bea

BEA WebLogic
Adapter for
RDBMS

User Guide

Copyright

Copyright © 2002 BEA Systems, Inc. All Rights Reserved.

Copyright © 2002 iWay Software. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

BEA WebLogic Adapter for RDBMS User Guide

Part Number	Date	Release
N/A	November 2002	7.0 with Service Pack 1

Table of Contents

3. Defining an Application View

4. Service XML Examples

5. Event Adapter Examples

About This Document

The *BEA WebLogic Adapter for RDBMS User Guide* is organized as follows:

- Chapter 1, "Introducing the BEA WebLogic Adapter for RDBMS," gives an overview of the adapter and how it works.

- Chapter 2, "Using the BEA Application Explorer With an RDBMS," describes how to use the BEA Application Explorer with an RDBMS.

- Chapter 3, "Defining an Application View," provides information on creating and deploying application views that include the RDBMS Adapter.

- Chapter 4, "Service XML Examples," provides service examples that use the RDBMS service adapter.

- Chapter 5, "Event Adapter Examples," provides event adapter examples that use the RDBMS event adapter.

What You Need to Know

This document is written for system integrators who need to develop client-server interfaces between RDBMS and third-party enterprise information system (EIS) applications.

It provides information about using the BEA WebLogic Adapter for RDBMS tools to develop connections between a WebLogic Integration client and an RDBMS.

It is assumed that readers have a general understanding of Microsoft Windows and UNIX systems, as well as:

■ Some experience using Enterprise Information System (EIS) and integration products and an understanding of RDBMS and SQL products with which this software will be integrating.

■ Knowledge of EIS concepts.

■ General knowledge of RDBMS concepts and configuration options.

■ Specific business application knowledge of the target schema.

■ Knowledge of integration processes and data models for the required application area.

■ General knowledge of BEA Integration architecture.

■ General knowledge of XML concepts.

Extensive internal knowledge of the specific SQL environment is not required, but may be helpful in learning about the BEA WebLogic Adapter for RDBMS.

Related Information

The following documents provide additional information for the associated software components:

- *BEA WebLogic Adapter for RDBMS Installation and Configuration Guide*

- *BEA WebLogic Adapter for RDBMS Release Notes*

- BEA WebLogic Server installation and user documentation, which is available at the following URL:

  `http://edocs.bea.com/more_wls.html`

- BEA WebLogic Integration installation and user documentation, which is available at the following URL:

  `http://edocs.bea.com/more_wli.html`

- Specific RDBMS installation and user documentation.

Contact Us!

Your feedback on the BEA WebLogic Adapter for RDBMS documentation is important to us. Send us e-mail at `docsupport@bea.com` if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the BEA WebLogic Adapter for RDBMS documentation.

In your e-mail message, please indicate which version of the BEA WebLogic Adapter for RDBMS documentation you are using.

If you have any questions about this version of BEA WebLogic Adapter for RDBMS, or if you have problems installing and running BEA WebLogic Adapter for RDBMS, contact BEA Customer Support through BEA WebSupport at `www.bea.com`. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

- Your name, e-mail address, phone number, and fax number

- Your company name and company address

- Your machine type and authorization codes

- The name and version of the product you are using

- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void `**`commit`**` ( )`
`monospace italic text`	Identifies variables in code. *Example*: `String `*`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR

Convention	Item
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

[ ]	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

...	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

1 Introducing the BEA WebLogic Adapter for RDBMS

This section gives an overview of the adapter and how it works. It includes the following topics:

■ Introduction

■ Event Adapter

■ Service Adapter

Introduction

Since most custom and packaged applications are built with relational databases, RDBMS systems must be taken into consideration in any enterprise integration strategy. The BEA WebLogic Adapter for RDBMS incorporates in-depth knowledge of relational database query access, and transaction, replication, and copy management technologies to optimize the use of databases with enterprise application systems.

The BEA WebLogic Adapter for RDBMS enables integration with RDBMS systems by functioning as a service adapter for accessing a database and as an event adapter for listening to a database. In both cases, the query or stored procedure call is expressed in XML. This provides a convenient and simple method for integrating databases with enterprise applications using WebLogic Integration.

Key features of the BEA WebLogic Adapter for RDBMS include:

- Asynchronous, bi-directional message interactions between applications and databases, including IBM DB2, IBM Informix, Microsoft SQL Server, Oracle, and Sybase RDBMSs.

- The BEA Application Explorer, which makes use of metadata on database servers to build application views (XML schemas for database events and SQL requests or responses) that can be used in workflows.

- Integration of requests and table event (outbound) operations in workflows.

- JDBC 2.0 standard SQL operations (DELETE, INSERT, SELECT, and UPDATE) and the execution of stored procedures against DB2, Informix, MS SQL Server, Oracle, and Sybase.

- Oracle object-relational extensions such as processing of nested tables and arrays in accessing PL/SQL stored procedures, or supporting outbound database rows on Oracle AQ queues.

Service Adapter

The service adapter is a generic service that is capable of processing SQL statements embedded in XML requests and forwarding them to an RDBMS. The RDBMS returns the data to the adapter, which returns the data to the client.

Service adapters receive an XML request document from a client and invoke a specific function in the target enterprise information system (EIS). The adapters receive request messages and provide responses depending on the request. There are two possible methods for invoking an RDBMS service adapter:

- Asynchronous. The client application issues a service request and then processes it without waiting for the response.

- Synchronous. The client waits for the response before proceeding with further processing.

WebLogic Integration supports both of these service adapter invocations, relieving you from having to provide this functionality within your own application code.

The service adapter can:

- Receive service requests from an external client.

- Transform the XML request document into the EIS data format. The request document conforms to the request XML schema for the service, which is based on metadata in the EIS.

- Invoke the underlying function in the EIS and wait for its response.

- Transform the response from the EIS data format to an XML document that conforms to the response XML schema for the service, which is based on metadata in the EIS.

Event Adapter

The event adapter supports the capture of events from applications that write to a relational table. It captures the data and performs operations based on the content of the rows. The event adapter reads one or more rows from the table and creates an XML document representing the column data in each row. Additional business logic facilities can then be applied to the constructed XML document, including transformation, validation, security management, and application processing. Transformations by business logic can include deleting rows or altering column values. The resulting XML is formatted as an application view and sent as an event to WebLogic Integration.

The event adapter can:

■ Monitor data changes by repeatedly performing an SQL query. The SQL listener also supports customized user exits with Java classes to define custom operations on the rowsets.

■ Be configured to operate one row at a time or to operate on sets of data, only sending events to a business process workflow when a specified minimum number of rows become available in the source DBMS.

■ Allow the configuration of an optional SQL post-query statement, which performs certain DBMS operations after the adapter has sent the rowset (formatted as XML) to a business process workflow. The default operation is to delete the rows that have been transferred to the workflow. However, other options may include moving the rows to an archive table or marking the rows with an SQL UPDATE.

■ Support complex configurations. For example, you may need to extract information periodically from a base table and incorporate reference data from an additional table. The base table and reference table cannot have records deleted from them. In this case, the adapter uses a temporary table to maintain the sequenced rows in the base table. The temporary table is seeded with a starting value for the sequence. It holds the last value of the sequence field processed by the SQL listener, allowing multiple event operations to collect updates while avoiding sending duplicates to a business process workflow.

■ Support user-defined exits, which can be implemented for more complex programming or external database operations. For example, an operating system program may be executed to facilitate an import or export process within a custom application.

Supported Data Types

The following are data types that are supported with the BEA WebLogic Adapter for RDBMS.

Database	Data Types Successfully Tested
Sybase	■ integer ■ smallint ■ char ■ float ■ double precision ■ smalldatetime ■ datetime ■ varchar ■ text ■ real ■ decimal ■ numeric
Oracle	■ number ■ date ■ varchar2 ■ char ■ float ■ long

Local XA Transaction Support

The LocalTransaction interface is exposed to adapter clients using the Common Client Interface (CCI) Connection class. Currently, the application view interface does not use the CCI LocalTransaction interface. To manage a local transaction, a user must first acquire a LocalTransaction from the Connection object.

Local Transaction Management Contracts

A local transaction management contract is created when an adapter implements the `javax.resource.spi.` LocalTransaction interface to provide support for local transactions that are performed on the system's underlying resource manager. This type of contract enables an application server to provide the infrastructure and run-time environment for transaction management. Application components rely on this transaction infrastructure to support their component-level transaction model.

For more information about transaction demarcation support, see the following URL:
`http://java.sun.com/blueprints/guidelines`
`/designing_enterprise_applications/transaction_management`
`/platform/index.html`

Connector Support for Local Transactions with No User-Defined Transaction Demarcation

The following is a scenario for supporting application view local transactions within the Application Integration Plug-in. This scenario is similar to `TX_REQUIRES_NEW` for EJB transactions because the connector supports only local transactions.

In this scenario, the connector supports only local transactions and the WebLogic Integration Studio does not explicitly demarcate the start and end of a local transaction. WebLogic Integration allows the connector to participate in the global transaction by providing an XA Wrapper around the LocalTransaction object. The XA Wrapper handles all the method calls on the XAResource interface that cannot be delegated to the LocalTransaction instance. WebLogic Integration allows only one non XA resource in the transaction chain. As a result, a user can have only one application view LocalTransaction within a workflow.

The BEA WebLogic Adapter for RDBMS supports XA local transactions.

2 Using the BEA Application Explorer With an RDBMS

This section describes how to use the BEA Application Explorer. The underlying technology used to access and create the appropriate schemas differs for the individual application systems being explored, but the user interface is consistent. In this section the functionality of the BEA Application Explorer is presented using the BEA WebLogic Adapter for RDBMS as an example.

The BEA Application Explorer supports the creation of schemas based on specific tables and resulting answer sets. To obtain the metadata about the relational database management system (RDBMS) tables and answer sets, the BEA Application Explorer connects to the RDBMS using the same JDBC drivers that the BEA WebLogic Adapter for RDBMS uses.

This section includes the following topics:

- Connecting to an RDBMS

- Connecting to an RDBMS Using an Existing Connection

- Disconnecting from an RDBMS

- Removing a Connection

- Viewing Table-Based Metadata

- Viewing Stored Procedures

- Generating Event Schemas

- Generating Service Schemas

- Combining Parameterized SQL Feature with Stored Procedures

- Generating Schemas for Stored Procedures

- Removing Schemas

Connecting to an RDBMS

Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer. When you first start the BEA Application Explorer, you can view the main panes. The left pane displays all the adapters supported by the version of the BEA Application Explorer being used.

Figure 2-1   BEA Application Explorer Window
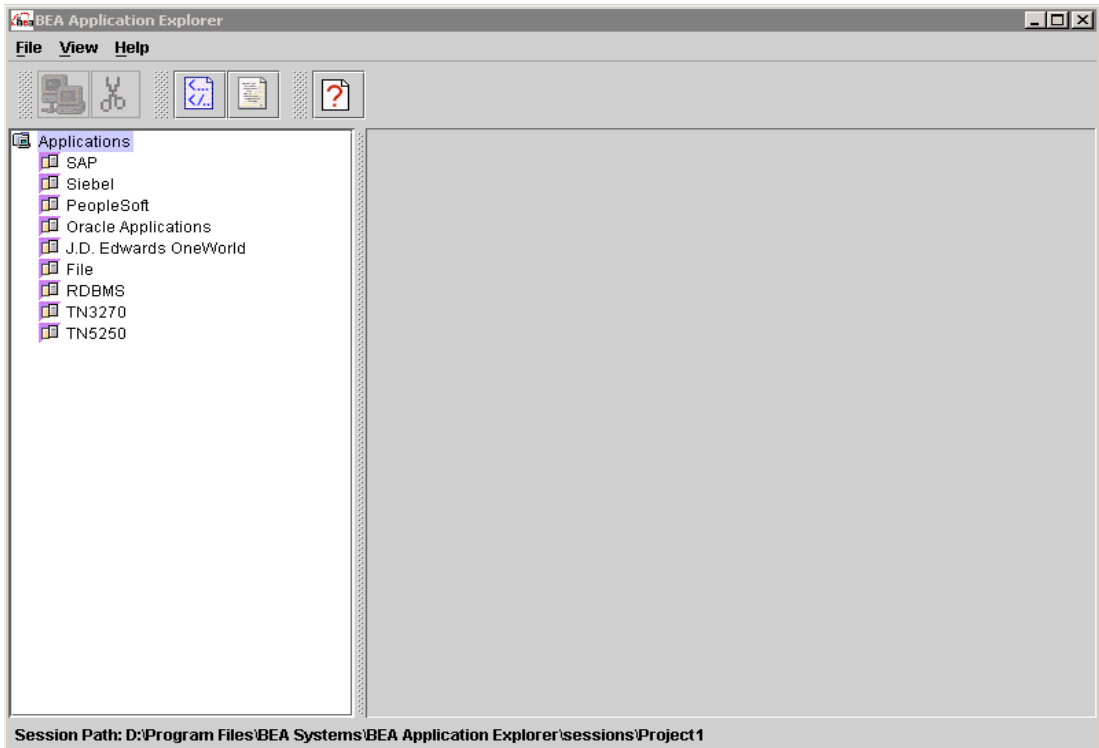
To connect to a specific RDBMS, you must first create a new connection.

1. Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer.

    The BEA Application Explorer opens.

