.NClob, java.sql.Clob, java.lang.String
NChar	java.lang.String
Numeric	java.math.BigDecimal
NVarChar	java.lang.String
Real	float
SmallInt	short
SmallInt (Unsigned)	int
Time	java.sql.Time
Time (Microseconds)	java.sql.Time
Timestamp	java.sql.Timestamp
Timestamp (Microseconds)	java.sql.Timestamp
TinyInt	byte
TinyInt (Unsigned)	short
VarBinary	byte[]
VarChar	java.lang.String
VarChar (Unicode)	java.lang.String
Unknown	java.lang.String

Data type mappings from InfoSphere DataStage data types to native data types

When the connector is configured to create target table and generate the CREATE TABLE statement at run time, it maps the InfoSphere DataStage data types of the input link columns to the native data types of the data source. It then includes the native data types in the column definitions of the generated statement.

To determine the native data type to specify in the CREATE TABLE statement for each input link column, the connector first maps the DataStage data type of the link column to the JDBC data type. It then queries the JDBC driver to determine the native data type that corresponds to that JDBC data type.

The following table shows the mapping rules between InfoSphere DataStage data types and JDBC data types.

Note: For certain InfoSphere DataStage data types the table lists more than one JDBC data type. In those cases the connector queries the JDBC driver for the native data type starting with the first JDBC data type in the list and in case the driver reports that no native data types correspond to it, the connector proceeds to query the driver for the remaining JDBC data types in the list. If the connector reaches the last listed JDBC data type and the driver still reports that no native data type corresponds to it, the connector includes the JDBC data type literal in the statement. In those cases manual modifications of the generated CREATE TABLE statement are likely to be needed.

Table 13. JDBC data types and their corresponding InfoSphere DataStage data types

InfoSphere DataStage data type	JDBC data type
BigInt	BIGINT
BigInt (Unsigned)	BIGINT
Binary	BINARY, VARBINARY
Bit	BIT, BOOLEAN, TINYINT, SMALLINT, INTEGER
Char	CHAR, VARCHAR
Char (Unicode)	NCHAR, NVARCHAR, CHAR
Date	DATE
Decimal	DECIMAL, NUMERIC
Double	DOUBLE, FLOAT
Float	FLOAT, REAL, DOUBLE
Integer	INTEGER
Integer (Unsigned)	INTEGER
LongNVarChar	NCLOB, LONGNVARCHAR, CLOB, LONGVARCHAR, NVARCHAR, VARCHAR
LongVarBinary	BLOB, LONGVARBINARY, VARBINARY
LongVarChar	CLOB, LONGVARCHAR, VARCHAR
LongVarChar (Unicode)	NCLOB, LONGNVARCHAR, CLOB, LONGVARCHAR, NVARCHAR, VARCHAR
NChar	NCHAR, NVARCHAR, CHAR
Numeric	NUMERIC, DECIMAL
NVarChar	NVARCHAR, NCHAR, VARCHAR
Real	REAL, FLOAT, DOUBLE
SmallInt	SMALLINT, INTEGER
SmallInt (Unsigned)	SMALLINT, INTEGER
Time	TIME
Time (Microseconds)	TIME
Timestamp	TIMESTAMP

Table 13. JDBC data types and their corresponding InfoSphere DataStage data types (continued)

InfoSphere DataStage data type	JDBC data type
Timestamp (Microseconds)	TIMESTAMP
TinyInt	TINYINT, SMALLINT, INTEGER
TinyInt (Unsigned)	TINYINT, SMALLINT, INTEGER
VarBinary	VARBINARY, BINARY
VarChar	VARCHAR, CHAR
VarChar (Unicode)	NVARCHAR, NCHAR, VARCHAR
Unknown	NCLOB, LONGNVARCHAR, CLOB, LONGVARCHAR, NVARCHAR, VARCHAR

Data type mappings from JDBC data types to InfoSphere DataStage data types

In an output link where DataStage columns are set from the JDBC data types, the Java Connector stage maps the JDBC data types to the InfoSphere DataStage data types.

The connector maps the JDBC data types to InfoSphere DataStage data types under the following scenarios:

- The connector has an output link, and the **Runtime column propagation** setting is selected for the link. The connector adds column definitions to the link at run time based on the column definitions in the JDBC data source. For each column in the data source that is included by the SELECT statement that is configured for the link, the connector maps the JDBC data type to the InfoSphere DataStage data type.
- The connector reads, writes or looks up data in the data source, and the **Report schema mismatch** property is set to **Yes**. The connector tries to determine whether the column definitions in the data source are compatible with the column definitions on the link. If the connector detects a mismatch, it logs a warning message.
- In InfoSphere Metadata Asset Manager (IMAM), the connector imports table definitions from the JDBC data source. When the connector obtains a table definition from the JDBC data source, the columns in that table definition are of JDBC data types, as reported by the JDBC driver. Before the connector can save this table definition to the metadata repository, the connector must map the JDBC column data types to DataStage column data types.

