SeeBeyond ICAN Suite

Oracle eWay Intelligent Adapter User's Guide

Release 5.0.2

SEEBEYOND®

Version 20031222115450.

Contents

Chapter 1

Introducing the Oracle eWay

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

This Chapter Includes:

1.1 Overview

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.

1.2 Supported Operating Systems

The Oracle eWay is available on the following operating systems:

- Windows Server 2003, Windows XP SP1a, and Windows 2000 SP3
- Solaris 8 and 9
- HP Tru64 5.1A
- HP-UX 11.0 and 11i
- IBM AIX 5.1 and 5.2
- Red Hat Enterprise Linux AS 2.1
- Red Hat Linux 8 (Intel Version)

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.

In addition to the above listed Operating Systems, this eWay in outbound mode is supported on WebLogic Application Servers when using Java Collaborations only. See the eGate Integrator User's Guide for additional information regarding the running of this eWay on this Application Server.

1.3 System Requirements

The system requirements for the Oracle eWay are the same as for eGate Integrator. For information, refer to the *SeeBeyond ICAN Suite Installation Guide*. It is also helpful to review the **Readme.txt** for any additional requirements prior to installation. The **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 enable Web Services, you must install and configure the SeeBeyond ICAN Suite eInsight Business Process Manager.*

1.4 External System Requirements

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

Chapter 2

Installing the Oracle eWay

This chapter describes how to install the Oracle eWay.

This Chapter Includes:

2.1 Before Installing the eWay

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

2.2 Installing the Oracle eWay

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:

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

- On the Enterprise Manager, install the **OracleeWayDocs.sar** (to install the documentation, the Javadoc, and the sample) file to upload.
- On the Enterprise Manager under the Documentation tab, click on the document link, the Javadoc link, or the sample file link. For the sample project, it is recommended that you extract the file to another file location prior to importing it using the Enterprise Explorer's Import Project tool.

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

Continue installing the eGate Enterprise Designer as instructed.

2.3 After Installation

Once the eWay is installed and configured it must then be incorporated 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.

Chapter 3

Properties of the Oracle eWay

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

This Chapter Includes:

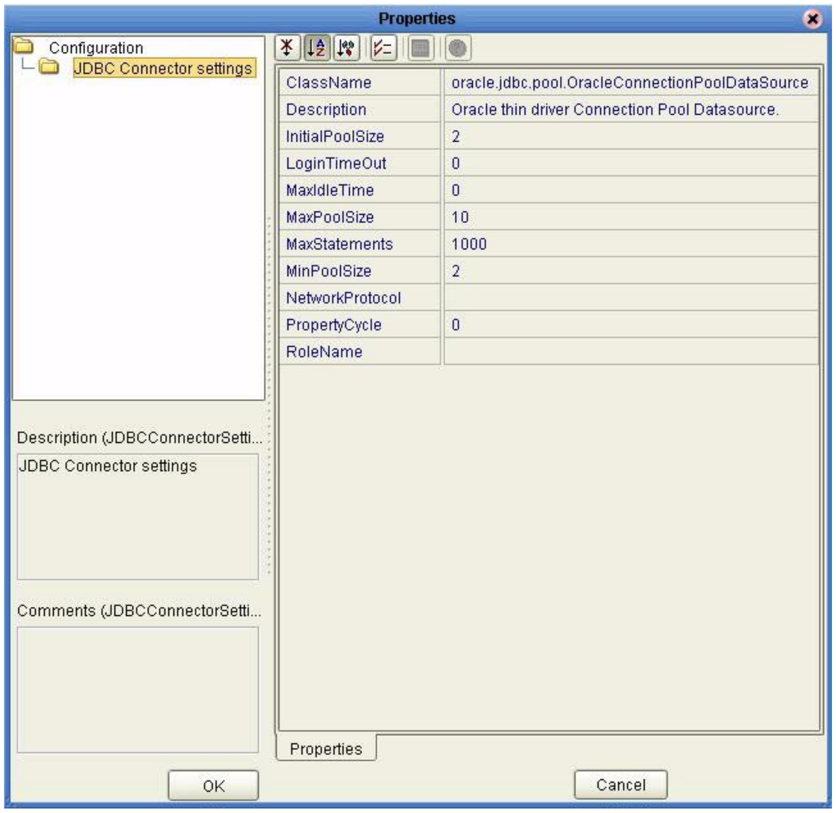
3.1 Working with eWay Property Sheets

On the Properties sheet window and 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.

Figure 1 The Outbound eWay Properties


Properties
Configuration
JDBC Connector settings
Description (JDBCConnectorSetti...
Comments (JDBCConnectorSetti...
OK
Cancel

JDBC Connector settings

ClassName	oracle.jdbc.pool.OracleConnectionPoolDataSource
Description	Oracle thin driver Connection Pool Datasource.
InitialPoolSize	2
LoginTimeOut	0
MaxIdleTime	0
MaxPoolSize	10
MaxStatements	1000
MinPoolSize	2
NetworkProtocol	
PropertyCycle	0
RoleName	

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

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 Value

A valid string.

InitialPoolSize

Description

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

Required Value

A valid numeric value.

LoginTimeOut

Description

The number of seconds driver will 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 Value

A valid numeric value.

MaxPoolSize

Description

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

Required Value

A valid numeric value.

MaxStatements

Description

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

Required Value

A valid numeric value.

MinPoolSize

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

Required Value

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 should 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

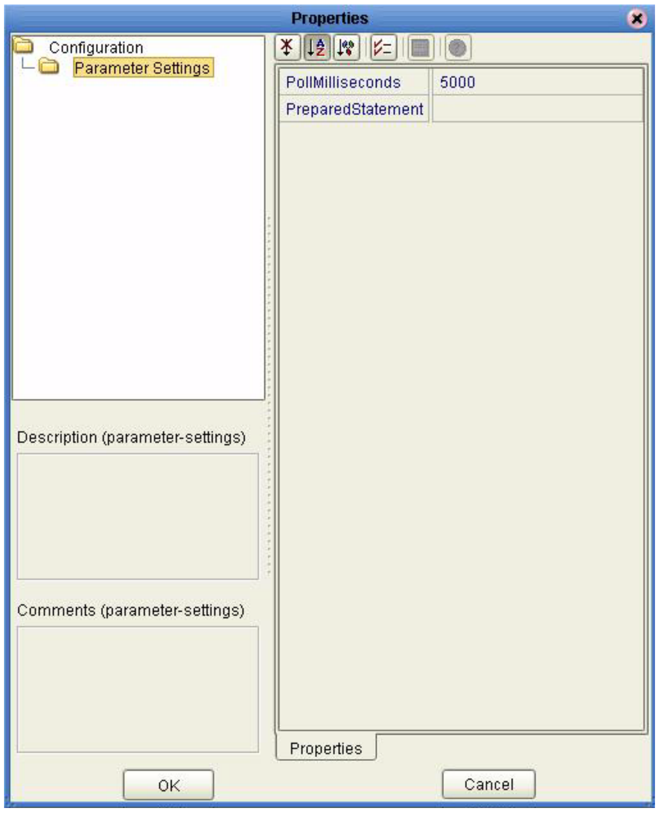
An initial SQL role name.

Required Values

Any valid string.

3.1.2. Setting the Properties in the Inbound eWay

Figure 2 Properties of the Inbound eWay


Properties
Configuration
Parameter Settings
Description (parameter-settings)
Comments (parameter-settings)
Properties
OK
Cancel

PollMilliseconds	5000
PreparedStatement	

Pollmilliseconds

Description

Polling interval in milliseconds.

Required Value

A valid numeric value. The default is 5000.

PreparedStatement

Description

Prepared Statement used for polling against the database.

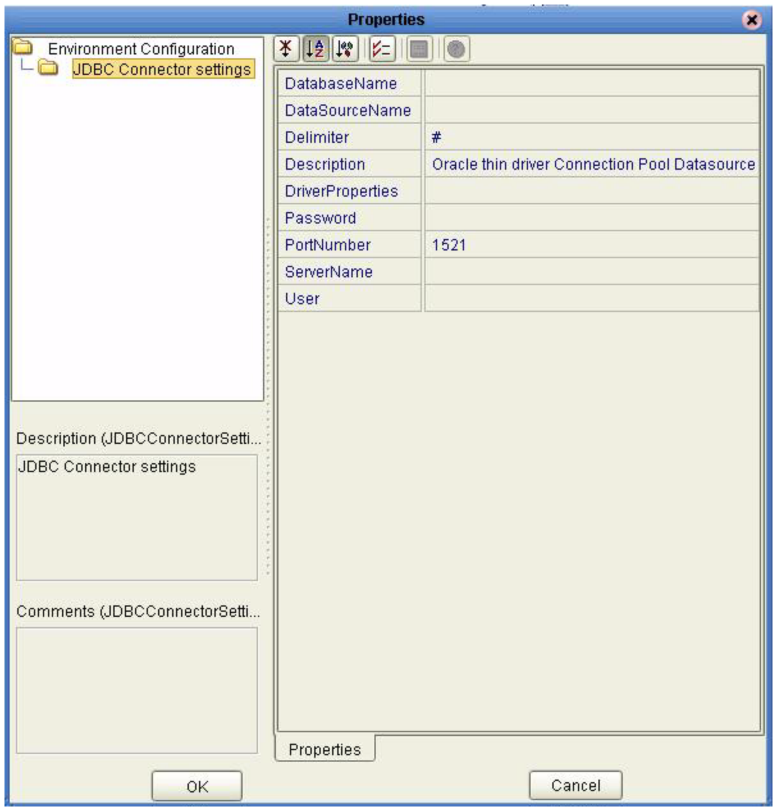
Required Value

The Prepared Statement must be the same Prepared Statement you created using the Database OTD Wizard. Only SELECT Statement is allowed. Additionally, no place holders should be specified. There should not be any "?" in the Prepared Query.

3.1.3. Setting the Properties in the Outbound eWay Environment

Before deploying your eWay, you will need to set the properties of the eWay environment using the following descriptions.

Figure 3 eWay Environment Configuration


Properties
Environment Configuration
JDBC Connector settings
Description (JDBCConnectorSetti...
Comments (JDBCConnectorSetti...
Properties
OK
Cancel

JDBC Connector settings

DatabaseName	
DataSourceName	
Delimiter	#
Description	Oracle thin driver Connection Pool Datasource
DriverProperties	
Password	
PortNumber	1521
ServerName	
User	

DatabaseName

Description

Specifies the name of the database instance.

Required Values

Any valid string.

DataSourceName

Description

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

Required Value

Optional. In most cases, leave this box empty.

Delimiter

Description

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

Required Value

The default is #

Description

Description

Enter a description for the database.

Required Value

A valid string.

DriverProperties

Description

If you choose to not use the JDBC driver that is shipped with this eWay, you will need to add the drivers properties to the eWay. Often times the DataSource implementation will need to execute additional properties to assure a connection. The additional methods will need to be identified in the Driver Properties.

Required Value

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.

User

Description

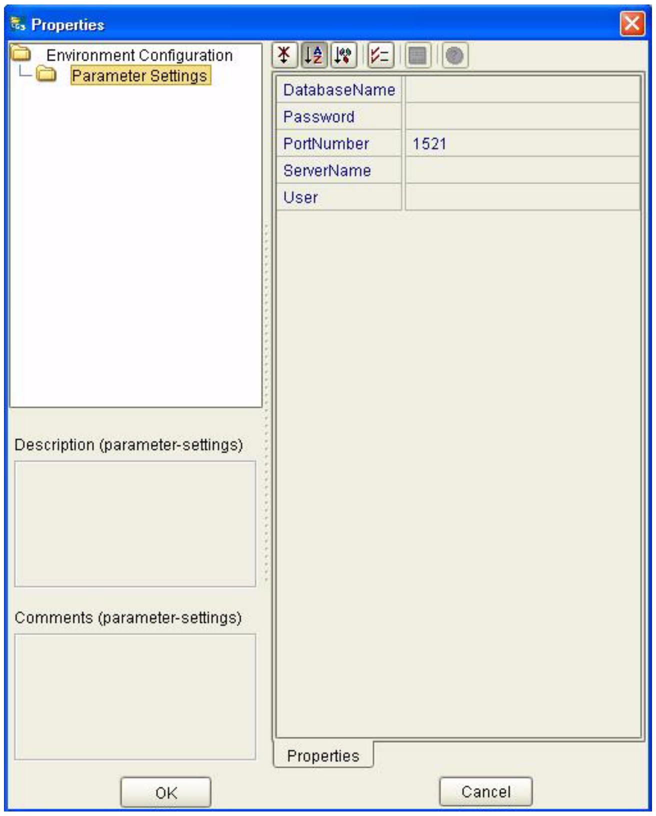
Specifies the user name the eWay uses to connect to the database.

Required Values

Any valid string.

3.1.4. Setting the Properties in the Inbound eWay Environment

Figure 4 Inbound eWay Environment


Properties
Environment Configuration
Parameter Settings
Description (parameter-settings)
Comments (parameter-settings)
Properties
OK
Cancel

DatabaseName	
Password	
PortNumber	1521
ServerName	
User	

DatabaseName

Description

Specifies the name of the database instance.

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

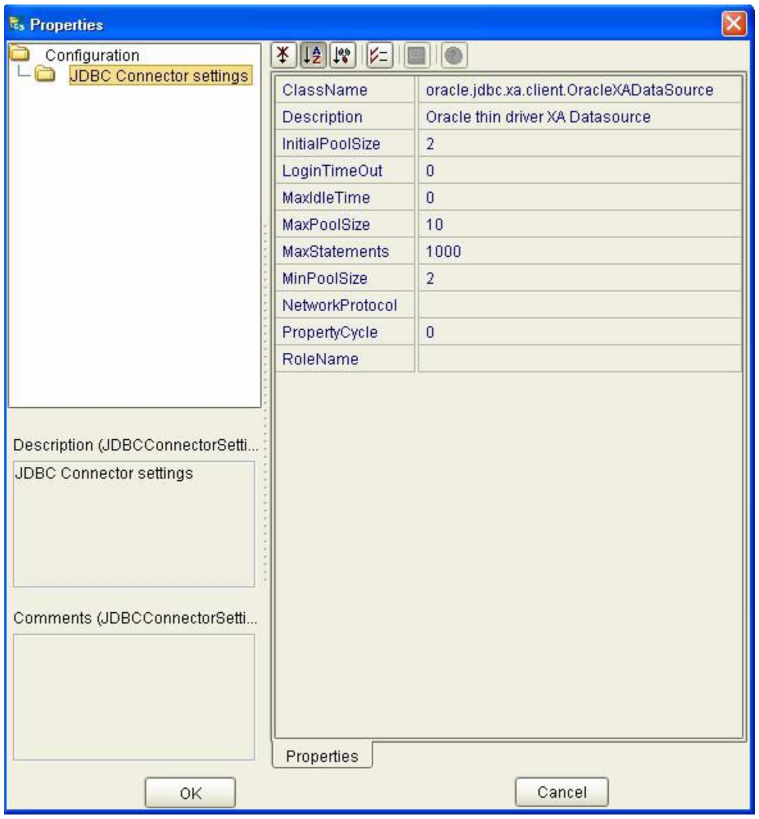
Specifies the user name the eWay uses to connect to the database.

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


Properties
Configuration
JDBC Connector settings
Description (JDBCConnectorSetti...
Comments (JDBCConnectorSetti...
Properties
OK
Cancel

JDBC Connector settings

ClassName	oracle.jdbc.xa.client.OracleXADataSource
Description	Oracle thin driver XA Datasource
InitialPoolSize	2
LoginTimeOut	0
MaxIdleTime	0
MaxPoolSize	10
MaxStatements	1000
MinPoolSize	2
NetworkProtocol	
PropertyCycle	0
RoleName	

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 Value

A valid string.

InitialPoolSize

Description

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

Required Value

A valid numeric value. The default is 2.

LoginTimeOut

Description

The number of seconds driver will 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 Value

A valid numeric value.

MaxPoolSize

Description

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

Required Value

A valid numeric value. The default is 10.

MaxStatements

Description

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

Required Value

A valid numeric value. The default is 1000.

MinPoolSize

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

Required Value

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 should 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

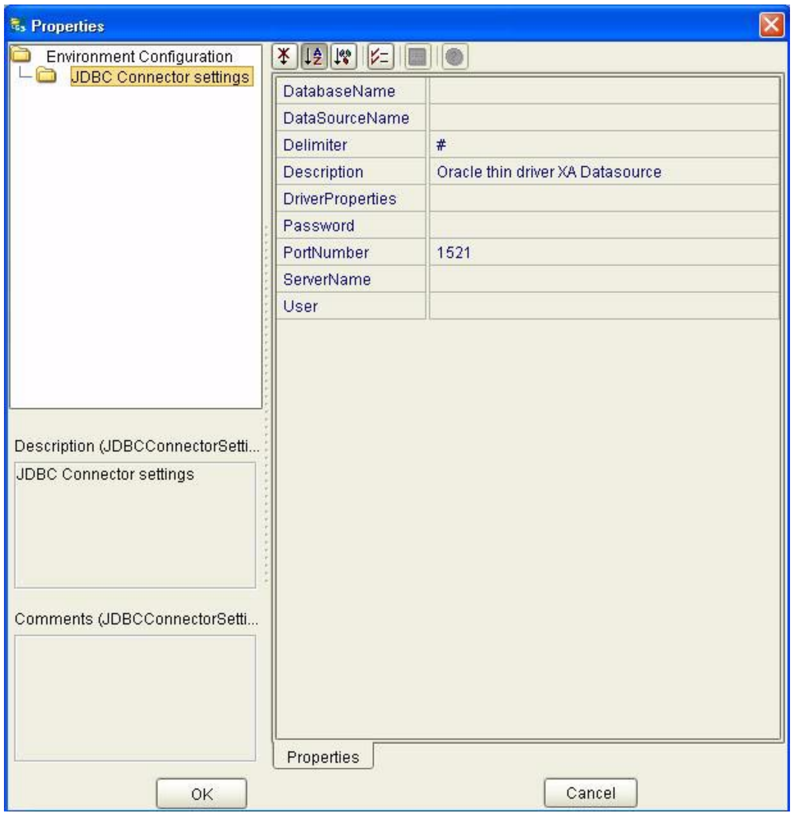
An initial SQL role name.

