SeeBeyond ICAN Suite

Oracle eWay Intelligent Adapter User's Guide

Release 5.0.6

Version 20050303114037.

Contents

Chapter 3

Configuring the Oracle eWay 16

Chapter 1

Introducing the Oracle eWay

This document describes how to install and configure the eWay Intelligent Adapter for Oracle.

What's in This Chapter

1.1 About Oracle Databases

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

The database has logical structures and physical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

1.2 About the Oracle eWay

The Oracle eWay enables eGate Integrator Projects to exchange data with external Oracle databases. This user's guide describes how to install and configure the Oracle eWay.

1.3 What's New in This Version

Relaunchable Oracle OTD

The Oracle OTD can now be saved under the same name and then be relaunched back to the same Java Collaboration or BPEL.

1.4 About This Document

This guide explains how to install, configure, and operate the SeeBeyond® Integrated Composite Application Network Suite™ (ICAN) Oracle eWay Intelligent Adapter, referred to as the Oracle eWay throughout this guide.

1.4.1. What's in This Document

This document includes the following chapters:

- **Chapter 1 "Introducing the Oracle eWay":** Provides an overview of the product as well as high-level information about this document.
- **Chapter 2 "Installing the Oracle eWay":** Describes the system requirements and provides instructions for installing the Oracle eWay.
- **Chapter 3 "Configuring the Oracle eWay":** Provides instructions for configuring the eWay to communicate with your legacy systems.
- **Chapter 4 "Using the Oracle eWay Database Wizard":** Provides information about .sag files and using the Oracle wizard.
- **Chapter 5 "Implementing the Oracle eWay":** Provides instructions for installing and running the sample Projects.

1.4.2 Scope

This document describes the process of installing, configuring, and running the Oracle eWay.

This document does not cover the Java methods exposed by this eWay. For information on the Java methods, download and view the Oracle eWay Javadoc files from the Enterprise Manager.

1.4.3 Intended Audience

This guide is intended for experienced computer users who have the responsibility of helping to set up and maintain a fully functioning ICAN Suite system. This person must also understand any operating systems on which the ICAN Suite is to be installed (Windows or UNIX) and must be thoroughly familiar with Windows-style GUI operations.

1.4.4 Document Conventions

The following writing conventions are observed throughout this document.

Table 1 Writing Conventions

Text	Convention	Example
Button, file, icon, parameter, variable, method, menu, and object names.	**Bold** text	▪ Click **OK** to save and close. ▪ From the **File** menu, select **Exit**. ▪ Select the **logicalhost.exe** file. ▪ Enter the **timeout** value. ▪ Use the **getClassName()** method. ▪ Configure the **Inbound** File eWay.
Command line arguments and code samples	`Fixed` font. Variables are shown in ***`bold italic`***.	`bootstrap -p` ***`password`***
Hypertext links	**Blue** text	**http://www.seebeyond.com**

1.4.5. Screenshots

Depending on what products you have installed, and how they are configured, the screenshots in this document may differ from what you see on your system.

1.5 Related Documents

The following SeeBeyond documents provide additional information about the ICAN product suite:

- *eGate Integrator User's Guide*
- *SeeBeyond ICAN Suite Installation Guide*

1.6 SeeBeyond Web Site

The SeeBeyond Web site is your best source for up-to-the-minute product news and technical support information. The site's URL is:

http://www.seebeyond.com

1.7 Feedback

If you have any feedback on any SeeBeyond documentation, please send an E-mail to:

docfeedback@seebeyond.com

Chapter 2

Installing the Oracle eWay

This chapter describes how to install the Oracle eWay.

What's in This Chapter

2.1 Supported Operating Systems

The Oracle eWay is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP Tru64 5.1A
- HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- IBM AIX 5.1L and 5.2
- Red Hat Enterprise Linux AS 2.1 (Intel x86)
- Red Hat Linux 8 (Intel x86)
- Sun Solaris 8 and 9
- SuSE Linux Enterprise Server 8 (Intel x86)
- Japanese Windows 2000, Windows XP, and Windows Server 2003
- Japanese HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- Japanese IBM AIX 5.1L and 5.2
- Japanese Sun Solaris 8 and 9
- Korean Windows 2000, Windows XP, and Windows Server 2003
- Korean HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)

- Korean IBM AIX 5.1L and 5.2
- Korean Sun Solaris 8 and 9

Note: *The Oracle eWay also supports Simplified and Traditional Chinese (Big5/GB2312/UTF-8) data using the United States version of Windows with the Chinese locale setting.*

Although the Oracle Universal Database eWay, the Repository, and Logical Hosts run on the platforms listed above, the Enterprise Designer requires the Windows operating system. Enterprise Manager can run on any platform that supports Internet Explorer 6.0.

2.1.1 WebLogic and WebSphere Application Server Support

In addition to the operating systems listed above, this eWay is also supported on the following application servers:

- WebSphere Application Server, version 5.0
- WebLogic Application Server, version 8.1

These are limited to outbound mode using Java Collaborations. For additional information, see the *eGate Integrator User's Guide*.

2.2 System Requirements

The system requirements for the Oracle eWay are the same as for eGate Integrator. Please review the **Readme.txt** file for any additional requirements prior to installation. **Readme.txt** is located on the installation CD-ROM.

The Oracle eWay installation installs the type 4 Oracle JDBC 9.0.1.3.0 driver required to connect to the external Oracle database.

Note: *To use the TimeStamp data type, which enables milliseconds to be stored, you need the Oracle 10g driver (Oracle JDBC version 10.1.0.2.0).*

To use the OCI driver with the Oracle eWay, you must install the Oracle client on the logical host. Because the OCI driver supports both the most current Oracle database and any previous versions, SeeBeyond recommends that you use the most current version of the OCI driver. By doing this, you can ensure compatibility between the various versions of any Oracle databases you may have.

Note: *To enable Web Services, you must install and configure the SeeBeyond ICAN Suite eInsight Business Process Manager.*

2.3 Supported External Applications

The Oracle eWay supports the following software for external systems:

- Oracle version 8i with patch 8.1.7
- Oracle version 9i with patch 9.0.1.3
- Oracle version 9i release 9.2.0
- Oracle 10*g*

2.3.1. XA Requirements

When using XA with multiple Oracle databases, SeeBeyond recommends that you use the latest version of the database.

2.4 Before You Install

Open and review the **Readme.txt** file for the Oracle eWay for any additional information or requirements, prior to installation. The **Readme.txt** file is located on the installation CD-ROM.

2.5 Installing the eWay Product Files

During the eGate Integrator installation process, the Enterprise Manager, a web-based application, is used to select and upload eWays (eWay.sar files) from the eGate installation CD-ROM to the Repository.

The installation process includes installing the following components:

- Installing the Repository
- Uploading products to the Repository
- Downloading components (such as Enterprise Designer and Logical Host)
- Viewing product information home pages

Follow the instructions for installing the eGate Integrator in the SeeBeyond *ICAN Suite Installation Guide*, and include the following steps:

- In the Enterprise Manager, select the **OracleeWay.sar** (to install the Oracle eWay) file to upload.
- In the Enterprise Manager, select the **FileeWay.sar** (to install the File eWay, used in the sample Project) file to upload.
- In the Enterprise Manager, select the **OracleeWayDocs.sar** (to install the documentation, the Javadoc, and the sample) file to upload.

- In the Enterprise Manager, in the Documentation tab, click on the document link, the Javadoc link, or the sample file link. For the sample project, SeeBeyond recommends that you extract the file to another file location before you import it using the Enterprise Explorer's Import Project tool.

For additional information on how to use eGate, please see the *eGate Integrator Tutorial*.

Continue installing the eGate Enterprise Designer as instructed.

2.6 After You Install

If you choose to use the OCI driver, you must copy the **classes12.jar** file from the Oracle client to the Logical Host.

Oracle 10*g* users must copy the **ojdbc14.jar** file from the Oracle client, and rename it to the **classes12.jar** file before copying into the Logical Host.

Note: *Make sure you create a back-up of the original* ***classes12.jar*** *file before applying these changes.*

You can load the **classes12.jar** file in a number of ways, including:

- Loading the file using Enterprise Designer
- Copying the file directly into the Logical Host

To Load the .jar File Using Enterprise Designer—Using 5.0.3

1. Right-click the Logical Host on the Environment tab, and select **Upload File** on the menu.

   If you are running the appplication on Unix, you must first copy the Oracle driver (**classes12.jar** or **ojdbc14.jar**) from the Oracle client on your Unix machine to a location on Windows.

2. In the Upload Third Party Files window, click **Add** and select the **classes12.jar** from the Oracle client installation directory. At run-time, the **classes12.jar** file is loaded into the following directory:

```
<egate_logicalhost>:\\ICAN50\logicalhost\stcis\lib\
```

If you do not follow the directions above, the eWay does not recognize your Oracle database.

Once the eWay is installed and configured, you must first incorporate it into a Project before it can perform its intended functions. See the *eGate Integrator User's Guide* for more information on incorporating the eWay into an eGate Project.

To Copy the .jar File Directly into the Logical Host—Using 5.0.1 or 5.0.2

1 Copy the **classes12.jar** file to:

```
<egate_logicalhost>:\\stcis\lib\
```

Replace the **classes12.jar** file already in this location with Oracle's **classes12.jar** file.

Note: *The <egate_logicalhost>:\\stcis\lib location only appears after you run the project.*

Note: *The Oracle eWay only supports the OCI driver for run-time.*

Chapter 3

Configuring the Oracle eWay

This chapter describes how to set the properties of the Oracle eWay.

What's in This Chapter

3.1 Configuring the Oracle eWay Properties

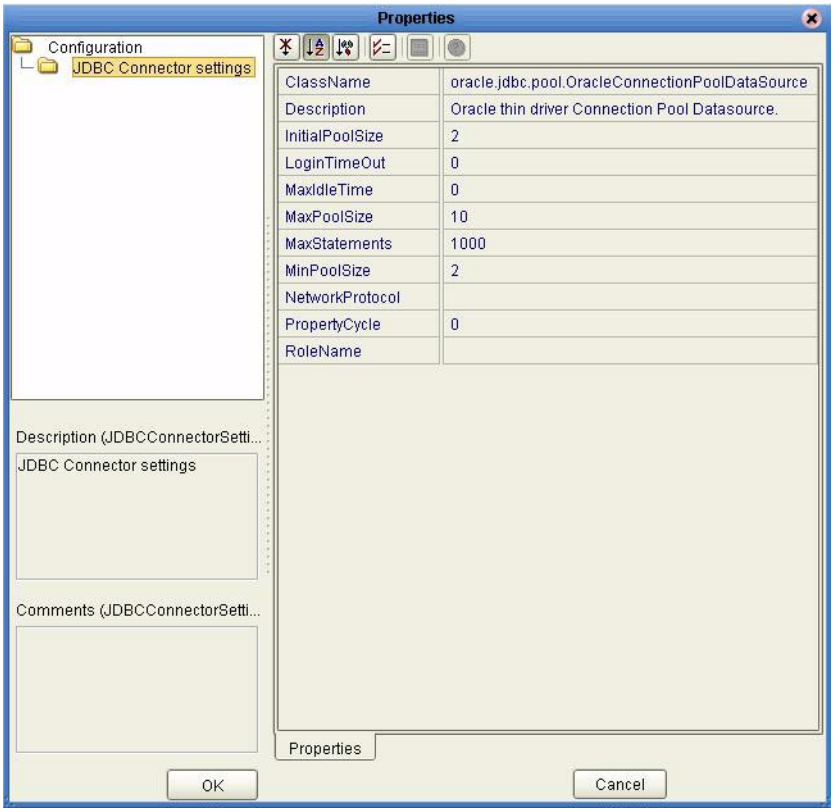
In the **Properties** window, using the descriptions below, enter the information necessary for the eWay to establish a connection to the external application.

3.1.1. Setting the Properties in the Outbound eWay

The DataSource settings define the properties used to interact with the external database.

Note: *Not all parameters are supported in the current release, please contact SeeBeyond for more information.*

Figure 1 The Outbound eWay Properties



ClassName

Description

Specifies the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface.

Required Values

A valid class name.

The default is **oracle.jdbc.pool.OracleConnectionPoolDataSource**.

Description

Description

Enter a description for the database.

Required Values

A valid string.

InitialPoolSize

Description

Enter a number for the physical connections the pool contains when it is created.

Required Values

A valid numeric value.

LoginTimeOut

Description

The number of seconds the driver must wait before attempting to log in to the database before timing out.

Required Value

A valid numeric value.

MaxIdleTime

Description

The maximum number of seconds that a physical connection may remain unused before it is closed. 0 (zero) indicates that there is no limit.

Required Values

A valid numeric value.

MaxPoolSize

Description

The maximum number of physical connections the pool must keep available at all times. 0 (zero) indicates that there is no maximum.

Required Values

A valid numeric value.

MaxStatements

Description

The maximum total number of statements that the pool keeps open. 0 (zero) indicates that the caching of statements is disabled.

Required Values

A valid numeric value.

MinPoolSize

The minimum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there is no physical connections in the pool and the new connections are created as needed.

Required Values

A valid numeric value.

NetworkProtocol

Description

The network protocol used to communicate with the server.

Required Values

Any valid string.

PropertyCycle

Description

The interval (in seconds) that the pool must wait before enforcing the current policy defined by the values of the other connection pool properties in this deployment descriptor.

Required Values

A valid numeric value.

RoleName

Description

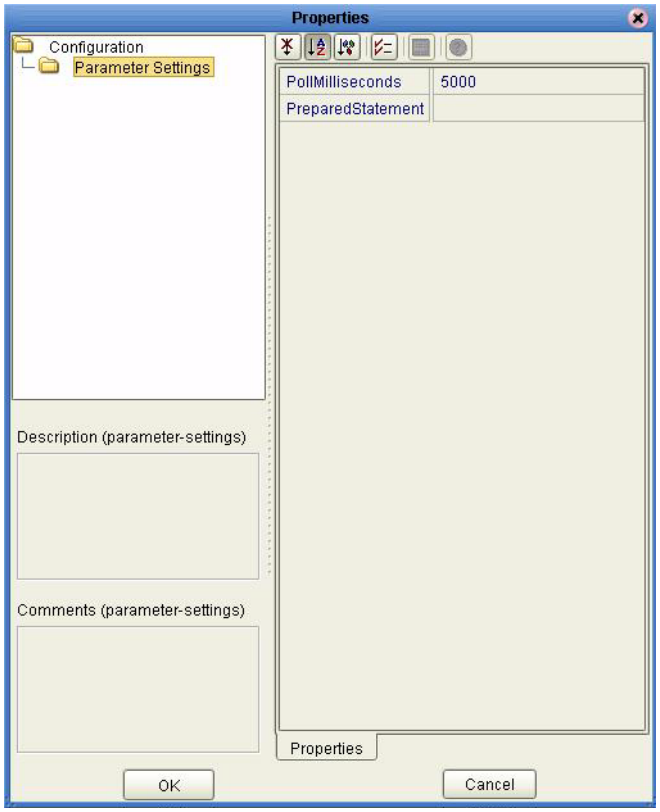
An initial SQL role name.

Required Values

Any valid string.

3.1.2. Setting the Properties in the Inbound eWay

Figure 2 Properties of the Inbound eWay



Pollmilliseconds

Description

Polling interval in milliseconds.

Required Values

A valid numeric value. The default is 5000.

PreparedStatement

Description

The Prepared Statement used for polling against the database.

Required Values

The Prepared Statement must be the same Statement you created using the Database OTD Wizard. Only a SELECT Statement is allowed. Additionally, no place holders should be used. This is a SQL statement that cannot contain any input data (i.e. you cannot use "?" in the Prepared Query).

3.1.3. Setting the Properties in the Outbound eWay Environment

Before deploying your eWay, you must set the properties of the eWay environment using the following descriptions.

To assure a successful database connection when using different Oracle drivers, make sure to fill in only the following fields:

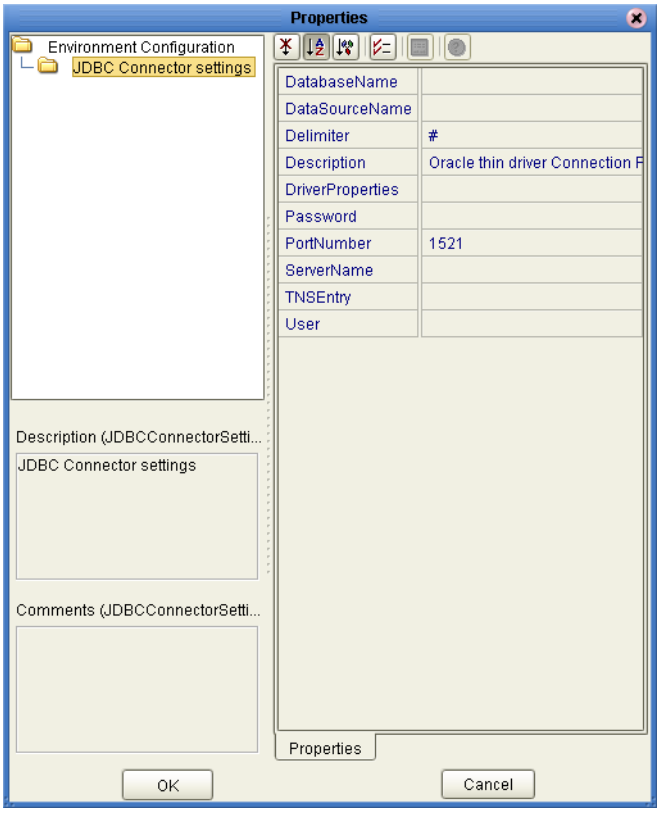
Oracle thin driver fields:

- DatabaseName
- PortNumber
- ServerName
- User
- Password

Oracle OCI driver fields:

- TNSEntry
- User
- Password
- PortNumber – not needed, but keep the default

Figure 3 eWay Environment Configuration



DatabaseName

Description

Specifies the name of the database instance used on the Server.

Required Values

Any valid string.

DataSourceName

Description

Provide the name of the XADataSource or ConnectionPoolDataSource object that the DataSource object delegates behind the scenes when connection pooling or distributed transaction management is being done.

Required Values

Optional. In most cases, leave this box empty.

Delimiter

Description

This is the delimiter character to be used in the DriverProperties prompt.

Required Values

The default is #.

Description

Description

Enter a description for the database.

Required Values

A valid string.

DriverProperties

Description

Use the JDBC driver that is shipped with this eWay. Often, the DataSource implementation needs to execute additional properties to assure a connection. You need to identify the additional methods in the Driver Properties.

Required Values

Any valid delimiter.

Valid delimiters are: “<method-name-1>#<param-1>#<param-2>##.....<param-n>##<method-name-2>#<param-1>#<param-2>#.....<param-n>##.....##”.

For example: to execute the method setURL, give the method a string for the URL “setURL#<url>##”.

Password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

Specifies the I/O port number on which the server is listening for connection requests.

Required Values

A valid port number. The default is 1521.

ServerName

Description

Specifies the host name of the external database server.

Required Values

Any valid string.

TNSEntry

Description

Specifies the TNS name for the Oracle instance specified in TNSNAMES.ORA. If a TNS name is specified, then the OCI driver is used, which requires further installation of the Oracle client. If a TNS name is not specified, then the thin driver is used.

Required Values

Any valid string.

User

Description

Specifies the user name that the eWay uses to connect to the database.

Required Values

Any valid string.

3.1.4. Setting the Properties in the Inbound eWay Environment

Figure 4 Inbound eWay Environment

DatabaseName

Description

Specifies the name of the database instance used on the Server.

Required Values

Any valid string.

Password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

Specifies the I/O port number on which the server is listening for connection requests.

Required Values

A valid port number. The default is 1433.

ServerName

Description

Specifies the host name of the external database server.

Required Values

Any valid string.

User

Description

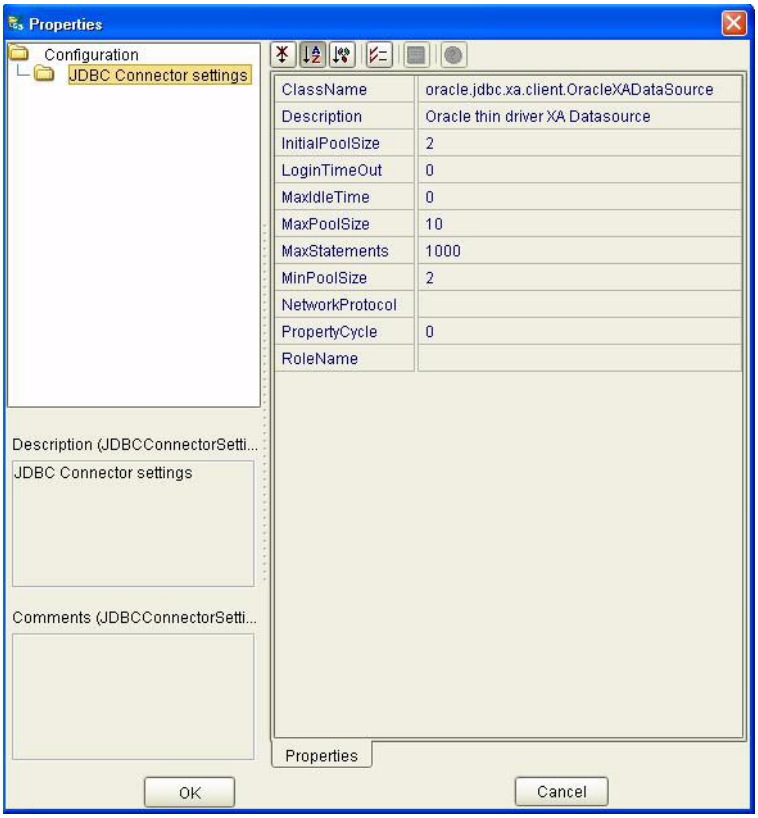
Specifies the user name that the eWay uses to connect to the database.