The following table shows the mapping rules between JDBC data types and InfoSphere DataStage data types.

Table 14. JDBC data types and their corresponding InfoSphere DataStage data types

JDBC data type	InfoSphere DataStage data type
ARRAY	VarChar
BIGINT	BigInt
BINARY	Binary
BIT	Bit
BLOB	LongVarBinary
BOOLEAN	Bit

Table 14. JDBC data types and their corresponding InfoSphere DataStage data types (continued)

JDBC data type	InfoSphere DataStage data type
CHAR	Char
CLOB	LongVarChar
DATALINK	VarChar
DATE	Date
DECIMAL	Decimal
DISTINCT	VarChar
DOUBLE	Double
FLOAT	Double
INTEGER	Integer
JAVA_OBJECT	VarChar
LONGNVARCHAR	LongNVarChar
LONGVARBINARY	LongVarBinary
LONGVARCHAR	LongVarChar
NCHAR	NChar
NCLOB	LongNVarChar
NULL	VarChar
NUMERIC	Numeric
NVARCHAR	NVarChar
OTHER	VarChar
REAL	Real
REF	VarChar
ROWID	VarChar
SMALLINT	SmallInt
SQLXML	VarChar
STRUCT	VarChar
TIME	Time
TIMESTAMP	Timestamp (Microseconds)
TINYINT	TinyInt
VARBINARY	VarBinary
VARCHAR	VarChar
any other data type	VarChar

When **Generate all columns as Unicode** property is set to **Yes**, the Char, VarChar and LongVarChar values in the second column as mentioned in the table change to NChar, NVarChar and LongNVarChar, respectively.

Related tasks:

"Importing JDBC metadata" on page 3
Before you design jobs that use the JDBC connector to read, write, or look up data you can use the InfoSphere Metadata Asset Manager to import the metadata that represents tables and views in the JDBC data source. The imported metadata is then saved in the metadata repository.

Related reference:

"Runtime column propagation" on page 22
You can configure a JDBC Connector stage to automatically add columns to its output link at run time.

"Report schema mismatch property" on page 19
Use the **Report schema mismatch** property to compare the column definitions on the link with the column definitions in the data source.

Performance optimization

You can configure various settings in the JDBC Connector stage that might optimize its performance.

Driver settings

Familiarize yourself with the requirements, capabilities, and configuration options for your driver and tune the settings to improve performance.

Some JDBC drivers support specifying custom connection properties within the URL connection string, which you specify in the **URL** connector property.

Some drivers support specifying custom connection properties, by listing them in `name=value` format in the **Attributes** property. If options are specified in this way, each option must be specified on a separate line.

The driver might also require that the Java Virtual Machine (JVM) is configured with a particular set of option to provide performance benefits. You can specify custom JVM options to be used in InfoSphere DataStage jobs when the JVM is initialized in each processing node for the stage. The **Java settings** property is used for this purpose.

Metadata import

When you use IBM InfoSphere Metadata Asset Manager to import metadata, you might be able to improve performance by importing the whole schema rather than individual data objects. The import methods that are available depend on the implementation of the JDBC driver and the number of objects to import.

When you use IBM InfoSphere Metadata Asset Manager to import metadata, you can import all the data objects that are contained in the schema or select individual data objects. When the JDBC connector imports individual data objects within a schema, it requests the metadata one object at a time. This method is optimal when you want to import a small subset of the overall schema.

However, if you select a large subset of the schema, the performance is enhanced if you import the complete schema.

For example, if a schema contains 50 tables and you select to import 30 tables in the schema, it might be faster to import the whole schema. If you import 15 tables, it might be faster to import only the 15 tables rather than the whole schema.

Related tasks:

"Importing JDBC metadata" on page 3
Before you design jobs that use the JDBC connector to read, write, or look up data you can use the InfoSphere Metadata Asset Manager to import the metadata that represents tables and views in the JDBC data source. The imported metadata is then saved in the metadata repository.

Length property values

Ensuring that all columns on the link have their **Length** property set to a concrete value might improve performance.