Required Values

Any valid string.

3.1.6. Setting the Properties in the eWay XA Environment

Figure 6 Environment Properties of the eWay with XA



Description (JDBCConnectorSetti...
JDBC Connector settings

Comments (JDBCConnectorSetti...

DatabaseName	
DataSourceName	
Delimiter	#
Description	Oracle thin driver XA Datasource
DriverProperties	
Password	
PortNumber	1521
ServerName	
User	

DatabaseName

Description

Specifies the name of the database instance.

Required Values

Any valid string.

DataSourceName

Description

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

Required Value

Optional. In most cases, leave this box empty.

Delimiter

Description

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

Required Value

The default is #

Description

Description

Enter a description for the database.

Required Value

A valid string.

DriverProperties

Description

If you choose to not to use the JDBC driver that is shipped with this eWay, you will need to add the drivers properties to the eWay. Often times the DataSource implementation will need to execute additional methods to assure a connection. The additional methods will need to be identified in the Driver Properties.

Required Value

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>##”.

If you are using Spy Log. Optional: “setURL#jdbc:Seebeyond:db2://<server>:50000;DatabaseName=<database>##setSpyAttributes#log=(file)c:/temp/spy.log;logTName=yes##”.

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.

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.

This Chapter Includes:

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.

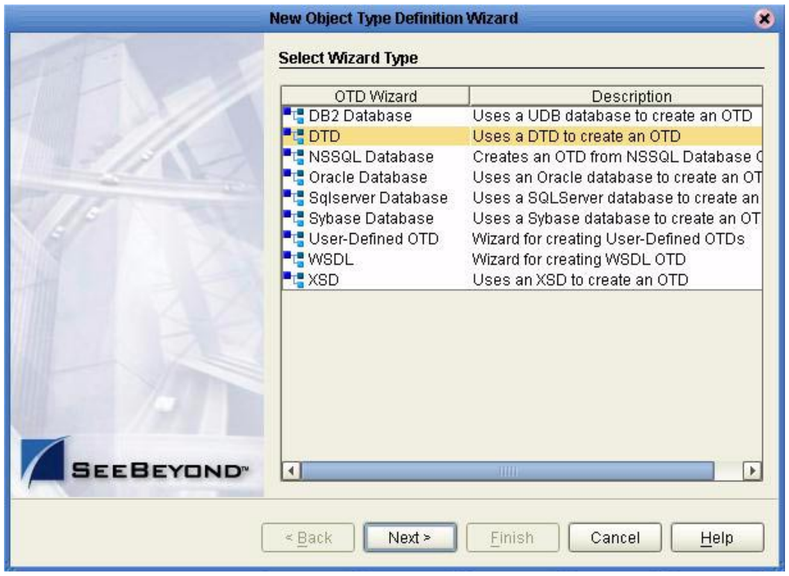
Field nodes are added to the OTD based on the Tables in the external data source. Java method and parameter nodes are added to provide the appropriate JDBC functionality. For more information about the Java methods, refer to your JDBC developer's reference.

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

Select Wizard Type

1. On the Enterprise Explorer, right click on the project and select **Create an Object Type Definition** from the shortcut menu.
2. From the OTD Wizard Selection window, select the **Oracle Database** and click **Next**. See **Figure 7**.

Figure 7 OTD Wizard Selection

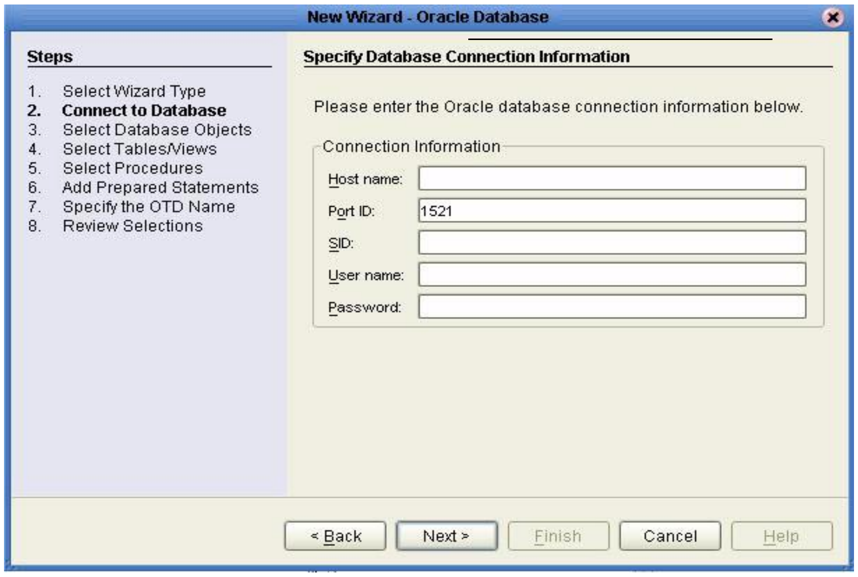


OTD Wizard	Description
DB2 Database	Uses a UDB database to create an OTD
DTD	Uses a DTD to create an OTD
NSSQL Database	Creates an OTD from NSSQL Database (
Oracle Database	Uses an Oracle database to create an OT
Sqlserver Database	Uses a SQLServer database to create an
Sybase Database	Uses a Sybase database to create an OT
User-Defined OTD	Wizard for creating User-Defined OTDs
WSDL	Wizard for creating WSDL OTD
XSD	Uses an XSD to create an OTD

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



Please enter the Oracle database connection information below.

Connection Information

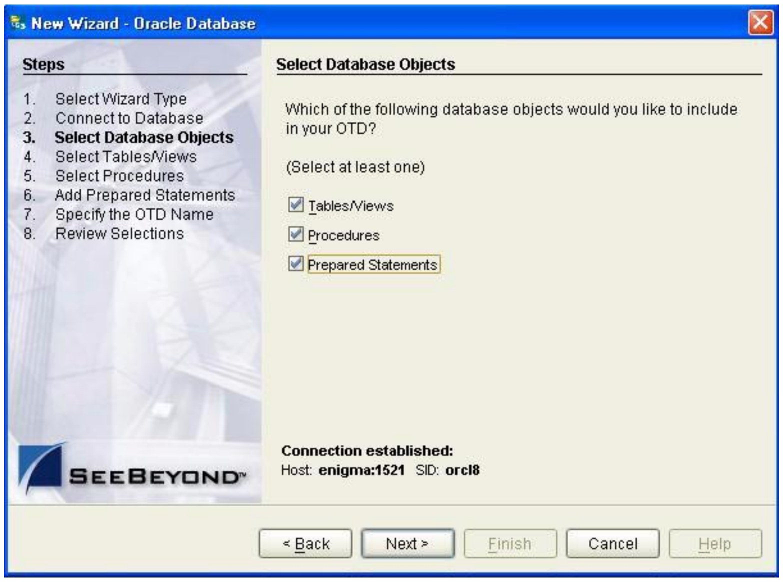
Host name:	
Port ID:	1521
SID:	
User name:	
Password:	

Select Database Objects

1 In the **Select Database Objects** window, see **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


New Wizard - Oracle Database

Steps
1. Select Wizard Type
2. Connect to Database
3. **Select Database Objects**
4. Select Tables/Views
5. Select Procedures
6. Add Prepared Statements
7. Specify the OTD Name
8. Review Selections

Select Database Objects

Which of the following database objects would you like to include in your OTD?

(Select at least one)

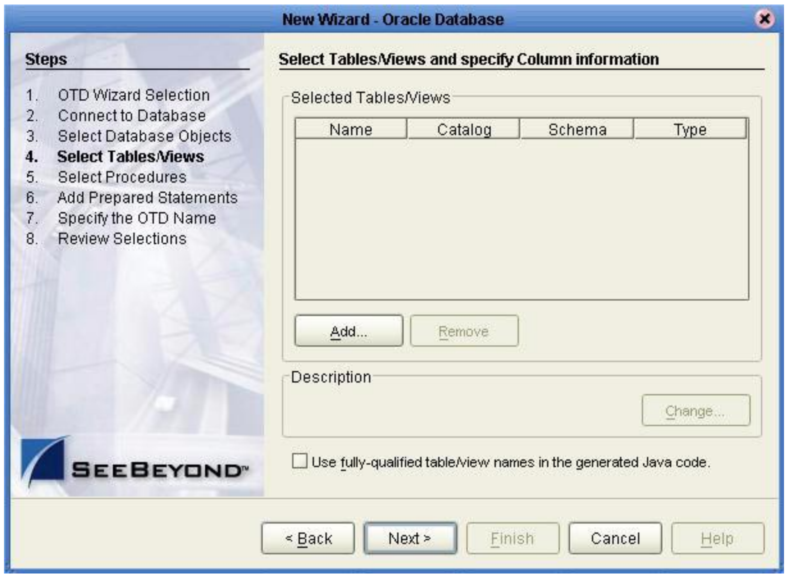
[✓] Tables/Views
[✓] Procedures
[✓] Prepared Statements

Connection established:
Host: **enigma:1521** SID: **orcl8**

Select Table/Views

1 In the **Select Tables/Views** window, click **Add**. See **Figure 10**.

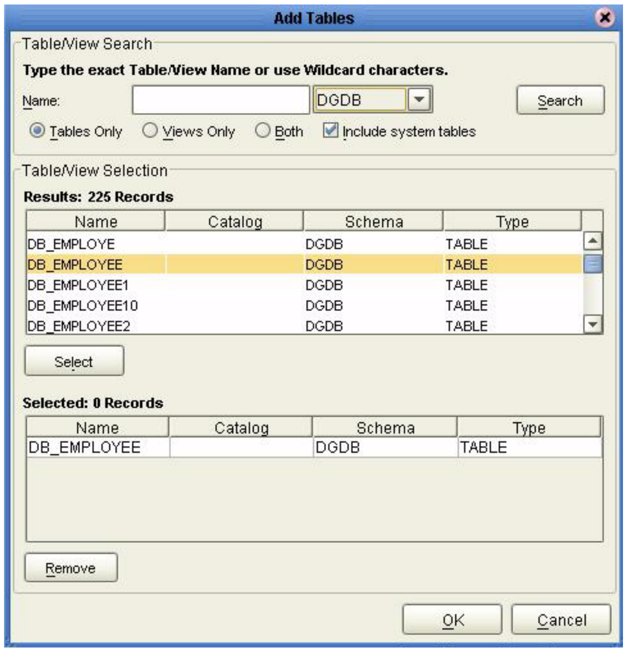
Figure 10 Select Tables/Views



2 In the **Add Tables** window, select if your selection criteria will include table data, view only data, both, and/or system tables.

3 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



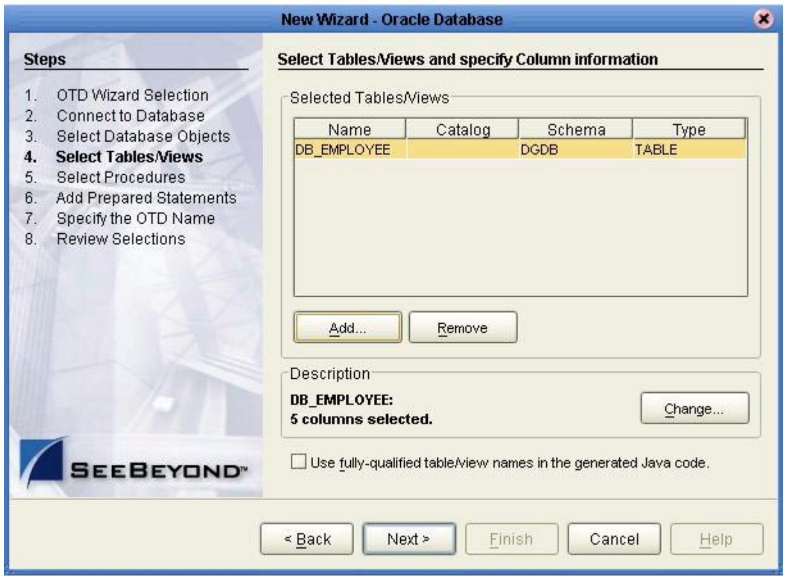
Name	Catalog	Schema	Type
DB_EMPLOYE		DGDB	TABLE
DB_EMPLOYEE		DGDB	TABLE
DB_EMPLOYEE1		DGDB	TABLE
DB_EMPLOYEE10		DGDB	TABLE
DB_EMPLOYEE2		DGDB	TABLE

Name	Catalog	Schema	Type
DB_EMPLOYEE		DGDB	TABLE

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

The table selected is added to the **Selected Tables/Views** window. See **Figure 12**.

Figure 12 Selected Tables/Views window with a table selected



Name	Catalog	Schema	Type
DB_EMPLOYEE		DGDB	TABLE

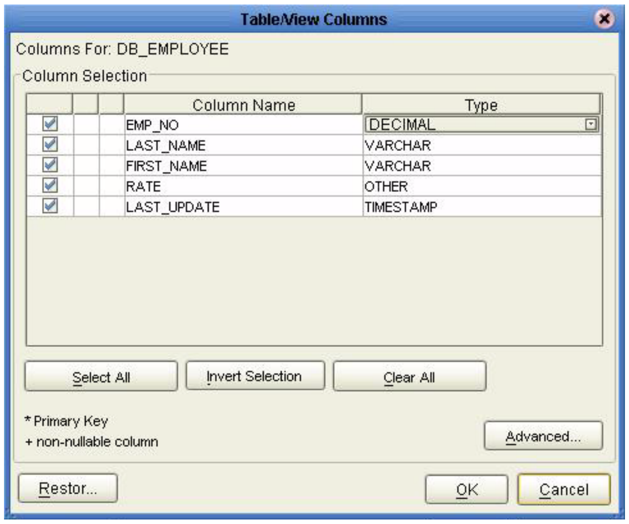
DB_EMPLOYEE:
5 columns selected.

☐ Use fully-qualified table/view names in the generated Java code.

5 In the **Selected Tables/Views** window, 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.

6 In the **Table/View Columns** window, 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 one from the drop down list. If you would like to change any of the tables columns, click **Change**. See **Figure 13**.

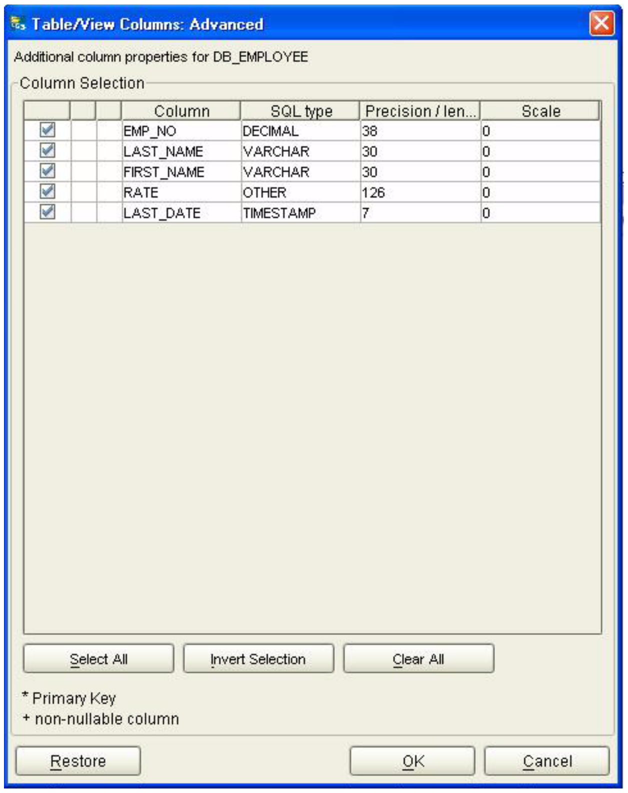
Figure 13 Table/View Columns



			Column Name	Type
✓			EMP_NO	DECIMAL
✓			LAST_NAME	VARCHAR
✓			FIRST_NAME	VARCHAR
✓			RATE	OTHER
✓			LAST_UPDATE	TIMESTAMP

7 Click **Advanced** to change the data type, percision/length, or scale. Once you have finished your table choices, click **OK**. See **Figure 14**. In general, you do not need to change the percision or scale.