2. Right-click the RDBMS node.
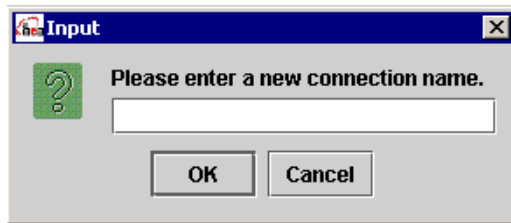
Figure 2-2   BEA Application Explorer - New Connection
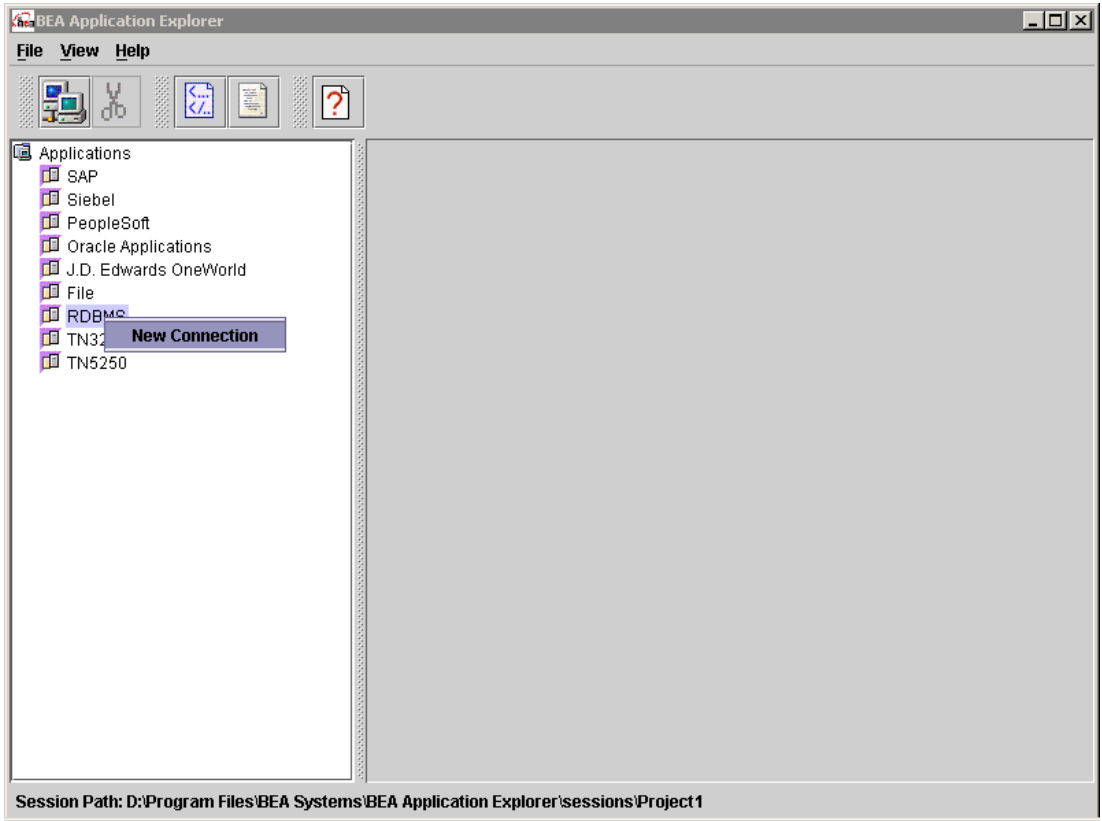
3. Select the New Connection option to create a new connection.

Note: If a connection was previously created, see "Connecting to an RDBMS Using an Existing Connection" on page 2-9.

The New Connection prompt displays.

Figure 2-3   New Connection Name Input Window

4. Enter a name for the connection. Use a descriptive name, for example, Oracle817.

   The name entered is used to build a subdirectory underneath the session path specified, as well as to identify the connection. Since the connection name is also used as a subdirectory name in your session path, it must contain characters that are considered to be valid for a directory name by the operating system on which the iAE is executed. For example, the connection name #SAP is invalid on a Windows operating system.

   Refer to your operating system reference material to determine which are valid characters for use in a directory name.

5. Click OK.
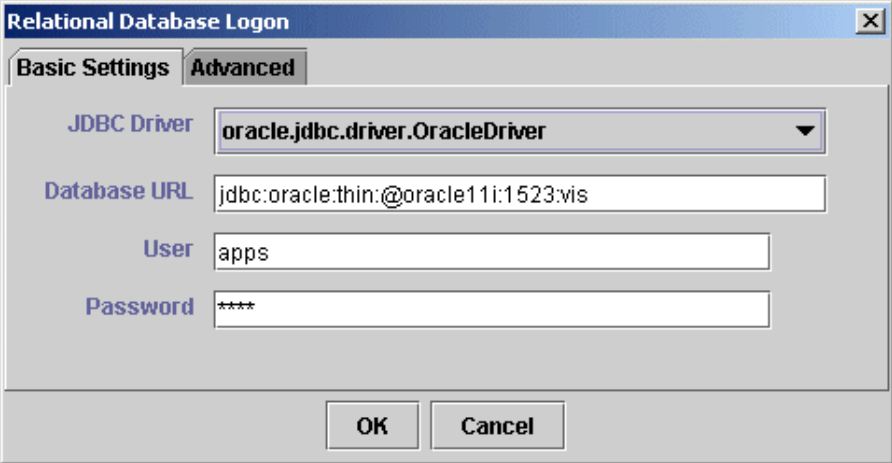
   The connection name is verified for the system. If you enter an invalid connection name, a new input box opens and asks for the connection name to be entered again.

Figure 2-4   Invalid Connection Name Message
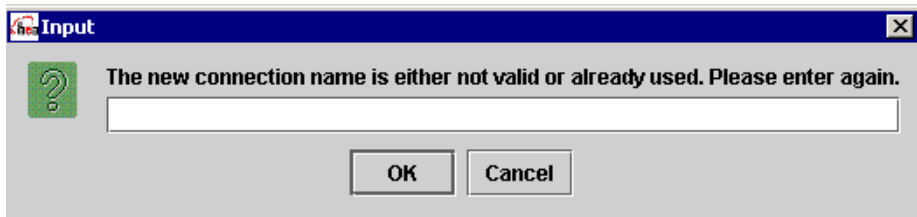
After you enter a valid connection name, the system prompts you for connection information.

Figure 2-5   Relational Database Logon Window

6. From the drop-down list, select the appropriate JDBC driver to use and enter the appropriate information for the connection, as follows:

   - Database URL: The JDBC driver-specific URL used to connect to the RDBMS.

   - User: A valid user ID for the RDBMS.

   - Password: The password associated with the user ID specified.

7. Click the Advanced tab.

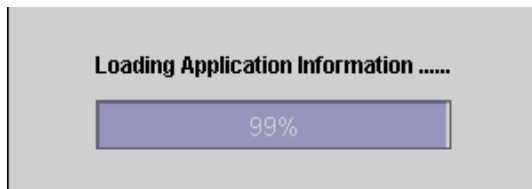8. Select the schema to use, as shown in the following figure.

Figure 2-6 Relational Database Logon - Advanced Tab
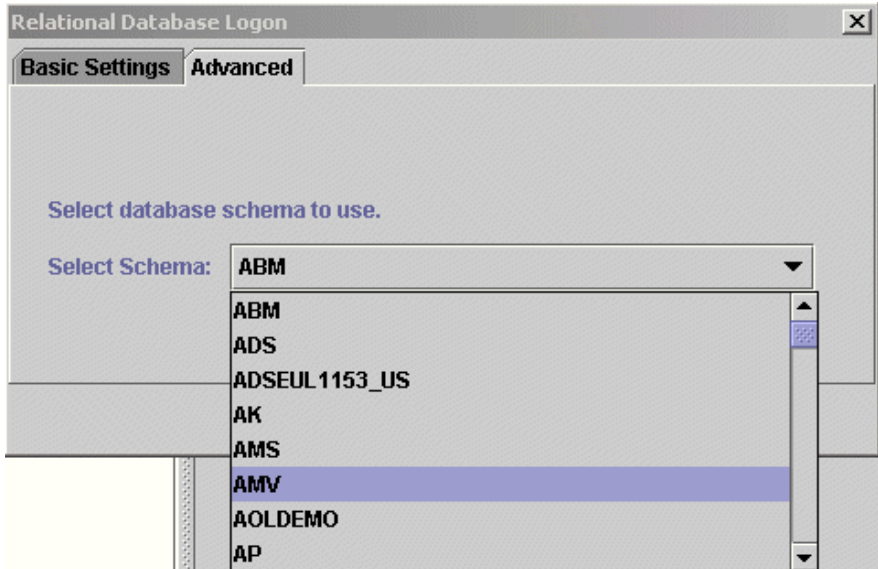
9. Click OK.

If the parameters are correct and the RDBMS is available, a progress bar displays, indicating that the RDBMS metadata is loading.

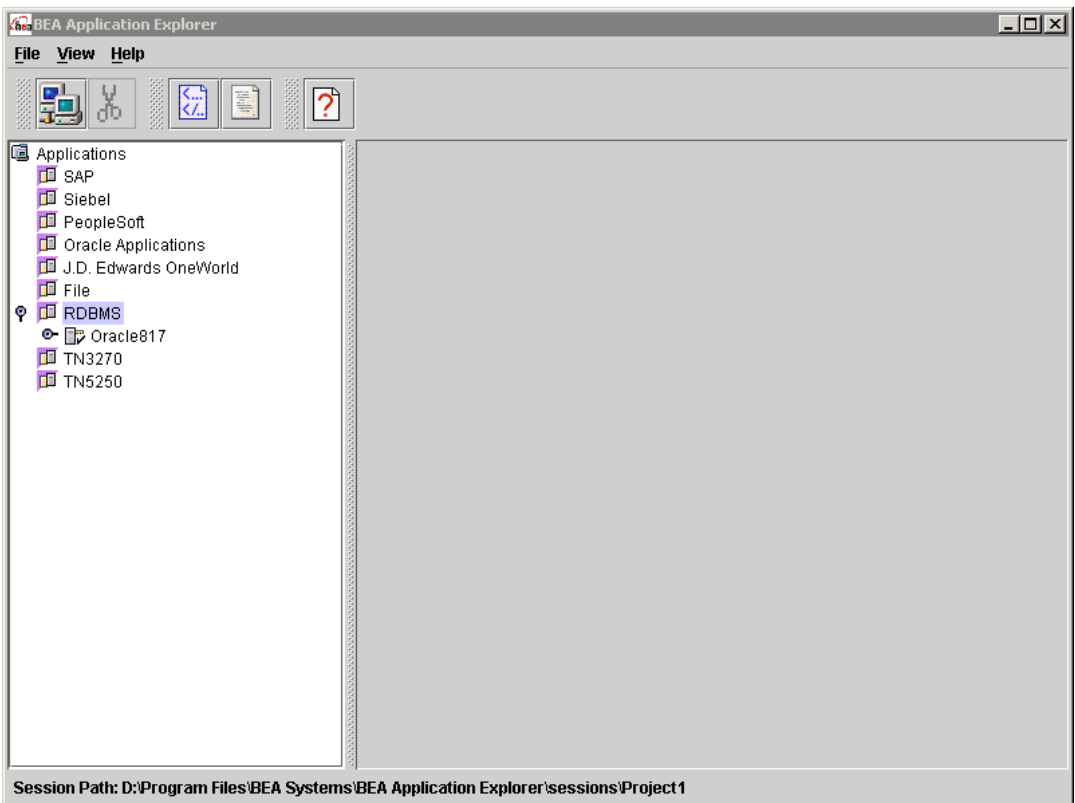**Figure 2-7 Loading Application Information Progress Indicator**

If there is a problem with the connection, a message appears.

Figure 2-8   Database Logon Error Message

After the loading of the application (RDBMS) information is complete, the connection appears as a node under the RDBMS node.
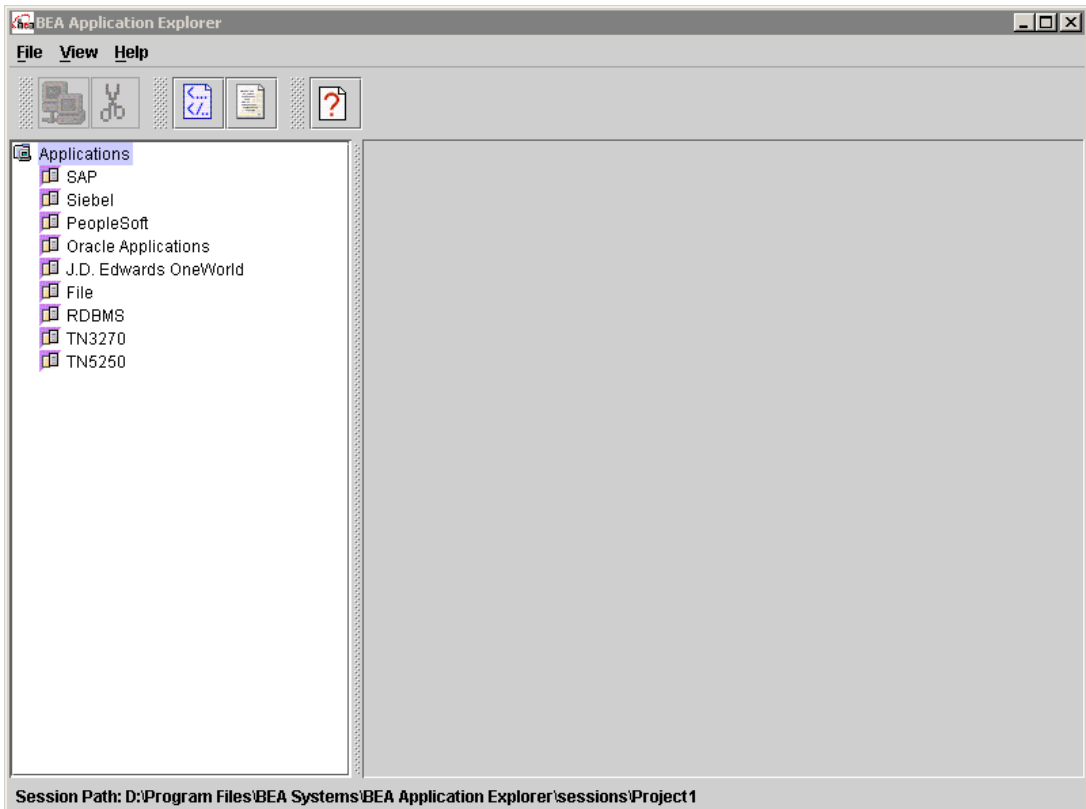
Figure 2-9   BEA Application Explorer - Connection Node

Connecting to an RDBMS Using an Existing Connection

When you first start the BEA Application Explorer, the main window opens. The left pane displays all the adapters supported by the version of the BEA Application Explorer you are using.

Figure 2-10 BEA Application Explorer Window

To connect to a specific RDBMS with an existing connection:

1. Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer.

   The BEA Application Explorer opens.

2. Right-click the RDBMS node and select Existing Connection from the shortcut menu.

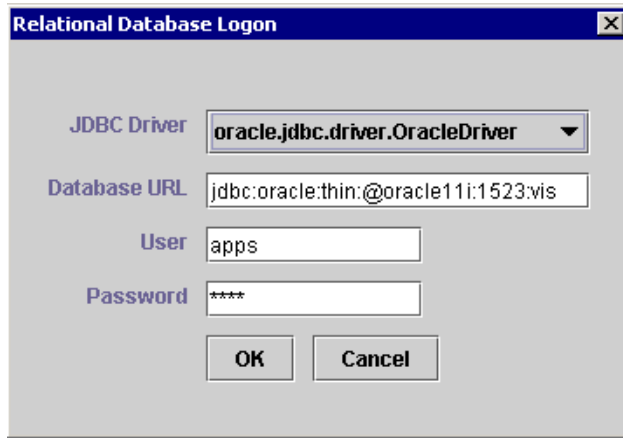   A list of existing connections from which to choose opens.