For Char, VarChar, NChar, NVarChar, Binary and VarBinary link columns that do not have their **Length** property set, the connector uses the value specified in the **Default length for columns** property as the column length. For LongVarChar, LongNVarChar and LongVarBnary link columns that do not have their **Length** property set, the connector uses the value specified in the **Default length for long columns** property as the column length.

To improve performance, specify a value for the **Length** property for all columns. Specify the smallest value that is still sufficiently large to accommodate the actual data represented by the column. When a value is specified for the **Length** property, the connector optimally allocates resources such as memory storage for exchanging data values with the JDBC driver and with the InfoSphere DataStage framework.

Character set conversions

Eliminating character set conversions in the stage can improve performance.

When the links to the JDBC Connector stage have Char, VarChar and LongVarChar columns for which the **Extended** property is not set to Unicode, the connector performs character set conversion. When the connector passes the values to the JDBC driver, the connector converts them from the character set encoding that is specified in the **Character set for non-Unicode columns** property to the UTF-16 character set encoding.

To prevent the connector from performing the character set conversions on the character data, use NChar, NVarChar, and LongNVarChar columns on the links instead of Char, VarChar, and LongVarChar columns. Alternatively, you can leave the Char, VarChar, and LongVarChar columns on the links but set the **Extended** property to Unicode. The connector then exchanges the values with the driver as UTF-16 values.

Other stages in the job might require Char, VarChar and LongVarChar columns on their links, with the **Extended** property for those columns that are not set to Unicode. In this case you can still use the NChar, NVarChar and LongNVarChar columns on the links of the JDBC Connector stage and perform mapping to or from Char, VarChar and LongVarChar columns in a stage located downstream or upstream of the JDBC Connector stage.

Compare the performance when converting character sets in a stage other than the JDBC Connector stage with the performance when converting character sets in the JDBC Connector stage, and, choose the option that better suits your environment.

Fetch size values

Tuning the **Fetch size** property value can improve the performance when reading records from the data source.

The connector provides the value of the **Fetch size** property to the driver. This value is a hint to the driver how many records to retrieve from the data source in a single round trip to the data source. The connector always requests one record from the driver at a time, and the driver might use the prefetching option to be able to provide some of the records to the connector locally.

The default value for the **Fetch size** property is **0**. The default value results in optimal performance in many cases. You can also set the **Fetch size** property to a specific value and check the results to see whether they are better than the results with the default fetch size.

Changing the value of the **Fetch size** property can improve performance when the stage reads many records such as hundreds of thousands of records or more. Run the job with the fetch size values of 20, 200, 2000 and 20000 and observe the performance trends to determine the optimal value. If none of the fetch size values improve the performance, use the default value.

Lock waits

The choice of the partitioning type for the records on input link can affect the performance of the connector.

When the stage is configured to write records to the data source in parallel on multiple processing nodes, you can configure the partitioning of records on the input link to reduce or prevent lock waits in concurrent write operations.

For example, if you configured the stage to run in Update then insert write mode, the connector first tries to run the UPDATE statement on each input record. In this example, the WHERE clause references the primary key columns in the target table. If that statement results in zero updates, the connector runs the INSERT statement for the record. If the input data set contains multiple records with the same values for the key columns, the UPDATE statement is run for each of the records. If two processing nodes attempt this operation on two different records with the same key values, in the transactions that are active at the same time, the processing node that first ran the statement acquires the lock on the respective row in the data source. The processing node holds the lock until its transaction completes. The other processing node must wait for the lock to be released.

To ensure that all input records, that have the same key field values are processed by the same processing node, configure the **Partition type** setting for the link so that those records are routed to the same processing node. The **Partition type** is specified on the Partitioning page for the link in the stage editor. For example, you can set the **Partition type** property to **Hash**. For the hashing key columns, choose the input link key columns that are referenced by the WHERE clause. Partitioning records on the input link prevents lock waits and ensures that the UPDATE operation is run on the records with the matching key values in the same order in which they are arriving on the input link.

Multiple links

Multiple input links are a convenient mechanism for reducing the numbers of JVM instances and data source connections but at the same time their use can negatively affect the performance of the connector.

The connector supports multiple input links. Each of the links can be configured to run different operations on the data source. The order in which the records are processed on the links is controlled by the **Record ordering** property. The processing of records for all of the input links takes place in a single JVM instance and all the input links share a connection to the data source.

When the stage is configured to run in parallel mode, one JVM instance and one connection to the data source are created in each processing node. The downside of the use of multiple input links is that it might result in increased consumption of the processor cycles by the connector.