Required Values

Any valid string.

3.1.5. Setting the Properties in the Outbound eWay with XA Support

Figure 5 Outbound Oracle eWay with XA Support



ClassName

Description

Specifies the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface.

Required Values

A valid class name.

The default is **oracle.jdbc.xa.client.OracleXADataSource**.

Description

Description

Enter a description for the database.

Required Values

A valid string.

InitialPoolSize

Description

Enter a number for the physical connections the pool contains when it is created.

Required Values

A valid numeric value. The default is 2.

LoginTimeOut

Description

The number of seconds the driver must wait before attempting to log into the database before timing out.

Required Values

A valid numeric value.

MaxIdleTime

Description

The maximum number of seconds that a physical connection may remain unused before it is closed. 0 (zero) indicates that there is no limit.

Required Values

A valid numeric value.

MaxPoolSize

Description

The maximum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there is no maximum.

Required Values

A valid numeric value. The default is 10.

MaxStatements

Description

The maximum total number of statements that the pool keeps open. 0 (zero) indicates that the caching of statements is disabled.

Required Values

A valid numeric value. The default is 1000.

MinPoolSize

The minimum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there are no physical connections in the pool and new connections are created as needed.

Required Values

A valid numeric value.

NetworkProtocol

Description

The network protocol used to communicate with the server.

Required Values

Any valid string. The default is 2.

PropertyCycle

Description

The interval, in seconds, that the pool waits before enforcing the current policy defined by the values of the other connection pool properties in this deployment descriptor.

Required Values

A valid numeric value.

RoleName

Description

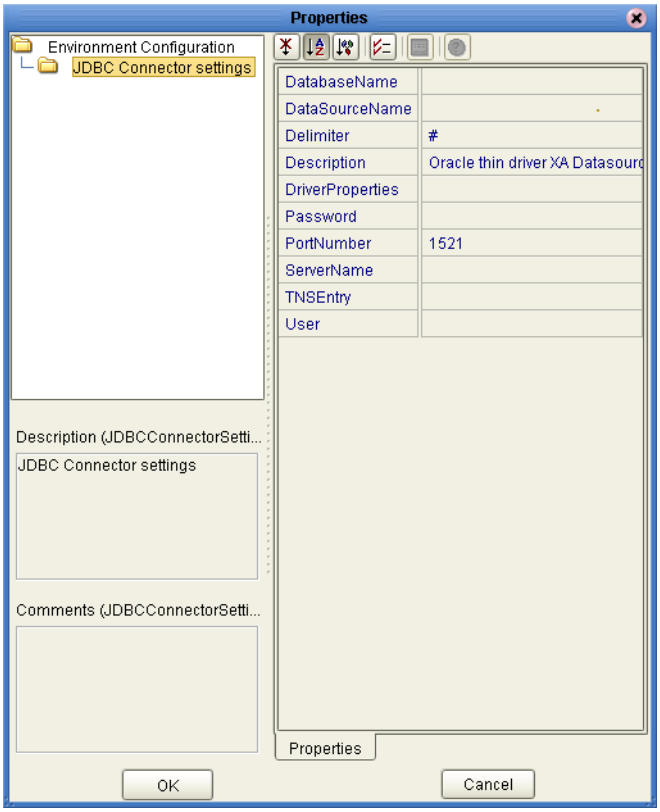
An initial SQL role name.

Required Values

Any valid string.

3.1.6. Setting the Properties in the eWay XA Environment

Figure 6 Environment Properties of the eWay with XA



DatabaseName

Description

Specifies the name of the database instance used on the Server.

Required Values

Any valid string.

DataSourceName

Description

Enter the name of the XADataSource or ConnectionPoolDataSource object that the DataSource object delegates (behind the scenes) when connection pooling or distributed transaction management is being done.

Required Values

Optional. In most cases, leave this box empty.

Delimiter

Description

This is the delimiter character to be used in the DriverProperties prompt.

Required Values

The default is #.

Description

Description

Enter a description for the database.

Required Values

A valid string.

DriverProperties

Description

Use the JDBC driver that is shipped with this eWay. Often times the DataSource implementation needs to execute additional methods to assure a connection. You must identify the additional methods in the Driver Properties.

Required Values

Any valid delimiter.

Valid delimiters are: “<method-name-1>#<param-1>#<param-2>##.....<param-n>##<method-name-2>#<param-1>#<param-2>#.....<param-n>##.....##”.

For example: to execute the method setURL, give the method a String for the URL “setURL#<url>##”.

Password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

Specifies the I/O port number on which the server is listening for connection requests.

Required Values

A valid port number. The default is 1521.

ServerName

Description

Specifies the host name of the external database server.

Required Values

Any valid string.

TNSEntry

Description

Specifies the TNS name for the Oracle instance specified in TNSNAMES.ORA. If a TNS name is specified, then the OCI driver is used, which further requires installation of the Oracle client. If a TNS name is not specified, then the thin driver is used.

Required Values

A valid TNS name if using the OCI driver; otherwise do not enter any value.

User

Description

Specifies the user name the eWay uses to connect to the database.

Required Values

Any valid string.

Chapter 4

Using the Oracle eWay Database Wizard

This chapter describes how to use the Oracle eWay Database Wizard to build OTD's, including using the Statement Builder Wizard.

What's in This Chapter

4.1 Using the Database OTD Wizard

The Database OTD Wizard generates OTDs by connecting to external data sources and creating corresponding Object Type Definitions. The OTD Wizard can create OTDs based on any combination of Tables and Stored Procedures or Prepared SQL Statements.

Field nodes are added to the OTD based on the Tables in the external data source. Java method and parameter nodes are added to provide the appropriate JDBC functionality. For more information about the Java methods, refer to your JDBC developer's reference.

The Oracle eWay also supports Double-Byte Character Set (DBCS) table and column names. The DBCS is a set of characters in which each character is represented by two bytes. Japanese and Korean languages require double-byte character sets.

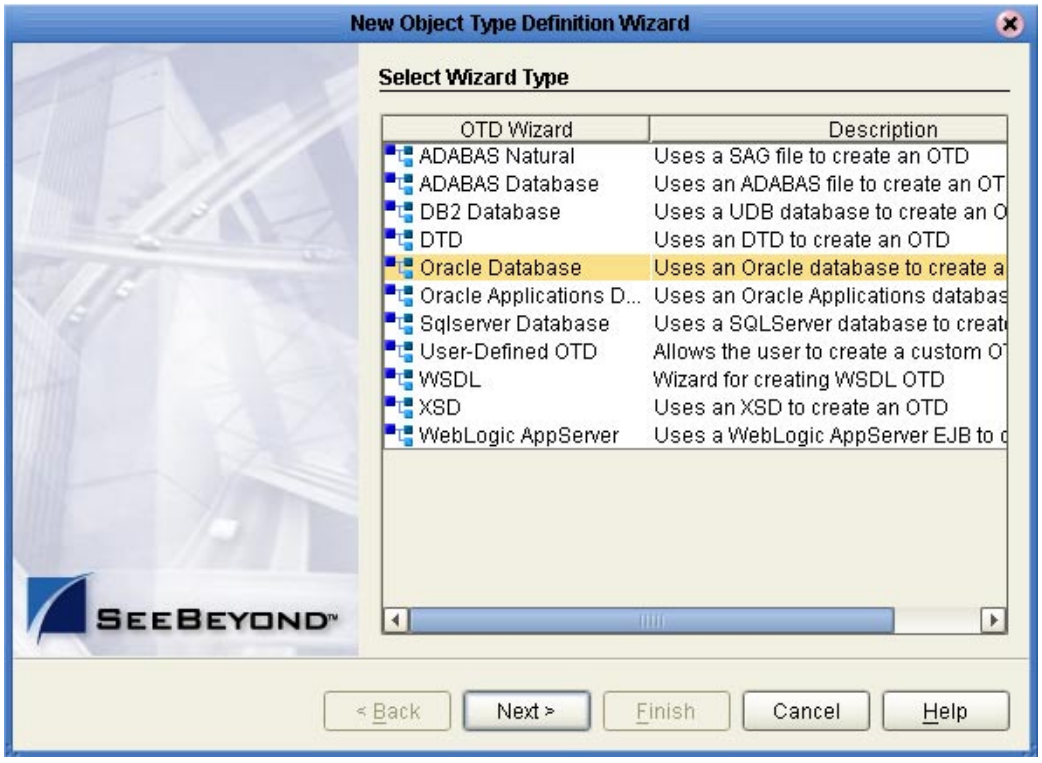
Note: *Database OTDs are not messagable. For more information on messagable OTDs, see the eGate Integrator User's Guide.*

Select Wizard Type

1 In the Enterprise Explorer, right-click on the project and select **Create an Object Type Definition** from the shortcut menu.

2 In the **OTD Wizard Selection** dialog box, select the **Oracle Database** and click **Next** (see Figure 7).

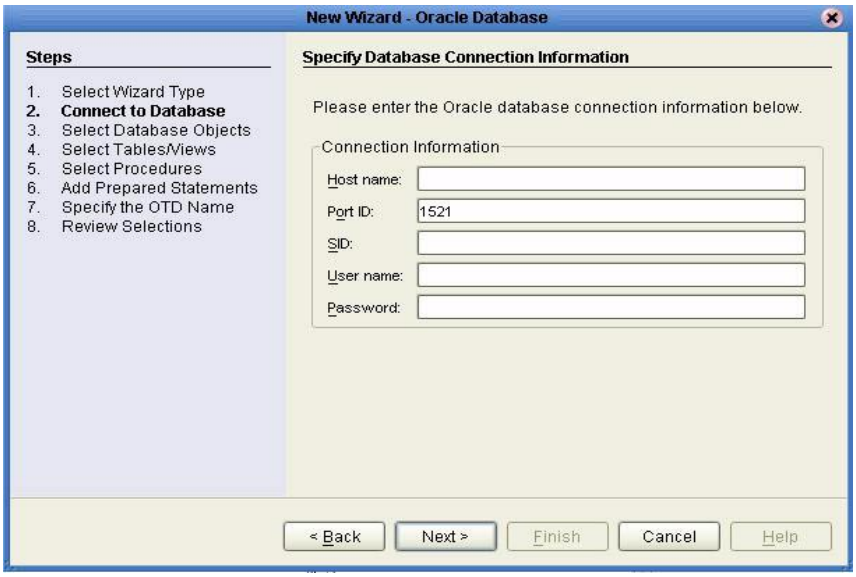
Figure 7 OTD Wizard Selection



Connect to Database

3 Specify the connection information for your database, including your **UserName** and **Password** and click **Next** (see Figure 8).

Figure 8 Database Connection Information

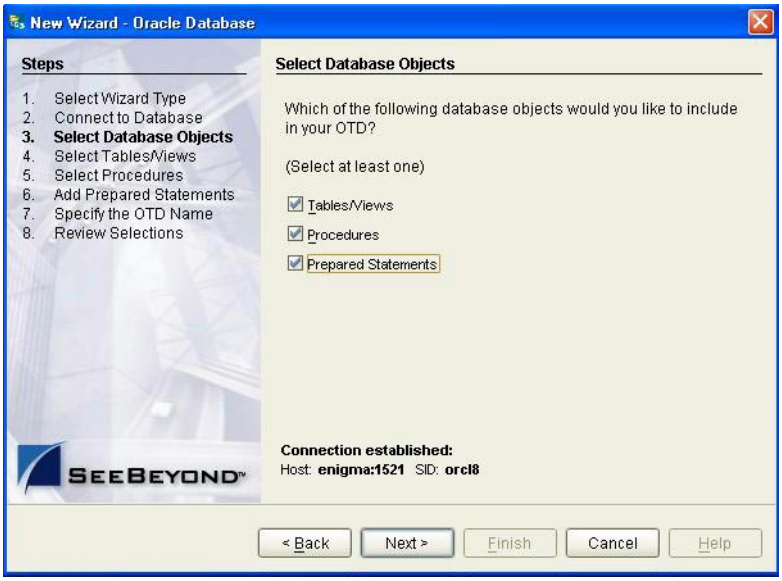


Select Database Objects

4 In the **Select Database Objects** dialog box, shown in Figure 9, select **Tables/Views** for this sample. When selecting Database Objects, you can select any combination of **Tables**, **Views**, **Procedures**, or **Prepared Statements** you would like to include in the .otd file. Click **Next** to continue.

Note: *Views are read-only and are for informational purposes only.*

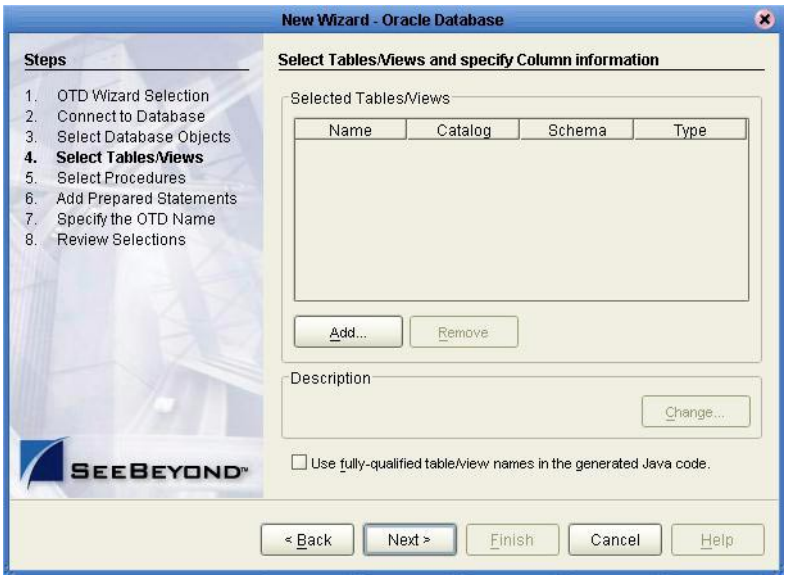
Figure 9 Select Database Objects



Select Table/Views

5 In the **Select Tables/Views** dialog box, click **Add** (see Figure 10).

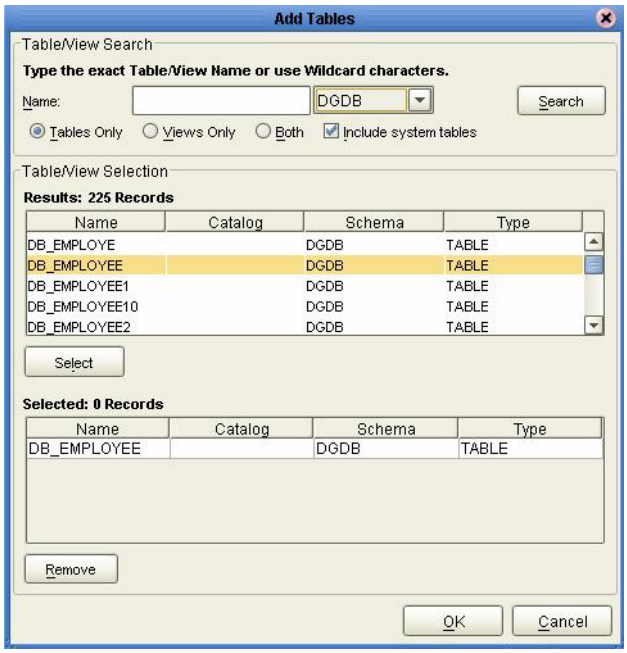
Figure 10 Select Tables/Views



6 In the **Add Tables** dialog box, if your selection criteria includes table data, view only data, both, and/or system tables, then select these options.

7 From the **Table/View Name** drop-down list, select the location of your database table and click **Search** (see Figure 11). You can search for **Table/View Names** by entering a table name.

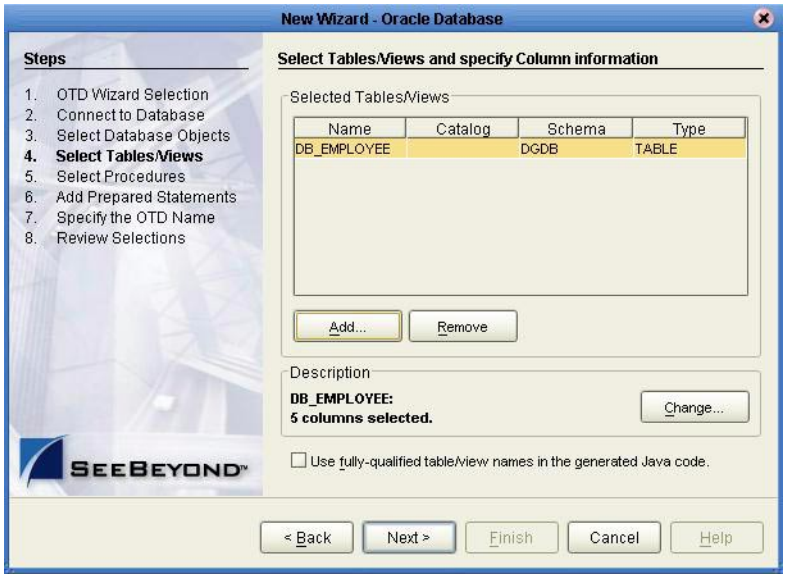
Figure 11 Database Wizard - All Schemes



8 Select the table of choice and click **OK**.

The table selected is added to the **Selected Tables/Views** dialog box (see Figure 12).

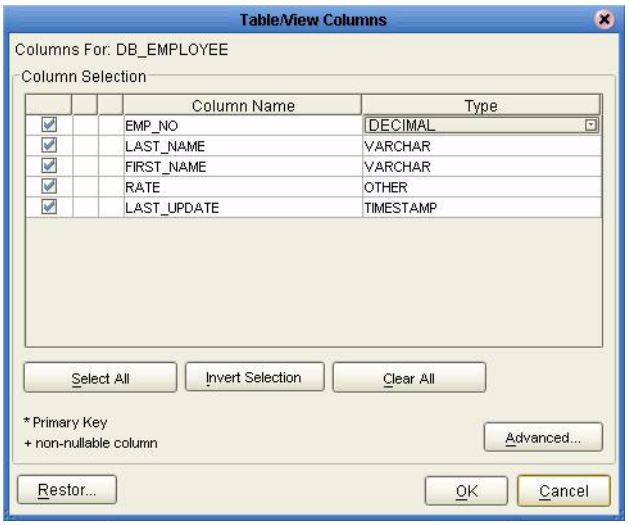
Figure 12 Selected Tables/Views dialog box with a table selected



9 In the **Selected Tables/Views** dialog box, review the table(s) you have selected. To make changes to the selected Table or View, click **Change**. If you do not wish to make any additional changes, click **Next** to continue.

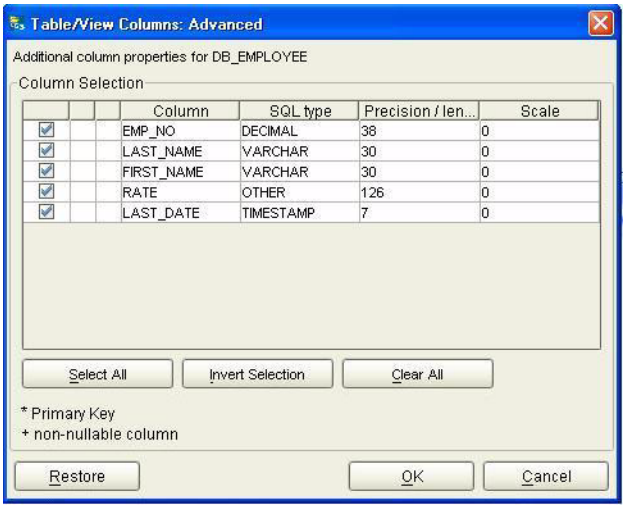
10 In the **Table/View Columns** dialog box, you can select or deselect your table columns. You can also change the data type for each table by highlighting the data type and selecting a different type from the drop-down list. If you need to change any of the tables columns, click **Change** (see Figure 13).

Figure 13 Table/View Columns



11 Click **Advanced** to change the data type, precision/length, or scale. Once you have finished your table choices, click **OK** (see Figure 14). In general, you do not need to change the precision or scale.

Figure 14 Table/View Columns — Advanced

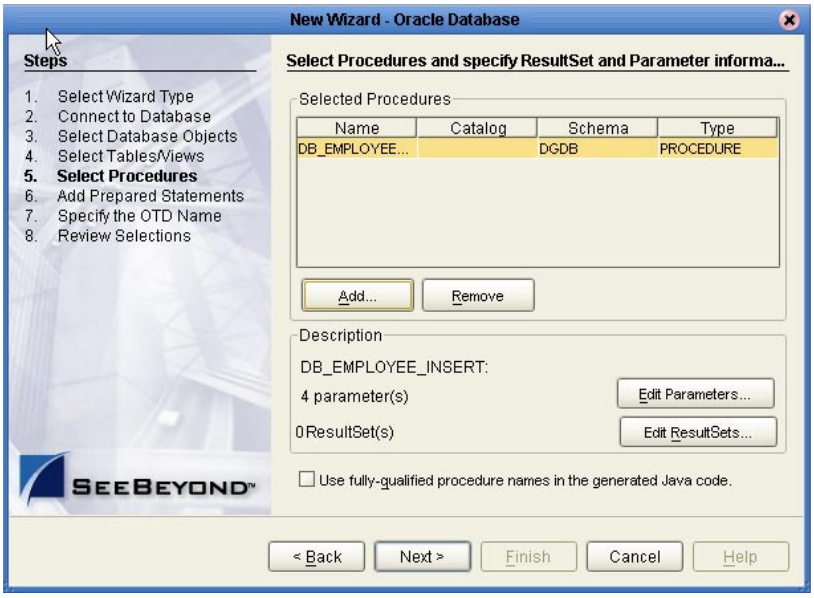


12 When using Oracle packages, select **Use fully qualified table/view names in the generated Java code** (see Figure 12).