Figure 2-11   BEA Application Explorer - Existing Connections
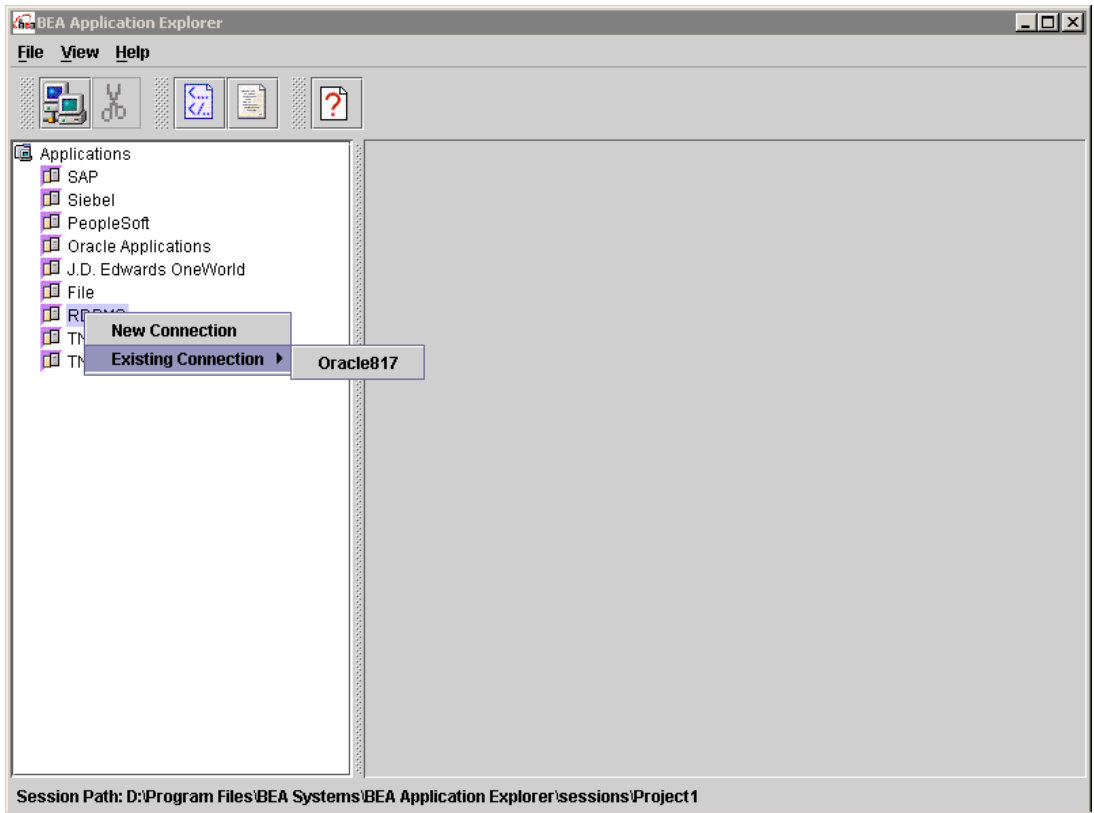
3. Select the desired connection.

A confirmation window opens, showing the connection information being used.

Figure 2-12   Relational Database Logon Confirmation Window

4. If this is the correct connection, click OK. If not, click Cancel.
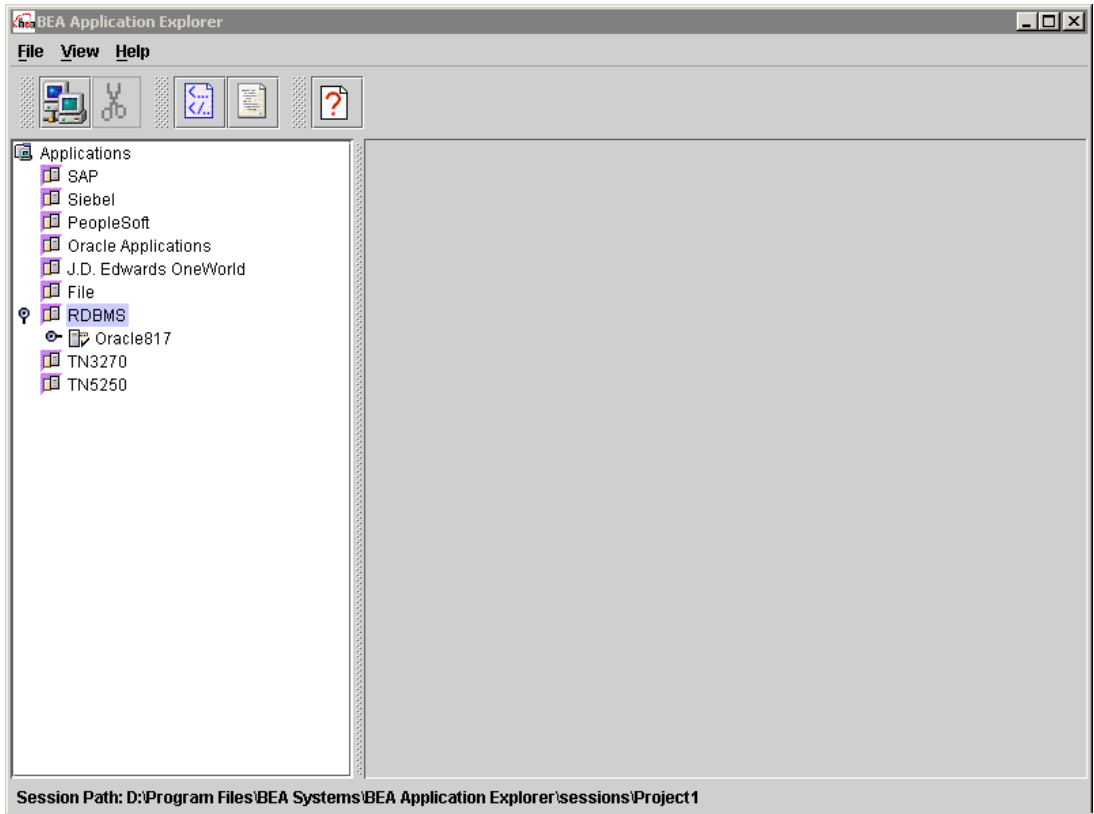
   If the parameters are correct and the RDBMS is available, a progress bar
   indicates that the RDBMS metadata is loading. If there is a problem with the
   connection, an error message appears.

After the application (RDBMS) information loads, the connection appears as a node under the RDBMS node.

Figure 2-13   BEA Application Explorer - Connection Node

Disconnecting from an RDBMS

Although you can maintain multiple open connections to different RDBMSs and applications, it is prudent to close connections when they are not being used. To close a connection to an RDBMS:

1. Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer.

2. When the BEA Application Explorer opens, right-click the connection node and select Disconnect.

Figure 2-14   BEA Application Explorer - Disconnect Node Option
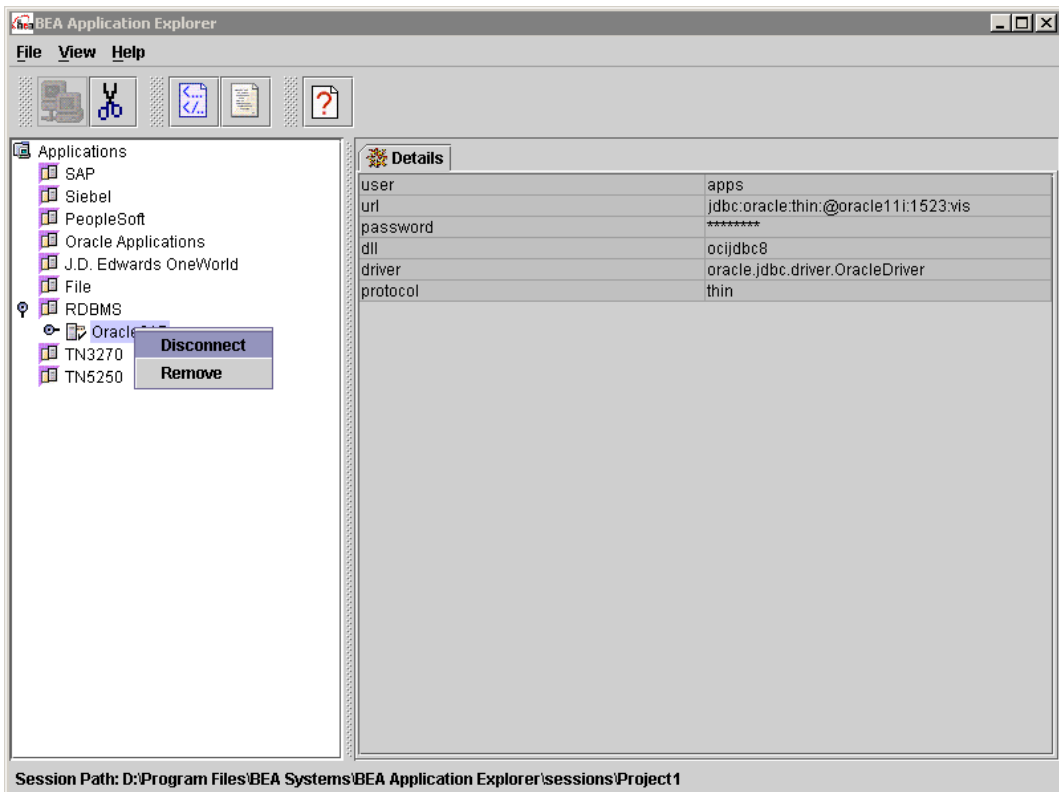
Disconnecting from the RDBMS drops the connection with the RDBMS, but the node remains. Note that the disconnected node has a different icon, 🔌, indicating that it is disconnected, as shown in the following figure.

Figure 2-15   BEA Application Explorer - Disconnected Node

3. To re-establish the connection, right-click the disconnected node and select
   Connect, as shown.

Figure 2-16   BEA Application Explorer - **Node Reconnection**

Removing a Connection

To remove a connection from the existing connections list:

1. Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer.

2. When The BEA Application Explorer opens, right-click the desired connection and select Remove.
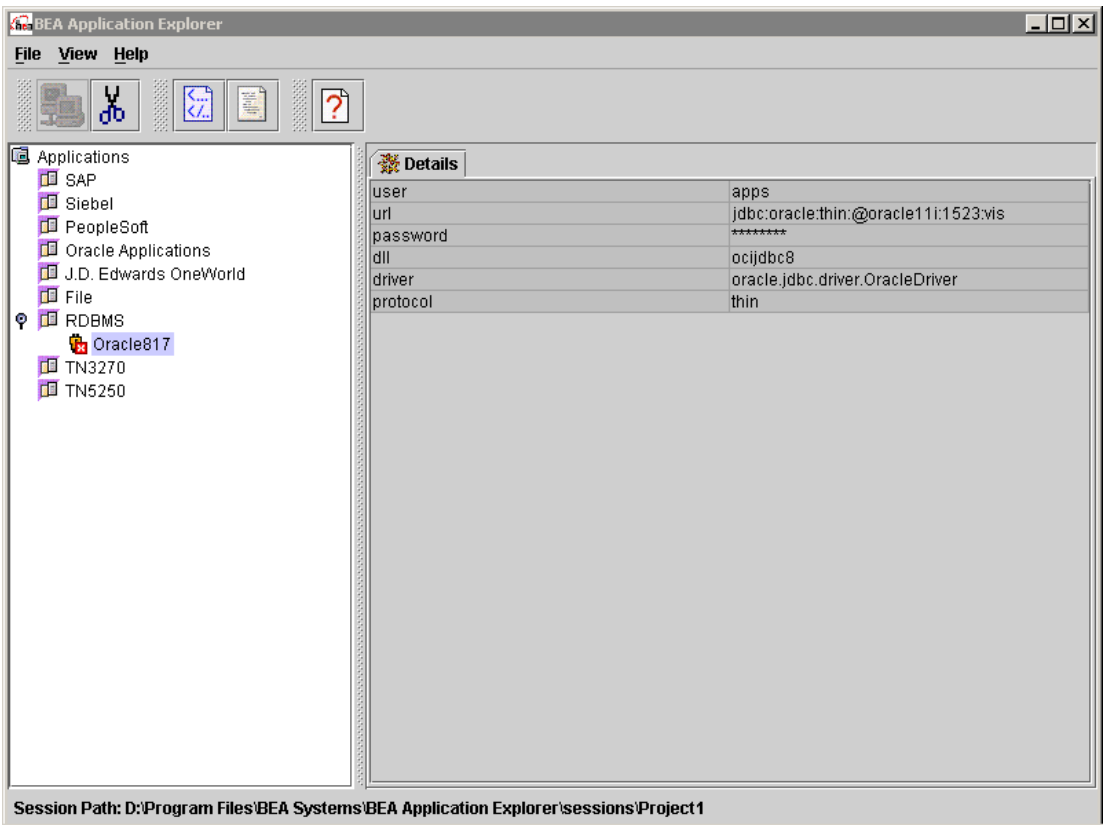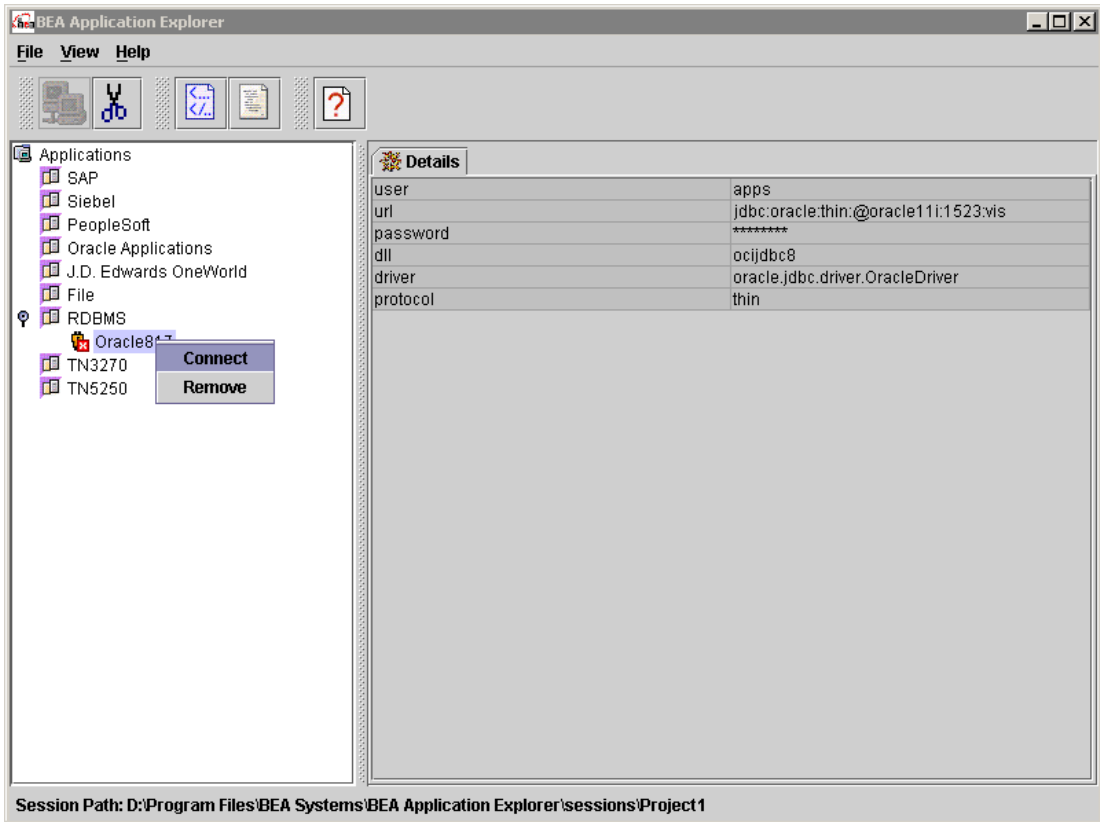
Figure 2-17   BEA Application Explorer - Connection Removal
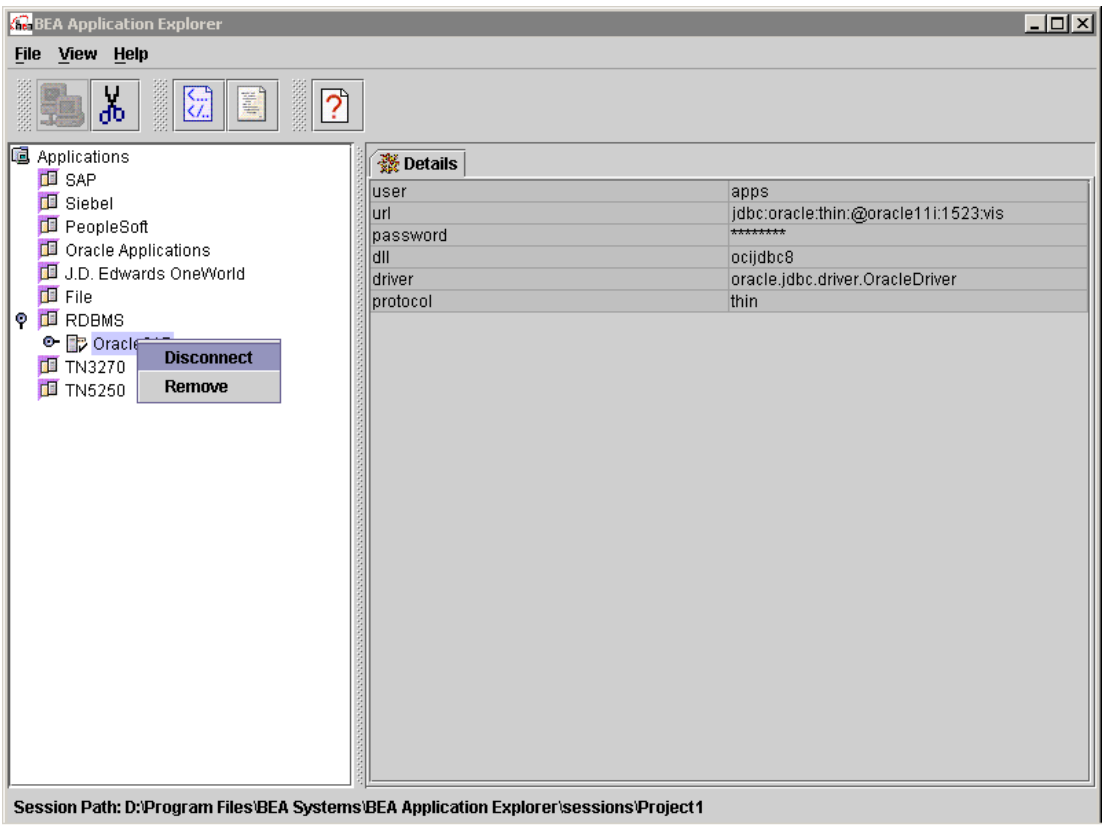
The connection is closed, if open. The entry is removed from the available connection list.