You must weigh this negative effect against the benefit of sharing JVM instance and data source connection between the links, and choose the configuration that better meets your requirements.

Reject links

Reject links are a convenient mechanism for handling input link records in error but at the same time their use can negatively affect the performance of the connector.

Reject links handle input link records that caused an error in the write operation. When reject links are used, instead of stopping the job, the stage can be configured to send the records in error to the reject links and continue processing input records. After the job is completed, you can check the rejected records to determine further actions.

However, reject links can affect the performance of the connector negatively. When the stage with reject links does not meet your performance requirements, consider replacing the reject links with an alternative mechanism to handle the input records in error. For example, if reject links were used to handle records that violated NULL or CHECK constraints of the target table, check the records for those violations and handle them in the processing stages upstream from the connector stage.

Alternatively, check for any mechanisms provided by the backend data source that allow for handling of records in error on the data source side. For example, you might be able to instruct the data source to redirect the records to a dedicated error table.

Transaction commits

Reducing the frequency of transaction commits when reading records from the data source can improve performance.

When the connector is configured to read records from the data source and send the records to the output link, it commits the transaction after every *N* record, where *N* is the value that is specified in the **Record count** property.

If the statement performed by the stage does not result in changes in the data source and the records do not need to be sent on the output link in waves, consider setting the **Record count** property to **0**. The value 0 for this property indicates that the transaction is committed only a single time after all the records are read from the data source.

Batch processing

Using batch processing when the connector writes records to the data source can improve performance.

The connector supports the batch mode of operation when it writes records to the data source. In this type of operation, the connector provides multiple records to the JDBC driver to be included in a batch and then requests the driver to submit the batch to the data source. This process can improve performance. The size for the batches is specified through the **Batch size** property.

When the **Record count** property is set to **0**, the transaction is committed after all the records are written to the data source. Setting the **Record count** property to **0** can result in long-running transactions, depending on the number of input records. But setting the **Record count** property to a small value results in frequent transaction commits.

If you reduce the batch size to a small value, the benefits that are provided by batch processing might be reduced or lost. Note that when the record count value is not `0`, the connector automatically adjusts the specified batch size value if necessary to ensure that the record count value is a multiple of it.

Write modes

The Insert then update and Update then insert write modes might result in different performance for the same input records.

You can improve performance by minimizing the total number of statements that are executed for each input record. To reduce the total number of statements that are executed on each input record during the job execution, choose the write mode based on your knowledge on the input data.

For an input record when executing the Insert then update write mode, the connector first attempts to execute the insert statement. If that fails the connector executes the update statement. When executing the Update then insert write mode, the connector first attempts to execute the update statement. If that fails the connector executes the insert statement.

You can improve performance by choosing the write mode based on your input data.

If most of the input records are expected to be inserted, and only a small percentage of them will be updated, then the Insert then update mode can have better performance.

If most of the input records are already in the data source and must be updated, and only a small percentage of the input records will be inserted, then the Update then insert write mode can have better performance.

Troubleshooting the JDBC Connector stage

Several common errors are specific to the JDBC Connector stage.

The configuration file cannot be accessed

The `isjdbc.config` file is configured, but an error indicates that the configuration file cannot be accessed.

Symptoms

An error indicates that the configuration file cannot be accessed.

Resolving the problem

Complete the following steps:

- Ensure that the configuration file is saved in the `DSEngine` directory of the InfoSphere Information Server engine tier installation.
  - On Windows operating systems, the default location for the `DSEngine` directory is `C:\IBM\InformationServer\Server\DSEngine`.
  - On Linux and UNIX operating systems, the default location is `/opt/IBM/InformationServer/Server/DSEngine`.

- Ensure that the name of the file is `isjdbc.config`, with all lowercase characters and no leading and trailing white space characters.
- Ensure that the file is saved on the engine tier computer.
- Ensure that when the engine tier spans multiple hosts, the file is accessible from the same location on all of the hosts. You can make a copy of the file on the same `DSEngine` location on each of the hosts, or use a shared `DSEngine` network location that is accessible by all of the hosts.
- Ensure that the file has the read access for the InfoSphere DataStage user and the local Administrator user (Windows) or system root user (Linux and UNIX). On Linux and UNIX, check the permissions of the `.odbc.ini` file that is stored at the same location and set the same permissions on the `isjdbc.config` file.
- Ensure that the content of the configuration file is in the correct format.

  For more information about the format, see "Setting up the driver configuration file" on page 1 topic.