Select Procedures

13 In the **Select Procedures and specify ResultSet and Parameter Information** dialog box, click **Add.**

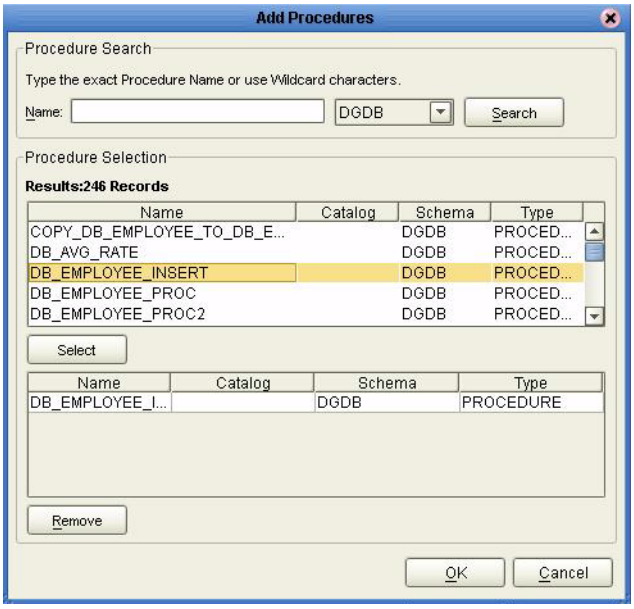
Figure 15 Select Procedures and specify ResultSet and Parameter Information



14 In the **Select Procedures** dialog box, enter the name (case-sensitive) of a Procedure or select a table from the drop-down list. Click **Search**. You can use Wildcard characters.

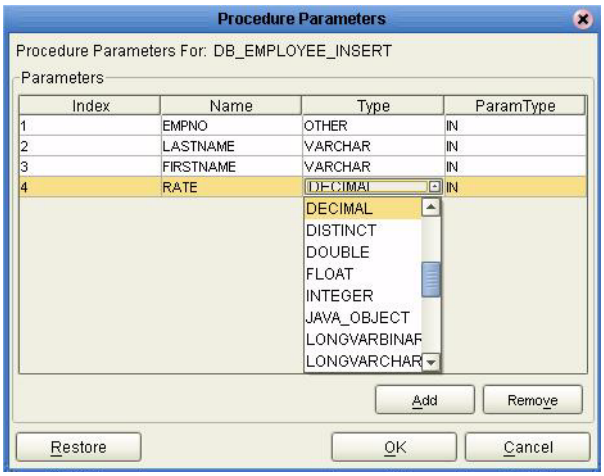
15 In the resulting **Procedure Selection** list box, select a Procedure. Click **OK**.

Figure 16 Add Procedures



16 In the **Select Procedures and specify ResultSet and Parameter Information** dialog box, click **Edit Parameters** to make changes (as needed) to the selected Procedure (see Figure 17).

Figure 17 Procedure Parameters

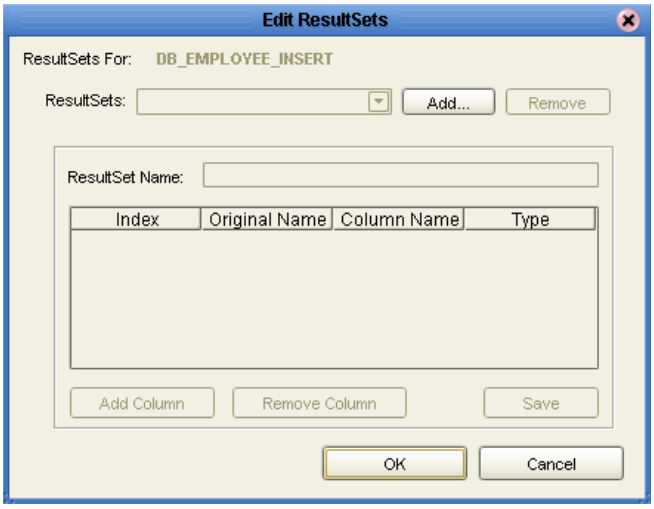


17 To restore the data type, click **Restore**. When you are finished, click **OK**.

18 Click **Edit ResultSets** to have the wizard generate the ResultSet(s) for the Oracle OTD.

19 Select the type of Resultset you want to generate and click **Add**.

Figure 18 Edit ResultSet



The DBWizard provides three different ways to generate the ResultSet nodes of a Stored Procedure. They are the "**By Executing**", "**Manually**", and "**With Assistance**" modes.

The "**By Executing**" mode automatically generates the ResultSet(s) returned by the Stored Procedure. Depending on the business logic of the Stored Procedure, zero or more ResultSets can be returned from the execution.

The "**With Assistance**" mode enables you to extract a query from the Stored Procedure and generate the ResultSet. To facilitate this operation, the DBWizard tries to retrieve the content of the specified Stored Procedure and display it. However, content retrieval is not supported by all types of Stored Procedures. Stored Procedures are loosely classified into two types: SQL and external. SQL Stored Procedures are created using CREATE PROCEDURE SQL statements while external Stored Procedures are created using host languages (e.g. Java). Since external Stored Procedures do not store their execution plans in the database, content retrieval is impossible. When using the "**Assist**" mode, highlight and extract the SELECT statement up to and including the table name(s) to build the ResultSet.

The "**Manually**" mode is the most flexible way to generate the ResultSet directly. It enables you to specify the ResultSet name, original name, column name, and data type manually. One drawback of this method is that you need to know the original column names and data types (which is not always possible). For example, the column name of 3*C in this query.

```
SELECT A, B, 3*C FROM table T
```

is generated by the database. In this case, the "**With Assistance**" mode is a better choice.

If you modify the ResultSet generated by the "**Execute**" mode of the Database Wizard, you need to make sure the indexes match the Stored Procedure. This assures your ResultSet indexes are preserved.

20 In the **Select Procedures and specify ResultSet and Parameter Information** dialog box, click **Next** to continue.

Add Prepared Statements

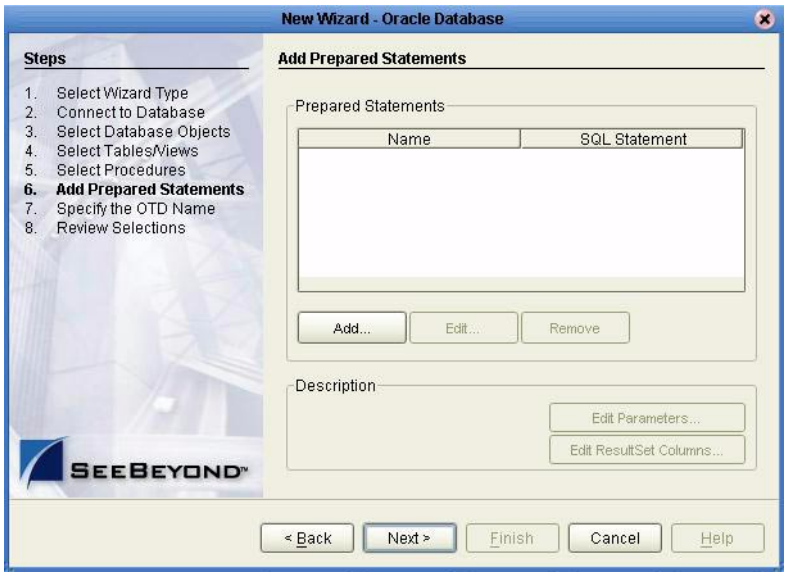
A Prepared Statement OTD represents a SQL statement that has been compiled. Fields in the OTD correspond to the input values that you need to provide.

You can use Prepared Statements to perform insert, update, delete, and query operations. A Prepared Statement uses a question mark (?) as a place holder for input. **For example**: insert into EMP_TAB(Age, Name, Dept No) values(?, ?, ?).

Note: *If the Prepared Statement performs either the insert, update, or delete operations, the* ***executeUpdate()*** *method is created when the OTD is built. If the Prepared Statement performs only the Select operation, the* ***executeQuery()*** *method is created when the OTD is built.*

21 In the **Add Prepared Statements** dialog box, click **Add**.

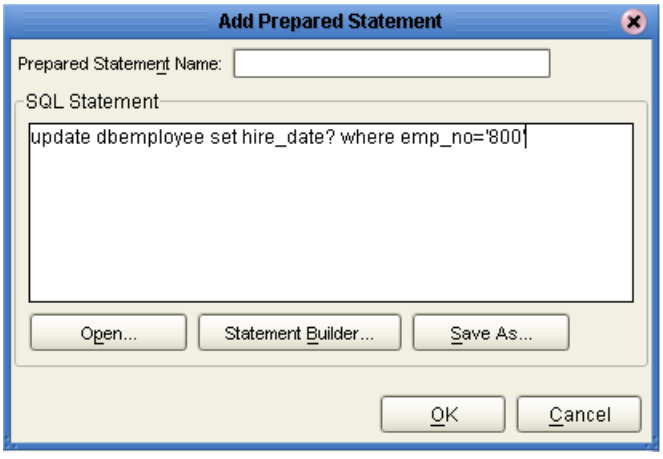
Figure 19 Prepared Statement



22 Enter the name of a Prepared Statement and create a SQL statement by clicking in the **SQL Statement** dialog box or by clicking the **Statement Builder** button. When you are finished creating the statement, click **Save As**, which gives the statement the name you just entered. This name appears as a node in the OTD (see Figure 20). Click **OK**.

Note: *For more information on creating a Prepared Statement using the* ***Statement Builder****, refer to* **"Using the Statement Builder Wizard" on page 44**.

Figure 20 Prepared SQL Statement

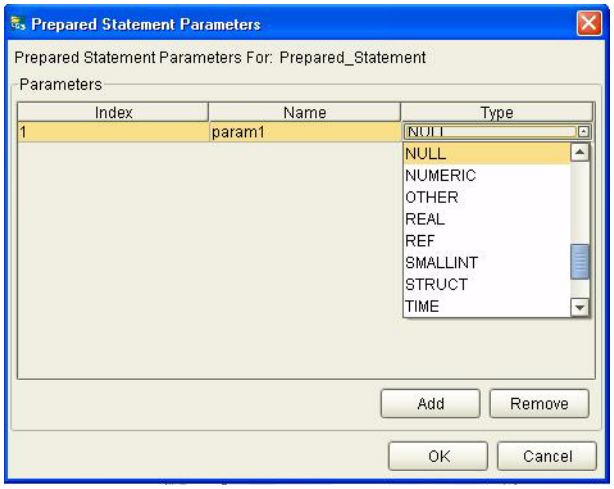


23 In the **Add Prepared Statement** dialog box, the name you assigned to the Prepared Statement appears. To edit the parameters, click **Edit Parameters**. You can change the datatype by clicking in the **Type** field and selecting a different type from the list.

24 Click **Add** if you want to add additional parameters to the Statement, or highlight a row and click **Remove** to remove the parameter (see Figure 21). Click **OK**.

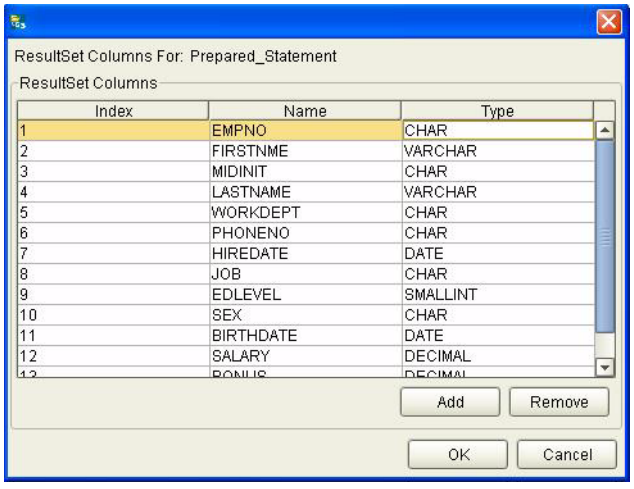
Note: *Once you save the Prepared Statement, make sure that the ResultSet Column Name (in the Prepared Statement parameters) is a valid alpha-numeric string.*

Figure 21 Edit the Prepared Statement Parameters



25 To edit the ResultSet Columns, click **Edit ResultSet Columns** (see Figure 22). Although both the Name and Type are editable, SeeBeyond recommends that you do not change the Name because it can cause a loss of integrity between the ResultSet and the Database. Click **OK**.

Figure 22 ResultSet Columns

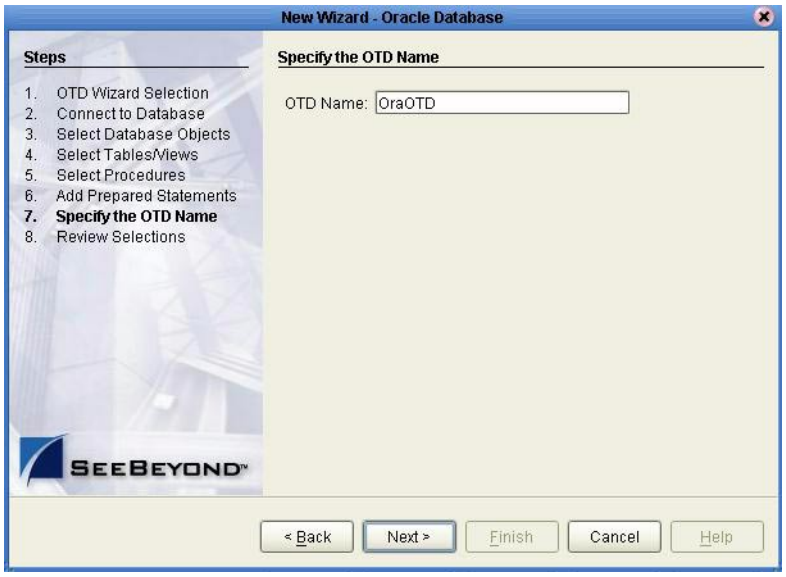


26 In the **Add Prepared Statements** dialog box, click **OK**.

Specify the OTD Name

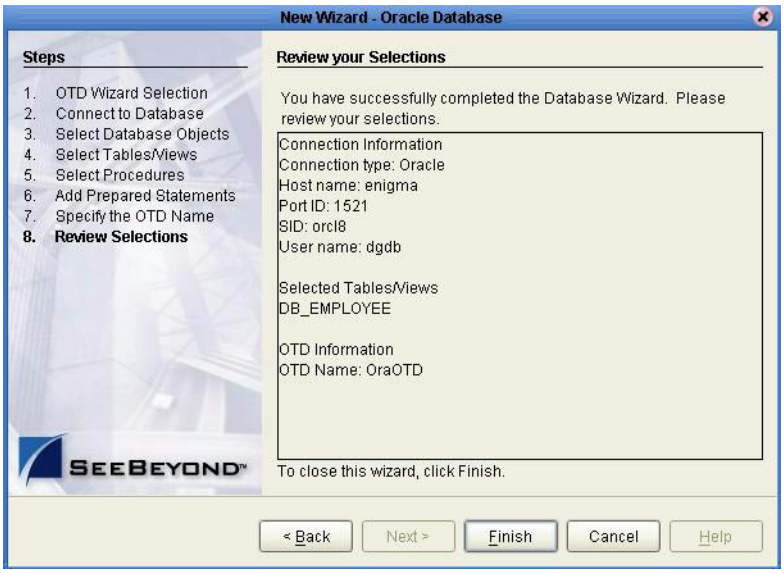
27 Enter a name for the OTD. The OTD contains the selected tables and the package name of the generated classes (see Figure 23).

Figure 23 Naming an OTD



28 View the summary of the OTD. If you find that you have made a mistake, click **Back** and correct the information. If you are satisfied with the OTD information, click **Finish** to begin generating the OTD (see Figure 24).

Figure 24 Database Wizard - Summary



The resulting **OTD** appears on the Enterprise Designer's canvas.

4.1.1 Using the Statement Builder Wizard

Use the Statement Builder wizard to construct a Prepared Statement with point-and-click ease. You can create Prepared Statements, using a graphical user interface, and preview them before incorporating them into an Oracle OTD.

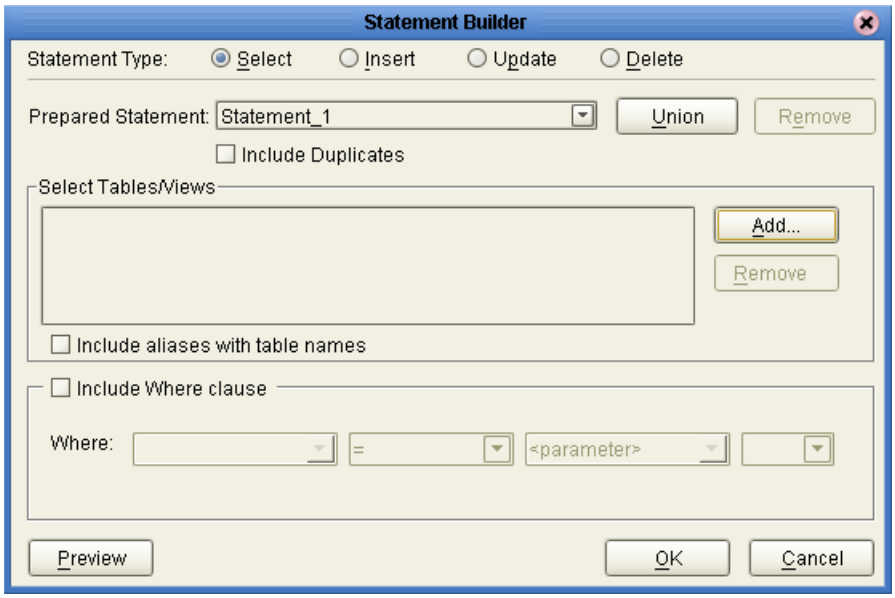
Please note that the Statement Builder wizard:

- is one option for constructing a Prepared Statement, but is not mandatory for building them.
- does not perform data validation.
- does not support the CLOB data type.

Using the Statement Builder

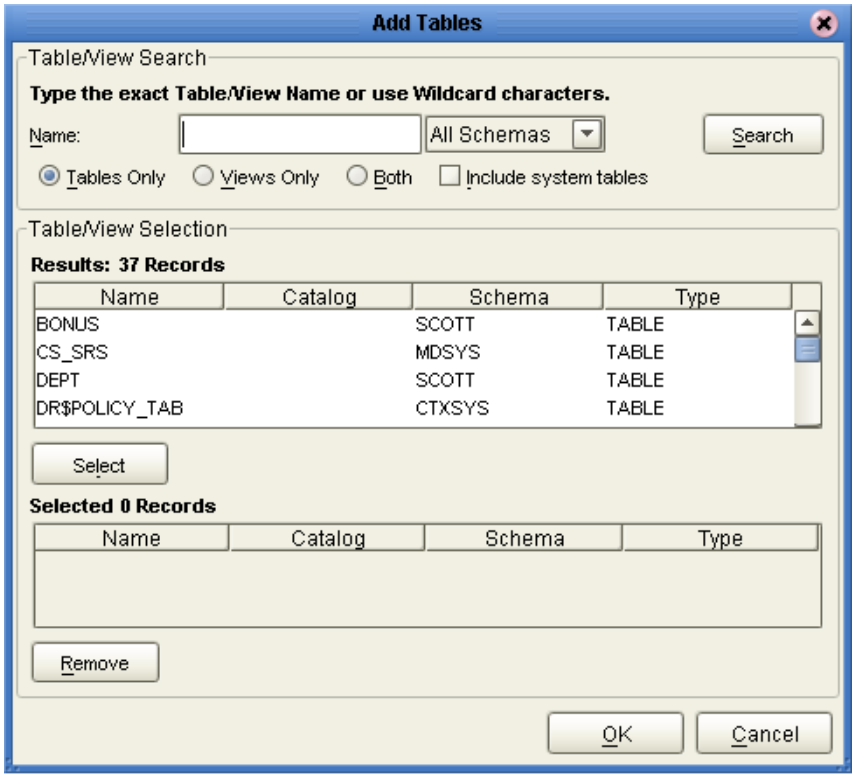
1. Add a Prepared Statement OTD by clicking **Add**, as described in **"Add Prepared Statements" on page 40**.
2. Enter the name for the Prepared Statement and create a SQL statement by clicking the **Statement Builder** button. The **Statement Builder** dialog box appears (as shown in **Figure 25 on page 44**).

Figure 25 Statement Builder



3. Select the type of statement you want to construct and click **Add**. The **Add Tables** dialog box appears (as shown in **Figure 26 on page 45**).