Viewing Table-Based Metadata

Viewing metadata is useful when creating schemas. For more information, see "Generating Event Schemas" on page 2-23 and "Generating Service Schemas" on page 2-30.

The BEA Application Explorer enables you to view the tables available in the RDBMS. To view the available tables:

1. Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer.

2. When the BEA Application Explorer opens, expand the table node under the desired connection.

Figure 2-18   BEA Application Explorer - Expanded Table Node
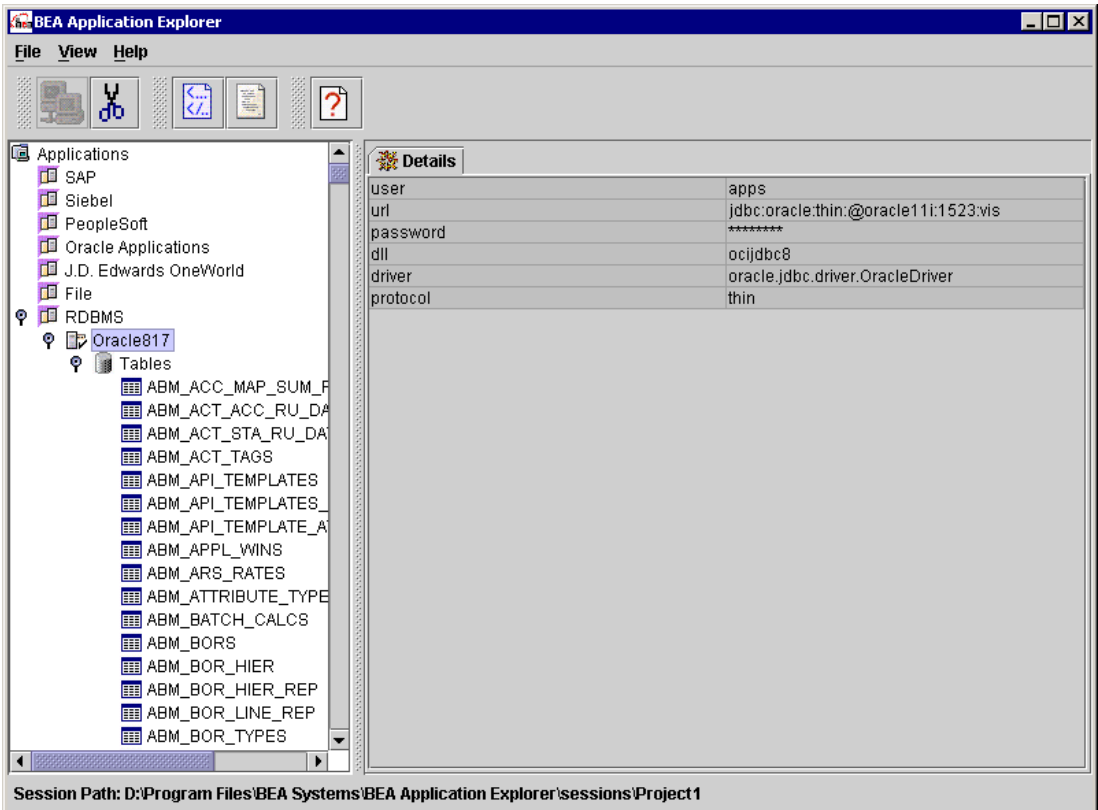
3. Scroll down and select the specific table to review.

Note: The list of tables includes all tables on the RDBMS. It is possible that the user ID used for the connection does not have access to the specified table. If this is the case, the creation of schemas fails.

When you select a specific table by clicking it, the table metadata appears in the right pane, as shown in the following figure. This information can be used to determine the table (or tables) and fields to use when creating the schema.

Figure 2-19   BEA Application Explorer - Table Metadata
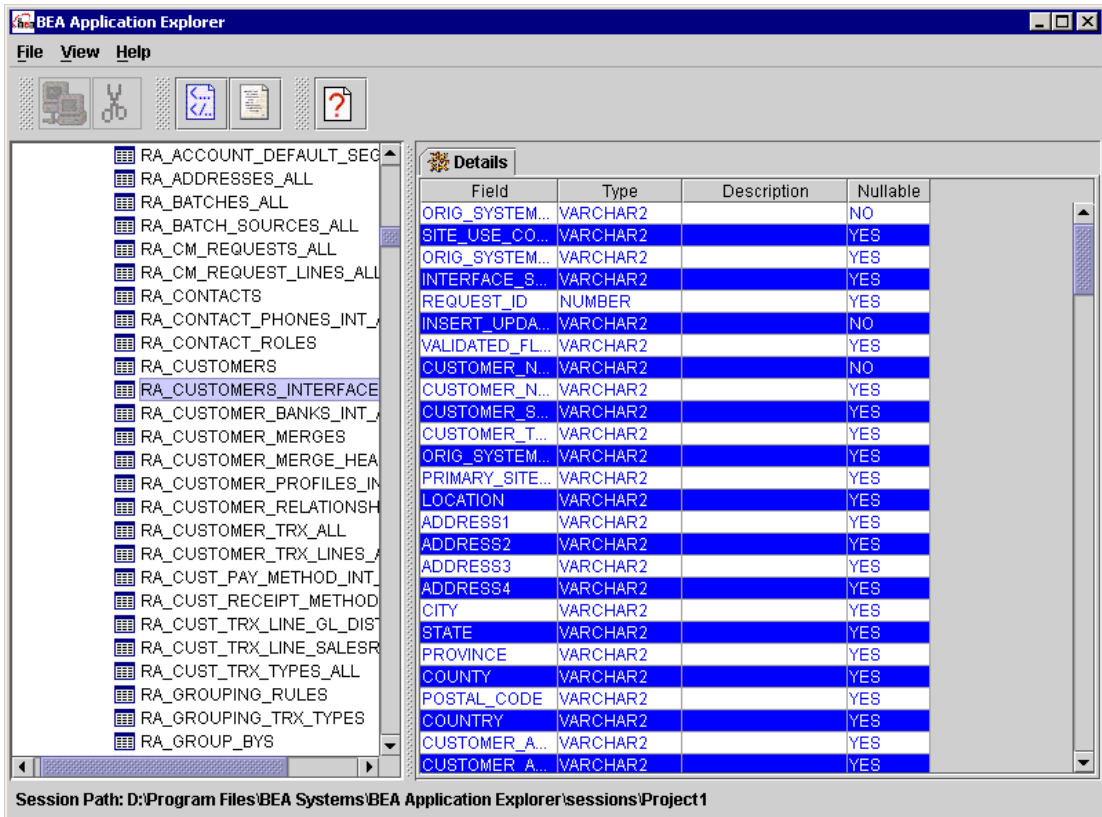
Viewing Stored Procedures

Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer. When you first start the BEA Application Explorer, in the left pane you see a list of all the adapters supported by the version of the BEA Application Explorer you are using.

Figure 2-20   BEA Application Explorer
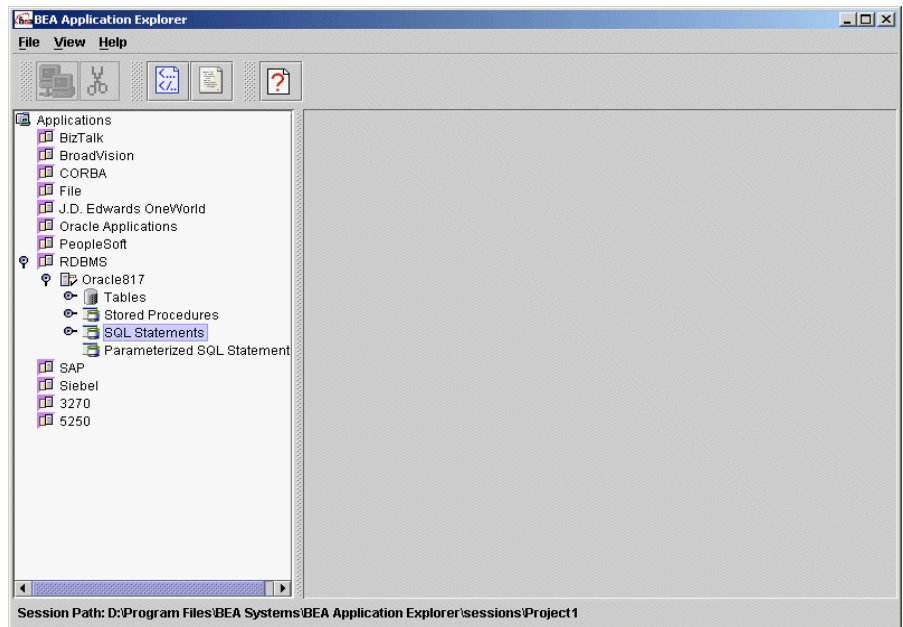
1. From the BEA Application Explorer main menu, expand the RDBMS branch.

2. Connect to a back-end Oracle system. For more information on connecting to back-end systems, see "Connecting to an RDBMS" on page 2-2 or "Connecting to an RDBMS Using an Existing Connection" on page 2-9.

3. Under the Connection name, expand the Stored Procedures branch by clicking the icon next to Stored Procedures. All stored procedures available under the schema you specified appear, as shown in the following figure.

Figure 2-21   BEA Application Explorer - Expanded Stored Procedures List
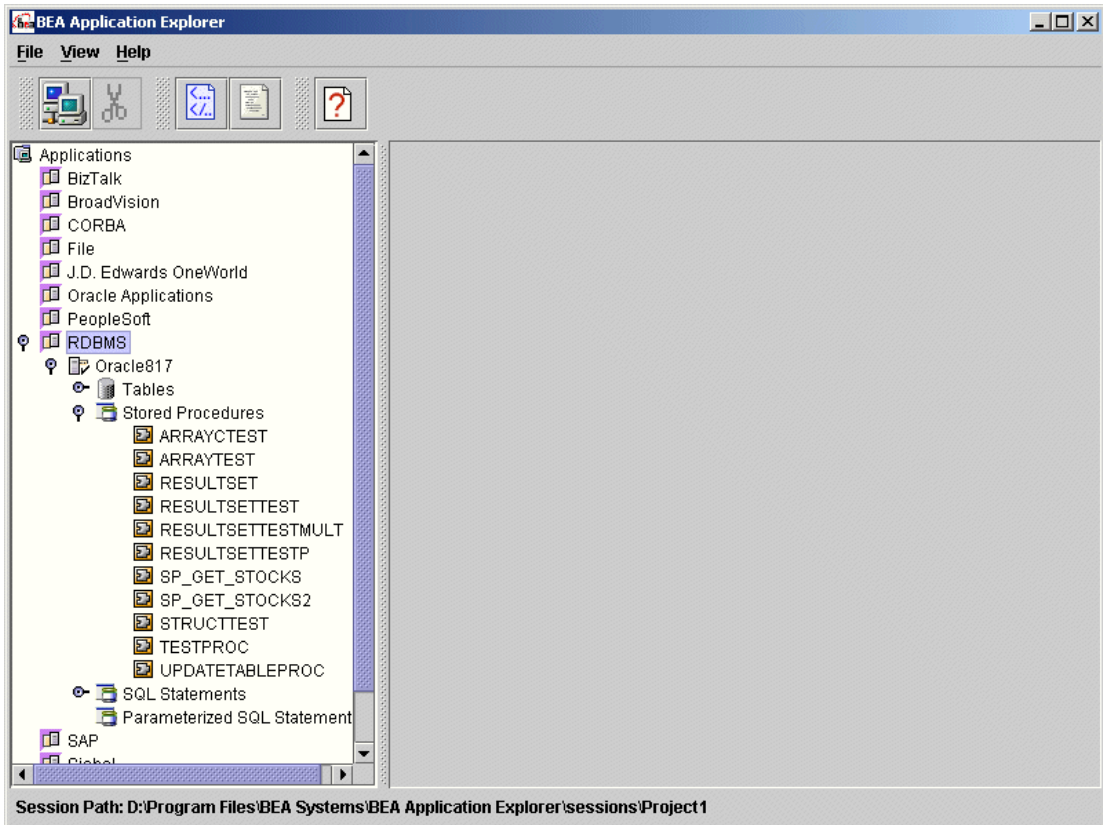
4. Highlight a stored procedure to view its parameters, as shown in the following figure.