Figure 14 Table/View Columns — Advanced


Table/View Columns: Advanced
Additional column properties for DB_EMPLOYEE
Column Selection

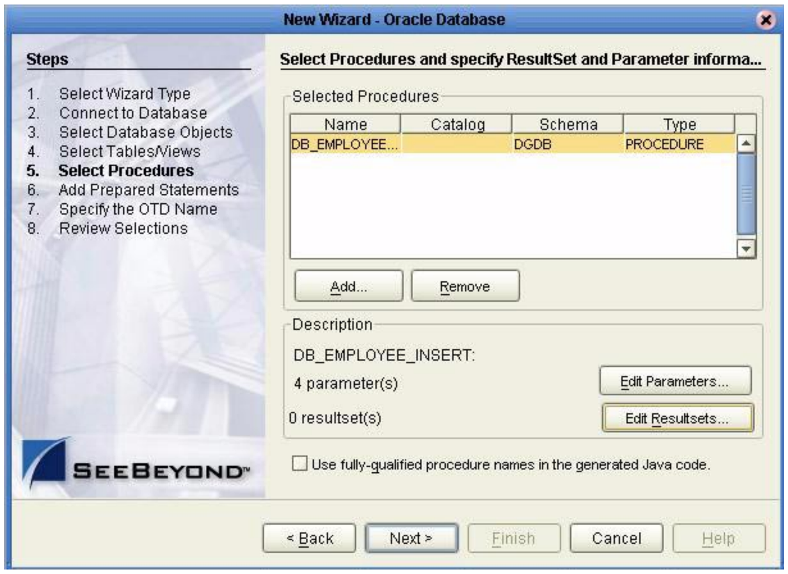
			Column	SQL type	Precision / len...	Scale
✓			EMP_NO	DECIMAL	38	0
✓			LAST_NAME	VARCHAR	30	0
✓			FIRST_NAME	VARCHAR	30	0
✓			RATE	OTHER	126	0
✓			LAST_DATE	TIMESTAMP	7	0

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

Select Procedures

1 On the **Select Procedures and specify Resultset and Parameter Information** window, click **Add.**

Figure 15 Select Procedures and specify Resultset and Parameter Information



Name	Catalog	Schema	Type
DB_EMPLOYEE...		DGDB	PROCEDURE

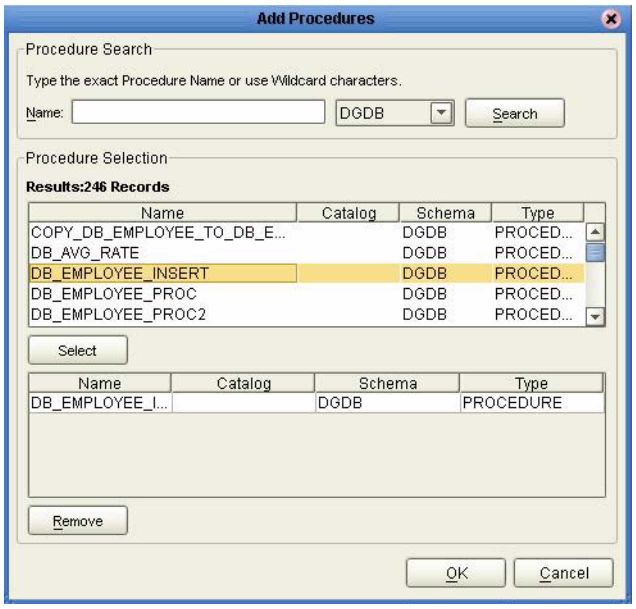
DB_EMPLOYEE_INSERT:
4 parameter(s)
0 resultset(s)

☐ Use fully-qualified procedure names in the generated Java code.

2 On the **Select Procedures** window, enter the name of a Procedure or select a table from the drop down list. Click **Search**. Wildcard characters can also be used.

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

Figure 16 Add Procedures

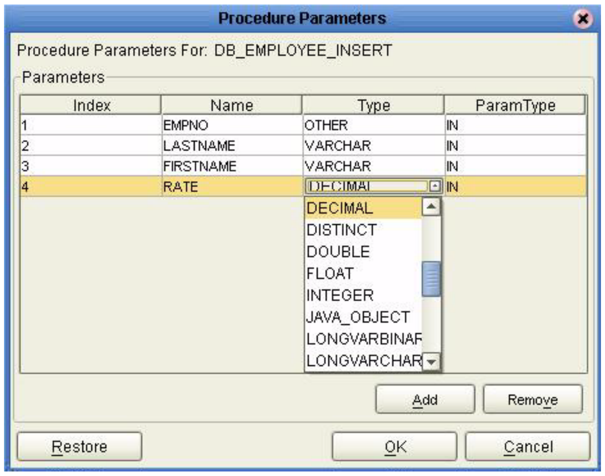


Name	Catalog	Schema	Type
COPY_DB_EMPLOYEE_TO_DB_E...		DGDB	PROCED...
DB_AVG_RATE		DGDB	PROCED...
DB_EMPLOYEE_INSERT		DGDB	PROCED...
DB_EMPLOYEE_PROC		DGDB	PROCED...
DB_EMPLOYEE_PROC2		DGDB	PROCED...

Name	Catalog	Schema	Type
DB_EMPLOYEE_I...		DGDB	PROCEDURE

4 On the **Select Procedures and specify Resultset and Parameter Information** window click **Edit Parameters** to make any changes to the selected Procedure. See **Figure 17**.

Figure 17 Procedure Parameters



Procedure Parameters For: DB_EMPLOYEE_INSERT

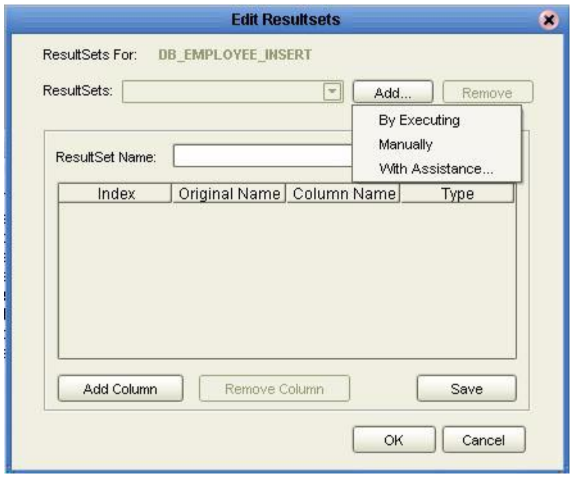
Index	Name	Type	ParamType
1	EMPNO	OTHER	IN
2	LASTNAME	VARCHAR	IN
3	FIRSTNAME	VARCHAR	IN
4	RATE	DECIMAL	IN

5 To restore the data type, click **Restore**. When finished, click **OK**.

6 To select how you would like the OTD to generate the nodes for the Resultset click **Edit Resultsets**.

7 Click Add to add the type of Resultset node you would like to generate. See

Figure 18 Edit Resultset



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

"**By Executing**" mode executes the specified Stored Procedure with default values to generate the ResultSet(s). Depending on the business logic of the Stored Procedure, zero or more ResultSets can be returned from the execution. In the case that there are multiple ResultSets and "**By Executing**" mode does not return all ResultSets, one should use the other modes to generate the ResultSet nodes.

"**With Assistance**" mode allows users to specify a query and execute it to generate the ResultSet node. 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. We can roughly classify Stored Procedures 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 "**Assist**" mode, highlight the execute statement up to and including the table name(s) before executing the query.

"**Manually**" mode is the most flexible way to generate the result set nodes. It allows users to specify the node name, original column name and data type manually. One drawback of this method is that users need to know the original column names and data types. This 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, "**With Assistance**" mode is a better choice.

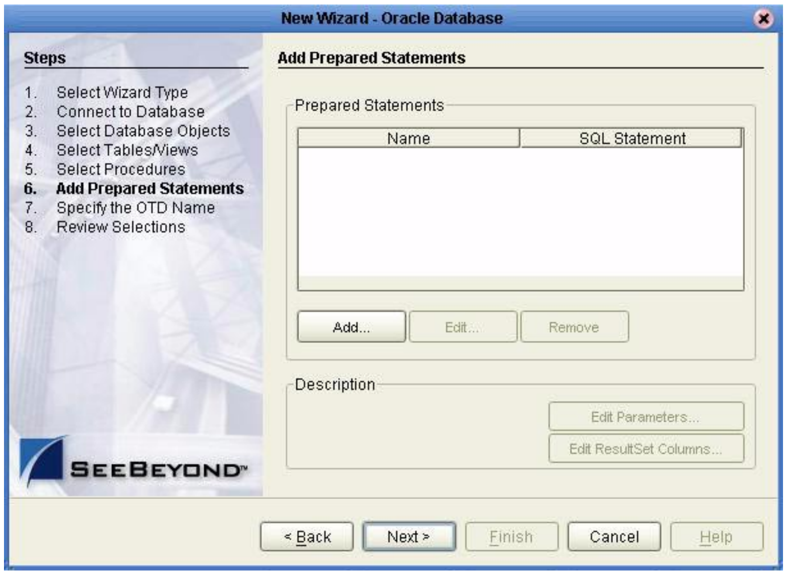
If you modify the ResultSet generated by the “**Execute**” mode of the Database Wizard you need to make sure the indexes match the Stored Procedure. This assures your ResultSet indexes are preserved.

8 On the **Select Procedures and specify Resultset and Parameter Information** window click **Next** to continue.

Add Prepared Statements

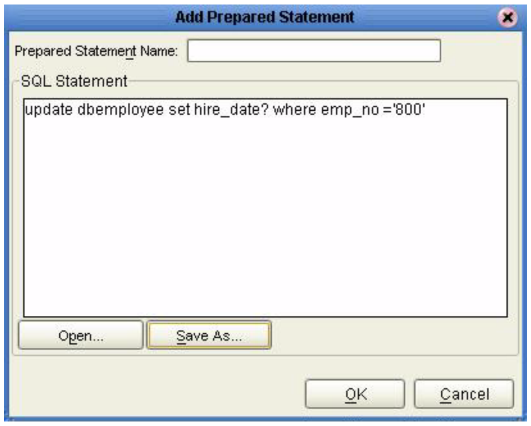
1 On the **Add Prepared Statements** window, click **Add**.

Figure 19 Prepared Statement



2 Enter the name of a Prepared Statement or create a SQL statement by clicking in the SQL Statement window. When finished creating the statement, click **Save As** giving the statement a name. This name will appear as a node in the OTD. Click **OK**. See Figure 20.

Figure 20 Prepared SQL Statement

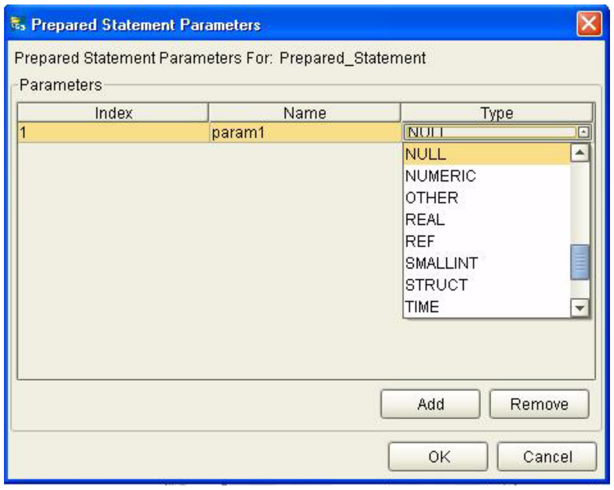


```
update dbemployee set hire_date? where emp_no ='800'
```

3 On the **Add Prepared Statement** window, 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.

4 Click **Add** if you want to add additional parameters to the Statement or highlight a row and click **Remove** to remove it. Click **OK**. See Figure 21.

Figure 21 Edit the Prepared Statement Parameters



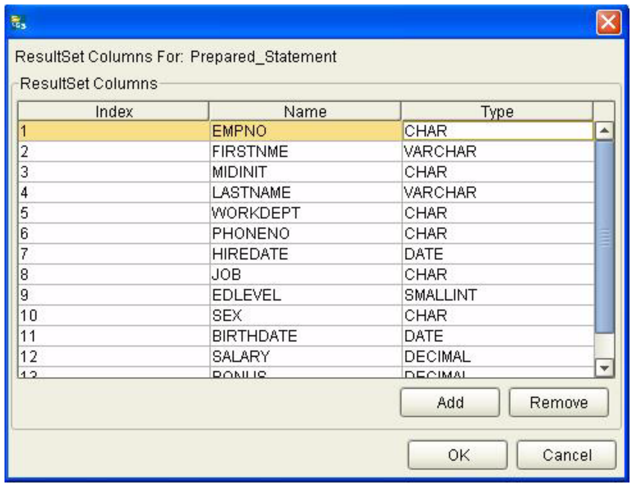
Prepared Statement Parameters For: Prepared_Statement

Parameters

Index	Name	Type
1	param1	NULL

1 To edit the Resultset Columns, click **Edit Resultset Columns**. Both the Name and Type are editable but it is recommend you do not changed the Name. Doing so will cause a loss of integrity between the Resultset and the Database. Click **OK**. See Figure 22.

Figure 22 ResultSet Columns



ResultSet Columns For: Prepared_Statement

ResultSet Columns

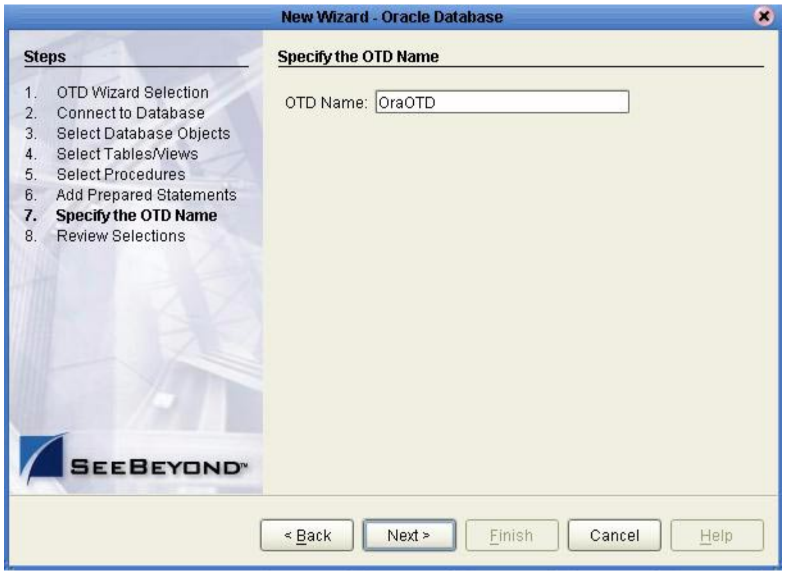
Index	Name	Type
1	EMPNO	CHAR
2	FIRSTNME	VARCHAR
3	MIDINIT	CHAR
4	LASTNAME	VARCHAR
5	WORKDEPT	CHAR
6	PHONENO	CHAR
7	HIREDATE	DATE
8	JOB	CHAR
9	EDLEVEL	SMALLINT
10	SEX	CHAR
11	BIRTHDATE	DATE
12	SALARY	DECIMAL
13	BONUS	DECIMAL

2 On the Add Prepared Statements window, click **OK**.

Specify the OTD Name

1 Enter a name for the OTD. The OTD contains the selected tables and the package name of the generated classes. See Figure 23.

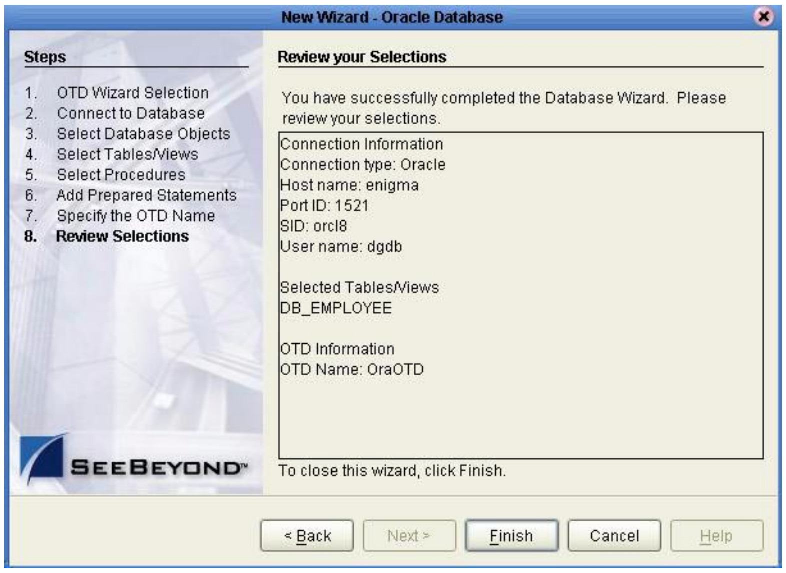
Figure 23 Naming an OTD



OTD Name: OraOTD

2 View the summary of the OTD. If you find 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



You have successfully completed the Database Wizard. Please review your selections.

```
Connection Information
Connection type: Oracle
Host name: enigma
Port ID: 1521
SID: orcl8
User name: dgdb

Selected Tables/Views
DB_EMPLOYEE

OTD Information
OTD Name: OraOTD
```

To close this wizard, click Finish.

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

Chapter 5

Importing the Project

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

This Chapter Includes:

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 will need to 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.