CREATE TABLE failed to run successfully

The connector is configured to create the target table automatically, but the CREATE TABLE that was generated failed to run successfully.

Symptoms

Depending on the column definitions on the input link, the driver, and the data source, the connector might not be able to generate a valid CREATE TABLE statement.

Causes

To generate the CREATE TABLE statement, the connector first maps each column definition on the input link from the InfoSphere DataStage data type to the closest matching JDBC data type. Then, the connector queries the driver to determine the data source native type that corresponds to that JDBC data type.

Depending on the input link data type, the connector might try several different JDBC data types. However in some cases the driver does not report a native data type for any of the JDBC data types that the connector presented. In those cases the connector uses the first JDBC data type in the CREATE TABLE statement. This in turn can cause the statement to fail to run. The connector writes the complete generated CREATE TABLE statement text in the job log.

Resolving the problem

To correct the error, complete one or more of the following steps:

- Use different data types for the input link columns. Ensure that the input link column definitions are valid for the actual data values that are represented by those columns when the job runs.
- Enter the CREATE TABLE statement manually. You can start with the generated CREATE TABLE statement by copying it from the job log to the **Create table statement** property in the stage editor.

  For more information, see the documentation for the data source where you want to create the table. Locate the native data types to use for the required columns in the statement, and then specify those data types in the statement directly.

  Make sure that the native data types that you specify for the table columns are valid for the actual data values that are written to those columns when the job runs.

For more information about mapping, see the following topic: "Data type mappings from InfoSphere DataStage data types to native data types" on page 32

The job fails and reports data truncation or character set conversion errors

When the connector reads values from or write values to a character-based column in the data source by using a character-based column on the link, the job fails and reports data truncation or character set conversion errors, even if the Length property of the column on the link matches the length of the column in the data source.

Symptoms

The job fails and reports data truncation or character set conversion errors.

Causes

For the Char, VarChar, LongVarChar, NChar, NVarChar, and LongNVarChar columns on the link, the connector exchanges values with the JDBC driver as double-byte Java™ Unicode values.

For the NChar, NVarChar and LongNVarChar columns, and, the Char, VarChar, and LongVarChar columns for which the **Extended** property is set to Unicode, the connector exchanges values with the InfoSphere DataStage framework as double-byte Unicode values.

For the Char, VarChar and LongVarChar columns for which the **Extended** property is not set to Unicode, the connector exchanges values with the InfoSphere DataStage framework based on the character set encoding specified for the stage in the **Character set for non-Unicode columns** property. When this property is set to value **Default**, the default system locale character set encoding is used. When this property is set to value **Custom**, the character set encoding to use is specified in the **Character set name** property.

The Length property for columns on the link is measured in double-byte units for NChar, NVarChar, and LongNVarChar columns and Char, VarChar, and LongVarChar columns for which **Extended** property is set to Unicode and is measured in bytes for Char, VarChar, and LongVarChar columns for which **Extended** property is not set to Unicode. These differences might lead in incompatible column definitions even when the same Length is specified for the column on the link and the column in the data source.

For example, a column in the data source can be defined as VARCHAR(10) column and be able to store 10 double-byte Unicode values. Assume that the corresponding column on the link is defined as VarChar(10) column and **Extended** property is not set to Unicode. When the connector fetches the value from this column in the data source, it fetches it as a Java Unicode string that consists of 10 double-byte Unicode characters.

Assume that the connector must convert this value to UTF-8 encoding, either because **Character set for non-Unicode columns** property was set to Default and UTF-8 is the character set encoding in the current system locale, or because **Character set for non-Unicode columns** property was set to value **Custom** and UTF-8 character set encoding was specified in the **Character set name** property. Assume further that some of the double-byte characters in the fetched value result in 2, 3 or 4-byte UTF-8 characters. The converted value in that case can require anywhere between 10 - 40 bytes of storage. If it requires more than 10 bytes of storage, the connector detects that the Length of the link column is not sufficient and reports an error.

Resolving the problem

To correct the error as mentioned in the previous example, complete one of the following actions:

- Change the data type of the column definition on the link to NVarChar(10).
- Set the **Extended** property for the column to Unicode.
- Increase the Length in the column definition to 40 to accommodate all the values.

For more information, see “Data type mappings” on page 31 and “Character set for non-Unicode columns property” on page 20.

The job fails and unused columns were detected

When the connector detects a column on the link that is not used by the statements that the connector runs for the records on that link, the connector reports an error for the detected unmatched column.

Symptoms

The connector reports an error for the unmatched column on the link and the job fails.