Figure 2-22   BEA Application Explorer - Stored Procedure Parameters
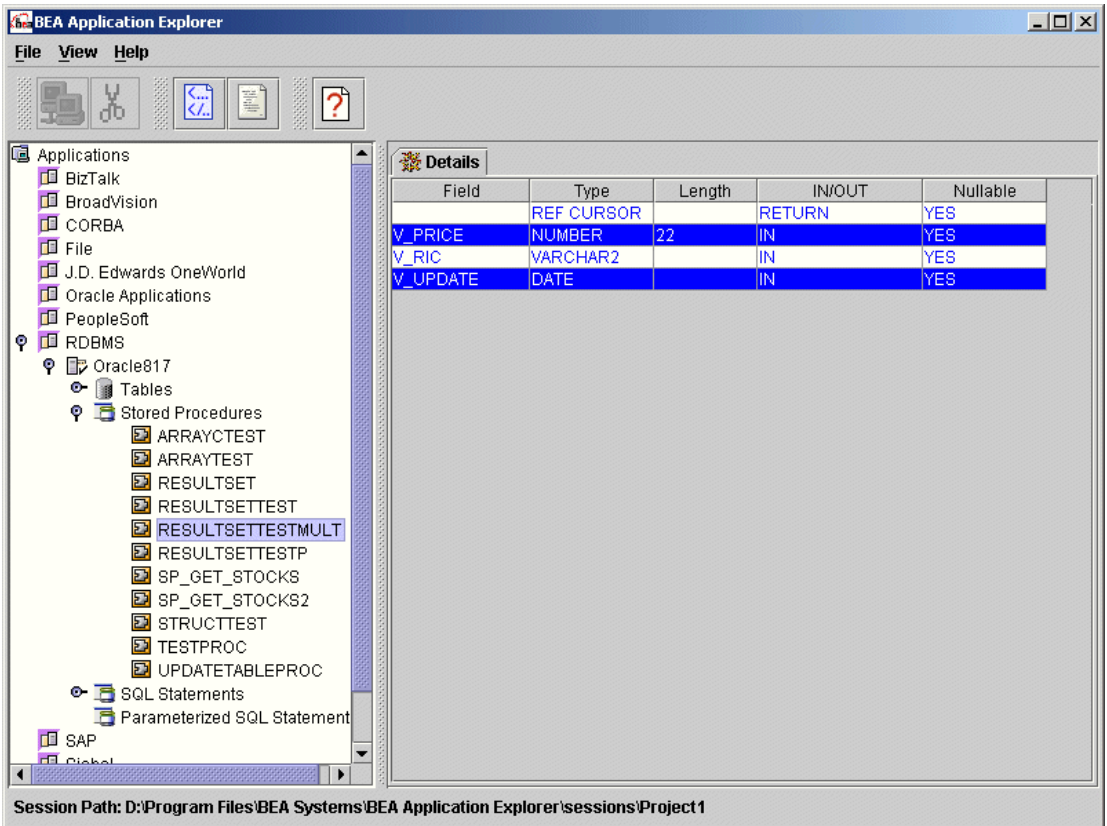
The RESULTSETTESTMULT Stored Procedure contains three input parameters, V_PRICE, V_RIC, and V_UPDATE, listed in the Field column.

Note: Output parameters are displayed without names in the field column.

For Oracle Stored Procedures returning result sets, the first parameter is designated as a type REF_CURSOR. This is actually the returning result set.

Generating Event Schemas

The generation of schemas is handled under the SQL statement node. To start the process of generating an event schema:

1. Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer.

2. When the BEA Application Explorer opens, right-click the SQL statement node and select Add SQL Statement, as shown in the following figure.

Figure 2-23   BEA Application Explorer - Add SQL Statement
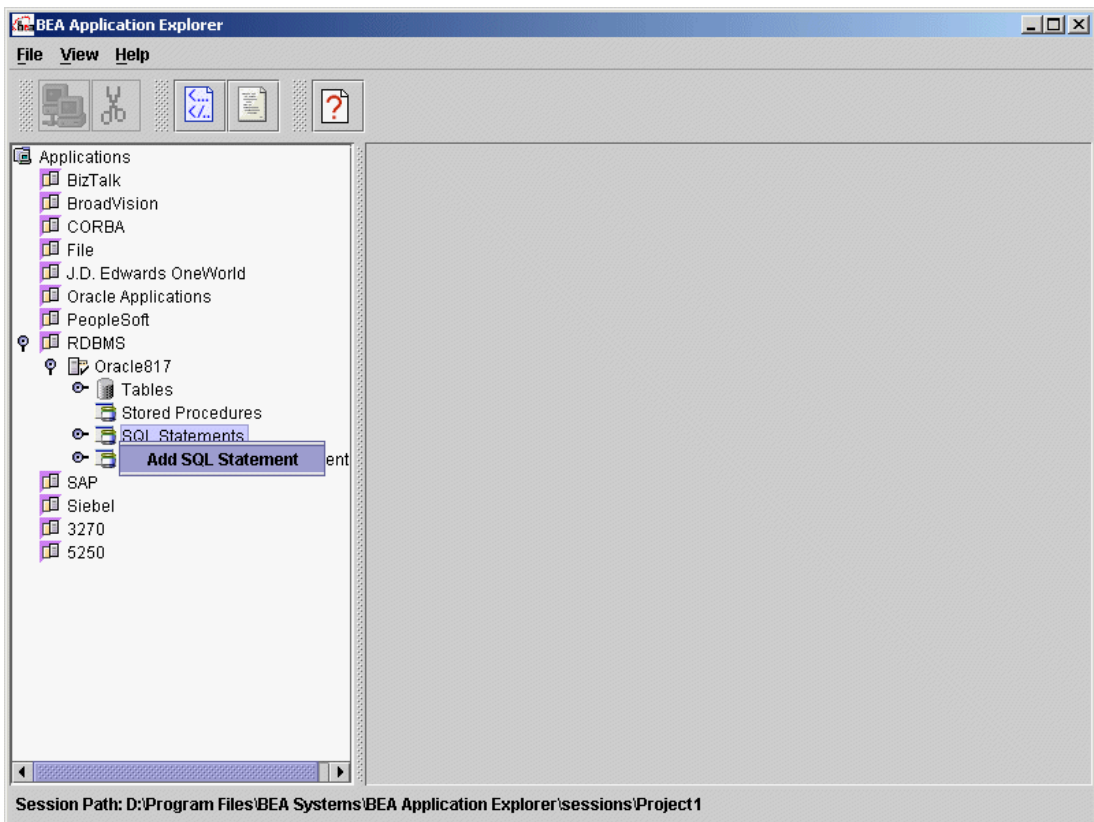
The SQL statement input box opens.

Figure 2-24 SQL Statement Name Input Box
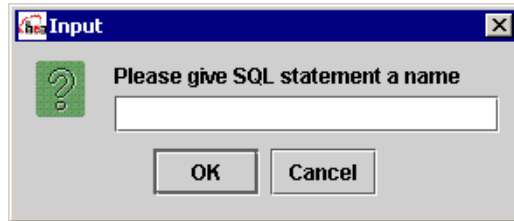
3. Enter a name for the schema group being generated.

   It is good practice to specify a name that describes the service. For example, a name of CustomerInt would represent an event on the Customer Interface table returning a Field format response document.

4. Click OK.

   After the SQL statement node is added, you are ready to build schemas.

5. To generate the schemas, right-click the SQL Statement and select the Create Event Schema option, as shown in the following figure.

Figure 2-25   BEA Application Explorer - Create Event Schema
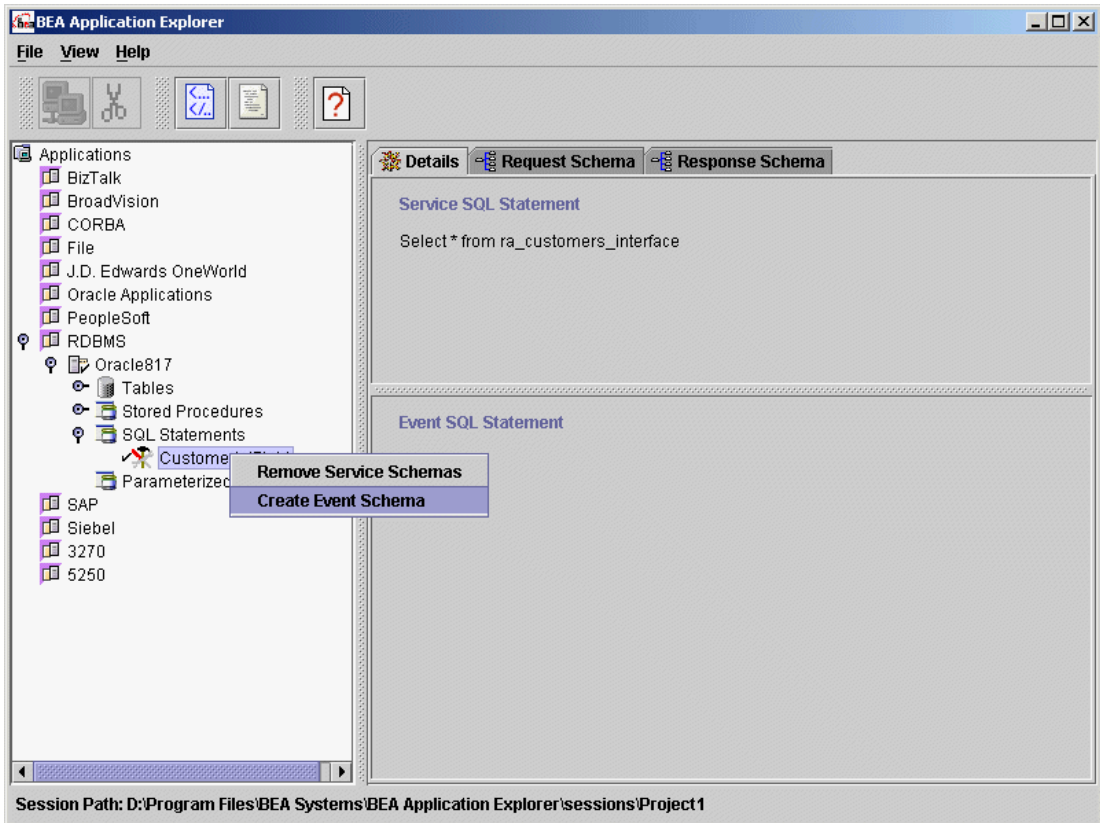
A Test SQL window appears.

Figure 2-26   Test SQL Statement
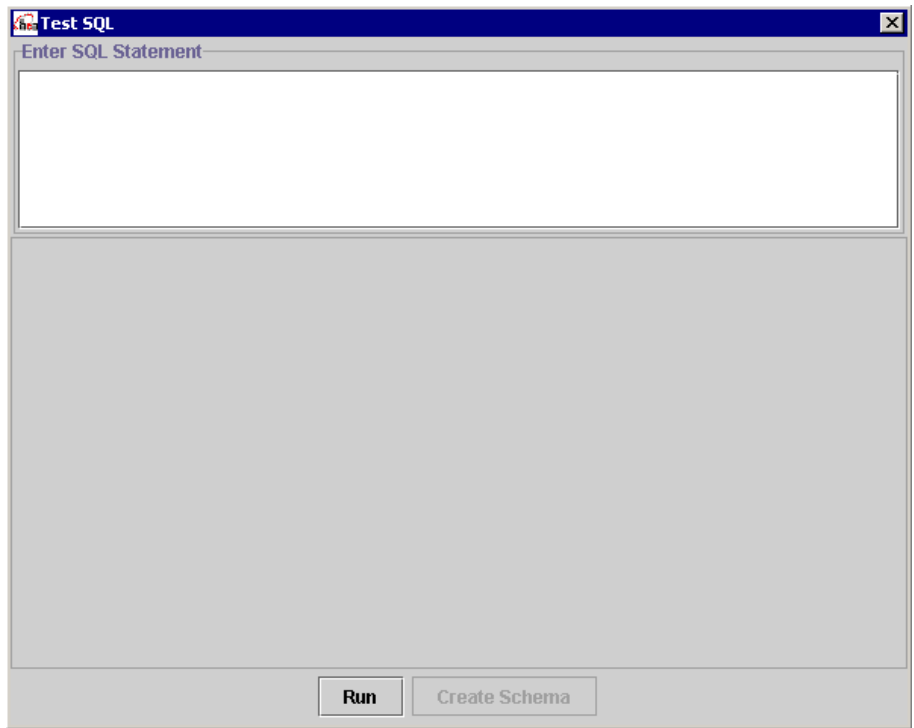
6. In the top pane, type the SQL statement to be used by the application view service.

   The RDBMS listener allows the creation of events that utilize complex SQL statements, including JOINs, WHEREs, and other verbs.

   When entering the SQL statement, ensure you use the SQL generating the input record for the event.

7. Click Run.

   If there is a problem executing the SQL statement, the error message from the RDBMS appears in an error window. When the SQL statement is correct, the following window appears.