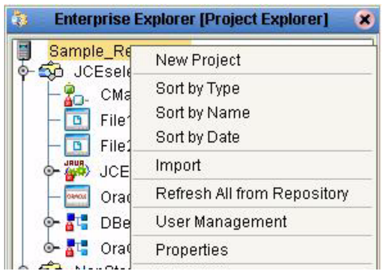
Note: *eInsight is a Business Process modeling tool. If you have not purchased eInsight, contact your sales representative for information on how to do so.*

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

Importing the Sample Project

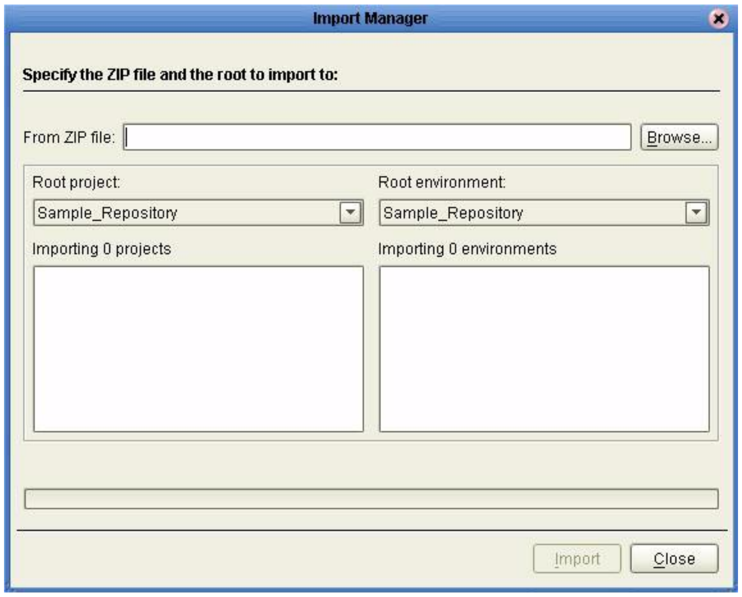
1 On the Enterprise Explorer highlight the repository and right click. Select **Import**. See **Figure 25**.

Figure 25 Importing the sample project



2 In the **Import Manager** window, **From ZIP file** browse to the location of the sample folder and select the following .zip file **Ora_BPEL_Sample.zip** and click **Import**.See **Figure 26**.

Figure 26 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 has the following columns:

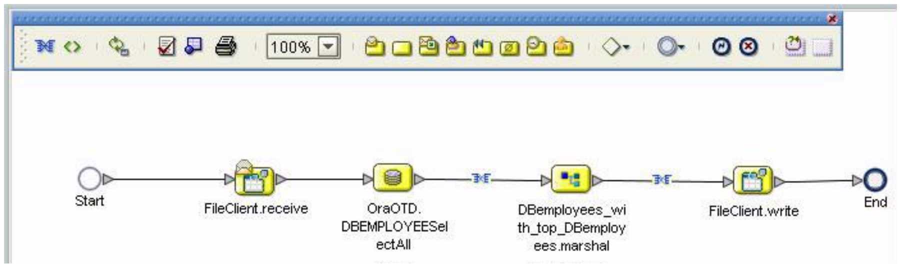
Table 1 Sample project data
Table 2

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 27



5.2.1. Working with the BPEL Operations

You can associate an eInsight Business Process Activity with the Oracle eWay, both 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.*

The table below shows the inputs and outputs to each of these 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
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.

whereClause()

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

For example:

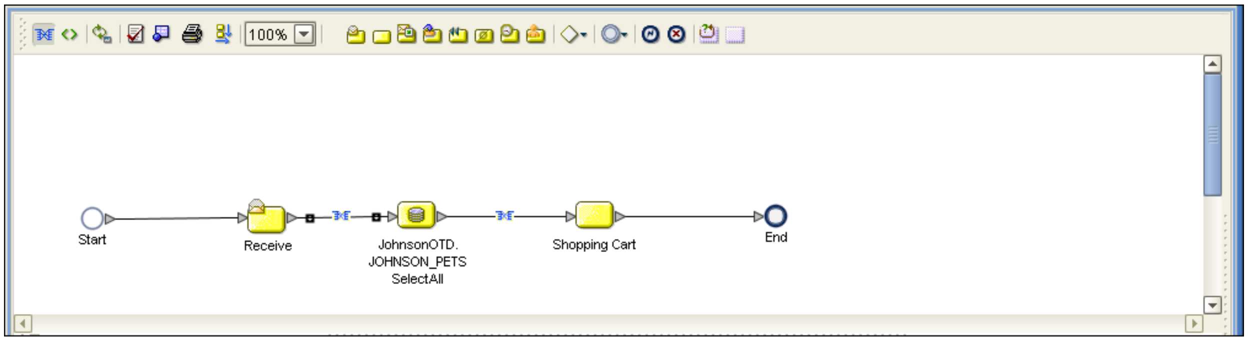
```
whereClause such as where column2=2 AND column1=1 OR column3=3 is
valid
```

SelectAll

The input to a SelectAll operation is an optional where() clause. The where() clause defines to which criteria rows must adhere to be returned. In the SelectAll operation, all items that fit the criteria are returned. If the where() clause is not specified, all rows all returned.

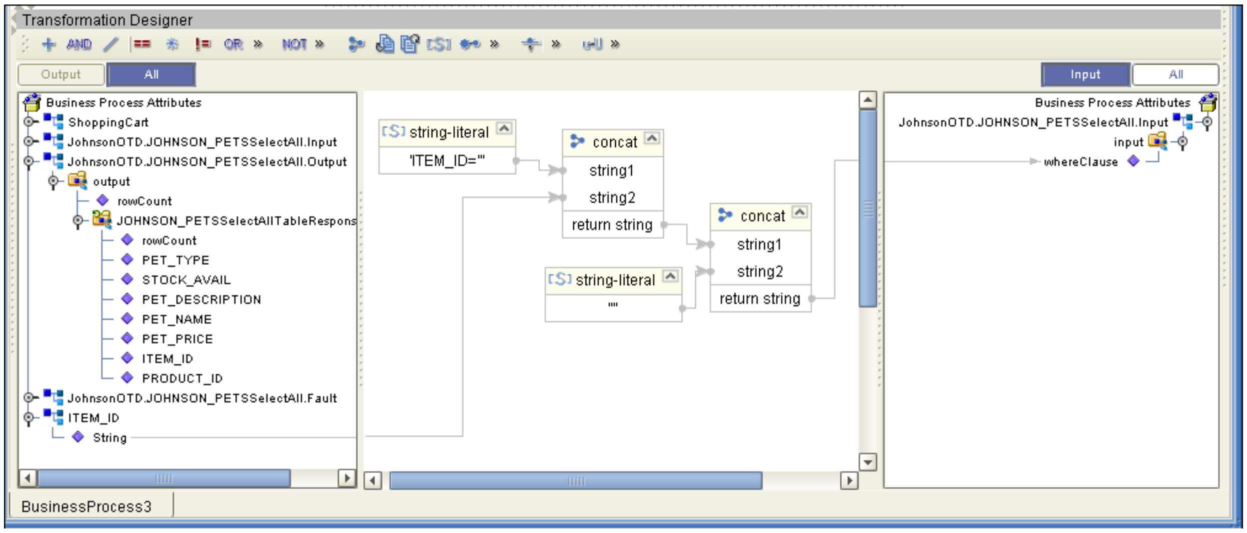
The figure below shows a sample eInsight Business Process using the SelectAll operation. In this process, the SelectAll operation returns all rows where the ITEM_ID matches the selected ITEM_ID to the shopping cart.

Figure 28 SelectAll Sample Business Process



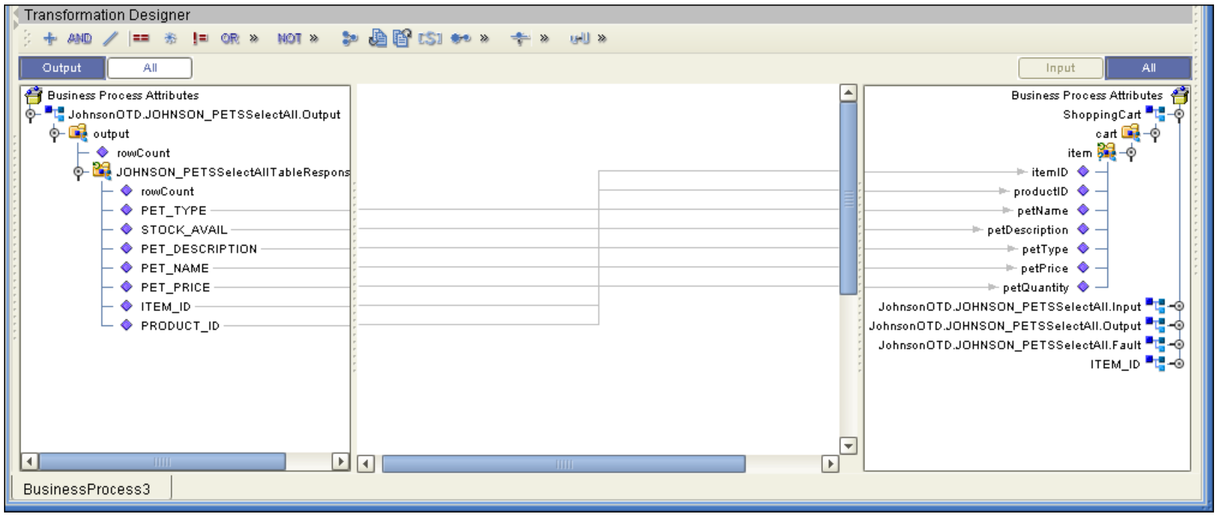
The figure below shows the definition of the where() clause for the SelectAll operation.

Figure 29 SelectAll Input



The figure below 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 30 SelectAll Output

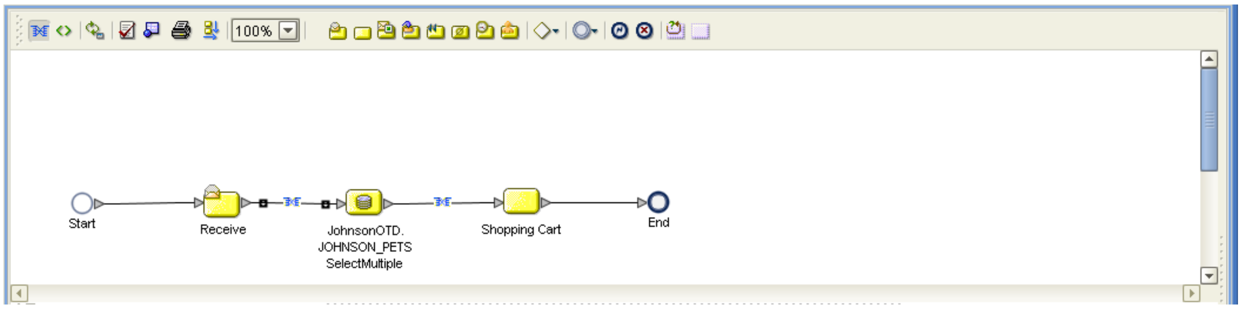

Transformation Designer
Output
All
Input
All
Business Process Attributes
JohnsonOTD.JOHNSON_PETSSelectAll.Output
output
rowCount
JOHNSON_PETSSelectAllTableRespons
rowCount
PET_TYPE
STOCK_AVAIL
PET_DESCRIPTION
PET_NAME
PET_PRICE
ITEM_ID
PRODUCT_ID
Business Process Attributes
ShoppingCart
cart
item
itemID
productID
petName
petDescription
petType
petPrice
petQuantity
JohnsonOTD.JOHNSON_PETSSelectAll.Input
JohnsonOTD.JOHNSON_PETSSelectAll.Output
JohnsonOTD.JOHNSON_PETSSelectAll.Fault
ITEM_ID
BusinessProcess3

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 to which criteria rows must adhere to be returned.

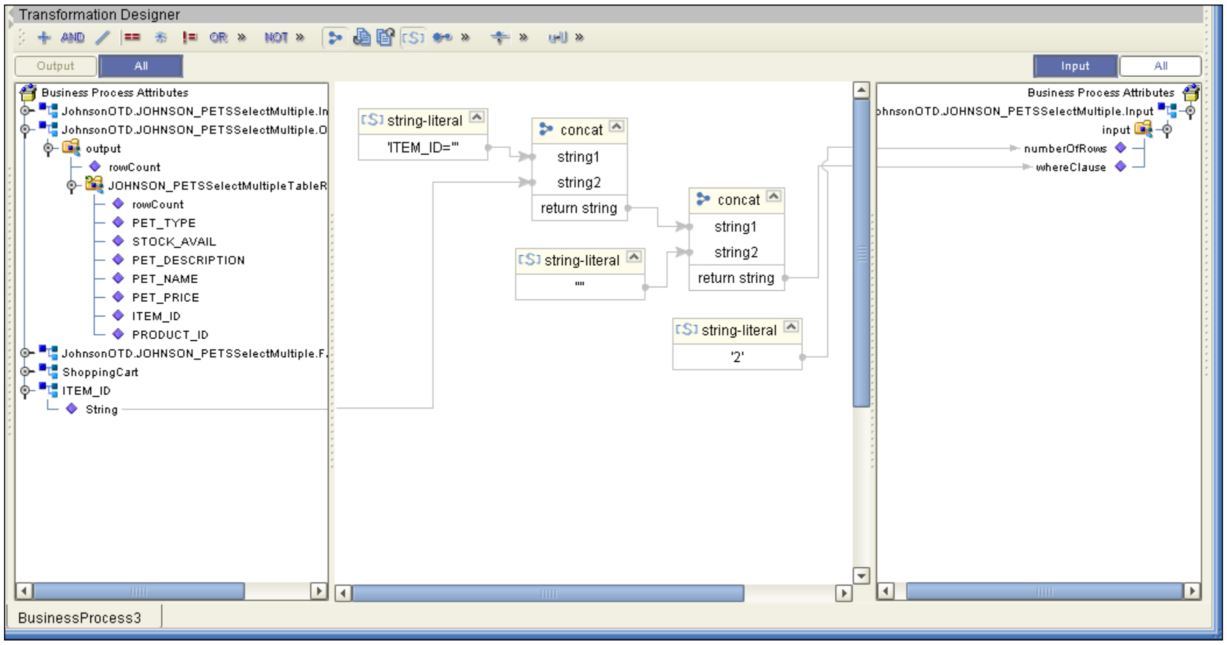
The figure below shows a sample eInsight Business Process using the SelectMultiple operation. In this process, the SelectMultiple operation returns the first two rows where the ITEM_ID matches the selected ITEM_ID to the shopping cart.

Figure 31 SelectMultiple Sample Business Process



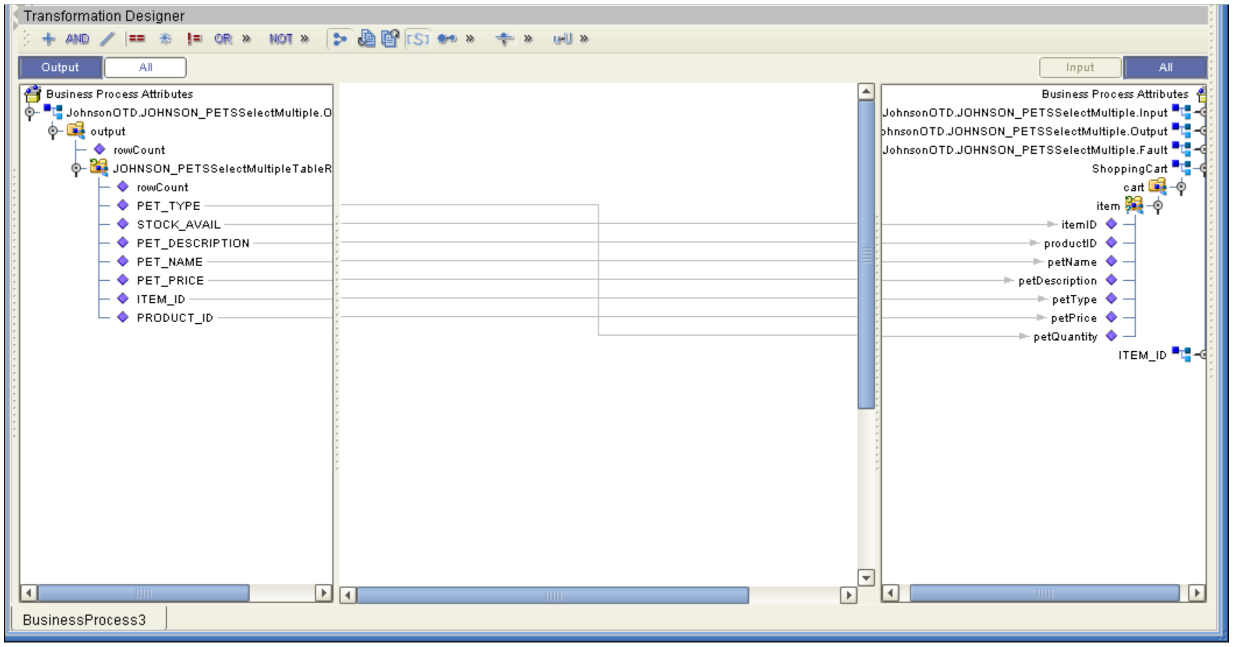
The figure below shows the definition of the number of rows and where() clause into the input for the SelectMultiple operation. You could also use an empty string or Item_ID=’123’.