Resolving the problem

The resolution of this error depends on whether the connector is configured to read data from a JDBC data source or write data to a JDBC data source

When the connector is configured to read rows from the data source, ensure that the SELECT statement that the connector uses to read the rows includes a column in its select list that corresponds to the column on the link. Alternatively, remove the unmatched column from the link.

The following table includes examples of scenarios and respective problems that you might encounter when the connector is configured to read data from a JDBC data source and the resolution for the problems.

SELECT statement	Output link columns	Problem	Resolutions
`SELECT C1, C3 FROM TABLE1`	C1, C2 and C3	The link column C2 is treated as an unmatched column and results in error	Complete one of the following steps: • Change the statement to `SELECT C1, C2, C3 FROM TABLE1` • Remove column C2 from the output link.

SELECT statement	Output link columns	Problem	Resolutions
The SELECT statement is generated automatically and the source table TABLE1 contains columns C1, C3 and C4	C1, C2 and C3	The connector does not include the table column C4 in the generated statement and does not fetch data for it because it determines that a corresponding column is not defined on the output link. But the connector includes column C2 in the generated statement because it is present on the output link and that leads to an error because the table does not have the column with that same name.	Complete one of the following steps: • Add column C2 to the table. • Remove column C2 from the output link.

When the connector is configured to write records from the input link to the data source, each column on the input link must be referenced by one or more SQL statements that the stage uses to write records to the data source. If a column on the input link is not referenced the behavior depends on the value of the **Drop unmatched fields** property. When this property is set to **No**, the connector reports an error, and the job fails. When the property is set to **Yes**, the connector reports an informational message for each unused input link column to indicate that the column is ignored, and the job continues.

The following table includes examples of scenarios and respective problems that you might encounter when the connector is configured to write data to a JDBC data source and the resolution for the problems.

Write modes	Input link columns	Problem	Resolutions
The connector is configured to run in Insert write mode and to run the statement `INSERT INTO TABLE1(C1, C2, C3) VALUES ( ORCHESTRATE.C1, 'job 1', ORCHESTRATE.C3)`	C1, C2 and C3	The connector determines that the input link column C2 is not referenced by the INSERT statement and acts based on the **Drop unmatched fields** property value.	Complete one of the following steps: • Change the INSERT statement to `INSERT INTO TABLE1(C1, C2, C3) VALUES (ORCHESTRATE.C1, ORCHESTRATE.C2, ORCHESTRATE.C3)` • Remove column C2 from the input link.

Write modes	**Input link columns**	**Problem**	**Resolutions**
The connector is configured in run in Insert write mode and automatically generate the INSERT statement to use to insert records into the table TABLE1 with columns C1, C3 and C4	C1, C2 and C4	The connector does not include the column C4 in the generated INSERT statement because a corresponding column is not on the input link. Because column C2 is on the input link and does not have a matching column in the target table, the connector treats this input link column as an unmatched column.	Complete one of the following steps: • Add column C2 to the table. • Remove column C2 from the input link.

The connector failed to detect and report all schema mismatches

The connector is configured to report mismatches between the column definitions on the link and the column definitions in the JDBC data source. However, the connector failed to detect and report all of the mismatches.

Symptoms

The connector tries to determine the JDBC data types of the columns in the data source that correspond to the columns on the link. If the connector determines that the data types are incompatible, it reports the schema mismatch warning.

The driver, the data source, and the usage scenario for the stage might prevent checks for data type mismatches from happening.

Causes

The connector checks for a schema mismatch on a best-effort basis. It might not always be able to detect all the discrepancies.

Resolving the problem

This feature must be used only as an aid for configuring the column definitions on the link. Schema mismatch checking happens during the job setup phase. Irrespective of the schema mismatch checking, the connector checks for more schema mismatches during the job processing phase. Extra checking happens when the connector provides values to the JDBC driver, or receives values from the JDBC driver. If then the connector detects data truncation or character set conversion issues, the connector reports the error and the job fails.

Suitable driver not found when you use the JDBC Connector stage

Errors occur when the driver class path is not correctly specified in the JDBC configuration file.

Symptoms

An error occurs indicating that no suitable driver is found. `The connector failed to locate a suitable driver for the specified URL value.`

Resolving the problem

Ensure that the information about the driver class path and the name of the Driver class is included in the `isjdbc.config` configuration file as follows:

- The *CLASSPATH* entry in the configuration file must include the fully qualified class path of the driver. If the driver consists of more than one JAR file or depends on more JAR files, all the JAR file locations must be included.