Figure 2-27   Test SQL Results
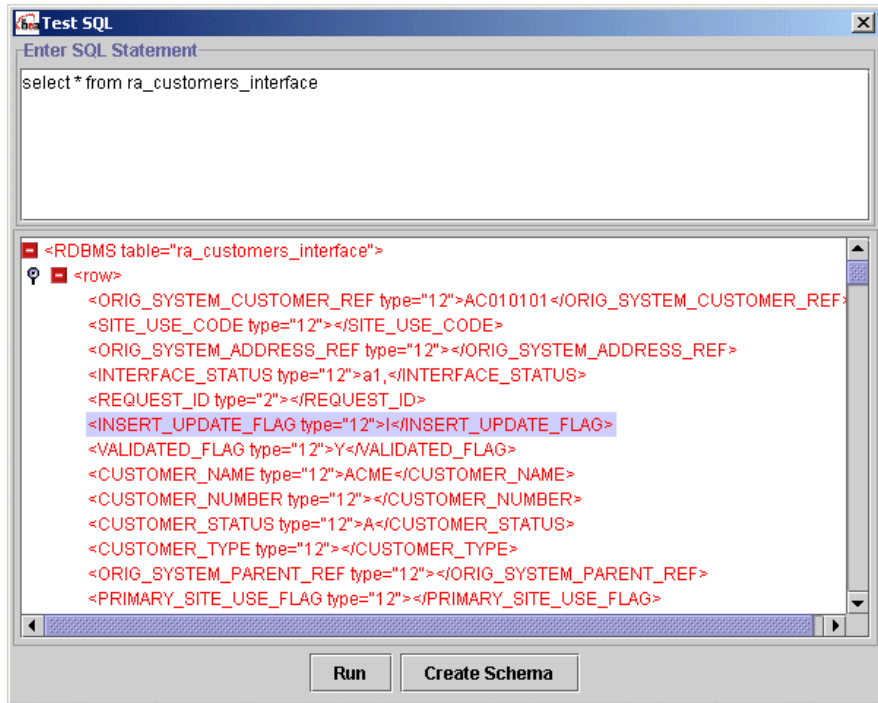
The SQL can be modified and the statement re-run, or the Create Schema button can be selected to produce the schema.

8. Click Create Schema.

The Select Table Format window is displayed.

Figure 2-28   Select Table Format Window
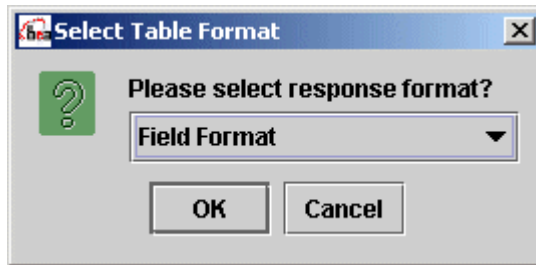
9. Select a formatting option from the field format drop-down list. The value you select must match the format used to generate the schema.

10. Click OK to create the schema.

The right pane of the BEA Application Explorer displays the SQL statement used, in addition to the Event, Request, and Response Schema tabs, as shown in the following figure.

Figure 2-29   Schema Display
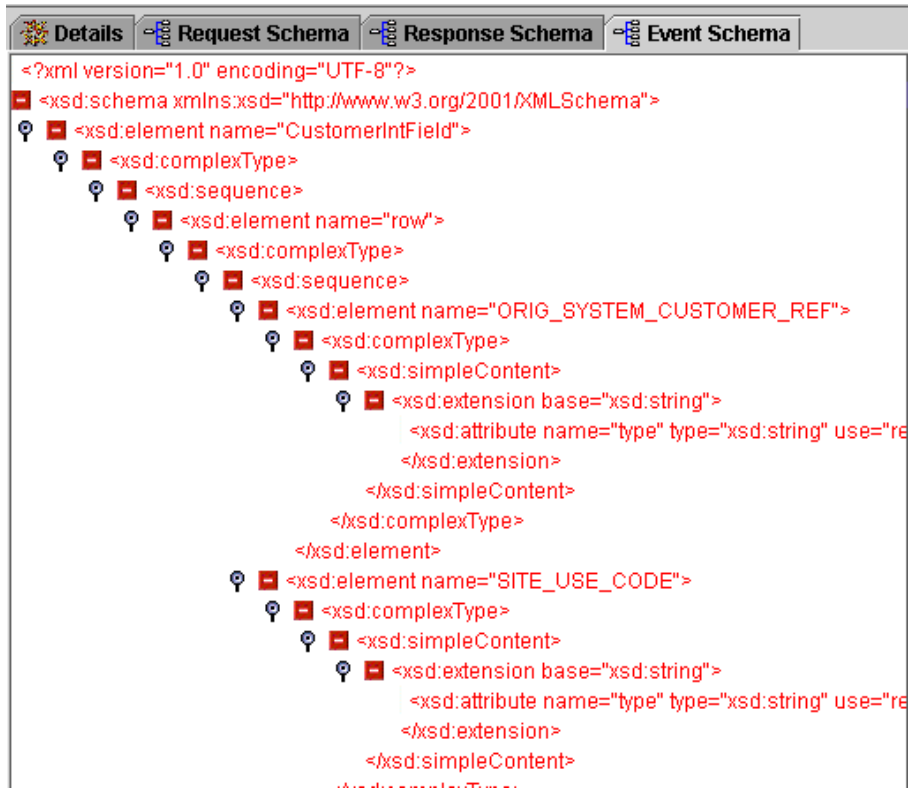
The schema is now generated and ready to use.

Generating Service Schemas

You can generate service schemas for:

- SQL statements.

- Parameterized SQL statements.

There are significant differences between request schemas generated by an SQL statement and a parameterized SQL statement. For more information about the use and format of the SQL statement and parameterized SQL statement options, see Chapter 3, "Defining an Application View."

Generating Service Schemas Under the SQL Statement Node

Generating service schemas under the SQL statement node produces the required request and response schemas for building a service adapter application view. To generate a service schema:

1. Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer.

The BEA Application Explorer opens.

Figure 2-30   BEA Application Explorer - Add SQL Statement
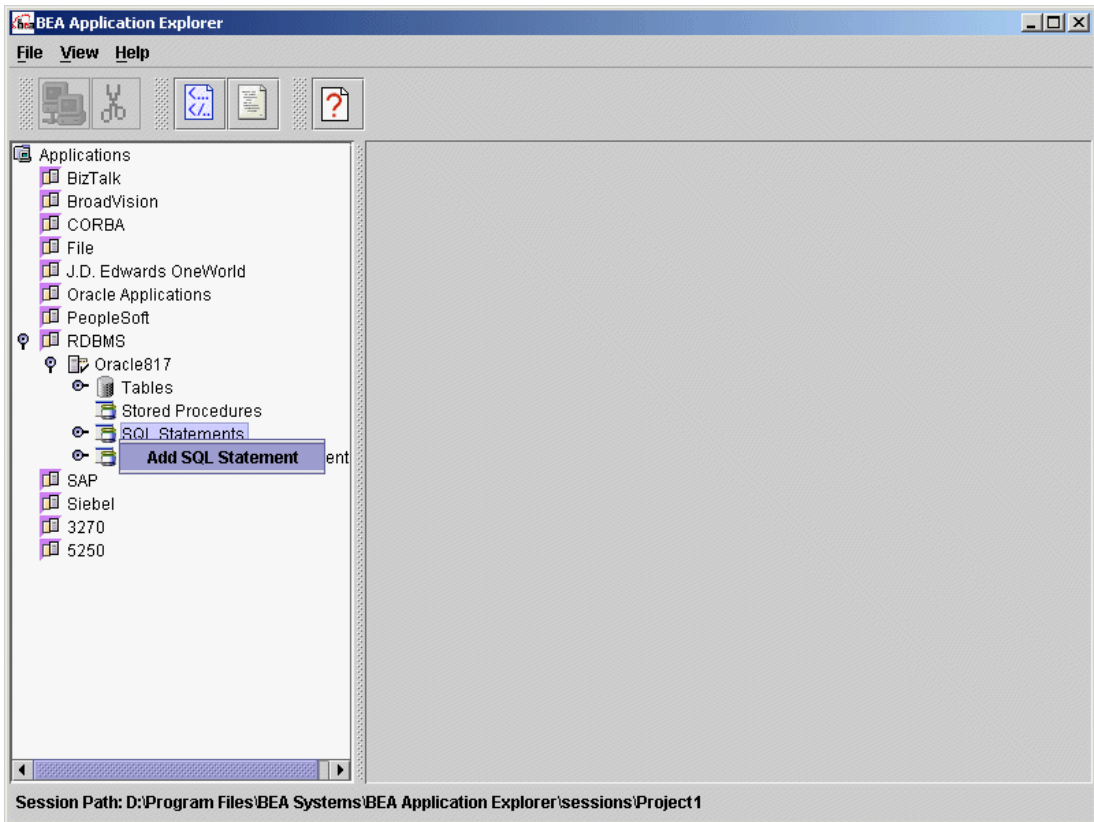
2. Right-click the SQL Statements node and select Add SQL Statement.

The SQL statement name input box appears.

Figure 2-31   SQL Statement Name Input Box
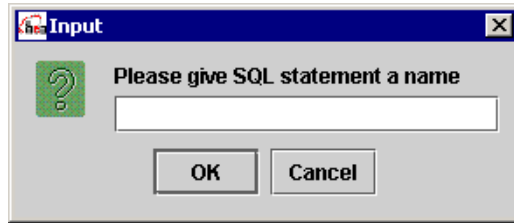
3. Provide a name for the schema group being generated.

   It is good practice to specify a name that describes the service. For example, a name of CustomerIntField would represent a request against the Customer Interface table returning a Field format response document.

4. Click OK.

   After the SQL statement node is built, you are ready to build schemas.

Figure 2-32 BEA Application Explorer - Create Service Schemas
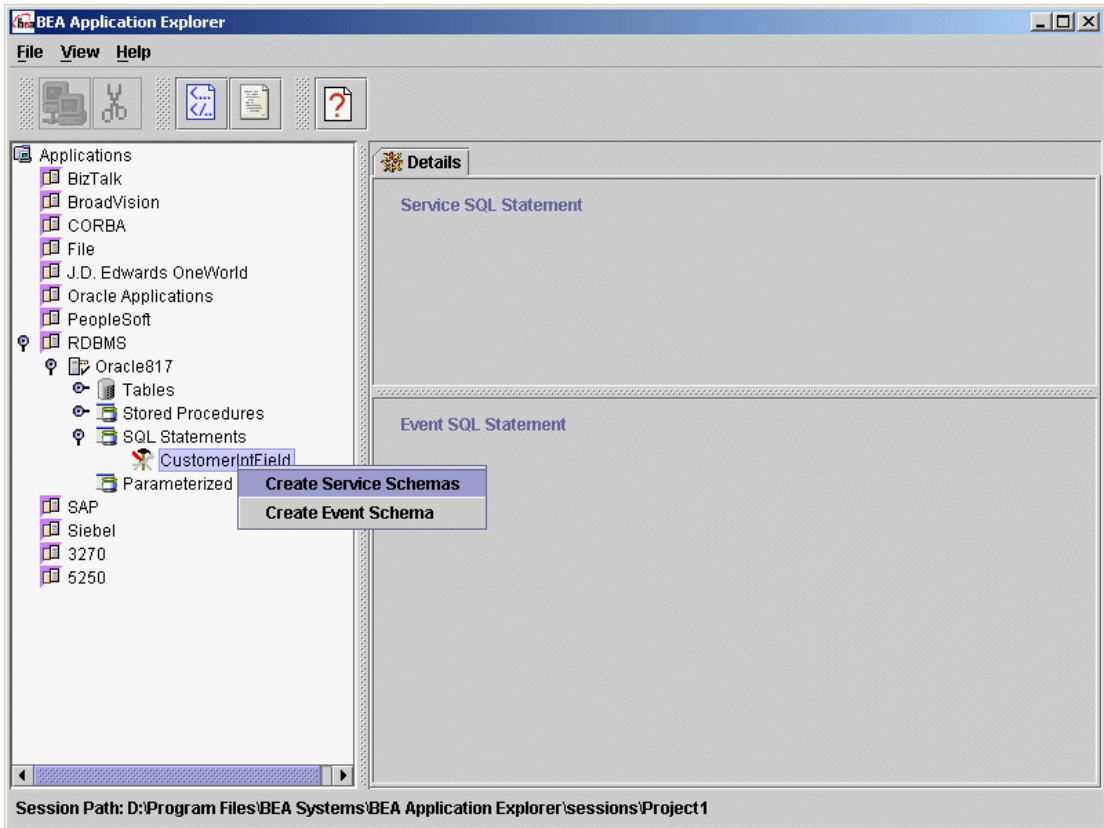
5. To generate the schemas, right-click the SQL statement and select the Create Service Schemas option.

A Test SQL window opens.

Figure 2-33   Test SQL Statement
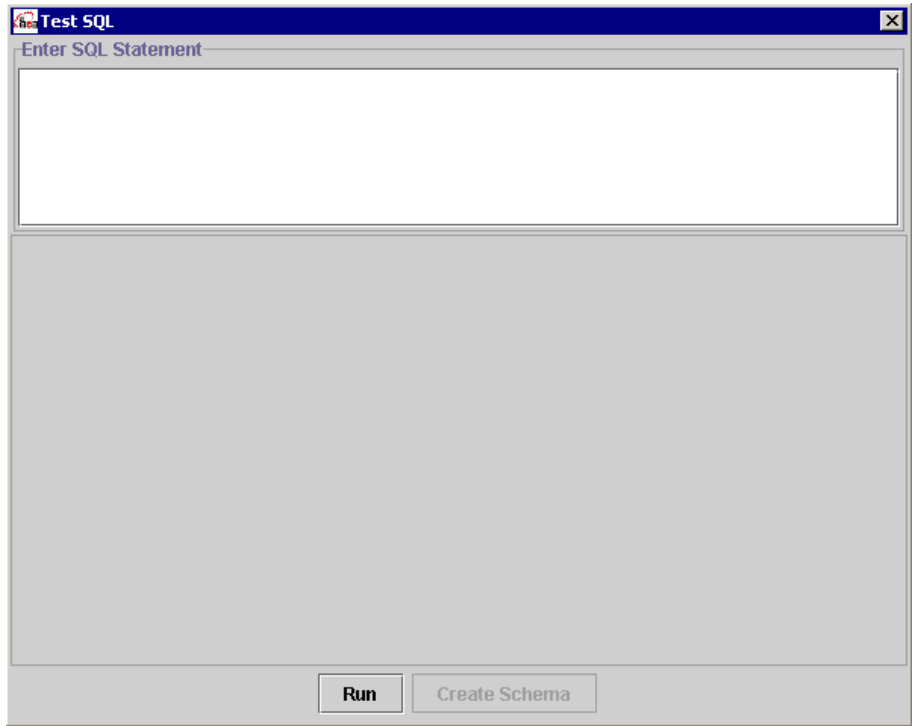
6. In the top pane, type the SQL statement to be used by the application view service.

7. Click Run.

   If there is a problem executing the SQL statement, the error message from the RDBMS appears in an error window.

When the SQL statement is correct, the following window appears.