Figure 32 SelectMultiple Input



The figure below 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 33 SelectMultiple Output

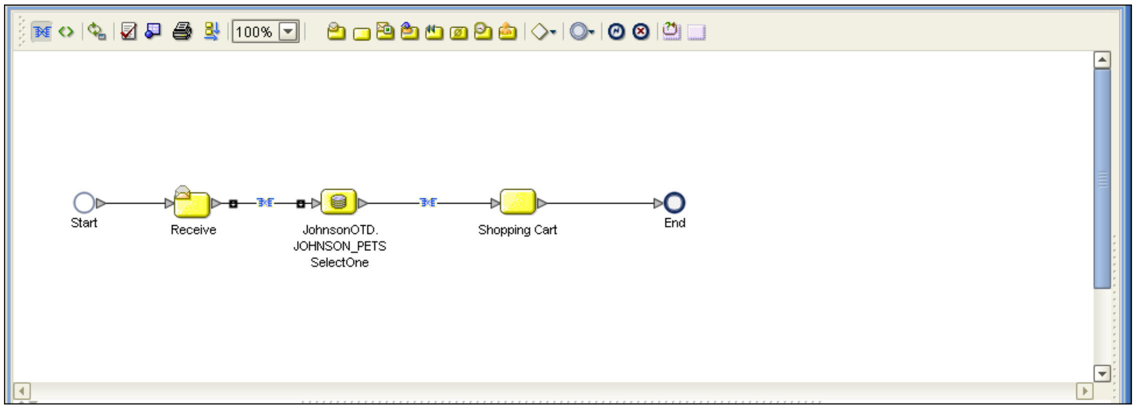

Transformation Designer
Output
All
Input
All
Business Process Attributes
JohnsonOTD.JOHNSON_PETSSelectMultiple.O
output
rowCount
JOHNSON_PETSSelectMultipleTableR
rowCount
PET_TYPE
STOCK_AVAIL
PET_DESCRIPTION
PET_NAME
PET_PRICE
ITEM_ID
PRODUCT_ID
Business Process Attributes
JohnsonOTD.JOHNSON_PETSSelectMultiple.Input
ohnsonOTD.JOHNSON_PETSSelectMultiple.Output
JohnsonOTD.JOHNSON_PETSSelectMultiple.Fault
ShoppingCart
cart
item
itemID
productID
petName
petDescription
petType
petPrice
petQuantity
ITEM_ID
BusinessProcess3

SelectOne

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

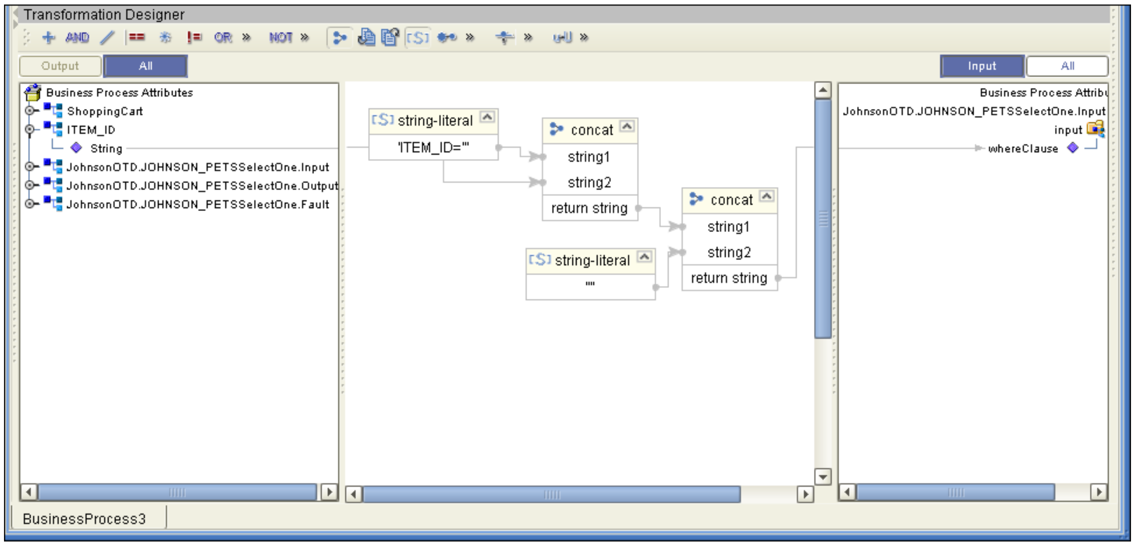
The figure below shows a sample eInsight Business Process using the SelectOne operation. In this process, the SelectOne operation returns the first row where the ITEM_ID matches the specified ITEM_ID to the shopping cart.

Figure 34 SelectOne Sample Business Process



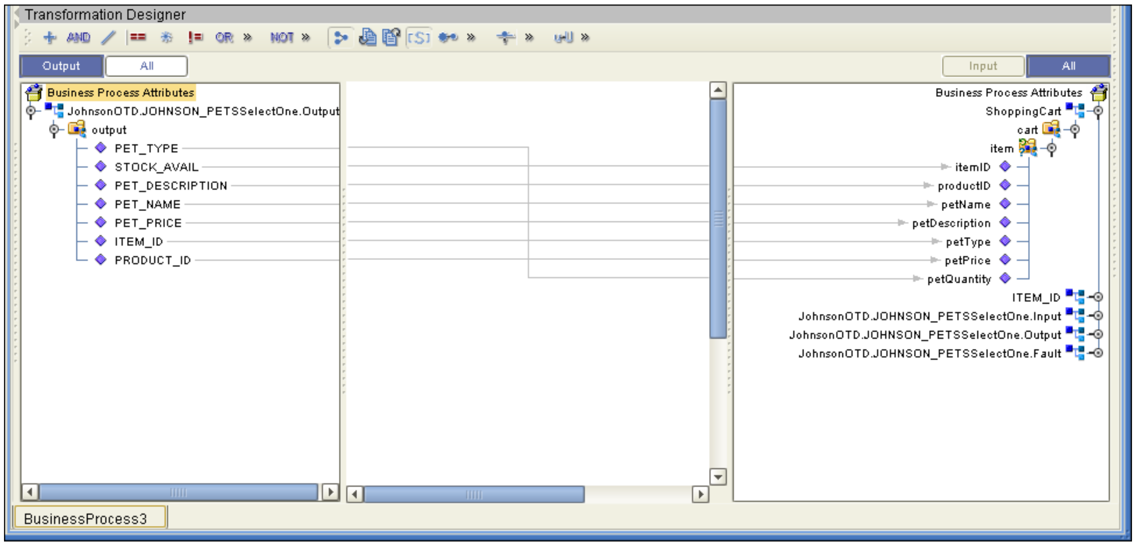
The figure below shows the definition of the where() clause for the SelectOne operation.

Figure 35 SelectOne Input



The figure below 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 36 SelectOne Output

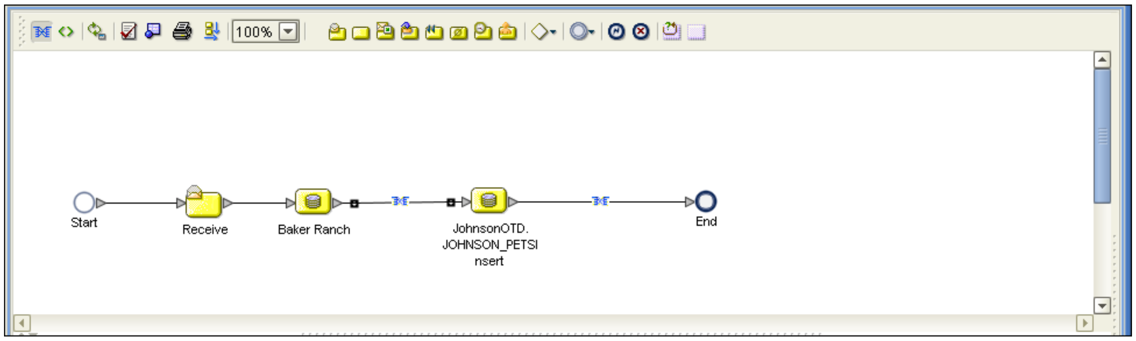


Insert

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

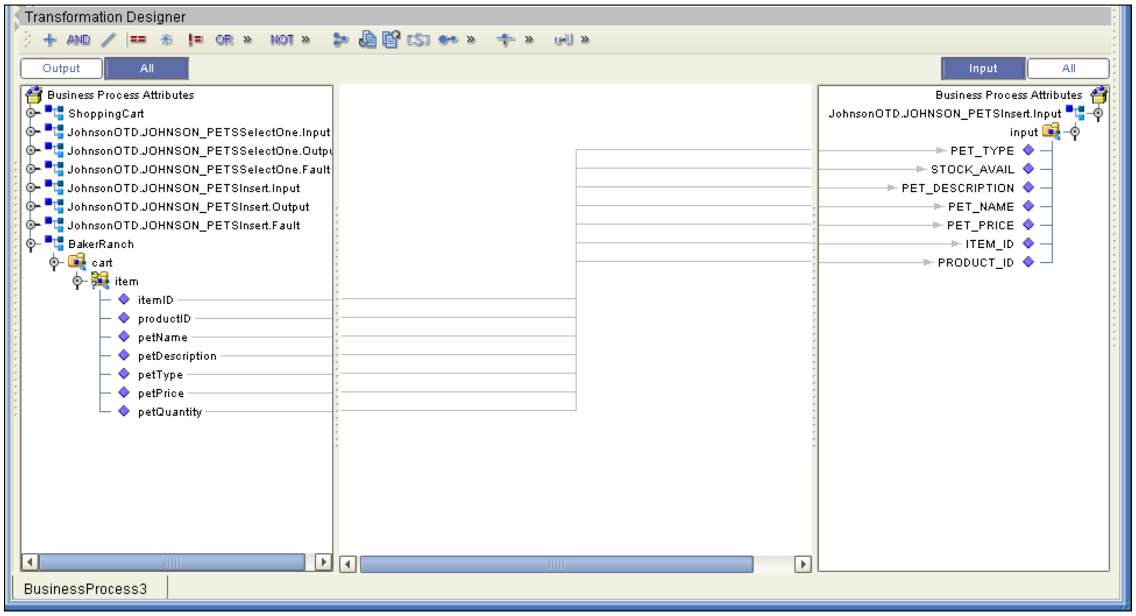
The figure below shows a sample eInsight Business Process using the Insert operation. In this process, the operation inserts a new row into the database to accommodate a new item provided by a vendor.

Figure 37 Insert Sample Business Process



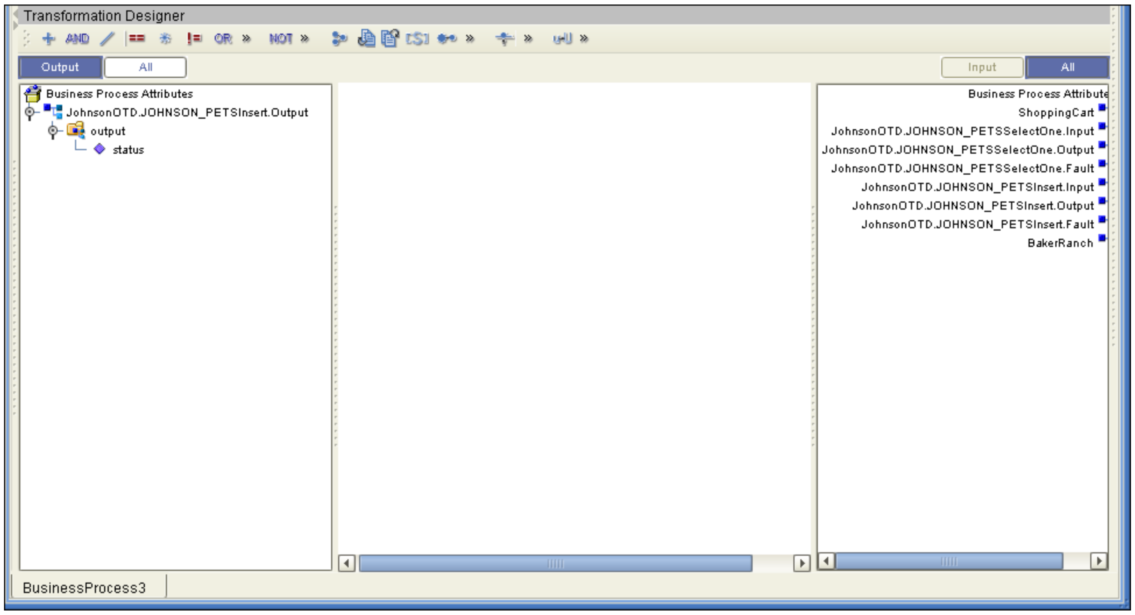
The figure below shows the definition of the input for the Insert operation.

Figure 38 Insert Input


Transformation Designer
Output
All
Input
All
Business Process Attributes
ShoppingCart
JohnsonOTD.JOHNSON_PETSSelectOne.Input
JohnsonOTD.JOHNSON_PETSSelectOne.Outpu
JohnsonOTD.JOHNSON_PETSSelectOne.Fault
JohnsonOTD.JOHNSON_PETSInsert.Input
JohnsonOTD.JOHNSON_PETSInsert.Output
JohnsonOTD.JOHNSON_PETSInsert.Fault
BakerRanch
cart
item
itemID
productID
petName
petDescription
petType
petPrice
petQuantity
Business Process Attributes
JohnsonOTD.JOHNSON_PETSInsert.Input
input
PET_TYPE
STOCK_AVAIL
PET_DESCRIPTION
PET_NAME
PET_PRICE
ITEM_ID
PRODUCT_ID
BusinessProcess3

The figure below shows the output of the Insert operation, which is a status indicating the number of rows created.

Figure 39 Insert Output

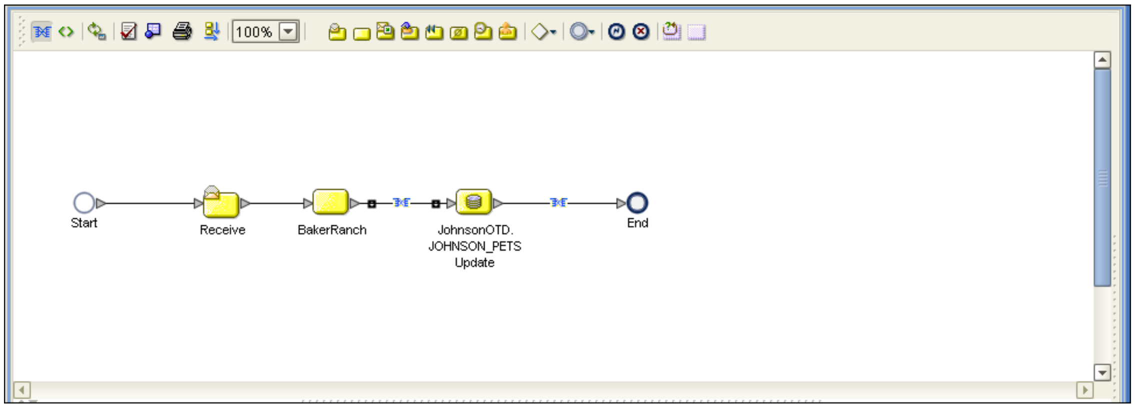

Transformation Designer
Output
All
Input
All
Business Process Attributes
JohnsonOTD.JOHNSON_PETSInsert.Output
output
status
Business Process Attribute
ShoppingCart
JohnsonOTD.JOHNSON_PETSSelectOne.Input
JohnsonOTD.JOHNSON_PETSSelectOne.Output
JohnsonOTD.JOHNSON_PETSSelectOne.Fault
JohnsonOTD.JOHNSON_PETSInsert.Input
JohnsonOTD.JOHNSON_PETSInsert.Output
JohnsonOTD.JOHNSON_PETSInsert.Fault
BakerRanch
BusinessProcess3

Update

The Update operation updates rows that fit certain criteria defined in a where() clause.

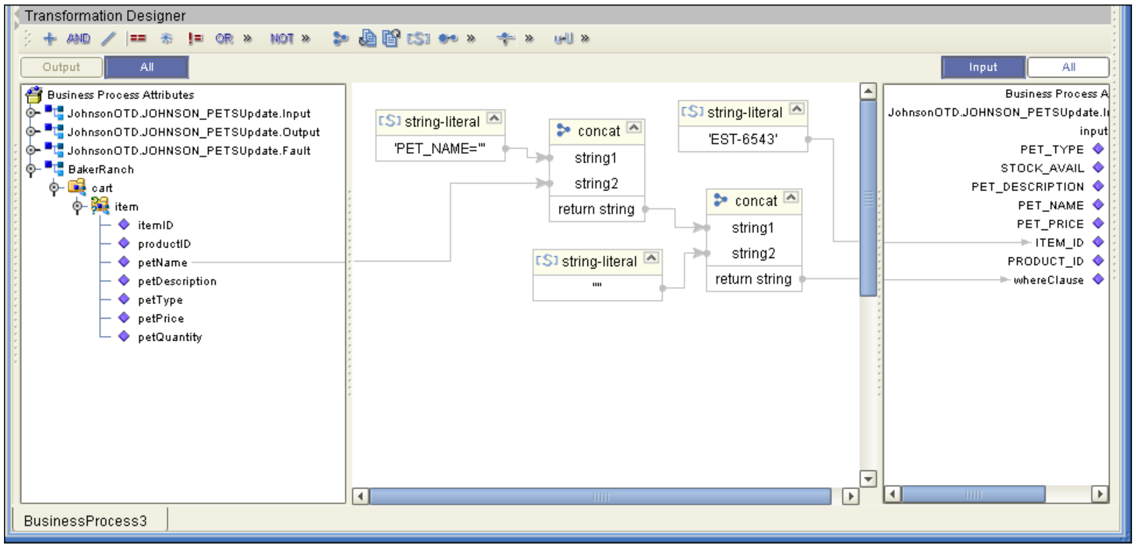
The figure below shows a sample eInsight Business Process using the Update operation. In this process, the operation updates the ITEM_ID for all items with a certain name to ESR_6543.