  If the driver is implemented in one or more JAR files, ensure that the specified JAR files are present in the system. Also, ensure that the JAR files have the read and run permissions that are enabled for the InfoSphere DataStage user and the local Administrator user on Windows or system root user on Linux and UNIX.
- If you are not sure about the JDBC version of your driver, check for the presence of the file `META-INF\services\java.sql.Driver` in the driver JAR archive. If this file is present and contains a single value, which indicates the Driver JDBC class name, you do not need to specify this class name in the *CLASS_NAMES* entry.

  The *CLASS_NAMES* entry in the configuration file must include the fully qualified name of the Java class in the driver that implements the `java.sql.Driver` JDBC interface. This information is only required if the driver is based on the JDBC API version before version 4.0. For the name of this class, see your driver documentation.

Product accessibility

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

The IBM InfoSphere Information Server product modules and user interfaces are not fully accessible.

For information about the accessibility status of IBM products, see the IBM product accessibility information at http://www.ibm.com/able/product_accessibility/index.html.

Accessible documentation

Accessible documentation for InfoSphere Information Server products is provided in an information center. The information center presents the documentation in XHTML 1.0 format, which is viewable in most web browsers. Because the information center uses XHTML, you can set display preferences in your browser. This also allows you to use screen readers and other assistive technologies to access the documentation.

The documentation that is in the information center is also provided in PDF files, which are not fully accessible.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.

Accessing the product documentation

Documentation is provided in a variety of formats: in the online IBM Knowledge Center, in an optional locally installed information center, and as PDF books. You can access the online or locally installed help directly from the product client interfaces.

IBM Knowledge Center is the best place to find the most up-to-date information for InfoSphere Information Server. IBM Knowledge Center contains help for most of the product interfaces, as well as complete documentation for all the product modules in the suite. You can open IBM Knowledge Center from the installed product or from a web browser.

Accessing IBM Knowledge Center

There are various ways to access the online documentation:

- Click the **Help** link in the upper right of the client interface.
- Press the F1 key. The F1 key typically opens the topic that describes the current context of the client interface.

  Note: The F1 key does not work in web clients.
- Type the address in a web browser, for example, when you are not logged in to the product.

  Enter the following address to access all versions of InfoSphere Information Server documentation:

  `http://www.ibm.com/support/knowledgecenter/SSZJPZ/`

  If you want to access a particular topic, specify the version number with the product identifier, the documentation plug-in name, and the topic path in the URL. For example, the URL for the 11.3 version of this topic is as follows. (The ⇒ symbol indicates a line continuation):

  ```
  http://www.ibm.com/support/knowledgecenter/SSZJPZ_11.3.0/⇒
  com.ibm.swg.im.iis.common.doc/common/accessingiidoc.html
  ```

  Tip:

  The knowledge center has a short URL as well:

  `http://ibm.biz/knowctr`

  To specify a short URL to a specific product page, version, or topic, use a hash character (#) between the short URL and the product identifier. For example, the short URL to all the InfoSphere Information Server documentation is the following URL:

  `http://ibm.biz/knowctr#SSZJPZ/`

  And, the short URL to the topic above to create a slightly shorter URL is the following URL (The ⇒ symbol indicates a line continuation):

  ```
  http://ibm.biz/knowctr#SSZJPZ_11.3.0/com.ibm.swg.im.iis.common.doc/⇒
  common/accessingiidoc.html
  ```

Changing help links to refer to locally installed documentation

IBM Knowledge Center contains the most up-to-date version of the documentation. However, you can install a local version of the documentation as an information center and configure your help links to point to it. A local information center is useful if your enterprise does not provide access to the internet.

Use the installation instructions that come with the information center installation package to install it on the computer of your choice. After you install and start the information center, you can use the **iisAdmin** command on the services tier computer to change the documentation location that the product F1 and help links refer to. (The ⇒ symbol indicates a line continuation):

Windows

```
IS_install_path\ASBServer\bin\iisAdmin.bat -set -key ⇒
com.ibm.iis.infocenter.url -value http://<host>:<port>/help/topic/
```

AIX® Linux

```
IS_install_path/ASBServer/bin/iisAdmin.sh -set -key ⇒
com.ibm.iis.infocenter.url -value http://<host>:<port>/help/topic/
```

Where `<host>` is the name of the computer where the information center is installed and `<port>` is the port number for the information center. The default port number is 8888. For example, on a computer named `server1.example.com` that uses the default port, the URL value would be `http://server1.example.com:8888/help/topic/`.