Figure 26 Add Tables

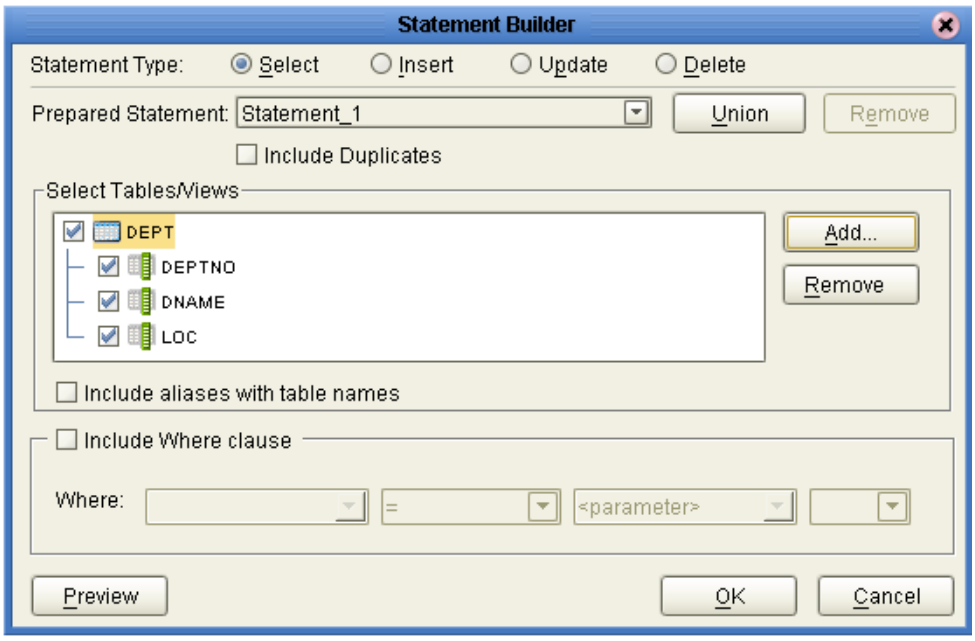


4 In the **Add Tables** dialog box, you can add applicable tables/views to the Statement by searching for a table, highlighting it, and clicking the **Select** button. You can also delete a table/view by highlighting it from the Selected list and clicking **Remove**.

Note: *You can also remove tables/views when you are in the Statement Builder dialog box by checking all applicable boxes and clicking* ***Remove****.*

5 Click **OK** when you have finished selecting the tables/views. The **Statement Builder** dialog box is displayed with all of your selections (as shown in Figure 27).

Figure 27 Statement Builder with Table Selections



6 Select the **Include aliases with table names** checkbox, if applicable.

7 Select the **Include Where clause** checkbox to enable the fields for creating Where clauses.

   A Select the field name to be used in the Where clause.

   B Select the applicable condition (for example +, >, <=, etc.).

   C Set the parameter.

   D Select the **And** or **Or** condition that enables you to create more Where clauses, if needed.

   You can create as many Where clauses to the Statement as needed.

8 Click **OK**. The **Add Prepared Statement** dialog box appears again, displaying the SQL statement you have created. If you need to make any changes to the statement, you can click the **Statement Builder** button again or change the statement in the dialog box.

9 Click **OK**. The Prepared Statement you constructed appears in the Oracle Database wizard. Click the **Edit** button to make changes to the Prepared Statement. If you only need to change parameters or ResultSet Columns, you can use the respective edit buttons.

10 Click the **Next** button. The **Specify the OTD Name** step of the Oracle Database wizard appears.

11 Name the OTD and click **Next**. The **Review your Selections** step appears.

12 Once you are satisfied with your selections, click **Finish**. Once the OTD is generated, you no longer have the ability to edit the Prepared Statement.

Chapter 5

Implementing the Oracle eWay

This chapter discusses an overview of the Sample Projects provided with this eWay.

What's in This Chapter

5.1 eInsight Engine and eGate Components

You can deploy an eGate component as an Activity in an eInsight Business Process. Once you have associated the desired component with an Activity, the eInsight engine can invoke it using a Web Services interface. Examples of eGate components that can interface with eInsight in this way are:

- Java Messaging Service (JMS)
- Object Type Definitions (OTDs)
- An eWay
- Collaborations

Using the eGate Enterprise Designer and eInsight, you can add an Activity to a Business Process, then associate that Activity with an eGate component, for example, an eWay. When eInsight runs the Business Process, it automatically invokes that component via its Web Services interface.

5.2 Using the Sample Project in eInsight

To begin using the sample eInsight Business Process project, you must import the project and view it from within the Enterprise Designer using the Enterprise Designer Project Import utility. Import the **Ora_BPEL_Sample.zip** file contained in the eWay sample folder on the installation CD-ROM.

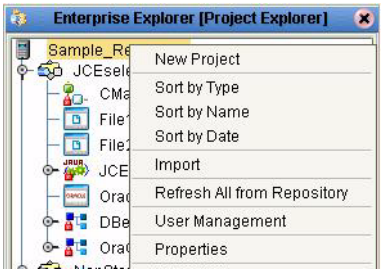
Note: *eInsight is a Business Process modeling tool. If you have not purchased eInsight, contact your sales representative for information on how to do so.*

Before recreating the sample Business Process, review the *eInsight Business Process Manager User's Guide* and the *eGate Integrator Tutorial*.

Importing the Sample Project

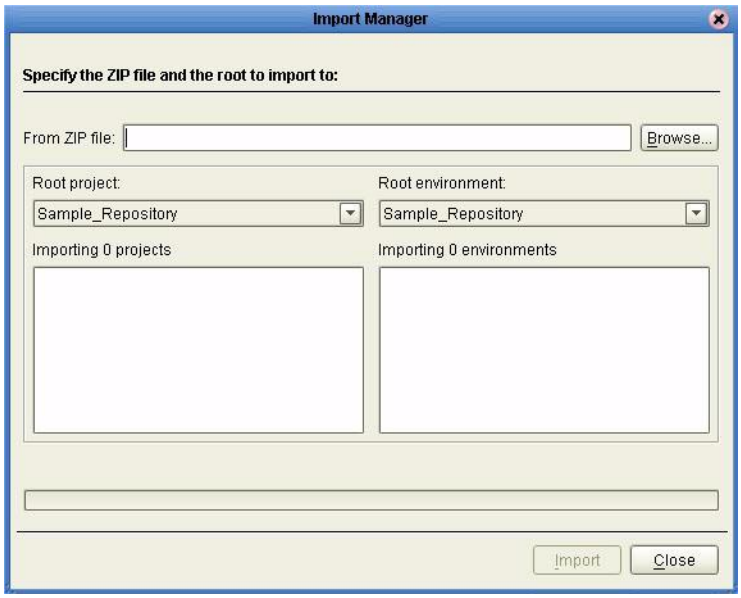
1 In the Enterprise Explorer, right-click on the repository and select **Import**. See **Figure 28**.

Figure 28 Importing the sample project



2 From the **Import Manager** dialog box, browse to the location of the sample folder, select the **Ora_BPEL_Sample.zip** file and click **Import**. See **Figure 29**.

Figure 29 Select the project file



3 Click the **Refresh All From Repository** icon located on the **Enterprise Explorer** toolbar.

The Business Process

The data used for this sample project is contained within a table called DBEmployee. The table contains the following columns:

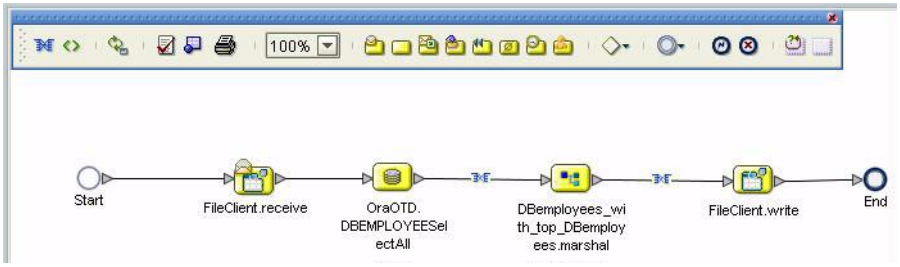
Table 2 Sample project data

Column Name	Mapping	Data Type	Data Length
EMP_NO	empno	varchar2	10
LAST_NAME	lastname	varchar2	30
FIRST_NAME	firstname	varchar2	30
SS_NUMBER	ssnumber	varchar2	20
HIRE_DATE	hiredate	varchar2	12

The sample project consists of an input file containing data that is passed into a database collaboration, marshalled, and then written out to an output file.

Refer to the *eInsight Business Process Manager User's Guide* for specific information on how to create and use a Business Process.

Figure 30



5.2.1. Working with the BPEL Operations

You can associate an eInsight Business Process Activity with the Oracle eWay during the system design phase. To make this association, select the desired **receive** or **write** operation under the eWay in the Enterprise Explorer and drag it onto the eInsight Business Process canvas.

For the Oracle eWay, the following operations are available:

- SelectAll
- SelectMultiple
- SelectOne
- Insert
- Update
- Delete

The operation automatically changes to an Activity with an icon identifying the component that is the basis for the Activity.

At run-time, the eInsight engine invokes each step in the order that you defined in the Business Process. Using the engine's Web Services interface, the Activity, in turn, invokes the Oracle eWay. You can open a file specified in the eWay and view its contents before and after the Business Process is executed.

Note: *Inbound Oracle eWays are only supported within BPEL Collaborations.*

Table 3 Inputs and outputs for eInsight operations:

eInsight Operation	Input	Output
SelectAll	Where clause (optional)	Returns all rows that fit the condition of the Where clause
SelectMultiple	number of rows Where clause (optional)	Returns the number of rows specified that fit the condition of the Where clause

eInsight Operation	Input	Output
SelectOne	Where clause (optional)	Returns the first row that fits the condition of the Where clause
Insert	definition of new item to be inserted	Returns status
Update	Where clause	Returns status
Delete	Where clause	Returns status

Where Clause

A BPEL Where clause statement may be joined by AND/OR with conditions of "=", "!=", "<>", "<", ">", "<=", ">=".

For example:

```
Where clause such as where column2=2 AND column1=1 OR column3=3 is
valid
```

SelectAll

The input to a SelectAll operation is an optional Where clause. The Where clause defines the criteria to which the rows must adhere for the purposes of being returned. In the SelectAll operation, all items that fit the criteria are returned. If the Where clause is not specified, all rows are returned.

The figure below shows a sample eInsight Business Process using the SelectAll operation. In this process, the SelectAll operation returns all rows where the ITEM_ID matches the selected ITEM_ID to the shopping cart.

Figure 31 SelectAll Sample Business Process

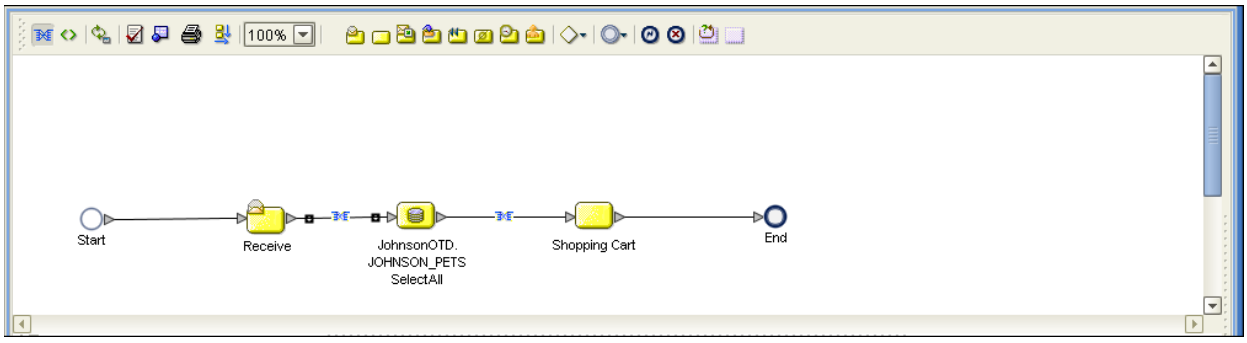


Figure 32 shows the definition of the Where clause for the SelectAll operation.

Figure 32 SelectAll Input

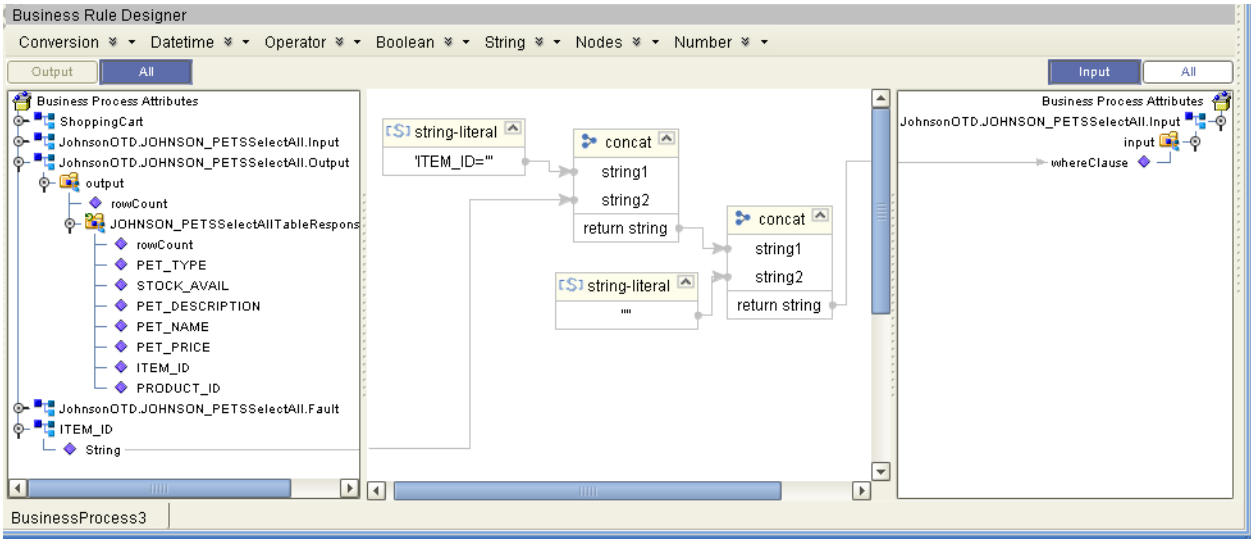
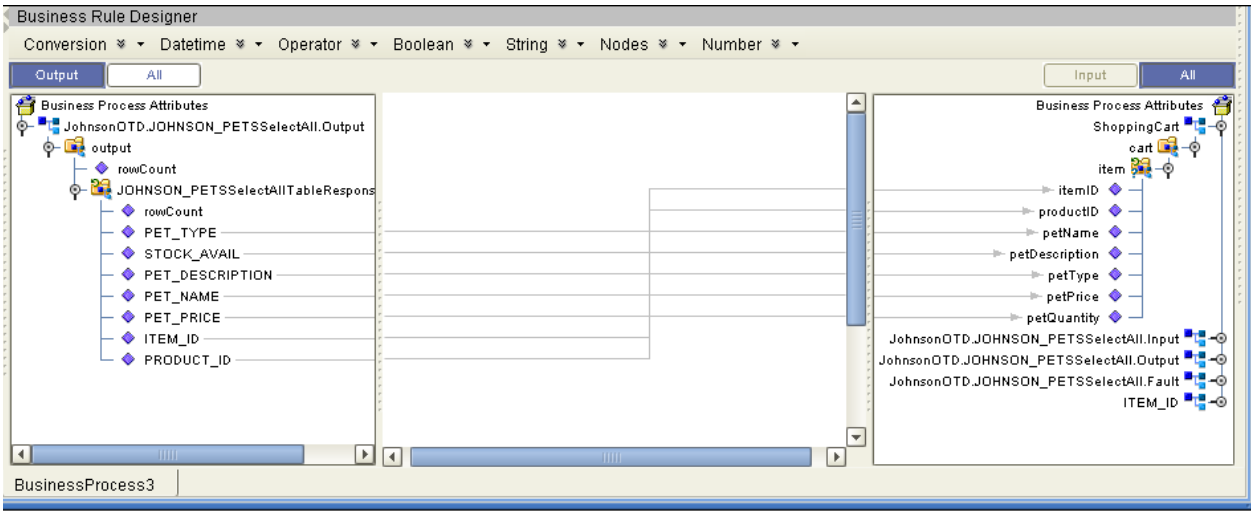


Figure 33 shows the definition of the output for the SelectAll operation. For each row selected during the operation, the shopping cart shows the columns of those rows as defined here.

Figure 33 SelectAll Output



SelectMultiple

The input to a SelectMultiple operation is the number of rows to be selected and a Where clause. The number of rows indicates how many rows the SelectMultiple operation returns. The Where clause defines the criteria to which rows must adhere to be returned.

Figure 34 shows a sample eInsight Business Process using the SelectMultiple operation. In this process, the SelectMultiple operation returns the first two rows where the ITEM_ID matches the selected ITEM_ID to the shopping cart.

Figure 34 SelectMultiple Sample Business Process

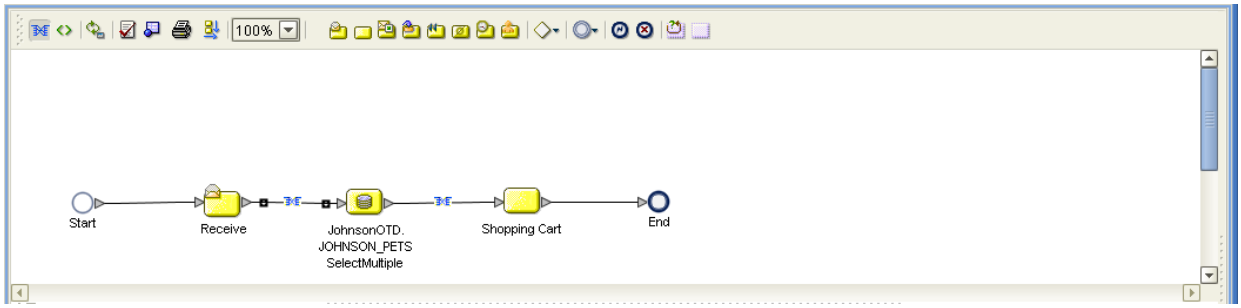


Figure 35 shows the definition of the input for the SelectMultiple operation, consisting of the definition of the number of rows and the Where clause. You can also use an empty string or Item_ID='123'.

Figure 35 SelectMultiple Input

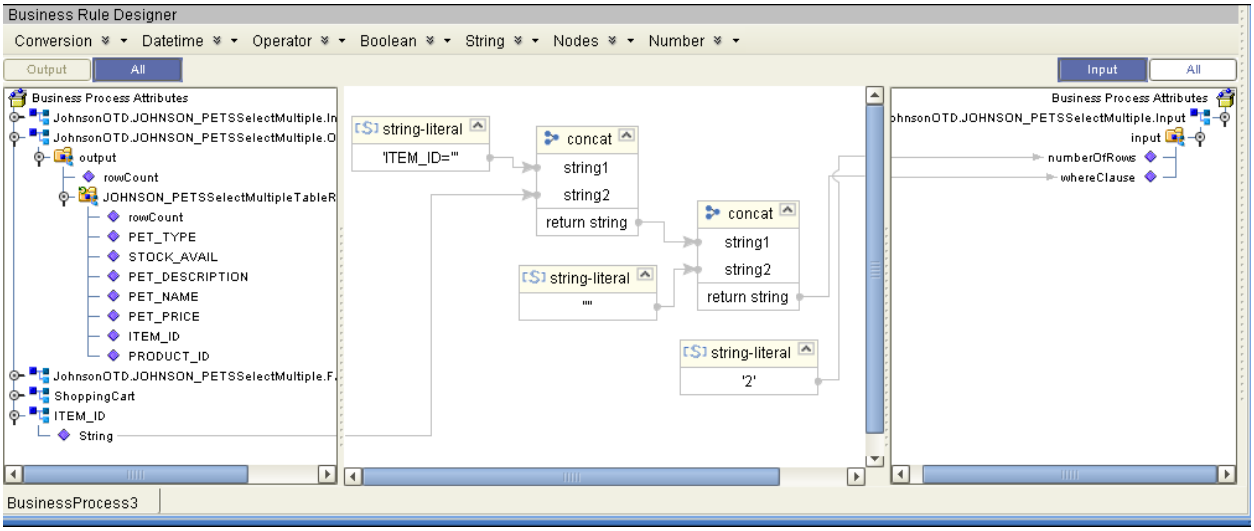
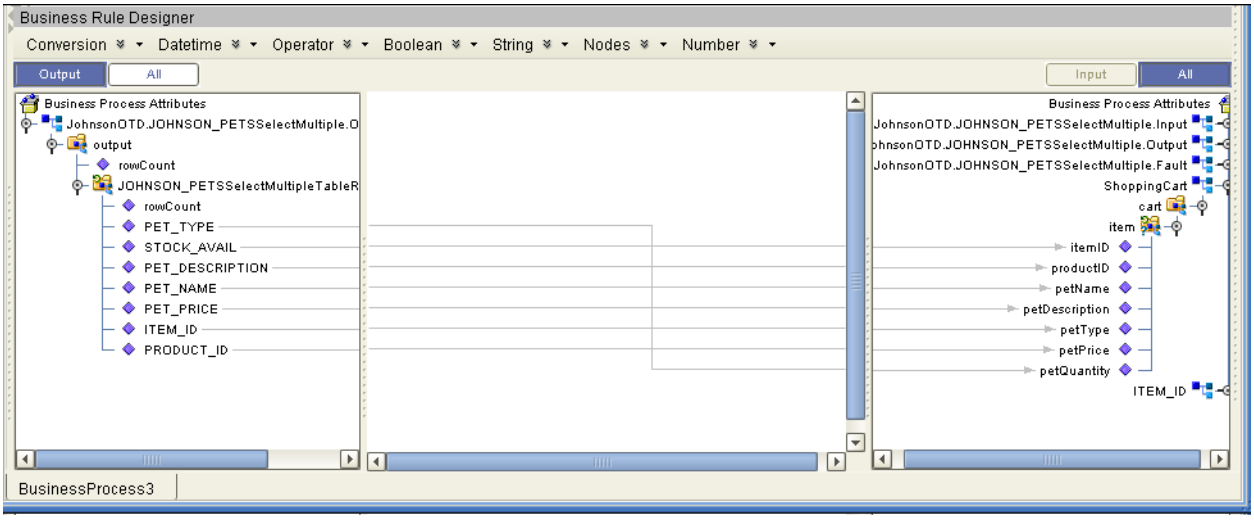


Figure 36 shows the definition of the output for the SelectMultiple operation. For each row selected during the operation, the shopping cart shows the columns of those rows as defined here.