Figure 40 Update Sample Business Process



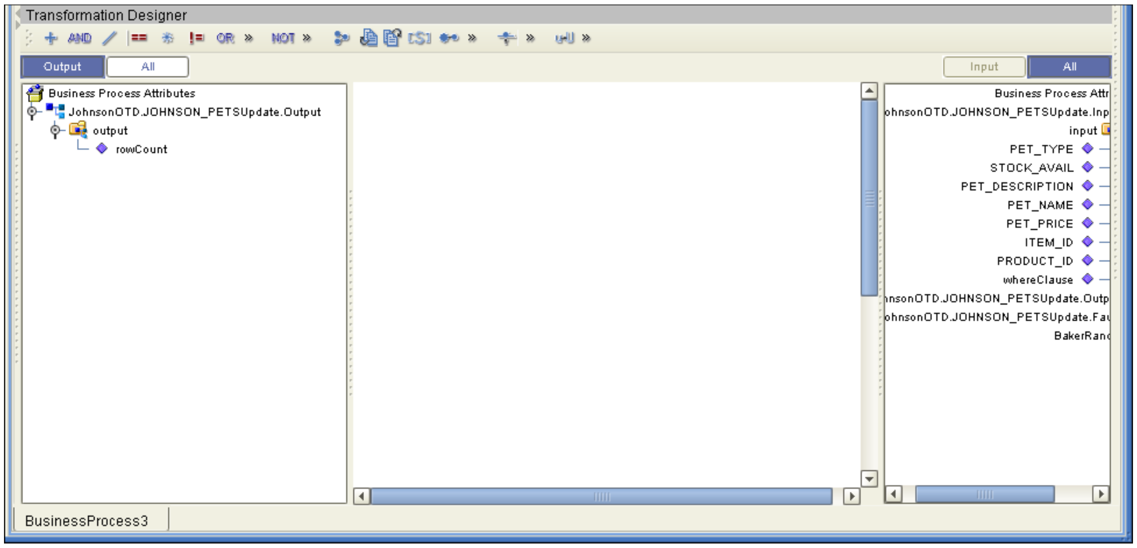
The figure below shows the definition of the where() clause for the Update operation.

Figure 41 Update Input



The figure below shows the output of the Update operation, which is a status indicating the number of rows updated.

Figure 42 Update Output

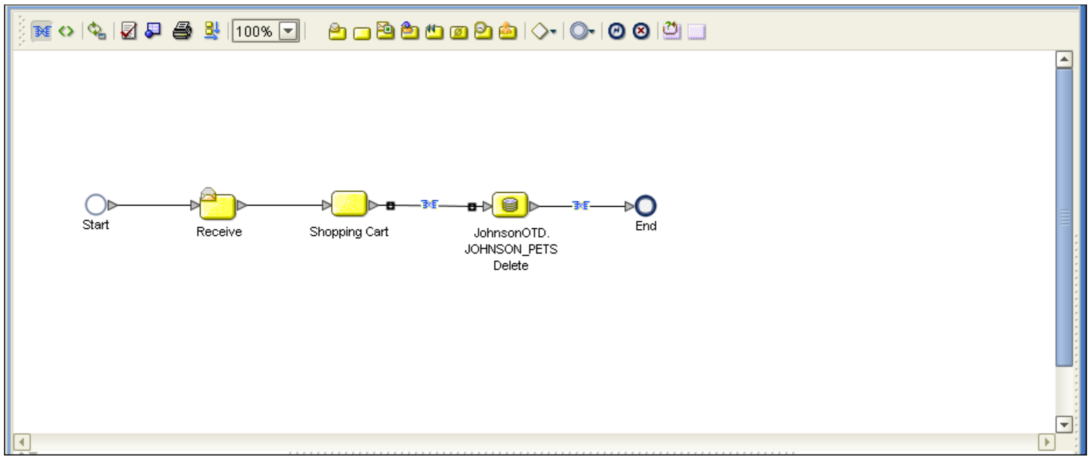


Delete

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.

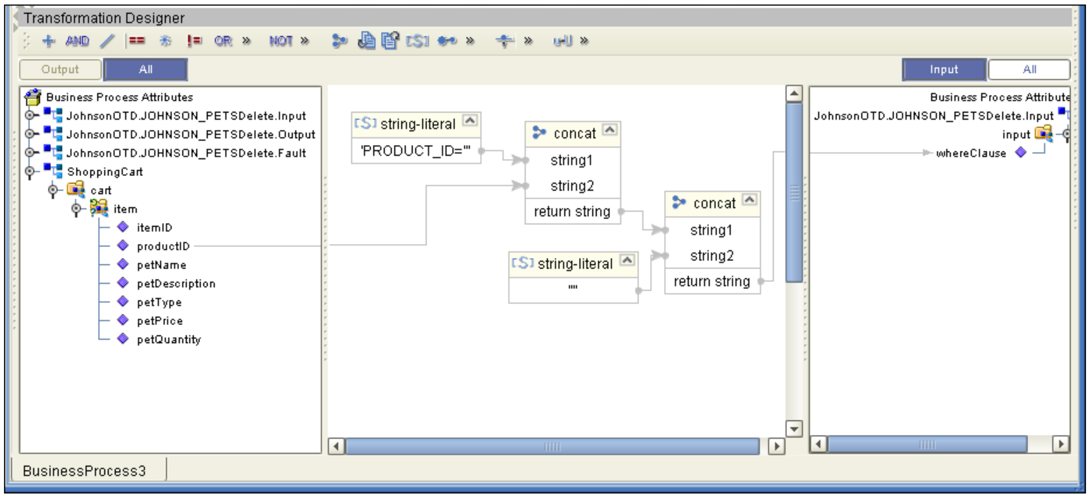
The figure below shows a sample eInsight Business Process using the Delete operation. In this process, the operation deletes rows with a certain product ID from the shopping cart.

Figure 43 Delete Sample Business Process



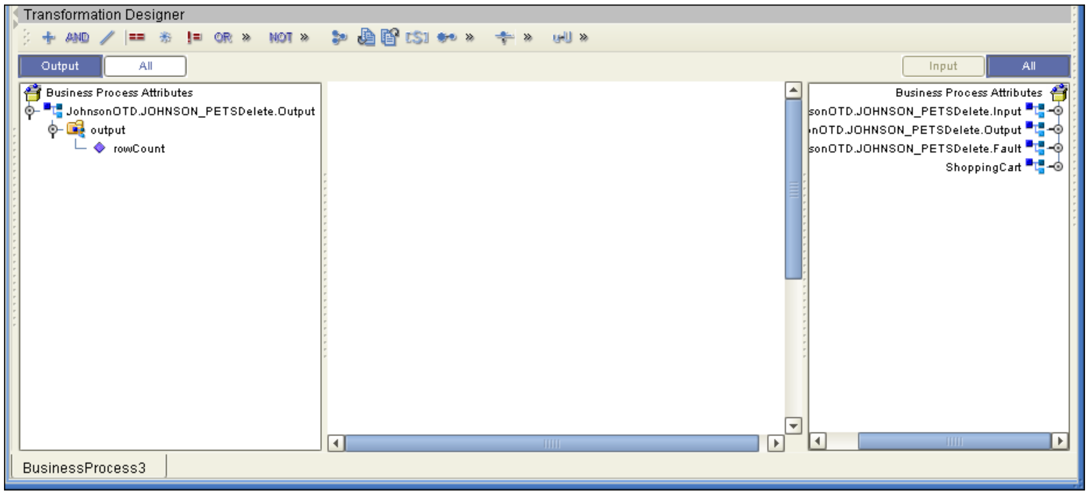
The figure below shows the definition of the where() clause for the Delete operation.

Figure 44 Delete Input



The figure below shows the output of the Delete operation, which is a status indicating the number of rows deleted.

Figure 45 Delete Output


Transformation Designer
Output
All
Input
All
Business Process Attributes
JohnsonOTD.JOHNSON_PETSDelete.Output
output
rowCount
sonOTD.JOHNSON_PETSDelete.Input
nOTD.JOHNSON_PETSDelete.Output
sonOTD.JOHNSON_PETSDelete.Fault
ShoppingCart
BusinessProcess3

5.3 Using the Sample Project in eGate

Follow the instructions given in **Importing the Sample Project** on page 38 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.

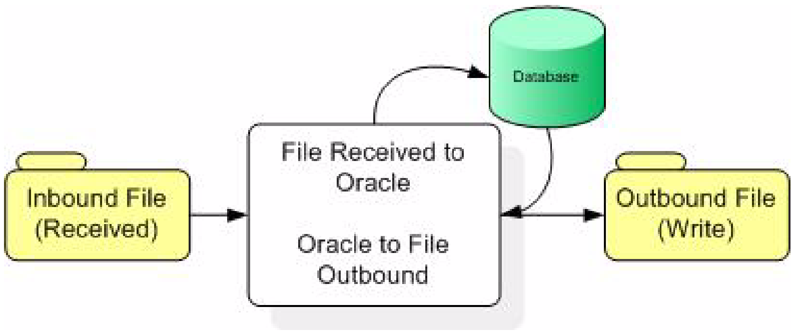
The data used for this projects is within a table called DBEmployee. The table contains the following columns with the following columns:

Table 3 Sample project data
Table 4

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 46 Database project flow



To work with the sample project, follow the instructions given in the *eGate 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 On the Connectivity Map canvas, double click the eWay icon located between the **File1** and **JCESelect1** service.

2 On the resulting **Templates** window, select **Inbound File eWay** and click **OK**.

3 On 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 Oracle1 eWay:

1. On the Connectivity Map canvas, double click the eWay icon located between the **JCESelect1** service and **Oracle1** database.
2. On the resulting **Templates** window, select **Outbound Oracle eWay** and click **OK**.
3. On the Properties window, enter the appropriate configurations for the Outbound Oracle eWay and click OK. See **Working with eWay Property Sheets** on page 10. For this sample, the default settings are used.
4. When you have completed your selections, click **OK**.

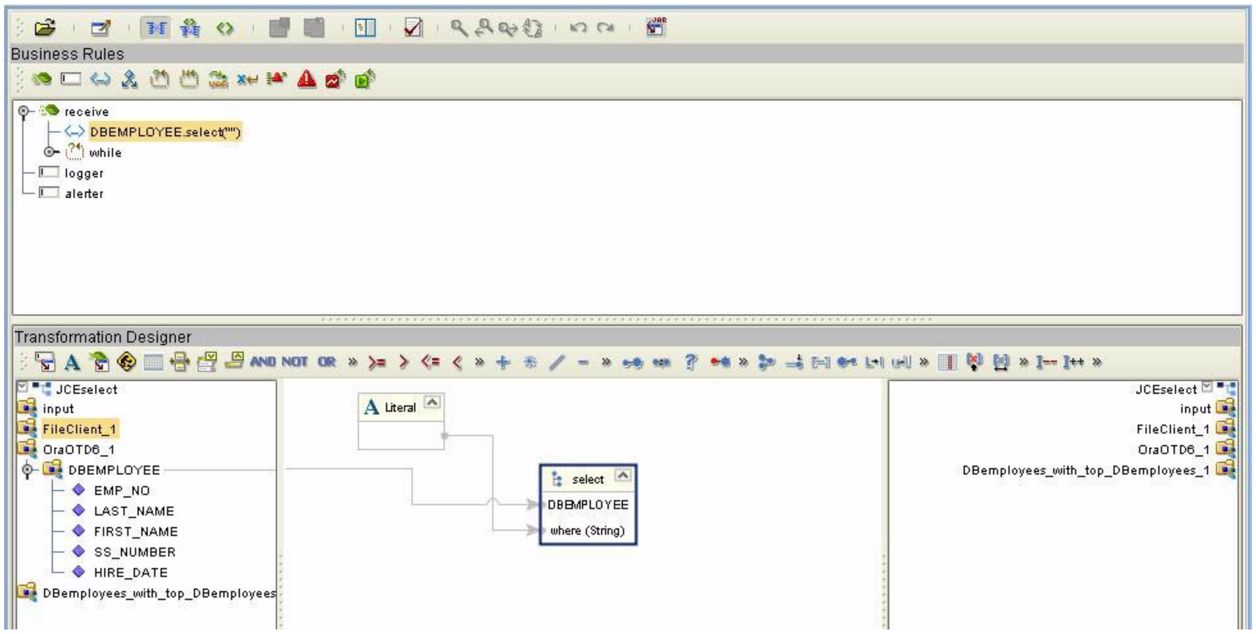
To configure the Outbound File eWay:

1. On the Connectivity Map canvas, double click the eWay icon located between the **JCESelect1** and **File2** service.
2. On the resulting **Templates** window, select **Outbound File eWay** and click **OK**.
3. On 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. On the Transformation Designer window's left pane, expand the **OraOTD6** node and the **DBEMPLOYEE** sub-node by selecting them and double-clicking. See **Figure 47**.

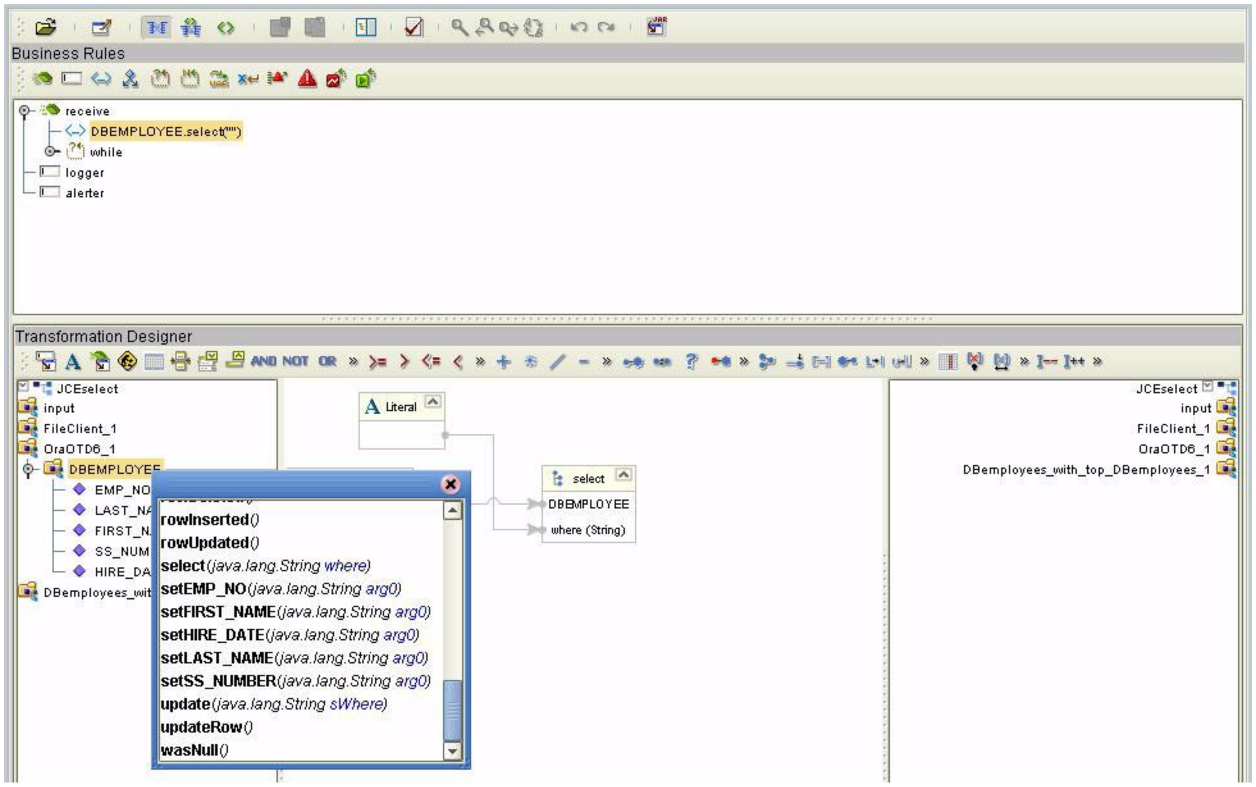
Figure 47 DBEMPLOYEE Select



2 To create a new rule, highlight **DBEMPLOYEE** and right click to open the Methods sub-menu.

3 From the **Select a method to call** submenu, highlight the **select(java.lang.String where)** method and enter. See **Figure 48**.