Figure 2-34   Test SQL Results
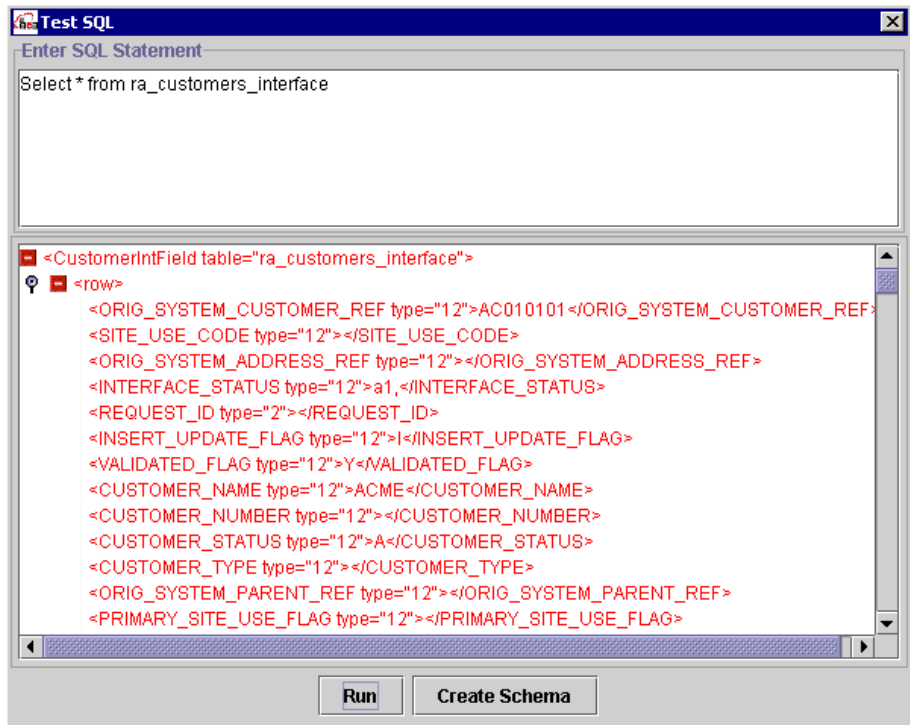
The SQL can be modified and the statement re-run, or the Create Schema button can be selected to produce the schemas.

Before the schemas are created, the systems prompts you for the format of the response.

Figure 2-35   Select Response Format
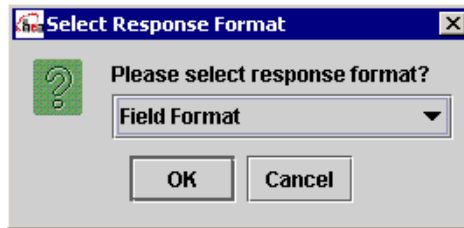
8. Select a formatting option from the response format drop-down list. The value selected must match the format value used to generate the application view service, or the schema of the response document will not validate.

9. After a format is selected, click OK to produce the schemas.

The right pane displays the SQL statement used, in addition to the Request and Response Schema tabs.

Figure 2-36   Schema Display
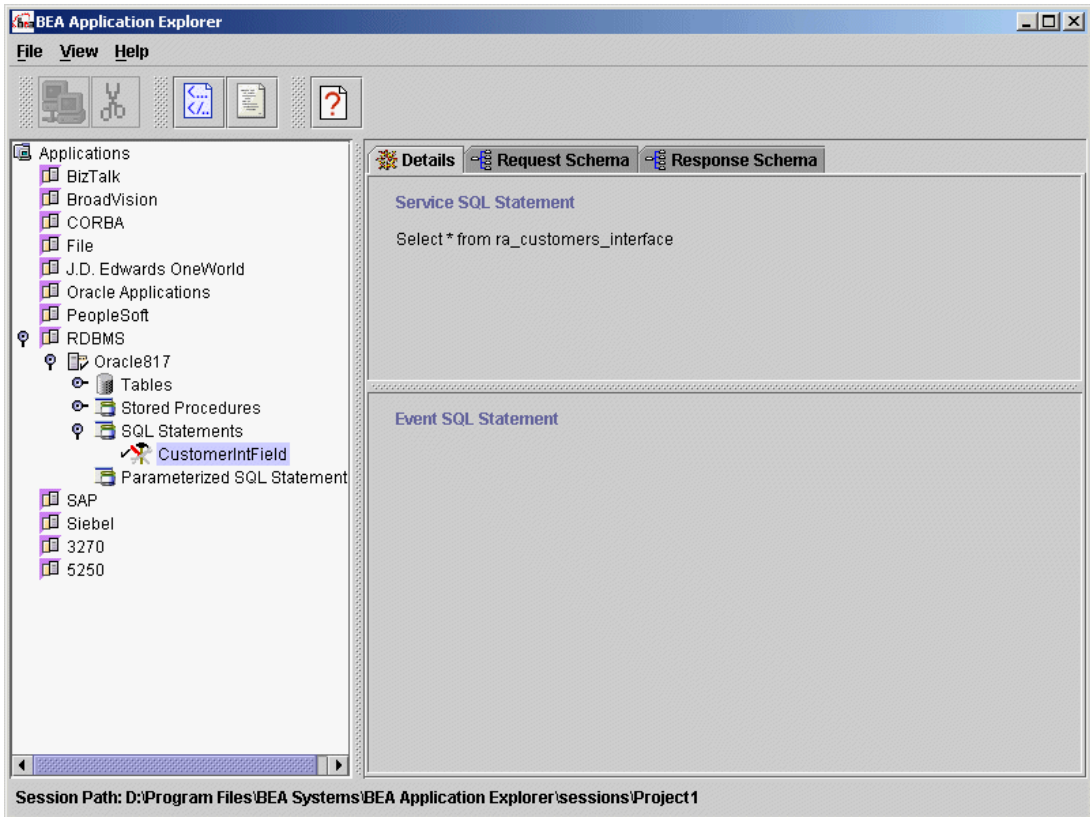
10. To view the schema, click the desired tab at the top of the right pane.

The following figure shows the request schema.

Figure 2-37   Request Schema
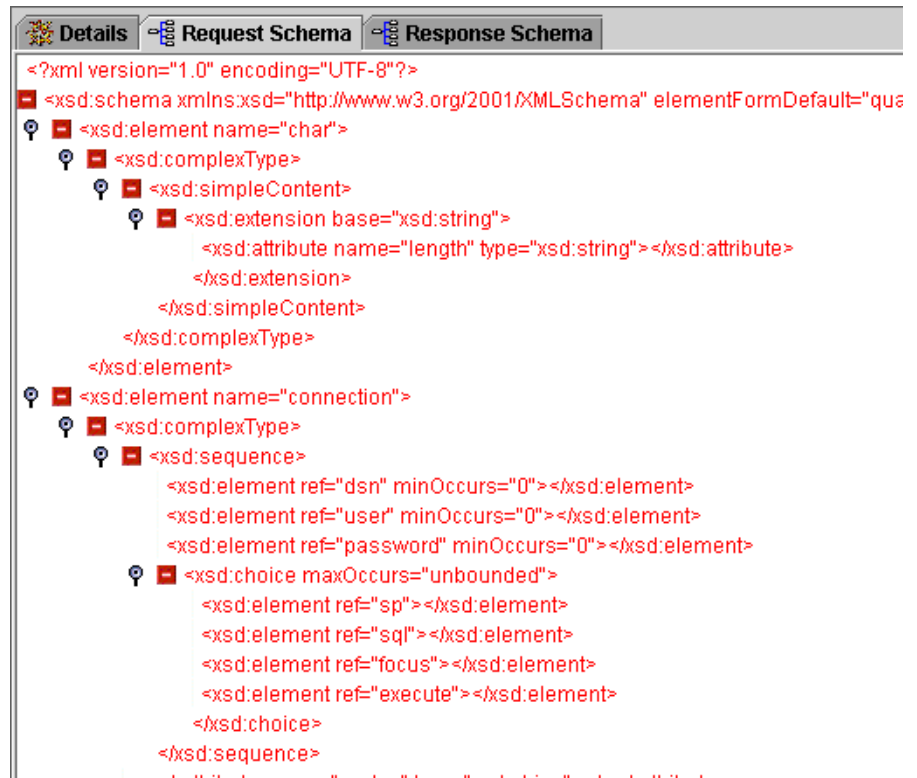
The following figure shows the response schema.

Figure 2-38   Response Schema
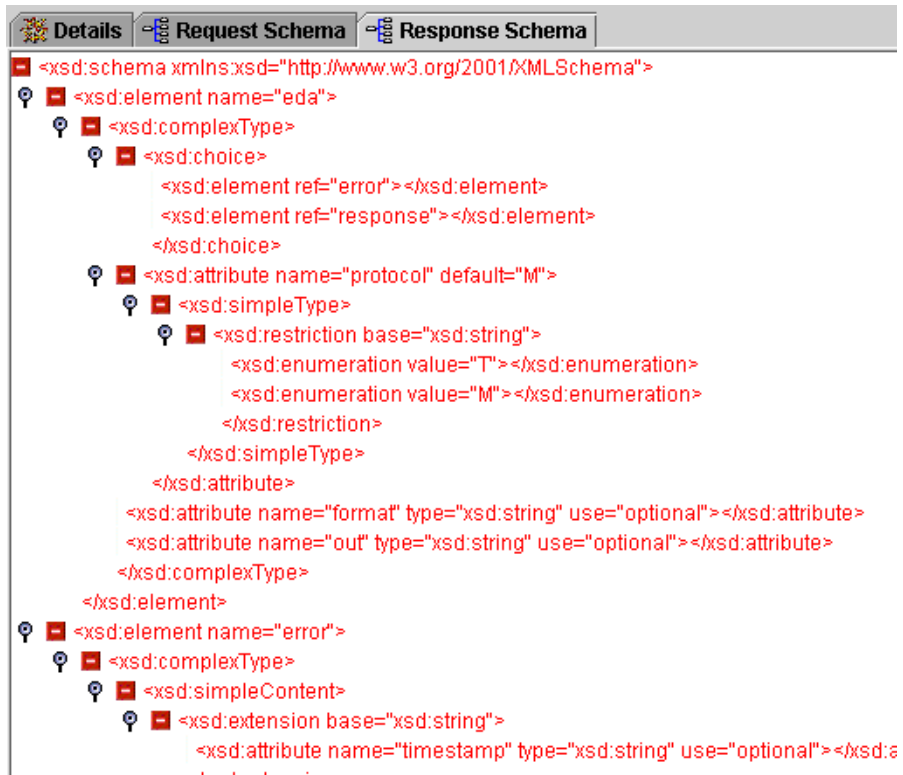
The schemas are now generated and ready to use.

Generating Service Schemas Under the Parameterized SQL Statement Node

Generating schemas under the Parameterized SQL statement node produces the required request and response schemas for building a service adapter application view.

To generate service schemas:

1. Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer.

2. When the BEA Application Explorer opens, open and expand an existing connection.

3. Right-click the Parameterized SQL Statements node and choose Add Parameterized SQL Statement.

Figure 2-39   BEA Application Explorer - Add Parameterized SQL Statement
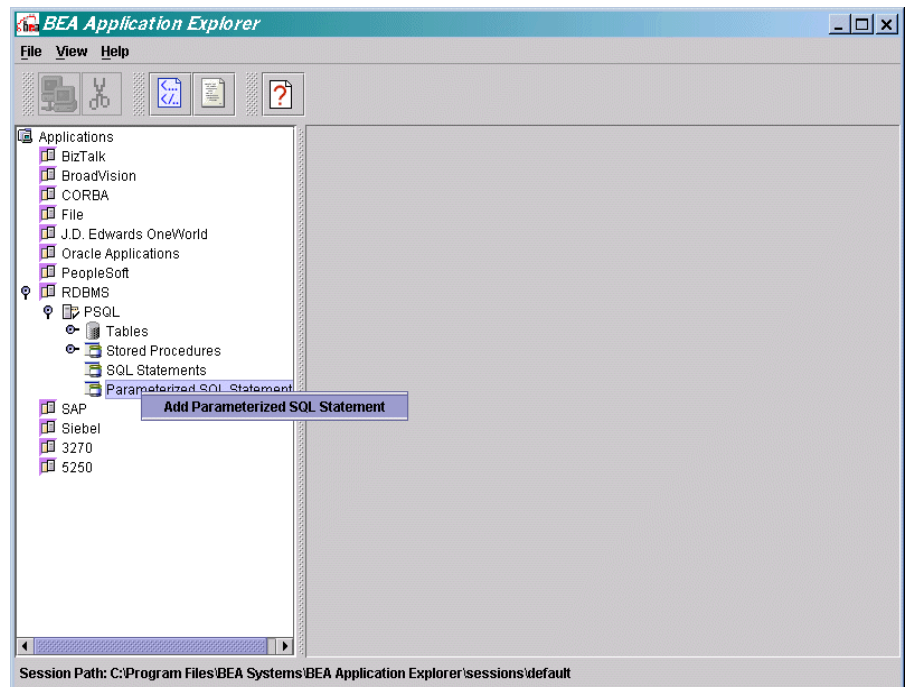
For more information on how to create and configure a connection, see "Connecting to an RDBMS" on page 2-2.

An input box prompts you for the name of the parameterized SQL statement.

Figure 2-40   Parameterized SQL Statement Name Input Box
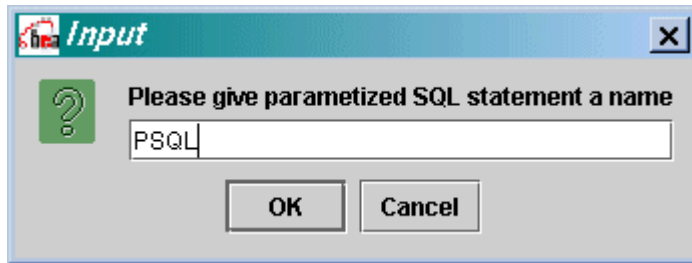
a. Enter a name for the schema group that you are generating.

   Note: It is good practice to specify a name that describes the service. For example, the name PSQL might represent a parameterized request against the PSQL table returning a field format response document.

b. Click OK.

You have created the parameterized SQL statement node, and you can generate schemas.