Figure 36 SelectMultiple Output



SelectOne

The input to a SelectOne operation is a Where clause. The Where clause defines the criteria to which rows must adhere to be selected for the operation. In the SelectOne operation, the first row that fits the criteria is returned.

Figure 37 shows a sample eInsight Business Process using the SelectOne operation. In this process, the SelectOne operation returns the first row where the ITEM_ID matches the specified ITEM_ID to the shopping cart.

Figure 37 SelectOne Sample Business Process

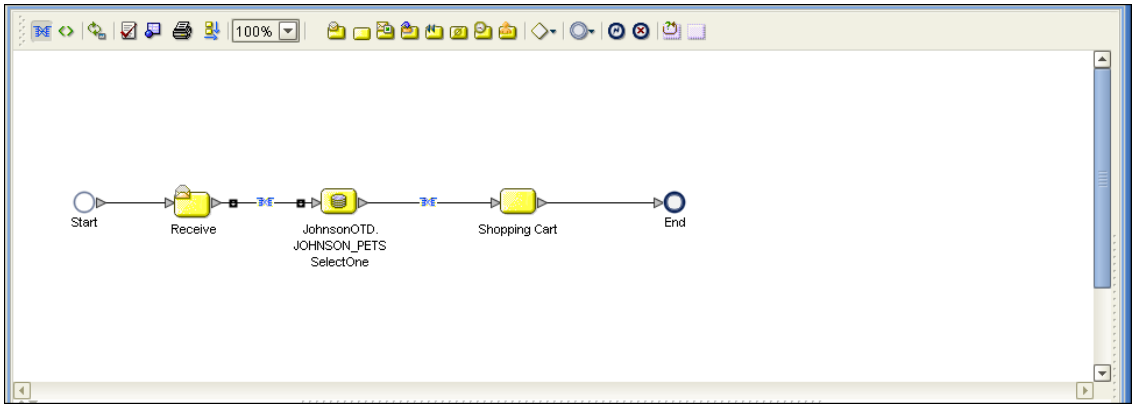


Figure 38 shows the definition of the Where clause for the SelectOne operation.

Figure 38 SelectOne Input

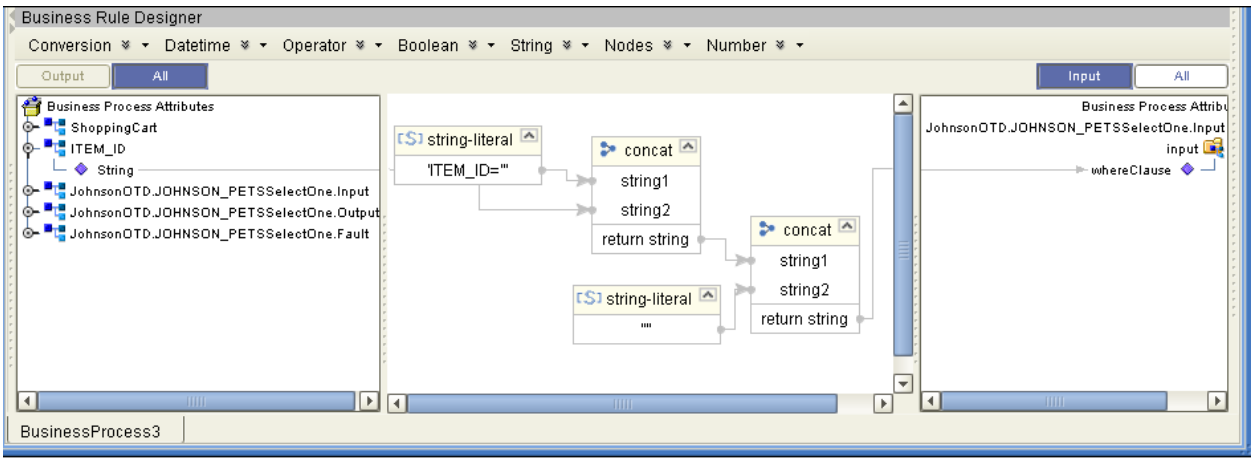
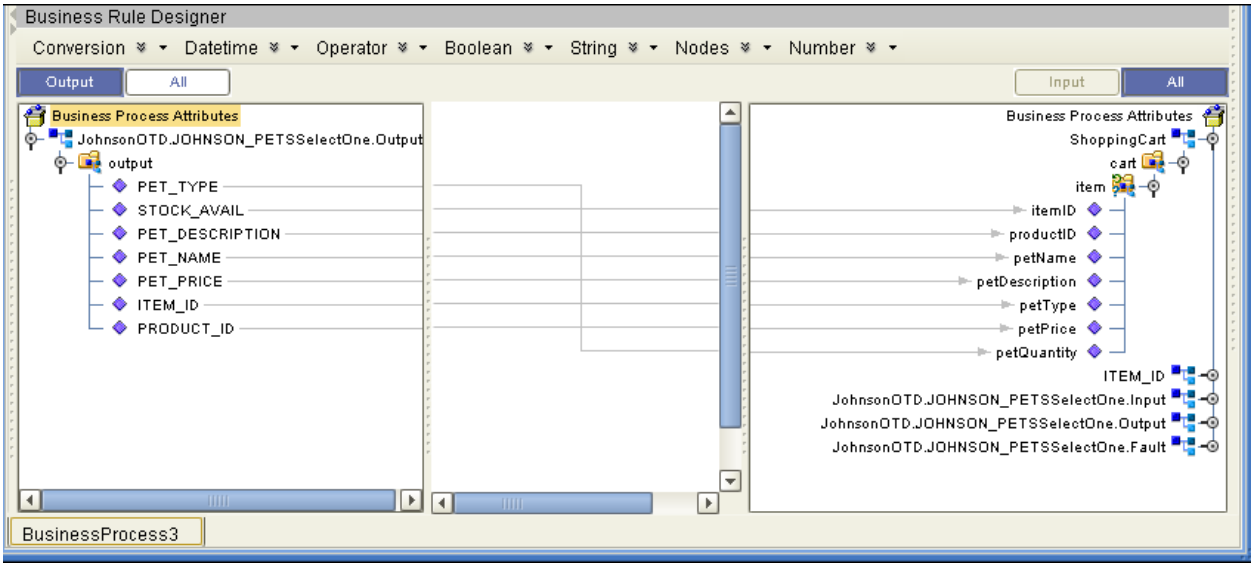


Figure 39 shows the definition of the output for the SelectOne operation. For the first row selected during the operation, the shopping cart shows the columns of that row as defined here.

Figure 39 SelectOne Output



Insert

The Insert operation inserts a row. The input to an Insert operation is a Where clause. The Where clause defines the criteria to which rows must adhere to be selected for the operation. In the Insert operation, the first row that fits the criteria is returned.

Figure 40 shows a sample eInsight Business Process using the Insert operation. In this process, the operation inserts a new row into the database to accommodate a new item provided by a vendor.

Figure 40 Insert Sample Business Process

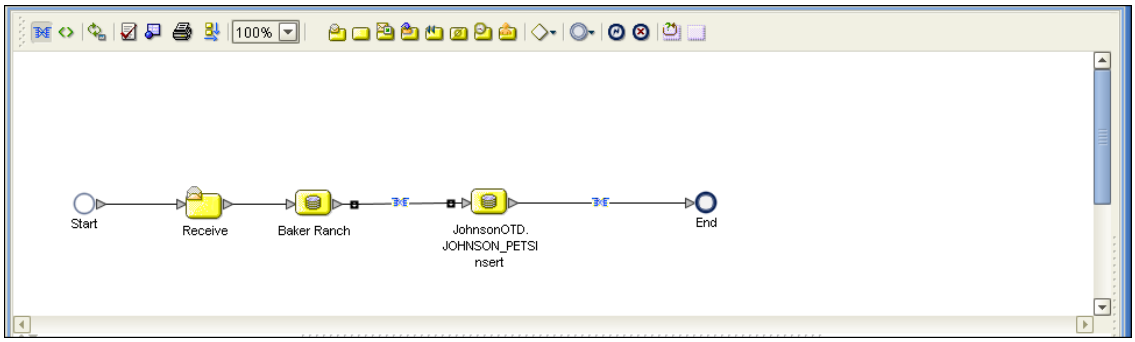


Figure 41 shows the definition of the input for the Insert operation.

Figure 41 Insert Input

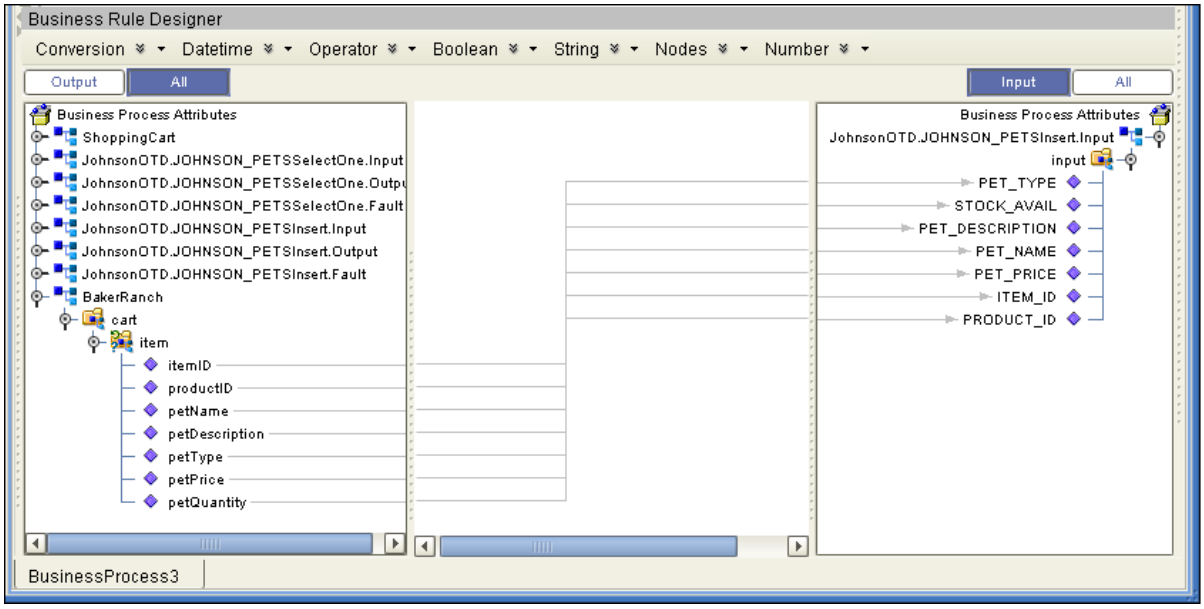
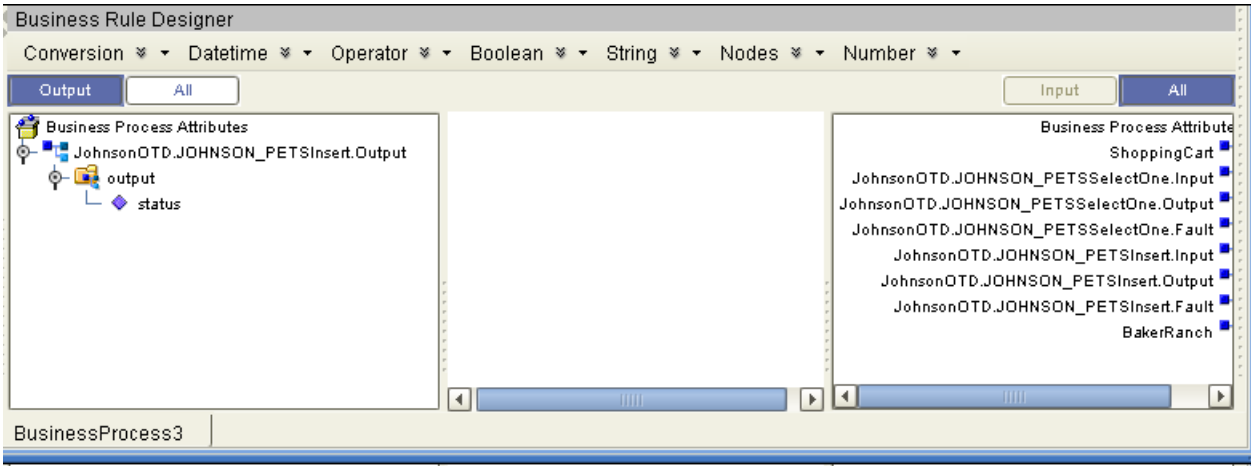


Figure 42 shows the output of the Insert operation, which is a status indicating the number of rows created.

Figure 42 Insert Output



Update

The Update operation updates rows that fit certain criteria defined in a Where clause.

Figure 43 shows a sample eInsight Business Process using the Update operation. In this process, the operation updates the ITEM_ID, for all items with a given name, to ESR_6543.

Figure 43 Update Sample Business Process

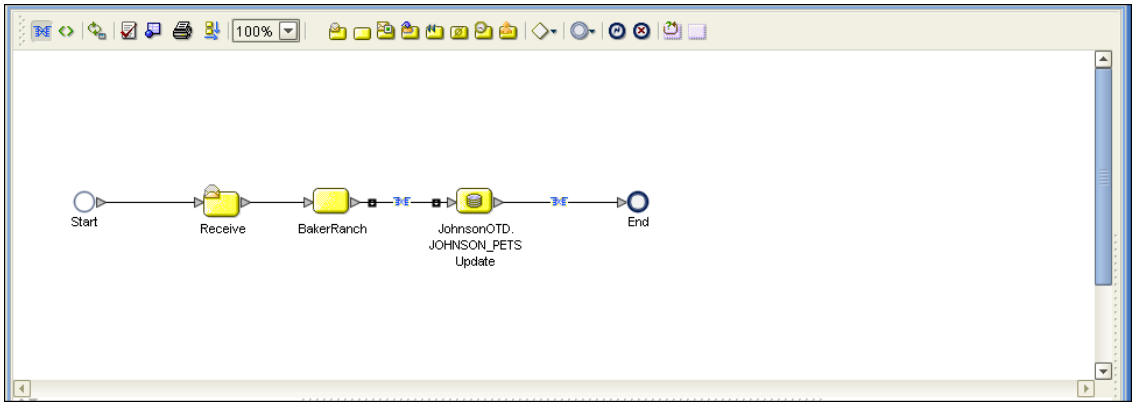


Figure 44 shows the definition of the Where clause for the Update operation.

Figure 44 Update Input

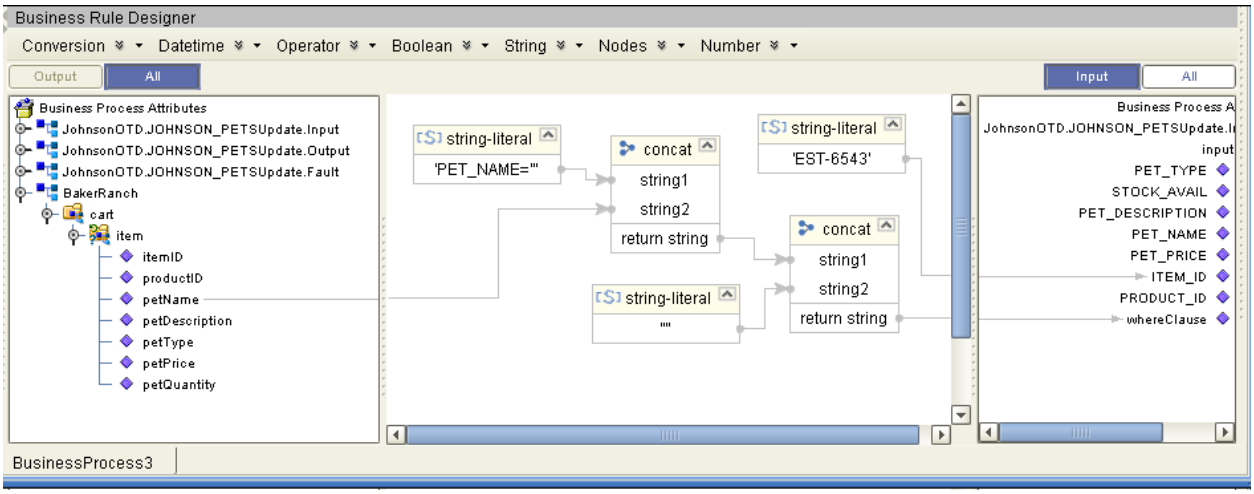
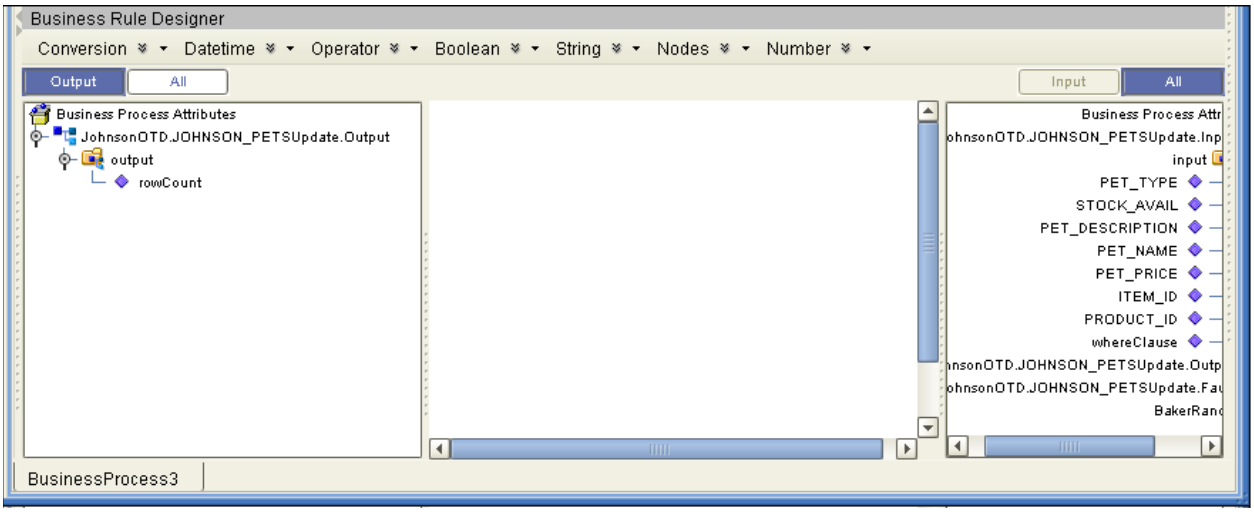


Figure 45 shows the output of the Update operation, which is a status indicating the number of rows updated.

Figure 45 Update Output



Delete

The Delete operation deletes rows that match the criteria defined in a Where clause. The output is a status of how many rows where deleted.

The figure below shows a sample eInsight Business Process using the Delete operation. In this process, the operation deletes rows with a certain product ID from the shopping cart.

Figure 46 Delete Sample Business Process

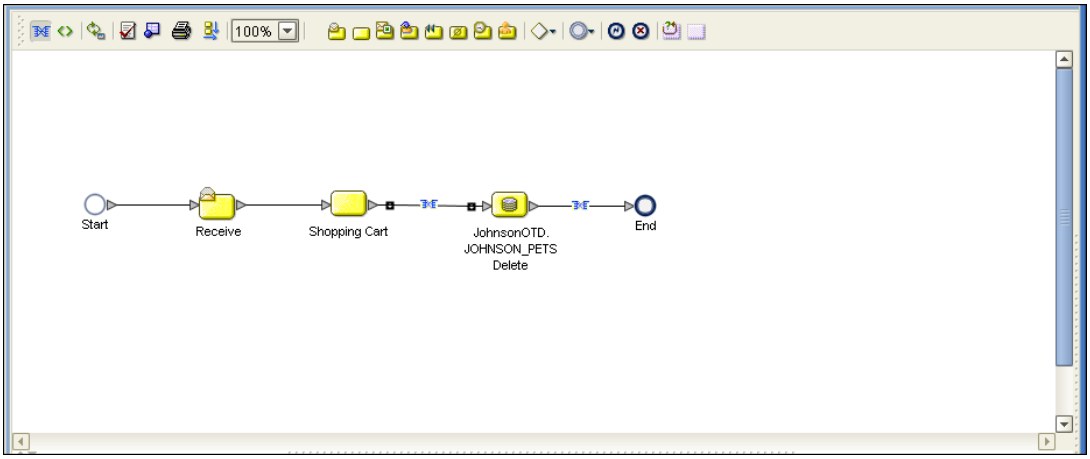


Figure 47 below shows the definition of the Where clause for the Delete operation.