Figure 48 Selecting a method

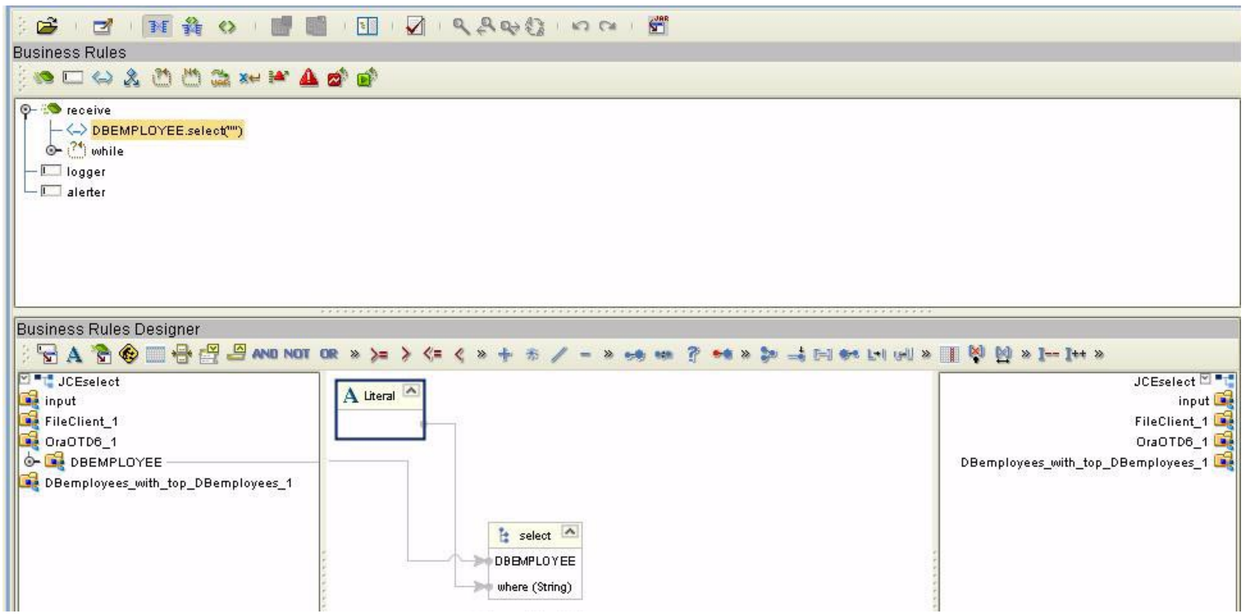

Business Rules
receive
DBEMPLOYEE.select("")
while
logger
alerter
Transformation Designer
JCEselect
input
FileClient_1
OraOTD6_1
DBEMPLOYEE
EMP_NO
LAST_NA
FIRST_N
SS_NUM
HIRE_DA
DBemployees_wit
Literal
select
DBEMPLOYEE
where (String)
DBemployees_with_top_DBemployees_1
rowInserted()
rowUpdated()
select(java.lang.String where)
setEMP_NO(java.lang.String arg0)
setFIRST_NAME(java.lang.String arg0)
setHIRE_DATE(java.lang.String arg0)
setLAST_NAME(java.lang.String arg0)
setSS_NUMBER(java.lang.String arg0)
update(java.lang.String sWhere)
updateRow()
wasNull()

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

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

6 On the **Literal** box, place your cursor inside the box and while depressing the left mouse key, drag a new connection line between the **Literal** box to the **where(String)** method node located next to the **select()** box. See **Figure 49**.

Figure 49 Literal mapping



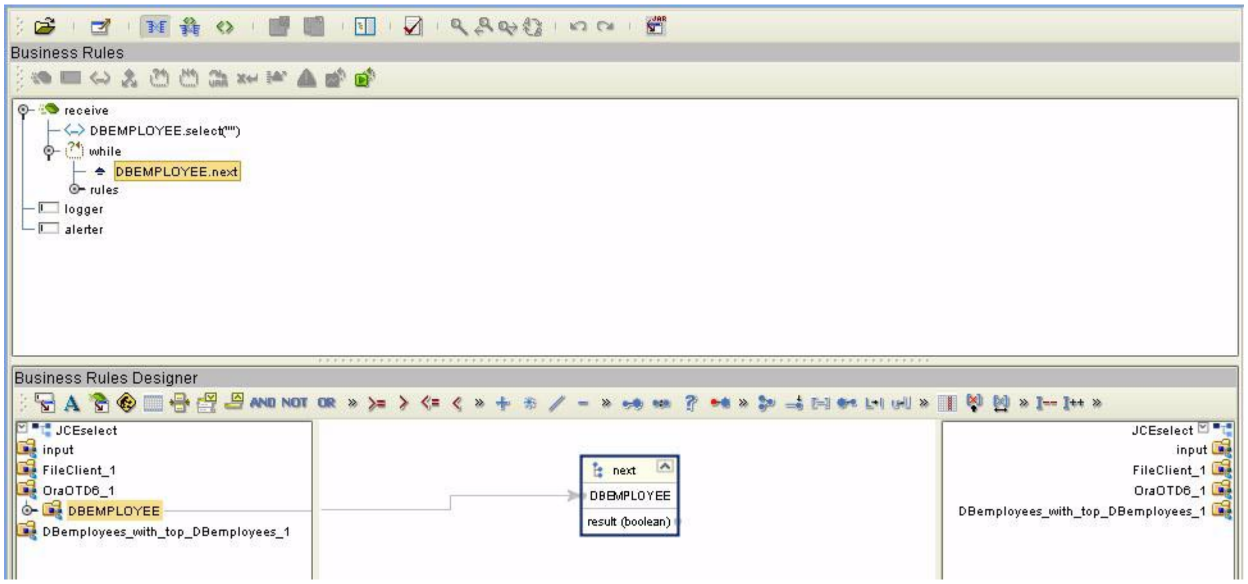
A new rule is created.

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

7 On 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 On the **Business Rules** list, highlight the condition. While highlighted, select DBEMPLOYEE located under the OraOTD6_1 node. Right click and select the next() method from the sub-menu. See **Figure 50**.

Figure 50 next()

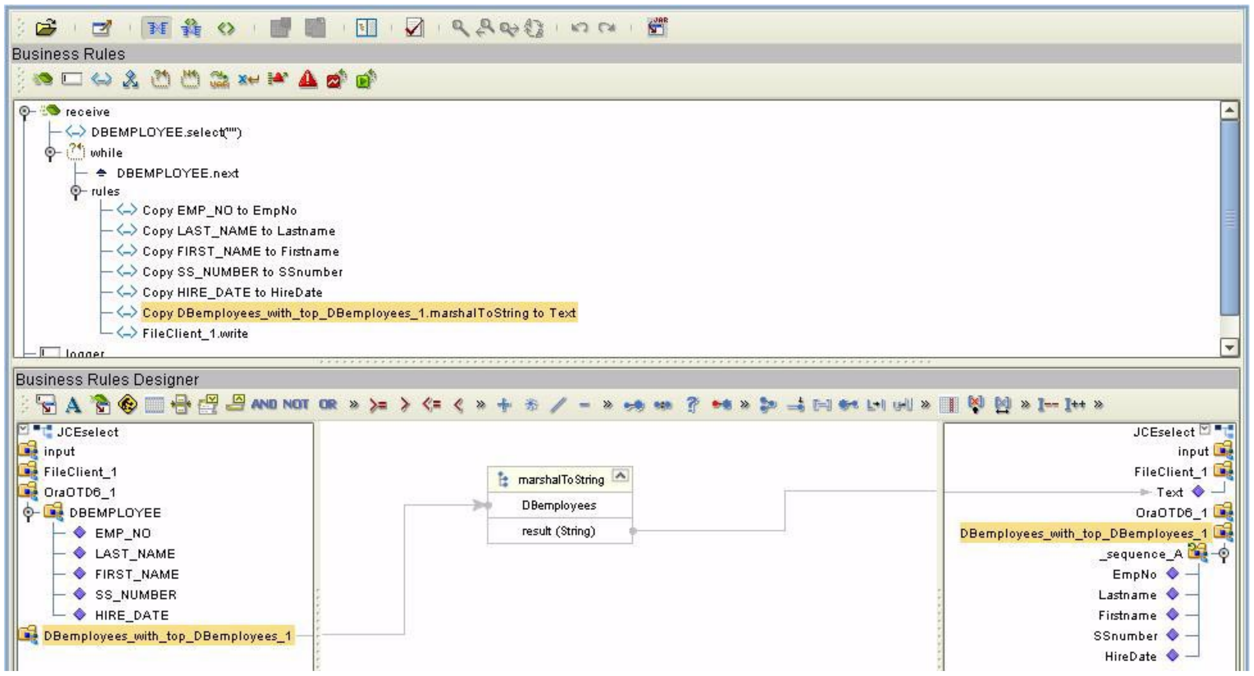


9 On the Business Rules list, highlight the **New Rule** located under the **DBEMPLOYEE.next** condition. On the left pane of the Business Rules canvas, expand the **DBEMOPLOYEE** sub-node located under the **OraOTD6_1** OTD node to expose the database columns. On the right side of the Business Rules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

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

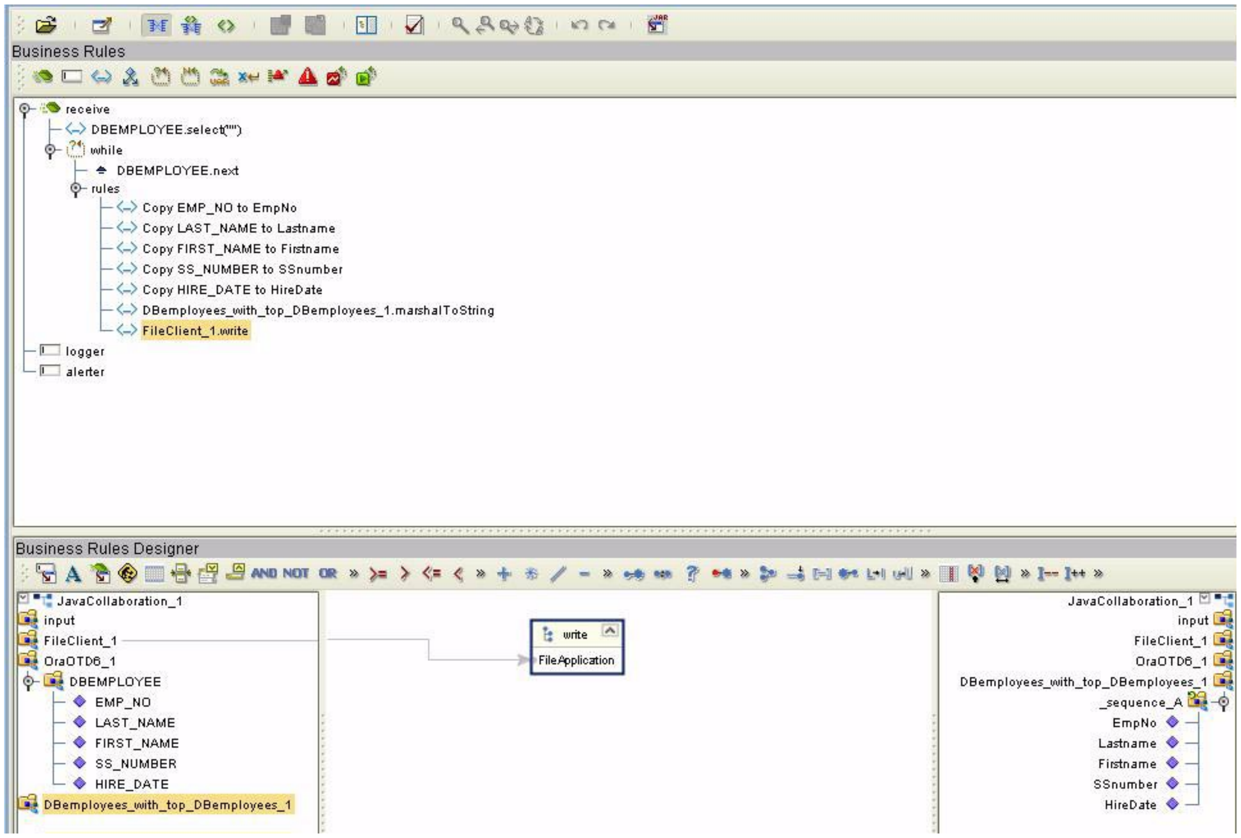
10 On the Business Rules list, highlight **Copy Hire_Date to HireDate**. On the Business Rules Designer's left pane, right-click on the **DBemployees_with_top_DBemployees_1** node. From the sub-menu, select and click **Select a method to call**. From the sub-menu, select and double click **marshalToString()**. See Figure 51.

Figure 51 marshalToString()



11 On the Business Rules list, highlight **DBemployees_with_top_DBemployees_1.marshalToString**. On the Business Rules Designer's left pane, right-click on the **FileClient_1** node. From the sub-menu, select and click **Select a method to call**. From the sub-menu, select and double-click **write()**. See

Figure 52 write()



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

5.3.3. Creating the External Evironment

To review the components of the Sample project, there is an Inbound and an Outbound File eWay, an eWay, and a Service.

To create the external environment for the Sample project:

On the Environment Explorer, highlight and right-click the Oracle eWay profile. Select **Properties**. Enter the configuration information required for your Outbound Oracle eWay. See **Setting the Properties in the Outbound eWay Environment** on page 15

5.3.4 Deploying a Project

To deploy a project, please see the "*eGate Integrators User's Guide*".

5.3.5. Running the Sample

For instruction on how to run the Sample project, see the *eGate Tutorial*.

Once the process has completed, the Output file in the target directory configured in the Outbound File eWay will contain 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 multibyte) before setting size. A maximum size must be specified.
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 1 byte per row. Consider the character set (single-byte or multibyte) before setting size.
Raw	Variable-length raw binary data.	Variable for each row in the table, up to 2000 bytes per row. A maximum size must be specified. Provided for backward compatibility.
Long	Variable-length character data.	Variable for each row in the table, up to 2^{32} - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.
Clob		Up to 2^{32} - 1 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 7 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 should help:

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
Varchar2	String	Direct Assign	Any character
Char	String	Direct Assign	Any character

Oracle Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
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

* When using Prepared Statements with the Inbound Oracle eWay, you will need to change the decimal type to a double data type in the OTD.

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
Varchar2	String	Direct Assign	Any characters
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:

- Real
- Float
- CLOB.

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

Long RAW for Prepared Statements and Stored Procedure support:

The following 2 parameters must be set prior to the Insert/Update/Delete statement.

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

5.6.2. The Table

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 allows you to perform query, update, insert, and delete SQL operations in a table.

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

- SetConcurrencytoUpdatable
- SetConcurrencytoReadOnly
- SetScrollTypetoForwardOnly
- SetScrollTypetoScrollSensitive
- SetScrollTypetoInsensitive

The methods should be called before executing the select() method if used. 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.

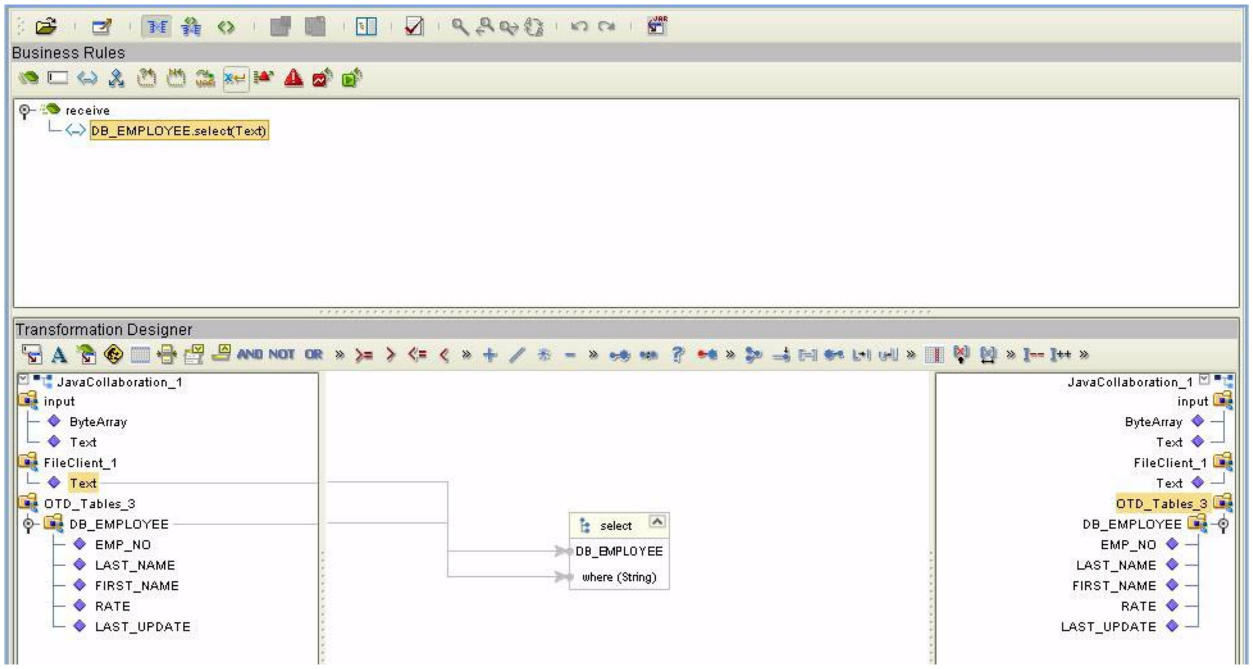
3 Process the return record within a **while()** loop.

For example:

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

//while DB_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 53 Select()



The Insert Operation

To perform an insert operation on a table

1 Execute the **insert()** method. Assign a field.

2 Insert the row by calling **insertRow()**

This example inserts an employee record.

```
//DB_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();

//Table_OTD_1.commit
  Table_OTD_1.commit();
}
```

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() );

//SALES_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() );

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

```
//DB_sales_orders_1.commit
  DB_sales_orders_1.commit();

   }
```

The Delete Operation

To perform a delete operation on a table

1. Execute the **delete()** method.
2. Move to the row that you want to delete.
3. Delete the row by calling **deleteRow()**.