Obtaining PDF and hardcopy documentation

- The PDF file books are available online and can be accessed from this support document: https://www.ibm.com/support/docview.wss?uid=swg27008803&wv=1.
- You can also order IBM publications in hardcopy format online or through your local IBM representative. To order publications online, go to the IBM Publications Center at http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss.

Providing feedback on the product documentation

You can provide helpful feedback regarding IBM documentation.

Your feedback helps IBM to provide quality information. You can use any of the following methods to provide comments:

- To provide a comment about a topic in IBM Knowledge Center that is hosted on the IBM website, sign in and add a comment by clicking **Add Comment** button at the bottom of the topic. Comments submitted this way are viewable by the public.
- To send a comment about the topic in IBM Knowledge Center to IBM that is not viewable by anyone else, sign in and click the **Feedback** link at the bottom of IBM Knowledge Center.
- Send your comments by using the online readers' comment form at www.ibm.com/software/awdtools/rcf/.
- Send your comments by e-mail to comments@us.ibm.com. Include the name of the product, the version number of the product, and the name and part number of the information (if applicable). If you are commenting on specific text, include the location of the text (for example, a title, a table number, or a page number).

Links to non-IBM Web sites

This information center may provide links or references to non-IBM Web sites and resources.

IBM makes no representations, warranties, or other commitments whatsoever about any non-IBM Web sites or third-party resources (including any Lenovo Web site) that may be referenced, accessible from, or linked to any IBM site. A link to a non-IBM Web site does not mean that IBM endorses the content or use of such Web site or its owner. In addition, IBM is not a party to or responsible for any transactions you may enter into with third parties, even if you learn of such parties (or use a link to such parties) from an IBM site. Accordingly, you acknowledge and agree that IBM is not responsible for the availability of such external sites or resources, and is not responsible or liable for any content, services, products or other materials on or available from those sites or resources.

When you access a non-IBM Web site, even one that may contain the IBM-logo, please understand that it is independent from IBM, and that IBM does not control the content on that Web site. It is up to you to take precautions to protect yourself from viruses, worms, trojan horses, and other potentially destructive programs, and to protect your information as you deem appropriate.

Notices and trademarks

This information was developed for products and services offered in the U.S.A. This material may be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

Notices

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session or persistent cookies. If a product or component is not listed, that product or component does not use cookies.

Table 15. Use of cookies by InfoSphere Information Server products and components

Product module	**Component or feature**	**Type of cookie that is used**	**Collect this data**	**Purpose of data**	**Disabling the cookies**
Any (part of InfoSphere Information Server installation)	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
Any (part of InfoSphere Information Server installation)	InfoSphere Metadata Asset Manager	• Session • Persistent	No personally identifiable information	• Session management • Authentication • Enhanced user usability • Single sign-on configuration	Cannot be disabled

Table 15. Use of cookies by InfoSphere Information Server products and components (continued)

Product module	Component or feature	Type of cookie that is used	Collect this data	Purpose of data	Disabling the cookies
InfoSphere DataStage	Big Data File stage	• Session • Persistent	• User name • Digital signature • Session ID	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere DataStage	XML stage	Session	Internal identifiers	• Session management • Authentication	Cannot be disabled
InfoSphere DataStage	IBM InfoSphere DataStage and QualityStage Operations Console	Session	No personally identifiable information	• Session management • Authentication	Cannot be disabled
InfoSphere Data Click	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
InfoSphere Data Quality Console		Session	No personally identifiable information	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere QualityStage Standardization Rules Designer	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
InfoSphere Information Governance Catalog		• Session • Persistent	• User name • Internal identifiers • State of the tree	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere Information Analyzer	Data Rules stage in the InfoSphere DataStage and QualityStage Designer client	Session	Session ID	Session management	Cannot be disabled

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Trademarks

Contacting IBM

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

The following table lists resources for customer support, software services, training, and product and solutions information.

Table 16. IBM resources

Resource	**Description and location**
IBM Support Portal	You can customize support information by choosing the products and the topics that interest you at www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server
Software services	You can find information about software, IT, and business consulting services, on the solutions site at www.ibm.com/businesssolutions/
My IBM	You can manage links to IBM Web sites and information that meet your specific technical support needs by creating an account on the My IBM site at www.ibm.com/account/
Training and certification	You can learn about technical training and education services designed for individuals, companies, and public organizations to acquire, maintain, and optimize their IT skills at http://www.ibm.com/training
IBM representatives	You can contact an IBM representative to learn about solutions at www.ibm.com/connect/ibm/us/en/

Index

Printed in USA

SC19-4345-00