Figure 47 Delete Input

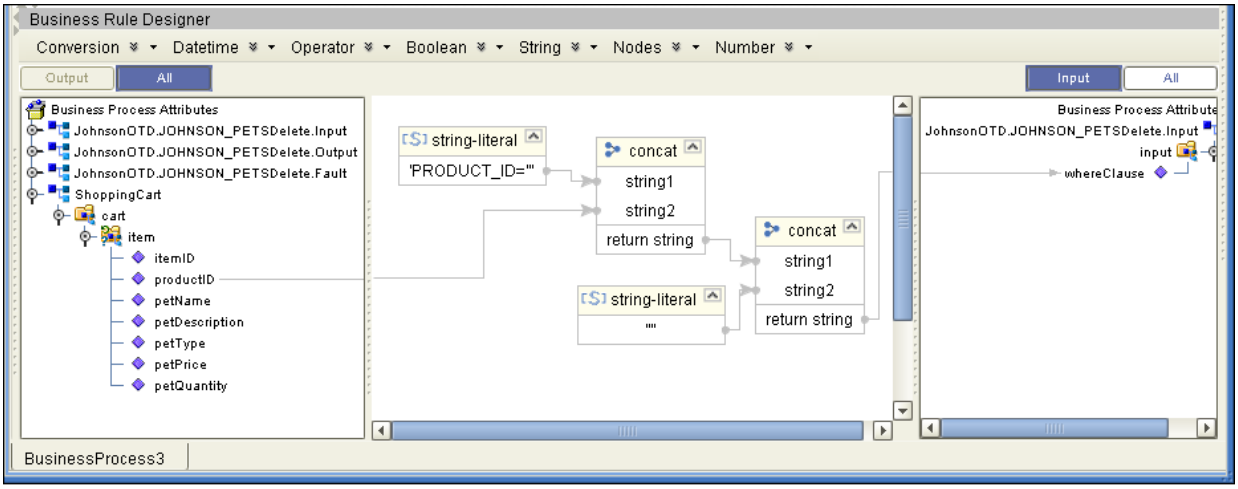
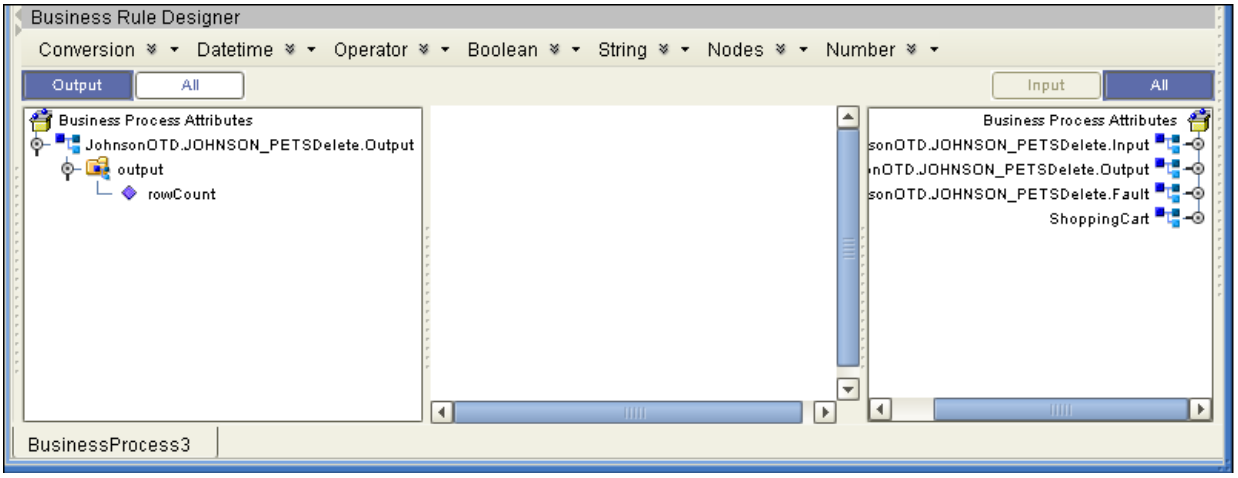


Figure 48 shows the output of the Delete operation, which is a status indicating the number of rows deleted.

Figure 48 Delete Output



5.3 Using the Sample Project in eGate

Follow the instructions given in **Importing the Sample Project** on page 48 importing the **Ora_JCE_Sample.zip** file.

5.3.1. Working with the Sample Project in eGate

This sample project updates the hire_date column within the DBEmployee table by selecting the employee whose employee number is equal to 800 and then updates the record.

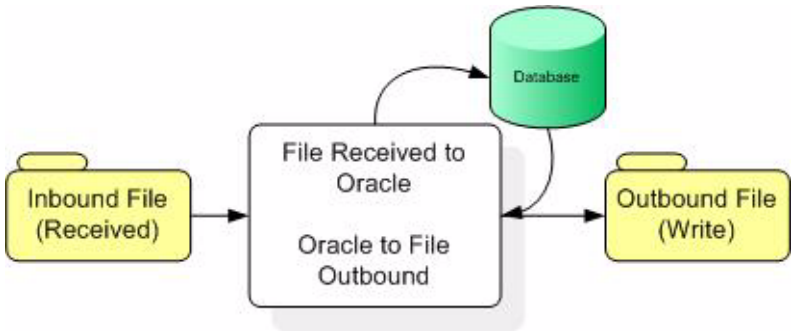
The data used for this project is within a table called DBEmployee. The table contains the following columns:

Table 4 Sample project data

Column Name	Mapping	Data Type	Data Length
EMP_NO	Empno	varchar2	10
LAST_NAME	Lastname	varchar2	30
FIRST_NAME	Firstname	varchar2	30
SS_NUMBER	SSnumber	varchar2	20
HIRE_DATE	HireDate	varchar2	12

The sample project consists of an input file containing data that is passed into a collaboration and out to the Oracle database from which data is retrieved and passed back into the collaboration and then to an output file.

Figure 49 Database project flow



To work with the sample project, follow the instructions given in the *eGate Integrator Tutorial*.

Note: *Outbound database eWays are available when using a JCE Collaboration. To poll the database, you must use the Scheduler.*

5.3.2. Configuring the eWays

To configure the Inbound File eWay:

1. In the Connectivity Map canvas, double-click the eWay icon located between the **File1** and **JCESelect1** service.
2. In the resulting **Templates** window, select **Inbound File eWay** and click **OK**.
3. In the **Properties** window, enter the appropriate configurations for the Inbound File eWay. See the ***File eWay User's Guide*** for information on how to specifically configure the File eWay. For this sample, the default settings are used.
4. When you have completed your selections, click **OK**.

To configure the Outbound Oracle eWay:

1. In the Connectivity Map canvas, double-click the eWay icon located between the **JCESelect1** service and **Oracle1** database.
2. In the resulting **Templates** window, select **Outbound Oracle eWay** and click **OK**.
3. In the **Properties** window, enter the appropriate configurations for the Outbound Oracle eWay and click **OK**. See **Configuring the Oracle eWay Properties** on page 16. For this sample, the default settings are used.
4. When you have completed your selections, click **OK**.

To configure the Outbound File eWay:

1. In the Connectivity Map canvas, double-click the eWay icon located between the **JCESelect1** and **File2** service.
2. In the resulting **Templates** window, select **Outbound File eWay** and click **OK**.
3. In the **Properties** window, enter the appropriate configurations for the Outbound File eWay. See the ***File eWay User's Guide*** for information on how to specifically configure the File eWay. For this sample, change the Directory field to **<valid path**

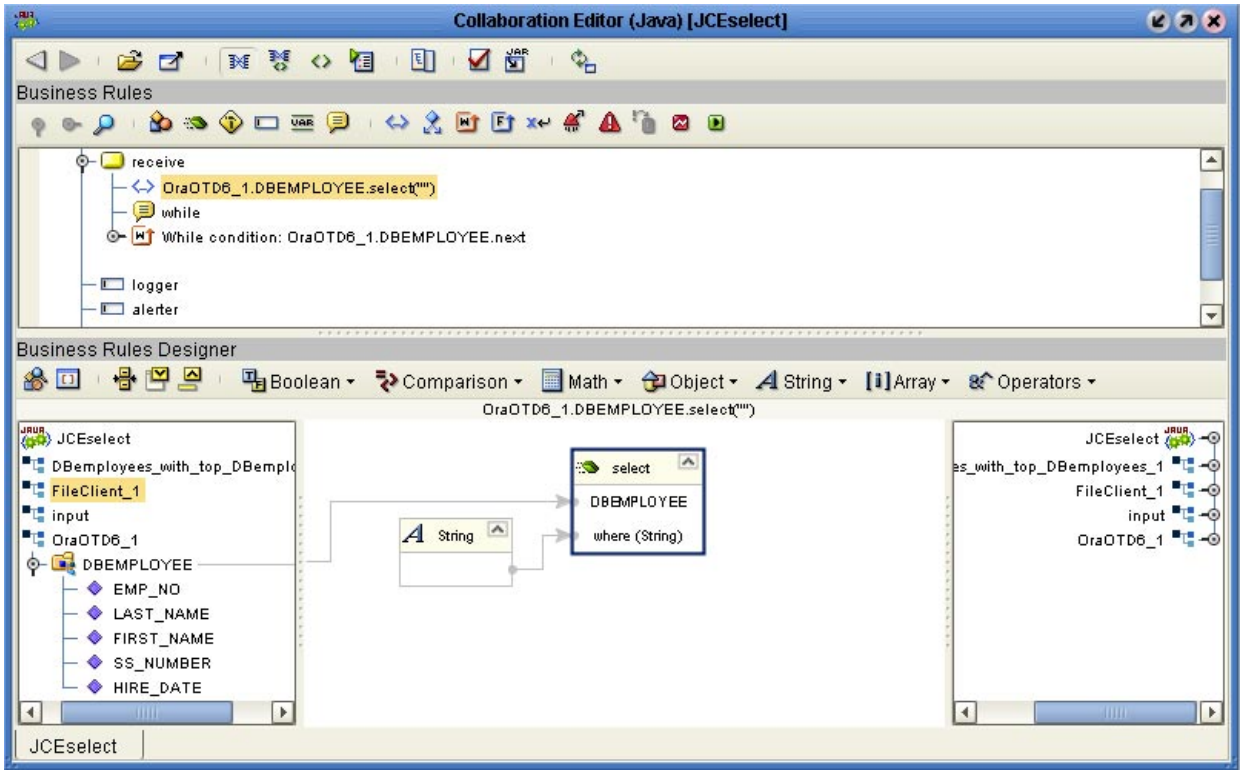
to the directory where the output file will be stored>. The Output File Name to **JCESelect_output%d.dat**. For the remaining parameters, the default settings are used.

4 When you have completed your selections, click **OK**.

Creating Rules Within the Collaboration

1 In the **Business Rules Designer** window's left pane, expand the **OraOTD6** node and the **DBEMPLOYEE** sub-node by selecting them and double-clicking.

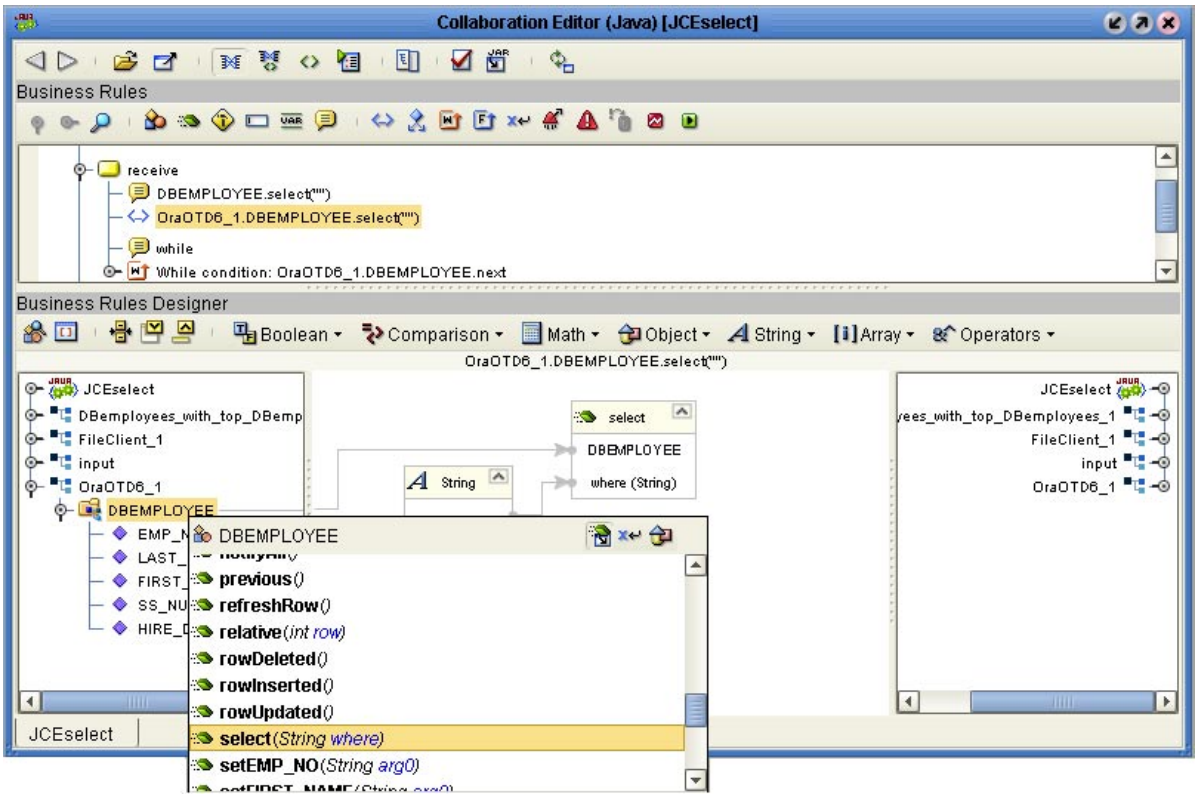
Figure 50 DBEMPLOYEE Select



2 To create a new rule, right-click on **DBEMPLOYEE** to open the Methods submenu.

3 From the **Select a method to call** submenu, choose the **select(String where)** method (as shown Figure 51).

Figure 51 Selecting a method



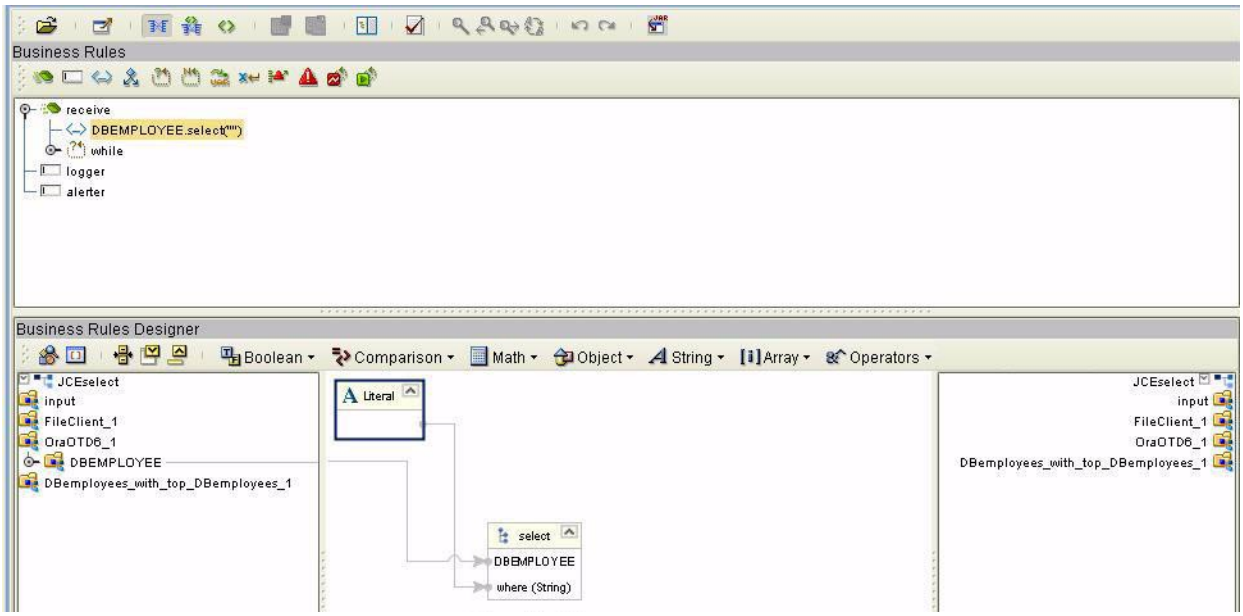
4 Next, you need to create a **Literal** by selecting the **Literal (A)** icon on the Business Rules Designer toolbar.

5 In the **Create Literal** popup window, select **String** from the drop-down list and click **OK**.

6 Place your cursor inside the **Literal** box, and while holding the left mouse key, drag a new connection line to the **where(String)** method node located next to the **select()** box (shown in Figure 52).

Note: *A **Literal** with no content implies that the Select clause has no WHERE clause and all rows in the table will be selected.*

Figure 52 Literal mapping



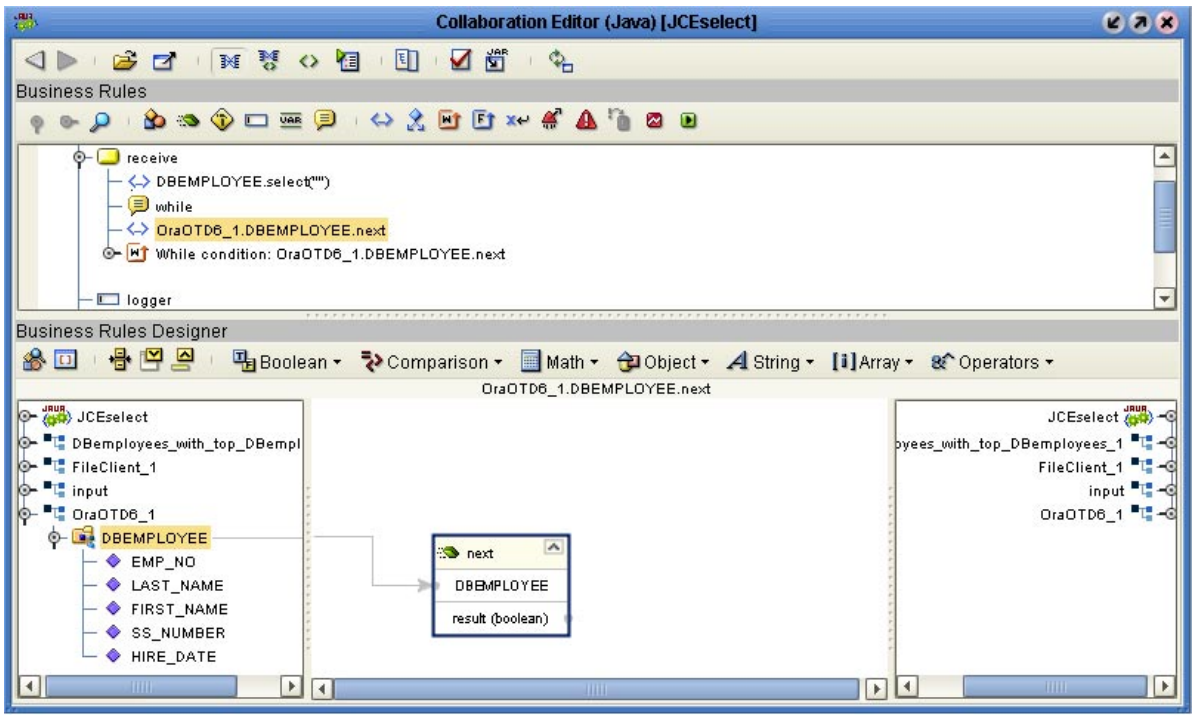
A new rule is created.

```
//DBEMPLOYEE.select("")
        OraOTD6_1.getDBEMPLOYEE().select("")
```

7 In the Business Rules canvas, highlight the `<_>DBEMPLOYEE.select("")` rule. Click the **while()** icon located on the toolbar. A new **while() condition** is waiting to be created within the list.

8 In the **Business Rules** list, highlight the condition. Select **DBEMPLOYEE** located under the **OraOTD6_1** node. Right-click and select the next() method from the submenu (see Figure 53).