   In this example DELETE an employee.

```
//DB_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. It allows you to execute a stored procedure. Remember that while in the Collaboration Editor you can drag and drop nodes from the OTD into the Collaboration Editor.

Note: *When creating a Package Stored Procedure in the Database Wizard, you must select* ***Use fully qualified names****.*

Note: *Stored Procedure Resultsets are support in Java collaborations only.*

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

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 PreparedStatementAgent class, simplifies the whole process of determining whether any results, be it 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.

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

Collaboration usability for a Stored Procedure ResultSet

The Column data of the ResultSets can be dragged-and-dropped from their XSC nodes to the Business Rules. Below is a code snippet that can be generated by the Collaboration Editor:

```
// resultsAvailable() will be 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 2
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.*

After calling "**resultsAvailable()**", 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, it is recommended that for maximum portability, all of the results in a ResultSet object should be retrieved before OUT parameters are retrieved. Therefore, you should 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. You should not call both methods in your java code. Doing so may result in skipped data from one of the ResultSets when more than one ResultSet is present.
- The methods **available()** and **getResultSet()** can not be used in conjunction with multiple ResultSets being 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 assure the indexes match the stored procedure. By doing this, your ResultSet indexes are preserved.
- Generally, getMoreResults does not need to be called. It is needed if you do not want to use our enhanced methods and you want to follow the traditional JDBC calls on your own.

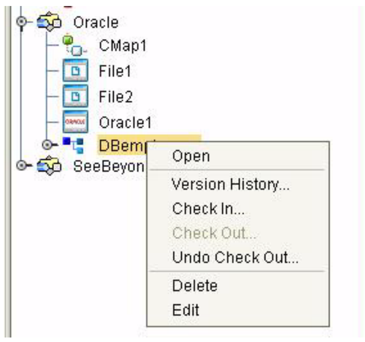
5.7 Creating or Editing an OTD from an Existing OTD

Editing an Existing OTD

To edit an OTD do the following:

1 On the Enterprise Explorer, right click on the OTD. From the submenu, click Edit. See **Figure 54**.

Figure 54 OTD Edit Menu Item



The Database Wizard will open 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 will fail 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. Following the steps in **Editing an Existing OTD** on page 68, make your edits and save the OTD with a new name.

5.8 Using XA

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

5.9 Alerting and Logging

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

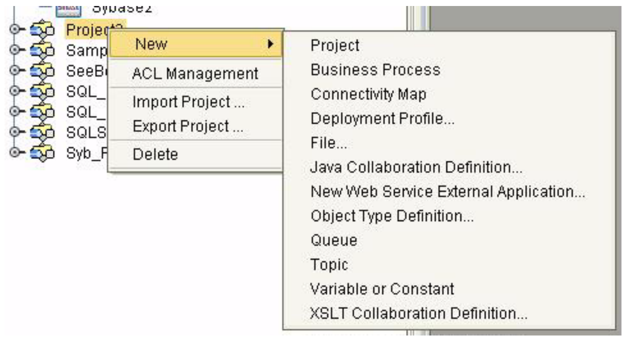
Appendix A

Using CLOB in the Oracle eWay

To use a CLOB in the Oracle eWay you will need to:

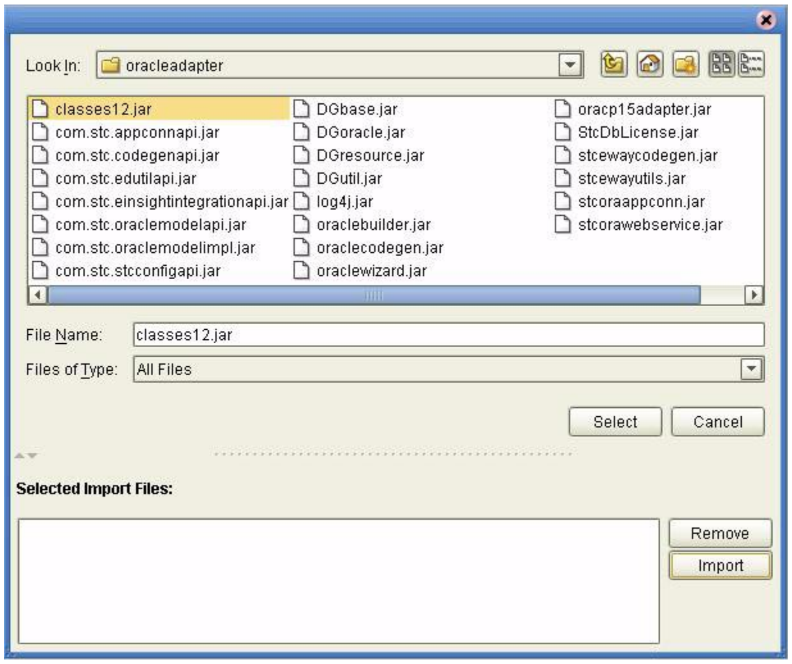
1 On the Enterprise Designer, select the project and right click. Select New, File. See **Figure 55**.

Figure 55 File Import Feature



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**.

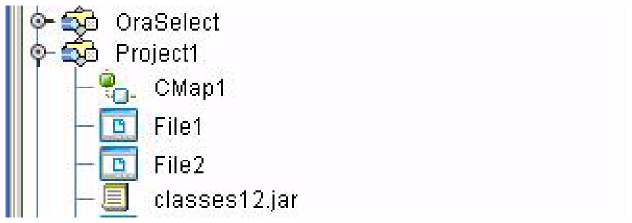
Figure 56 Importing Classes12.jar



3 Click Select.

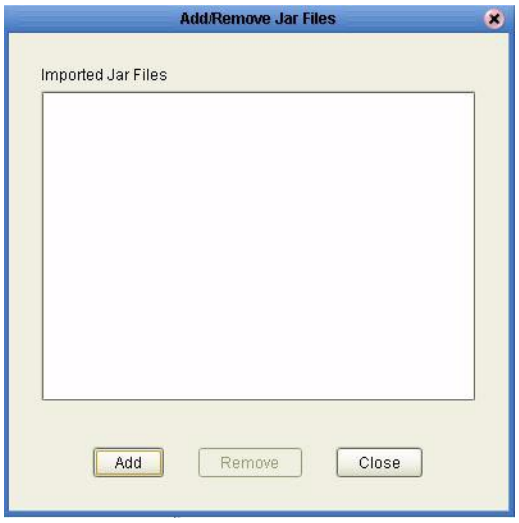
4 Click Import. See **Figure 57**.

Figure 57 Classes12.jar in a Project



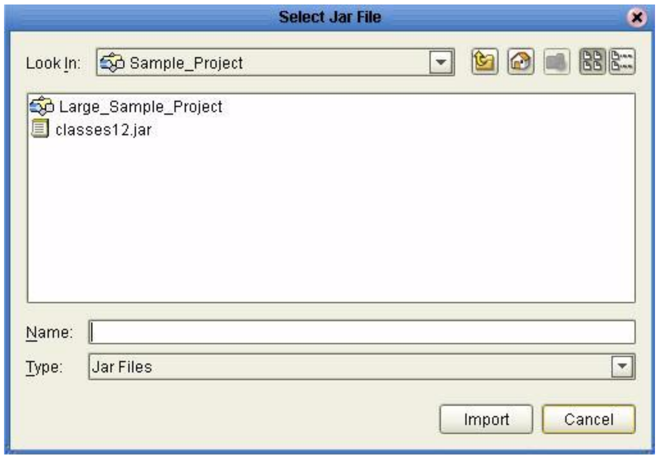
5 To load the **classes12.jar** file into your Java Collaboration, select the Import JAR File icon. On the **Add/Remove Jar Files** click **Add**. See **Figure 58**.

Figure 58 Add/Remove Jar Files



6 On the Select Jar File window, select the classes12.jar file and click Import. See **Figure 59**.

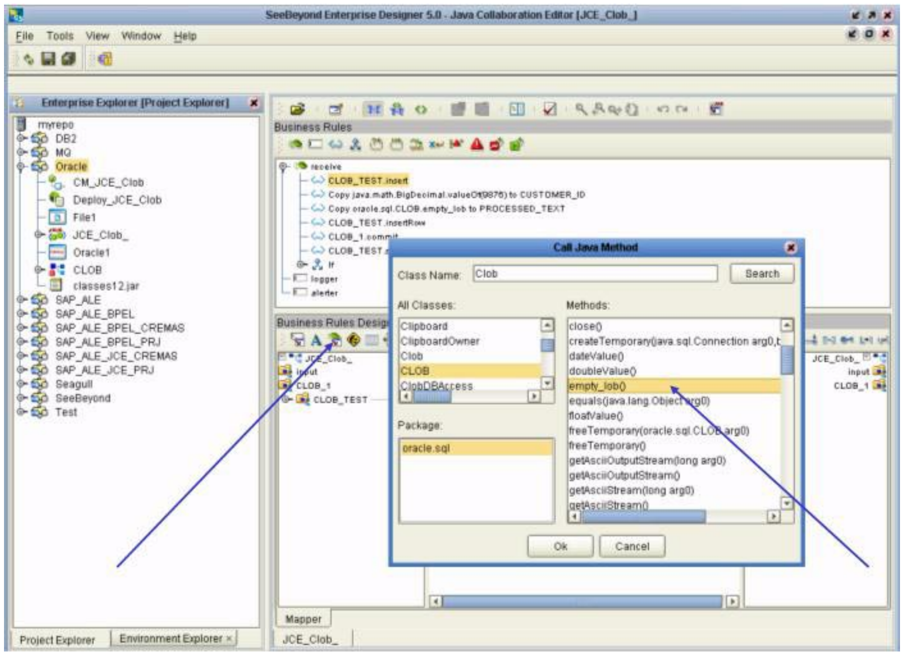
Figure 59 Select Jar File



7 On the **Add/Remove Jar Files** window, click **Close**.

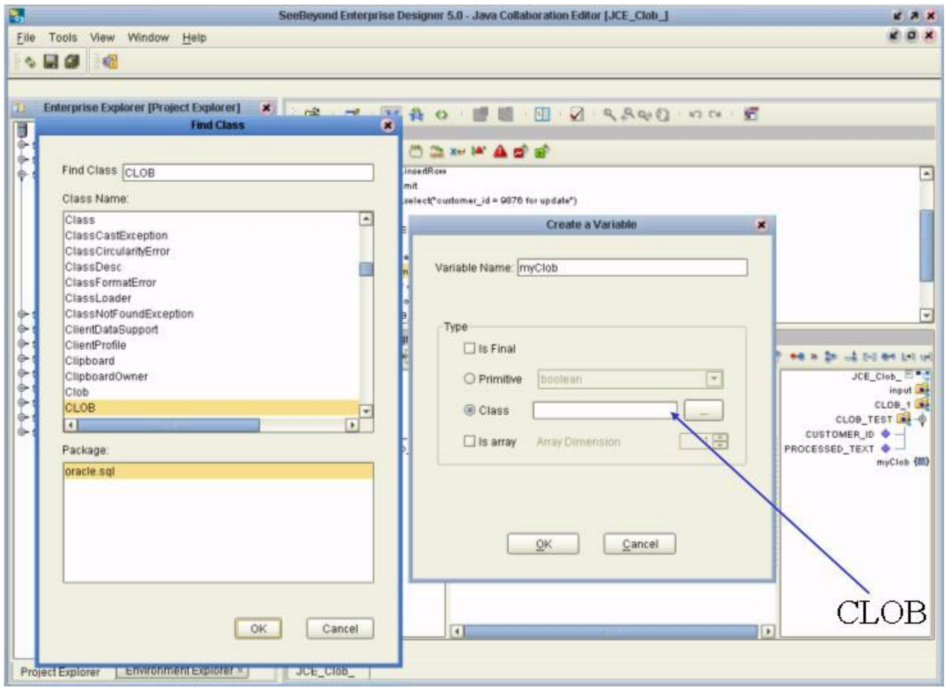
8 On the Business Rules Designer, call the CLOB method. See **Figure 60**.

Figure 60 Call the CLOB Method


SeeBeyond Enterprise Designer 5.0 - Java Collaboration Editor [JCE_Clob_]
Call Java Method
Class Name: Clob
Search
All Classes:
Clipboard
ClipboardOwner
Clob
CLOB
ClobDBAccess
Methods:
close()
createTemporary(java.sql.Connection arg0,
dateValue()
doubleValue()
empty_lob()
equals(java.lang.Object arg0)
floatValue()
freeTemporary(oracle.sql.CLOB arg0)
freeTemporary()
getAsciiOutputStream(long arg0)
getAsciiOutputStream()
getAsciiStream(long arg0)
getAsciiStream()
Package:
oracle.sql
Ok
Cancel

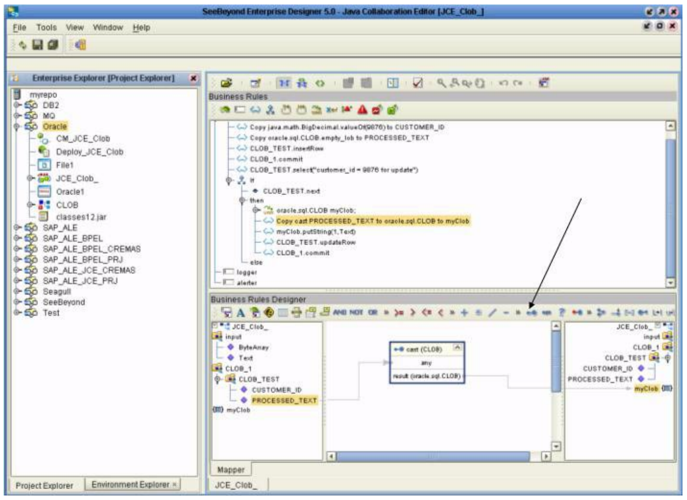
9 Create a local variable. See **Figure 61**.

Figure 61 Create a Local Variable



10 In the Business Rules Designer, drag the **CLOB** to the Local Variable using the Cast method. Click **Yes** to the Incompatible DataType warning. See **Figure 62**.

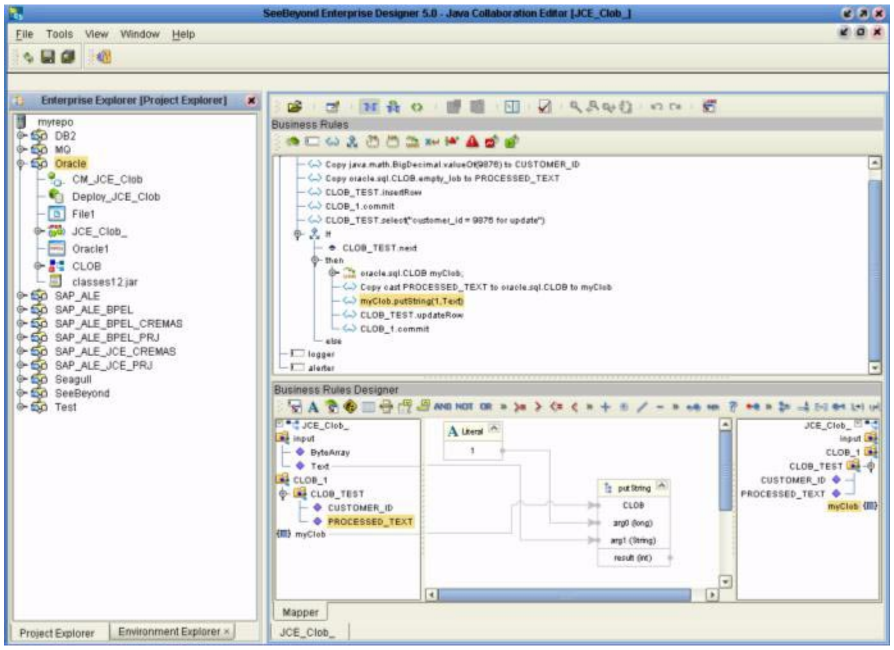
Figure 62 Drag CLOB to the Local Variable


SeeBeyond Enterprise Designer 5.0 - Java Collaboration Editor [JCE_Clob_]
File Tools View Window Help
Enterprise Explorer [Project Explorer]
Business Rules
Business Rules Designer
Mapper
Project Explorer
Environment Explorer
cast (CLOB)
any
result (oracle.sql.CLOB)

```
Copy java.math.BigDecimal.valueOf(9876) to CUSTOMER_ID
Copy oracle.sql.CLOB.empty_lob to PROCESSED_TEXT
CLOB_TEST.insertRow
CLOB_1.commit
CLOB_TEST.select("customer_id = 9876 for update")
if
    CLOB_TEST.next
  then
    oracle.sql.CLOB myClob;
    Copy cast PROCESSED_TEXT to oracle.sql.CLOB to myClob
    myClob.putString(1,Text)
    CLOB_TEST.updateRow
    CLOB_1.commit
  else
logger
alerter
```

11 Use the CLOB putString method assigning 1 to Arg(). See **Figure 63**.

Figure 63 CLOB putString Method


SeeBeyond Enterprise Designer 5.0 - Java Collaboration Editor [JCE_Clob_]
File Tools View Window Help
Enterprise Explorer [Project Explorer]
Business Rules
Business Rules Designer
Literal
1
putString
CLOB
arg0 (long)
arg1 (String)
result (int)
Mapper
Project Explorer
Environment Explorer

```
Copy java.math.BigDecimal.valueOf(9876) to CUSTOMER_ID
Copy oracle.sql.CLOB.empty_lob to PROCESSED_TEXT
CLOB_TEST.insertRow
CLOB_1.commit
CLOB_TEST.select("customer_id = 9876 for update")
if
    CLOB_TEST.next
  then
    oracle.sql.CLOB myClob;
    Copy cast PROCESSED_TEXT to oracle.sql.CLOB to myClob
    myClob.putString(1,Text)
    CLOB_TEST.updateRow
    CLOB_1.commit
  else
logger
alerter
```

12 In the Java Collaboration Editor, the java code should look something like:

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

        //@map:CLOB_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();

            //@map:CLOB_1.commit
            CLOB_1.commit();
        }
    }
```

Index