Figure 53 next()

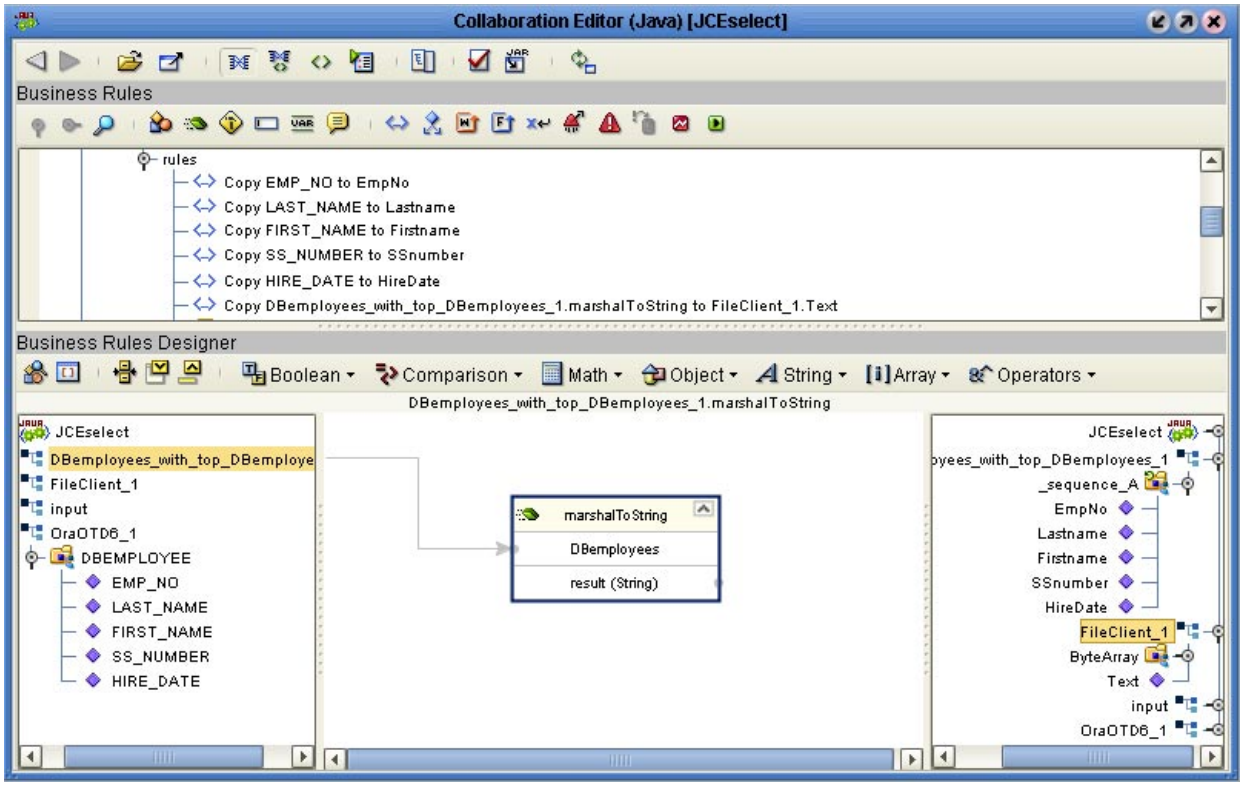


9 In the Business Rules list, highlight the **New Rule** located under the **DBEMPLOYEE.next** condition. In the left pane of the Business Rules canvas, expand the **DBEMOPLOYEE** subnode located under the **OraOTD6_1** OTD node to expose the database columns. In the right side of the Business Rules canvas, expand the **DBemployees_with_top_DBemployees_2** OTD node. Expand the **_sequence_A** node exposing the table's columns. Use your mouse to drag and drop the following fields from the left pane under the **DB_EMPLOYEE** subnode to the right pane under **_sequence_A**:

EMP_NO	---------->	EmpNo
LAST_NAME	---------->	Lastname
FIRST_NAME	--------- >	Firstname
SS_NUMBER	---------->	SSnumber
HIRE_DATE	---------->	HireDate

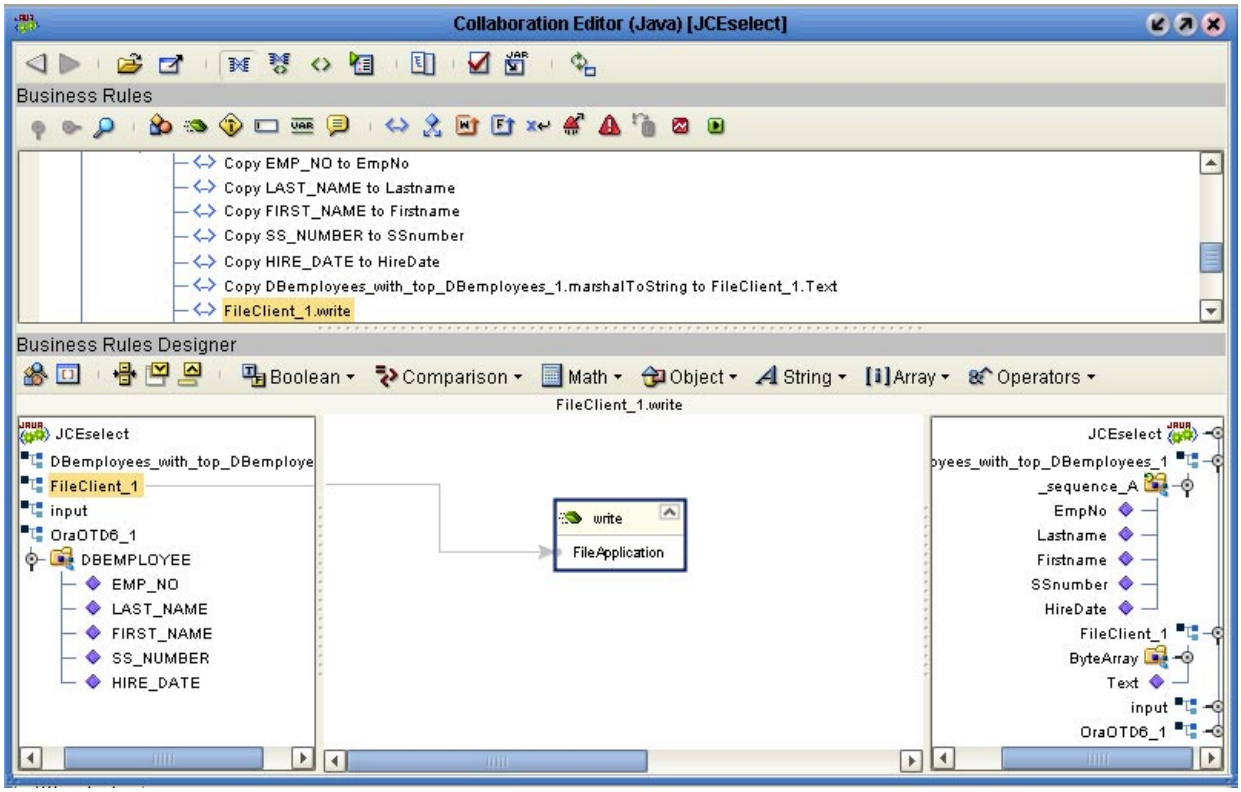
10 In the Business Rules list, highlight **Copy Hire_Date to HireDate**. In the Business Rules Designer's left pane, right-click on the **DBemployees_with_top_DBemployees_1** node. From the submenu, select and click **Select a method to call**. From the submenu, select and double-click **marshalToString()** (see Figure 54).

Figure 54 marshalToString()



11 In the Business Rules list, highlight **DBemployees_with_top_DBemployees_1.marshalToString**. In the Business Rules Designer's left pane, right-click on the **FileClient_1** node. From the submenu, select and click **Select a method to call**. From the submenu, select and double-click **write()** (see Figure 55).

Figure 55 write()



12 Click **Save** to save the Collaboration.

5.3.3. Creating the External Environment

The components of the Sample project include:

- Inbound File eWay
- Outbound File eWay
- Oracle eWay

To create the external environment for the Sample project:

In the Environment Explorer, highlight and right-click the Oracle eWay and select **Properties**. Enter the configuration information required for your Outbound Oracle eWay. See **Setting the Properties in the Outbound eWay Environment** on page 21.

5.3.4 Deploying a Project

For instructions on how to deploy a project, please refer to the "*eGate Integrators User's Guide*."

5.3.5. Running the Sample

For instructions on how to run the Sample project, see the "*eGate Integrator Tutorial*."

Once the process has completed, the Output file in the target directory, configured in the Outbound File eWay, contains all records retrieved from the database in an .xml format.

5.4 Supported Data Types

The Oracle eWay supports the following data types:

Table 5 Standard Data Types Supported by the Oracle eWay

Data Type	Description	Column Length
Number	Variable-length numeric data. Maximum precision p and/or scale s is 38.	Variable for each row. The maximum space required for a given column is 21 bytes per row.
VarChar2	Variable-length character data, with maximum length size bytes or characters.	Variable for each row, up to 4000 bytes per row. Consider the character set (single-byte or multi-byte) before setting size. You must specify a maximum size.
Char	Fixed-length character data of length size bytes or characters.	Fixed for every row in the table (with trailing blanks); maximum size is 2000 bytes per row, default size is one byte per row. Consider the character set (single-byte or multi-byte) before setting size.
Raw	Variable-length raw binary data.	Variable for each row in the table, up to 2000 bytes per row. You must specify a maximum size. Provided for backward compatibility.
Long	Variable-length character data.	Variable for each row in the table, up to 2^{32} - one bytes, or two gigabytes, per row. Provided for backward compatibility.
Clob	Built-in data type that stores a Character Large Object as a column value in a row of a database table.	Up to 2^{32} - one bytes, or 64k.

Data Type	Description	Column Length
Date	Fixed-length date and time data, ranging from Jan. 1, 4712 B.C.E. to Dec. 31, 4712 C.E.	Fixed at seven bytes for each row in the table. Default format is a string (such as `DD-MON-RR`) specified by the `NLS_DATE_FORMAT` parameter. Oracle expects a format of: "YYYY-MM-DD; hh:mm:ss.x".

5.5 Converting Data Types in the Oracle eWay

When working with data in the Oracle eWay OTD's, you may need to do a data conversion. The following tables show input/output data for data conversion:

Table 6 Insert and Update Operations Datatype Conversions (Text/String input data)

Oracle Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Int	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
Smallint	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
Number	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
Decimal*	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	147.78
Real	Double	**Double**: java.lang.Double.parseDouble(String)	147.78
Float	Double	**Double**: java.lang.Double.parseDouble(String)	147.78
Double	Double	**Double**: java.lang.Double.parseDouble(String)	147.78
Date	TimeStamp	**TimeStamp**: java.sql.TimeStamp.valueOf(String)	2003-08-11 11:47:39.0
TimeStamp	TimeStamp	**TimeStamp**: java.sql.TimeStamp.valueOf(String) ***Note:*** *To use the TimeStamp data type, (allowing milliseconds to be stored) you must use the Oracle 10g driver (Oracle JDBC version 10.1.0.2.0).*	11:47:39.001

Oracle Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Varchar2	String	Direct Assign	Any character
Char	String	Direct Assign	Any character
Long Char	String	Direct Assign	Any character
Raw	Byte[]	**String**: java.lang.String.getBytes()	Any character
Long Raw	Byte[]	**String**: java.lang.String.getBytes()	Any character
CLOB	Clob	See Appendix	Any character

Table 7 Select Operation Datatype Conversion (Text/String output data)

Oracle Data Type	OTD/Java Data Type	Methods To Use	Sample Data
Int	BigDecimal	**BigDecimal**: java.math.toString()	123
SmallInt	BigDecimal	**BigDecimal**: java.math.toString()	123
Number	BigDecimal	**BigDecimal**: java.math.toString()	123
Decimal	BigDecimal	**BigDecimal**: java.math.toString()	123.67
Real	Double	**Double**: java.lang.Double.toString(double)	123
Float	Double	**Double**: java.lang.Double.toString(double)	123
Double	Double	**Double**: java.lang.Double.parseDouble(String)	123
Date	TimeStamp	**TimeStamp**: java.sql.TimeStamp.valueof()	2003-08-11 11:47:39.0
TimeStamp	TimeStamp	**TimeStamp**: java.sql.TimeStamp.valueOf(String) ***Note:*** *To use the TimeStamp data type, (allowing milliseconds to be stored) you must use the Oracle 10g driver (Oracle JDBC version 10.1.0.2.0).*	11:47:39.001
Varchar2	String	Direct Assign	Any characters

Oracle Data Type	OTD/Java Data Type	Methods To Use	Sample Data
Char	String	Direct Assign	Any Characters
Raw	Byte[]	N/A	N/A
Long Raw	Byte[]	N/A	N/A
CLOB	Clob	N/A	N/A

5.6 Using OTDs with Tables/Views and Stored Procedures

Tables/Views and Stored Procedures are manipulated through OTDs. Common operations include insert, delete, update, and query.

5.6.1. Data Types

Oracle tables support the following data types:

- **Real** - an approximate numeric data type.
- **Float** - a data type where all platforms have values of the least specified minimum precision.
- **CLOB** - a built-in data type that stores a Character Large Object as a column value in a row of a database table.

For all others, use the data types Float, Double, or CLOB and build them using a data type of "Other".

Note: *The Oracle driver does not support the boolean and PL/SQL RECORD datatypes in the Function and Stored Procedure.*

Long RAW for Prepared Statements and Stored Procedure support:

The following two parameters must be set prior to the Insert/Update/Delete statement.

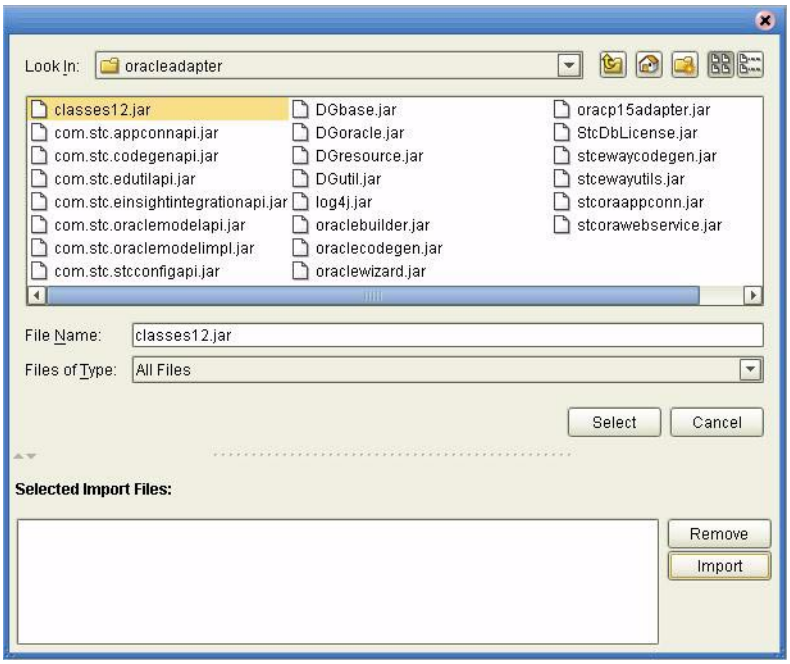
```
setConcurrencyToReadOnly()
setScrollTypeToForwardOnly()
```

Using CLOBs

To use a CLOB in the Oracle eWay, do the following:

1. In the Enterprise Designer, right-click on the project and select **New**. From the submenu, select **File**.
2. Navigate to the **classes12.jar file**, `<Client_eDesigner>\usrdir\modules\ext\oracleadapter\classes12.jar` using the Enterprise Designer's Project File Import feature (see Figure 56).

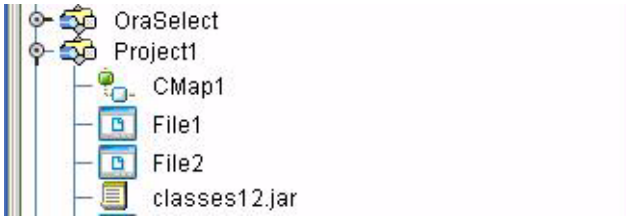
Figure 56 Importing Classes12.jar



3 Click **Select**.

4 Click **Import**. The **classes12.jar** file appears, as shown in Figure 57.

Figure 57 Classes12.jar in a Project

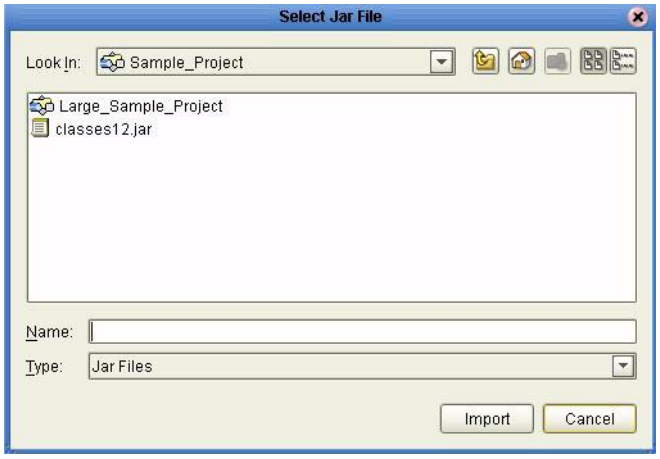


5 To load the **classes12.jar** file into your Java Collaboration, select the **Import JAR File** button. Click **Add** in the **Add/Remove Jar Files** dialog box to add the Jar files (see Figure 58).

Figure 58 Add/Remove Jar Files

6 In the **Select Jar File** dialog box, select the **classes12.jar** file and click **Import** (see Figure 59).

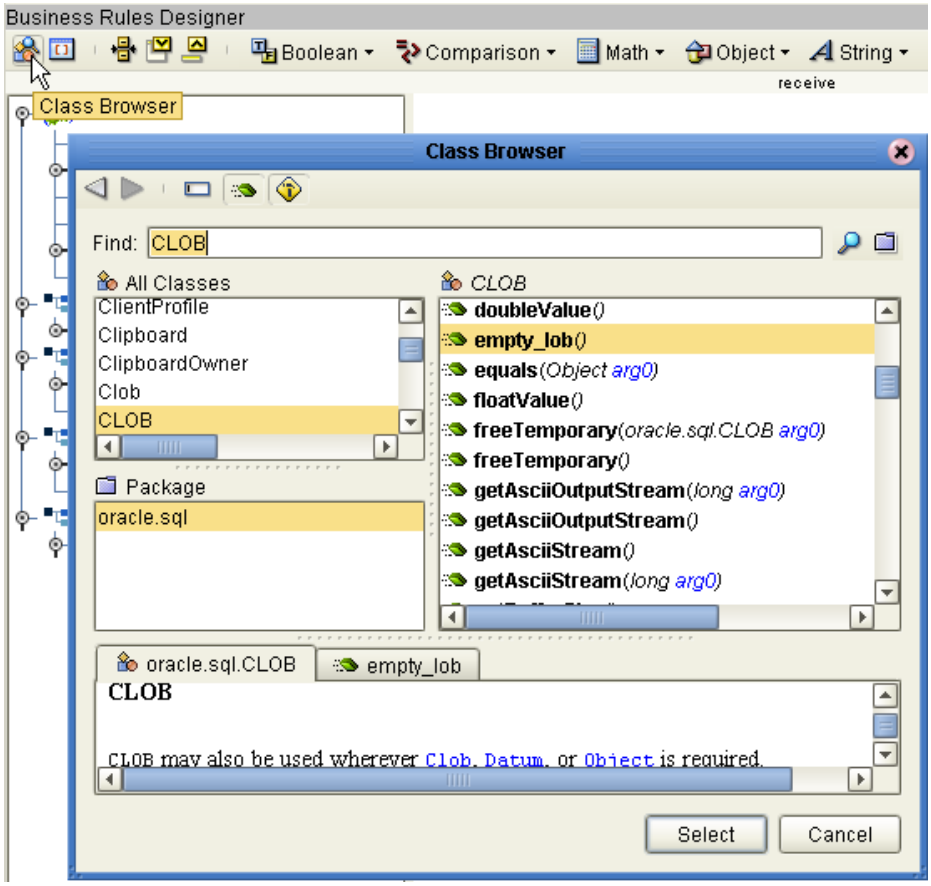
Figure 59 Select Jar File



7 In the **Add/Remove Jar Files** dialog box, click **Close**.

8 Call the CLOB method by clicking the **Class Browser** button, in the Business Rules Designer. The **Class Browser** dialog box appears. (see Figure 60).

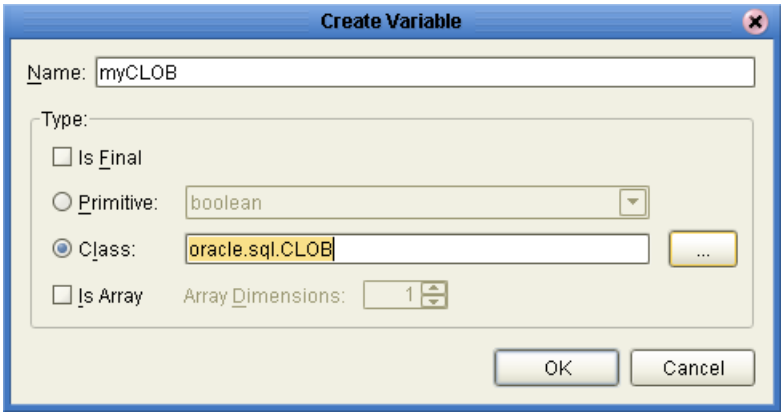
Figure 60 Call the CLOB Method



9 Select **empty_lob** from the list of CLOB variables and click **Select**.

10 Create a local variable by clicking the **Local Variable** button, on the Business Rules toolbar. The **Create Variable** dialog box appears. (see Figure 61).

Figure 61 Create a Local Variable

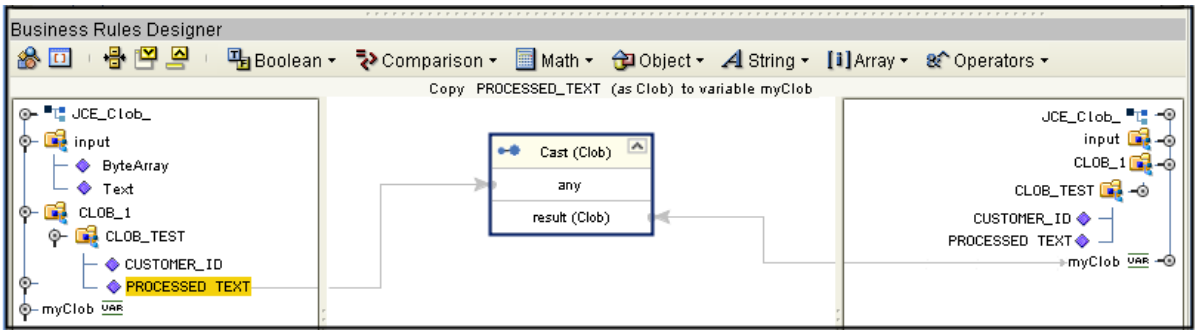


11 Name the variable **myCLOB**, select the Class type, and choose CLOB as the Class type.

12 Click **OK** to create the variable.

13 In the Business Rules Designer, drag the **CLOB** to the Local Variable using the Cast method. Click **Yes** when the Incompatible DataType warning appears. (see Figure 62).

Figure 62 Drag CLOB to the Local Variable



14 Use the CLOB putString method to assign 1 to Arg().

15 In the Java Collaboration Editor, the Java code resembles the following:

```
public void receive( com.stc.connector.appconn.file.FileTextMessage
input,cLOB.CLOBOTD CLOB_1 )
throws Throwable
{
    //@map:CLOB_1.getCLOB_TEST.insert
        CLOB_1.getCLOB_TEST().insert();

        //@map:Copy java.math.BigDecimal.valueOf(9876) to CUSTOMER_ID
        CLOB_1.getCLOB_TEST().setCUSTOMER_ID(
java.math.BigDecimal.valueOf( 9876 ) );

        //@map:Copy oracle.sql.CLOB.empty_lob to PROCESSED_TEXT
        CLOB_1.getCLOB_TEST().setPROCESSED_TEXT(
oracle.sql.CLOB.empty_lob() );

        //@map:CLOB_TEST.insertRow
        CLOB_1.getCLOB_TEST().insertRow();

//@map:CLOB_1.getCLOB_TEST.select("customer_id = 9876 for update")
       CLOB_1.getCLOB_TEST().select( "customer_id = 9876 for update"
);
//If
        if (CLOB_1.getCLOB_TEST().next()) {
            //@map:oracle.sql.CLOB myClob;
            oracle.sql.CLOB myClob;

            //@map:Copy cast PROCESSED_TEXT to oracle.sql.CLOB to
myClob
            myClob = (oracle.sql.CLOB)
CLOB_1.getCLOB_TEST().getPROCESSED_TEXT();

           //@map:myClob.putString(1,Text)
            myClob.putString( 1,input.getText() );

            //@map:CLOB_TEST.updateRow
            CLOB_1.getCLOB_TEST().updateRow();

}
    }
```

5.6.2. Table/Views

A table OTD represents a database table. It consists of fields and methods. Fields correspond to the columns of a table while methods are the operations that you can apply to the OTD. This enables you to perform query, update, insert, and delete SQL operations in a table.

By default, the Table OTD only has UpdatableConcurrency and ScrollTypeForward. You can specify the type of result returned by the select() method by using:

- SetConcurrencytoUpdatable
- SetConcurrencytoReadOnly
- SetScrollTypetoForwardOnly
- SetScrollTypetoScrollSensitive
- SetScrollTypetoInsensitive

If you use the methods listed above, they must be called before executing the select() method.

For example,

```
getDBEmp().setConcurToUpdatable();
getDBEmp().setScroll_TypeToScrollSensitive();
getDBEmp().getDB_EMPLOYEE().select("");
```

The Query Operation

To perform a query operation on a table:

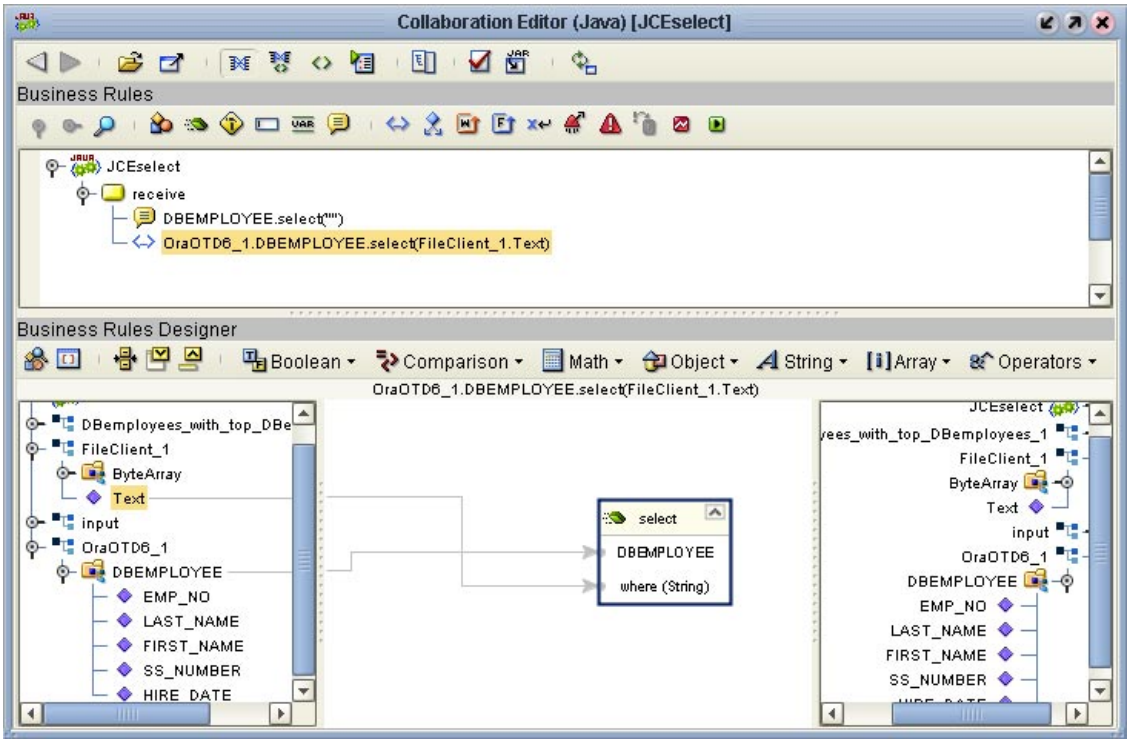
1. Execute the **select()** method with the "**where**" clause specified if necessary.
2. Loop through the ResultSet using the **next()** method.
3. Process the return record within a **while()** loop.

For example:

```
//Table_OTD_1.getDB_EMPLOYEE().select(Text)
    Table_OTD_1.getDB_EMPLOYEE().select( read.getText() );

//while Table_OTD_1.getDB_EMPLOYEE.next
    while (Table_OTD_1.getDB_EMPLOYEE().next()) {
//Copy LAST_NAME to LAST_NAME
      Table_OTD_1.getDB_EMPLOYEE().setLAST_NAME(
Table_OTD_1.getDB_EMPLOYEE().getLAST_NAME() );
}
```

Figure 63 Select()



The Insert Operation

To perform an insert operation on a table:

1. Execute the **insert()** method and assign a field.
2. Insert the row by calling **insertRow().**

   This example inserts an employee record.

```
//Table_OTD_1.getDB_EMPLOYEE.insert
    Table_OTD_1.getDB_EMPLOYEE().insert();
//Copy EMP_NO to EMP_NO
   insert_DB_1.getInsert_new_employee().setEmployee_no(
   java.lang.Integer.parseInt(
     employeedb_with_top_db_employee_1.getEmployee_no() ) );

//@map:Copy Employee_lname to Employee_Lname
   insert_DB_1.getInsert_new_employee().setEmployee_Lname(
   employeedb_with_top_db_employee_1.getEmployee_lname() );

//@map:Copy Employee_fname to Employee_Fname
   insert_DB_1.getInsert_new_employee().setEmployee_Fname(
   employeedb_with_top_db_employee_1.getEmployee_fname() );

//@map:Copy java.lang.Float.parseFloat(Rate) to Rate
   insert_DB_1.getInsert_new_employee().setRate(
   java.lang.Float.parseFloat(
   employeedb_with_top_db_employee_1.getRate() ) );

//@map:Copy java.sql.Timestamp.valueOf(Update_date) to Update_date
   insert_DB_1.getInsert_new_employee().setUpdate_date(
```

```
java.sql.Timestamp.valueOf(
   employeedb_with_top_db_employee_1.getUpdate_date() ) );
   Table_OTD_1.getDB_EMPLOYEE().insertRow();

}
```

The Update Operation

To perform an update operation on a table:

1. Execute the **update()** method.
2. Using a while loop together with **next()**, move to the row that you want to update.
3. Assign updating value(s) to the fields of the table OTD.
4. Update the row by calling **updateRow()**.

   In this example, we move to the third record and update the SALES_ORDERS Table and Cust_Name, and Cust_phone columns.

```
//SalesOrders_with_top_SalesOrders_1.unmarshalFromString(Text)
  SalesOrders_with_top_SalesOrders_1.unmarshalFromString(
  input.getText() );

//DB_sales_orders_1.getSALES_ORDERS.update("SO_num =99")
  DB_sales_orders_1.getSALES_ORDERS().update( "SO_num ='01'" );

//while
  while (DB_sales_orders_1.getSALES_ORDERS().next()) {

//Copy SalesOrderNum to SO_num
  DB_sales_orders_1.getSALES_ORDERS().setSO_num(
  SalesOrders_with_top_SalesOrders_1.getSalesOrderNum() );

//Copy CustomerName to Cust_name
  DB_sales_orders_1.getSALES_ORDERS().setCust_name(
  SalesOrders_with_top_SalesOrders_1.getCustomerName() );

//Copy CustomerPhone to Cust_phone
  DB_sales_orders_1.getSALES_ORDERS().setCust_phone(
  SalesOrders_with_top_SalesOrders_1.getCustomerPhone() );

//DB_sales_orders_1.getSALES_ORDERS.updateRow
  DB_sales_orders_1.getSALES_ORDERS().updateRow();
}


   }
```

The Delete Operation

To perform a delete operation on a table:

1. Execute the **delete()** method.

   In this example, DELETE an employee.

```
//Table_OTD_1.getDB_EMPLOYEE.delete("EMP_NO =
'".concat(EMP_NO).concat("'"))
```

```
    Table_OTD_1.getDB_EMPLOYEE().delete( "EMP_NO = '".concat(
Table_OTD_1.getDB_EMPLOYEE().getEMP_NO() ).concat( "'" ) );
  }
```

5.6.3. The Stored Procedure

A Stored Procedure OTD represents a database stored procedure. Fields correspond to the arguments of a stored procedure, while methods are the operations that you can apply to the OTD, allowing you to execute a stored procedure. Remember that from within the Collaboration Editor, you can drag and drop nodes from the OTD into the Collaboration Editor.

Note: *The parameter name(s) for a packaged stored procedure cannot be the same as the table column name.*

Note: *There are 2 ways to insert NULL into a Prepared Statement or Stored Procedure parameters in JCD. In advance mode, you can set the field to NULL. In the GUI mode, you can pull down* ***Object*** *and you will see a NULL field.*

Executing Stored Procedures

The OTD represents the stored procedure "LookUpGlobal" with two parameters, an inbound parameter (INLOCALID) and an outbound parameter (OUTGLOBALPRODUCTID). These inbound and outbound parameters are generated by the DataBase Wizard and are represented in the resulting OTD as nodes. Within the Transformation Designer, you can drag values from the input parameters, execute the call, collect data, and drag the values to the output parameters.

Below are the steps for executing the stored procedure:

1. Specify the input values.
2. Execute the stored procedure.
3. Retrieve the output parameters, if any.

For example:

```
package Storedprocedure;

public class sp_jce
{

public com.stc.codegen.logger.Logger logger;

public com.stc.codegen.alerter.Alerter alerter;

public void receive( com.stc.connector.appconn.file.FileTextMessage
input,com.stc.connector.appconn.file.FileApplication
FileClient_1,employeedb.Db_employee
employeedb_with_top_db_employee_1,insert_DB.Insert_DBOTD insert_DB_1
)
  throws Throwable
    {

//@map:employeedb_with_top_db_employee_1.unmarshalFromString(Text)
```

```
   employeedb_with_top_db_employee_1.unmarshalFromString(
   input.getText() );
//@map:Copy java.lang.Integer.parseInt(Employee_no) to Employee_no
   insert_DB_1.getInsert_new_employee().setEmployee_no(
   java.lang.Integer.parseInt(
   employeedb_with_top_db_employee_1.getEmployee_no() ) );
//@map:Copy Employee_lname to Employee_Lname
   insert_DB_1.getInsert_new_employee().setEmployee_Lname(
   employeedb_with_top_db_employee_1.getEmployee_lname() );
//@map:Copy Employee_fname to Employee_Fname
   insert_DB_1.getInsert_new_employee().setEmployee_Fname(
   employeedb_with_top_db_employee_1.getEmployee_fname() );
//@map:Copy java.lang.Float.parseFloat(Rate) to Rate
   insert_DB_1.getInsert_new_employee().setRate(
   java.lang.Float.parseFloat(
   employeedb_with_top_db_employee_1.getRate() ) );
//@map:Copy java.sql.Timestamp.valueOf(Update_date) to Update_date
   insert_DB_1.getInsert_new_employee().setUpdate_date(
   java.sql.Timestamp.valueOf(
   employeedb_with_top_db_employee_1.getUpdate_date() ) );
//@map:Insert_new_employee.execute
   insert_DB_1.getInsert_new_employee().execute();
//@map:insert_DB_1.commit
   insert_DB_1.commit();
//@map:Copy "procedure executed" to Text
   FileClient_1.setText( "procedure executed" );
//@map:FileClient_1.write
   FileClient_1.write();
    }
}
```

Manipulating the ResultSet and Update Count Returned by Stored Procedure

For Stored Procedures that return ResultSets and Update Count, the following methods are provided to manipulate the ResultSet:

- enableResultSetOnly
- enableUpdateCountsOnly
- enableResultSetandUpdateCounts
- resultsAvailable
- next
- getUpdateCount
- available

Note: *Stored Procedure ResultSets are supported in Java collaborations only.*

Oracle stored procedures do not return records as ResultSets; instead, the records are returned through output reference cursor parameters. Reference Cursor parameters are essentially ResultSets.

The **resultsAvailable()** method, added to the OTD, simplifies the whole process of determining whether any results, whether they are update Counts or ResultSets, are available after a stored procedure has been executed. Although JDBC provides three

methods (**getMoreResults()**, **getUpdateCount()**, and **getResultSet()**) to access the results of a stored procedure call, the information returned from these methods can be quite confusing to the inexperienced Java JDBC programmer and they also differ between vendors. You can simply call **resultsAvailable()** and if Boolean true is returned, you can expect either a valid Update Count when **getUpdateCount()** is called and/or the next ResultSet has been retrieved and made available to one of the ResultSet nodes defined for the Stored Procedure OTD, when that node's **available()** method returns true.

Update Counts information that is returned from Stored Procedures is often insignificant. Process returned ResultSet information and avoid looping through all of the Update Counts. The following three methods control exactly what information is returned from a stored procedure call. The **enableResultSetsOnly()** method, added to the OTD allows only ResultSets to be returned and thus every **resultsAvailable()** called only returns Boolean true if a ResultSet is available. Likewise, the **enableUpdateCountsOnly()** method causes **resultsAvailable()** to return true only if an Update Count is available. The default case of the **enableResultSetsAndUpdateCount()** method enables both ResultSets and Update Counts to be returned.

Collaboration usability for a Stored Procedure ResultSet

You can use your mouse to drag and drop the Column data of the ResultSets from their OTD nodes to the Business Rules. Below is a code snippet that can be generated by the Collaboration Editor:

```
// resultsAvailable() true if there's an update count and/or a result
set available.
// note, it should not be called indiscriminantly because each time
the results pointer is
// advanced via getMoreResults() call.
while (getSPIn().getSpS_multi().resultsAvailable())
{
     // check if there's an update count
     if (getSPIn().getSpS_multi().getUpdateCount() > 0)
     {
          logger.info("Updated
"+getSPIn().getSpS_multi().getUpdateCount()+" rows");
     }
// each result set node has an available() method (similar to OTD's)
that tells the user
// whether this particular result set is available. note, JDBC does
support access to
// more than one result set at a time, i.e., cannot drag from two
distinct result sets
// simultaneously
     if (getSPIn().getSpS_multi().getNormRS().available())
     {
     while (getSPIn().getSpS_multi().getNormRS().next())
     {
     logger.info("Customer Id =
"+getSPIn().getSpS_multi().getNormRS().getCustomerId());
     logger.info("Customer Name =
"+getSPIn().getSpS_multi().getNormRS().getCustomerName());
     }
     if (getSPIn().getSpS_multi().getDbEmployee().available())
     {
     while (getSPIn().getSpS_multi().getDbEmployee().next())
     {
```

```
    logger.info("EMPNO =
"+getSPIn().getSpS_multi().getDbEmployee().getEMPNO());
    logger.info("ENAME =
"+getSPIn().getSpS_multi().getDbEmployee().getENAME());
    logger.info("JOB =
"+getSPIn().getSpS_multi().getDbEmployee().getJOB());
    logger.info("MGR =
"+getSPIn().getSpS_multi().getDbEmployee().getMGR());
    logger.info("HIREDATE =
"+getSPIn().getSpS_multi().getDbEmployee().getHIREDATE());
    logger.info("SAL =
"+getSPIn().getSpS_multi().getDbEmployee().getSAL());
    logger.info("COMM =
"+getSPIn().getSpS_multi().getDbEmployee().getCOMM());
    logger.info("DEPTNO =
"+getSPIn().getSpS_multi().getDbEmployee().getDEPTNO());
    }
}
```

Note: ***resultsAvailable()*** *and* ***available()*** *cannot be indiscriminately called because each time they move ResultSet pointers to the appropriate locations.*

Once the "**resultsAvailable()**" method has been called, the next result (if available) can be either a **ResultSet** or an **UpdateCount**, if the default "**enableResultSetsAndUpdateCount()**" was used.

Because of limitations imposed by some DBMSs, SeeBeyond recommends that for maximum portability, all of the results in a ResultSet object should be retrieved before OUT parameters are retrieved. Therefore, you must retrieve all ResultSet(s) and update counts first, followed by retrieving the OUT type parameters and return values.

The following list includes specific ResultSet behavior that you may encounter:

- The method **resultsAvailable()** implicitly calls **getMoreResults()** when it is called more than once. Do not call both methods in your Java code. If you do, there may be skipped data from one of the ResultSets when more than one ResultSet is present.
- The methods **available()** and **getResultSet()** cannot be used when mulitple ResultSets are open at the same time. Attempting to open more the one ResultSet at the same time closes the previous ResultSet. The recommended working pattern is:
  - Open one Result Set, ResultSet_1 and work with the data until you have completed your modifications and updates. Open ResultSet_2, (ResultSet_1 is now closed) and modify. When you have completed your work in ResultSet_2, open any additional ResultSets or close ResultSet_2.
- If you modify the ResultSet generated by the Execute mode of the Database Wizard, you need to make sure that the indexes match the stored procedure; if you do this, your ResultSet indexes are preserved.
- Generally, you do not need to call getMoreResults; you need to call it only if you do not want to use our enhanced methods and you want to follow the traditional JDBC calls on your own.

5.7 Creating or Editing an OTD from an Existing OTD

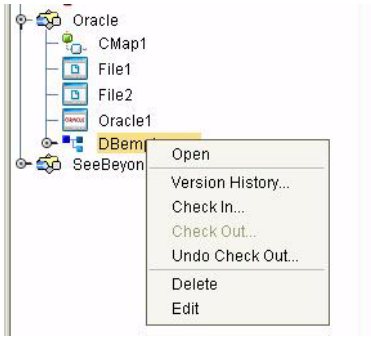
Editing an Existing OTD

When a minor change is needed for an Oracle OTD, there is no need to rebuild it from scratch. Another option is to edit the OTD. After making changes, save the OTD under a different name (temporarily), delete the old OTD, and then save the new OTD under the oldname. The mappings will not be lost, except for the one you have changed.

To edit an OTD, do the following:

1 In the Enterprise Explorer, right-click on the OTD. From the submenu, click **Edit**. See **Figure 64**.

Figure 64 OTD Edit Menu Item



The Database Wizard opens, allowing you to change the OTD. You may select additional Tables or edit the existing Tables.

Caution: *If during the edit process you delete a table that is included in a Collaboration, the Collaboration fails at run-time.*

You can also edit your Prepared Statements or Stored Procedures.

2 Save the new or edited OTD using **Save As** and give the OTD a new name.

Creating a New OTD from an Existing OTD

You can also create a new OTD from an existing OTD. Open the OTD that you would like to be the basis for your new OTD. Follow the steps in **Editing an Existing OTD** on page 83 to make your edits and save the OTD with a new name.

5.8 Using XA

When using XA, do not call the commit() method.

Note: *Due to a limitation with the Oracle JDBC Thin Driver, XA is not supported for multiple connections if your tables are in the same physical database. Because of this limitation, you must either use multipe tables (in the same database) within one OTD or use a different database for each table.*

5.9 Alerting and Logging

eGate provides an alerting and logging feature. This enables monitoring of messages and captures any adverse messages in order of severity, based on the configured severity level and higher. To enable Logging, please see the ***eGate Integrator User's Guide***.

Index

P

R

S

U

W