4. Right-click the new SQL Parameterized statement node and choose Create Service Schemas.

Figure 2-41   BEA Application Explorer - Creating Service Schemas
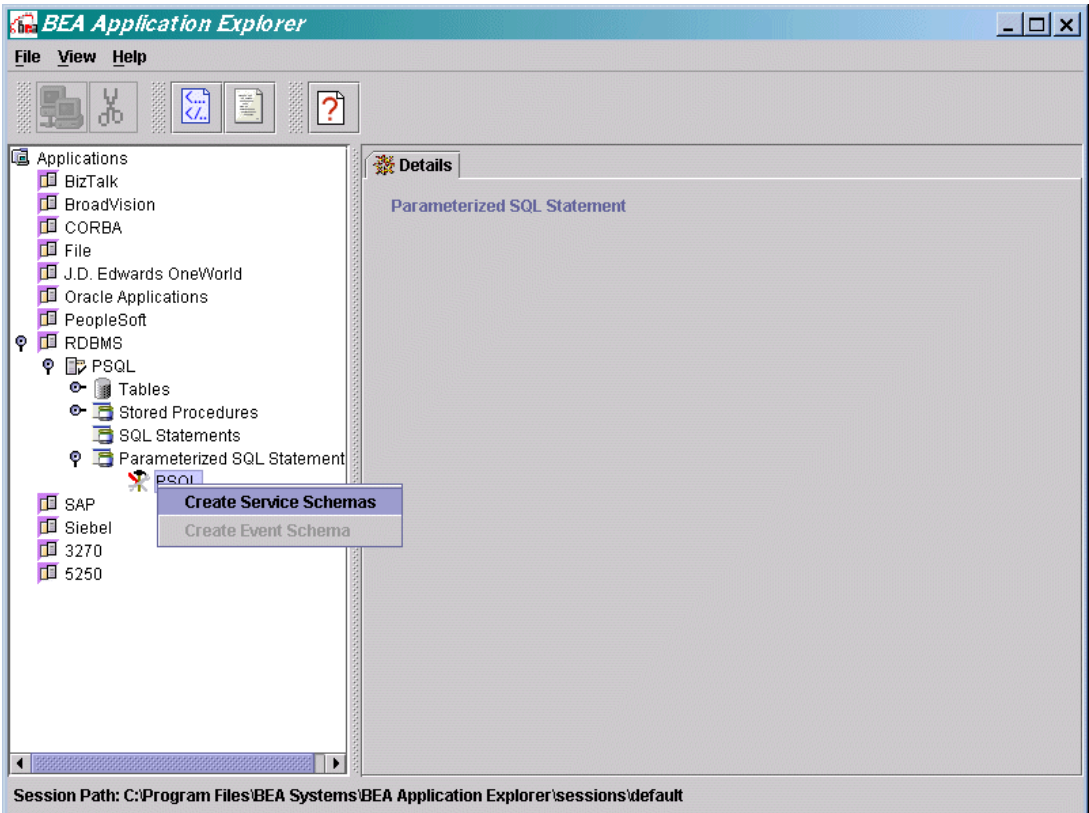
The Choose XML Request Schema dialog box appears.

Figure 2-42   RDBMS: Choose XML Request Schema Window
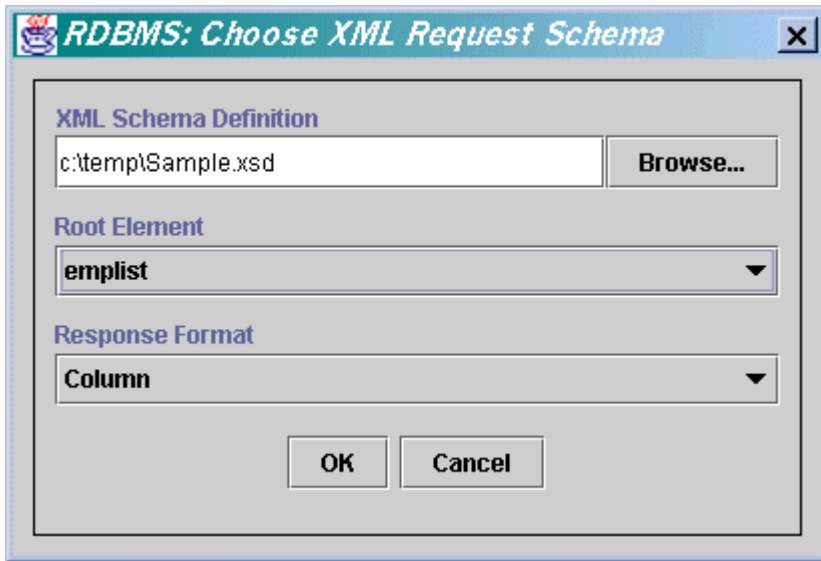
For this example, the following input XML is used (the corresponding schema is not displayed):

Listing 2-1   Input XML Sample

```
<emplist>
<emp>
    <name age='32'>Bess Armstrong</name>
    <address>211 Chadwick Rd</address>
    <citystatezip>New York, NY 10009</citystatezip>
</emp>
<emp>
   <name age='40'>Merle Pensk</name>
   <address>41 Church Rd</address>
   <citystatezip>Marlboro, NJ 07601</citystatezip>
</emp>
</emplist>
```

a. Click Browse and select the XML Input Schema Definition for the input XML document.

The input XML document is the XML that contains the values that will be used to substitute for the parameters.

b. Click the arrow for the Root Element drop-down to select the node in the XML document that corresponds to the root node of the input XML document.

c. Click the arrow for the Response Format drop-down to select the response format.

The choices are Column, Row, and Field, defined as defined in the following tables:

Table 2-1 Format Definitions

Table 2-2

format*
(*Required)
	Choose one of the following:
	■ **row.** The data that is produced is returned on a single line (per record) enclosed in `<row>` tags.
	■ **column.** The data produced is returned field by field, and each field is enclosed in `<column>` tags. The column tag has an attribute whose value is the name of the field; for example,
	`<row>`
	`<column name="ID">1000</column>`
	`<column name="First_Name">Scott</column>`
	`</row>`
	■ **field.** The data produced is returned field by field, and each field is enclosed in a tag that bears the field name; for example,
	`<row>`
	`<ID>1000></column>`
	`<FIRST_NAME>Scott</column>`
	`</row>`

Note: Due to the flexibility of parameterized SQL statement processing, the input document can be in virtually any valid XML format, from a standards-based format such as an OAGIS Business Object Document or xCBL document, to a non-standards-based XML document, to the output of a previous task. To support this flexibility, you must provide the predefined `.xsd` file and the root name for the request document.

d. Click OK.

The Parameterized SQL Mapper window opens.

Figure 2-43   Parameterized SQL Mapper Window
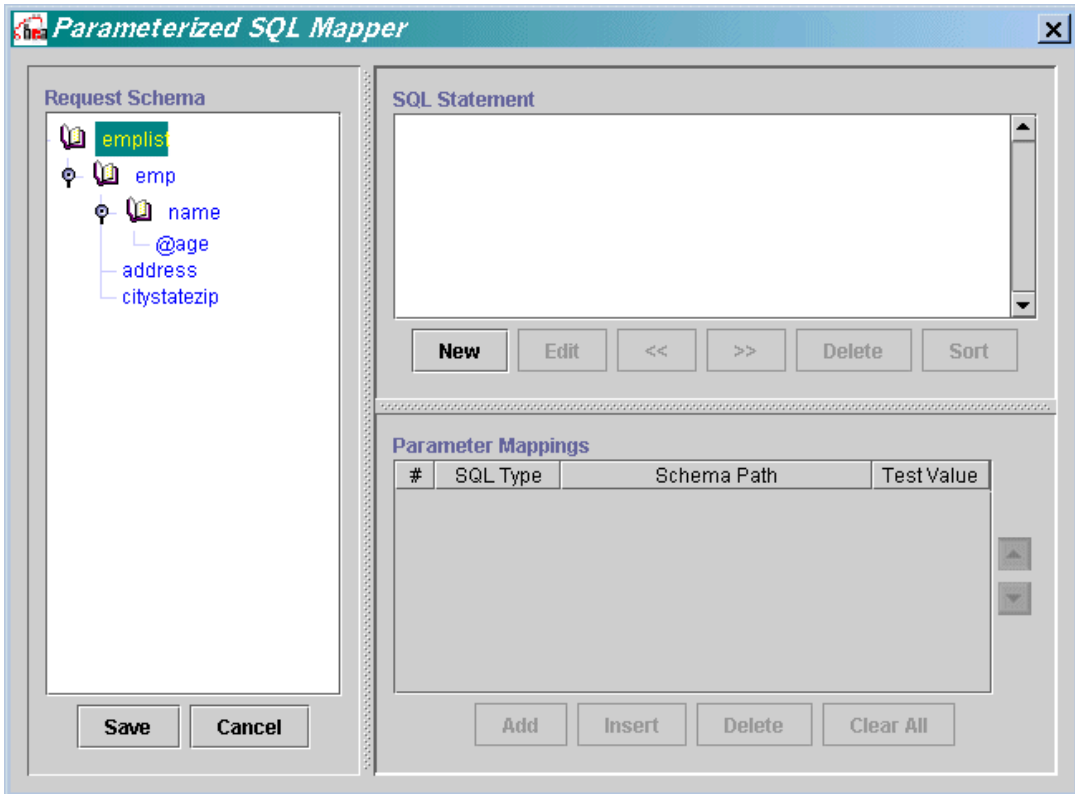
This mapping tool can be used to map fields from an input XML document to the parameterized SQL you will enter.

a. Highlight the root node for the input document in the Request Schema box and click New.

The SQL Statement test box becomes active.

Figure 2-44   Parameterized SQL Mapper with Active Test Box
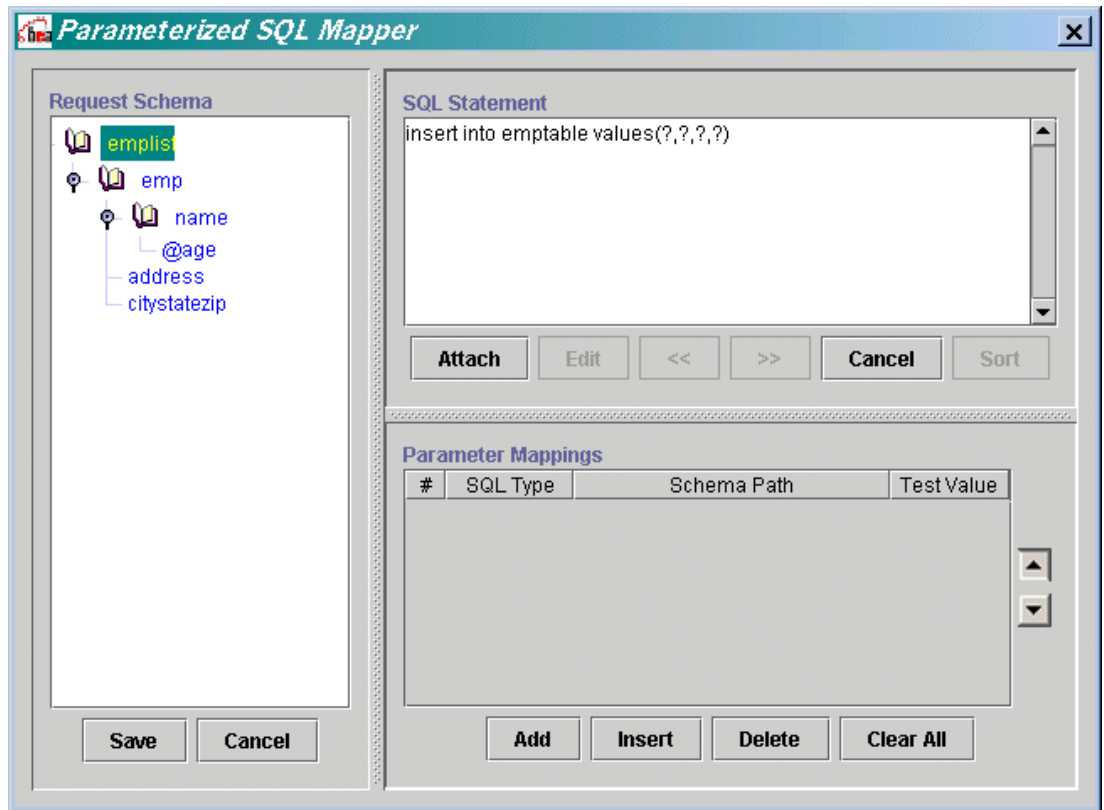
b. In the active test box, enter a parameterized SQL statement.

Substitute question marks for parameters.

In this example, four fields (corresponding with four parameters) are being inserted into the table called emptable (that is, table field names: Name, Address, CityStateZip, and Age).

Note: For combining stored procedures with parameterized SQL, issue a CALL statement in the SQL statement box.

Syntax: CALL Stored_Procedure_Name(?,?,? and so forth). For example, the following is a stored procedure named MyStoredProcedure that takes three parameters:

```
Call MyStoredProcedure(?,?,?)
```

c.   Click Add.

The Parameter Mappings dialog box becomes active.

Figure 2-45   Parameterized SQL Mapper - Active Parameter Mappings
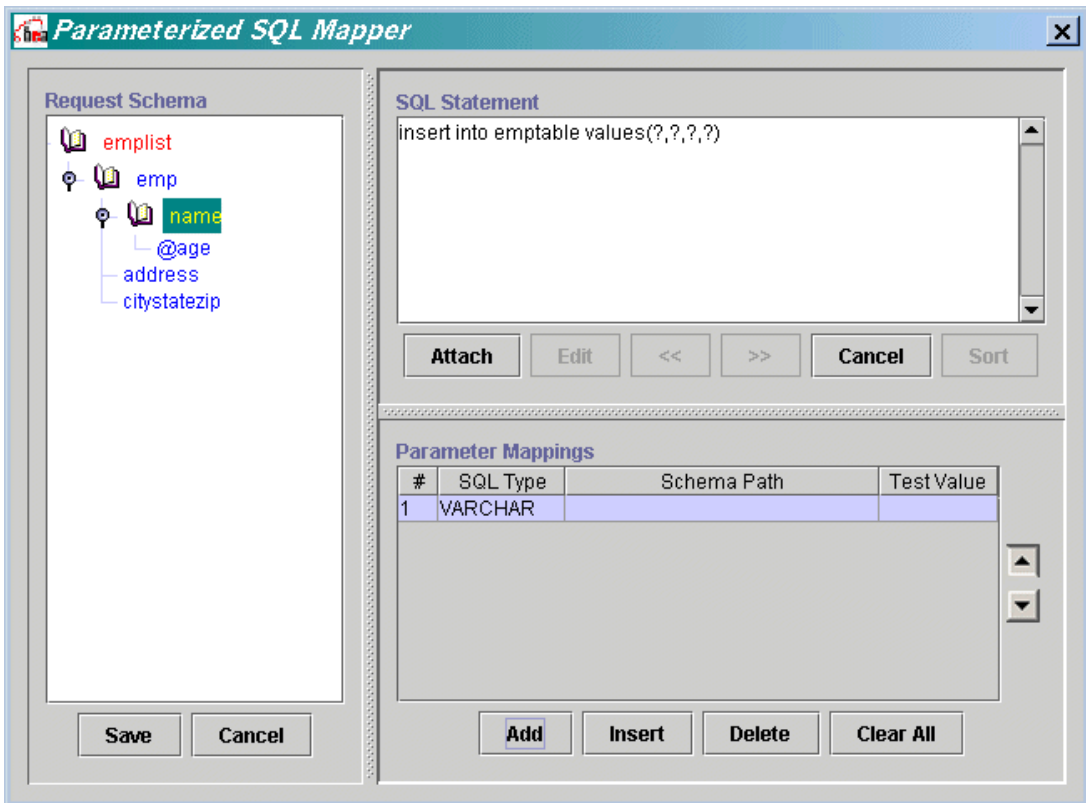
You now can map values taken from the sample XML to the four parameters specified in the Parameterized SQL. For each parameter value there must be one entry in the Parameter Mappings dialog box.

d. Click the SQL Type field in the Parameter Mappings box.

A drop-down list opens.

Figure 2-46   Parameterized SQL Mapper - Parameter Mappings Drop-Down
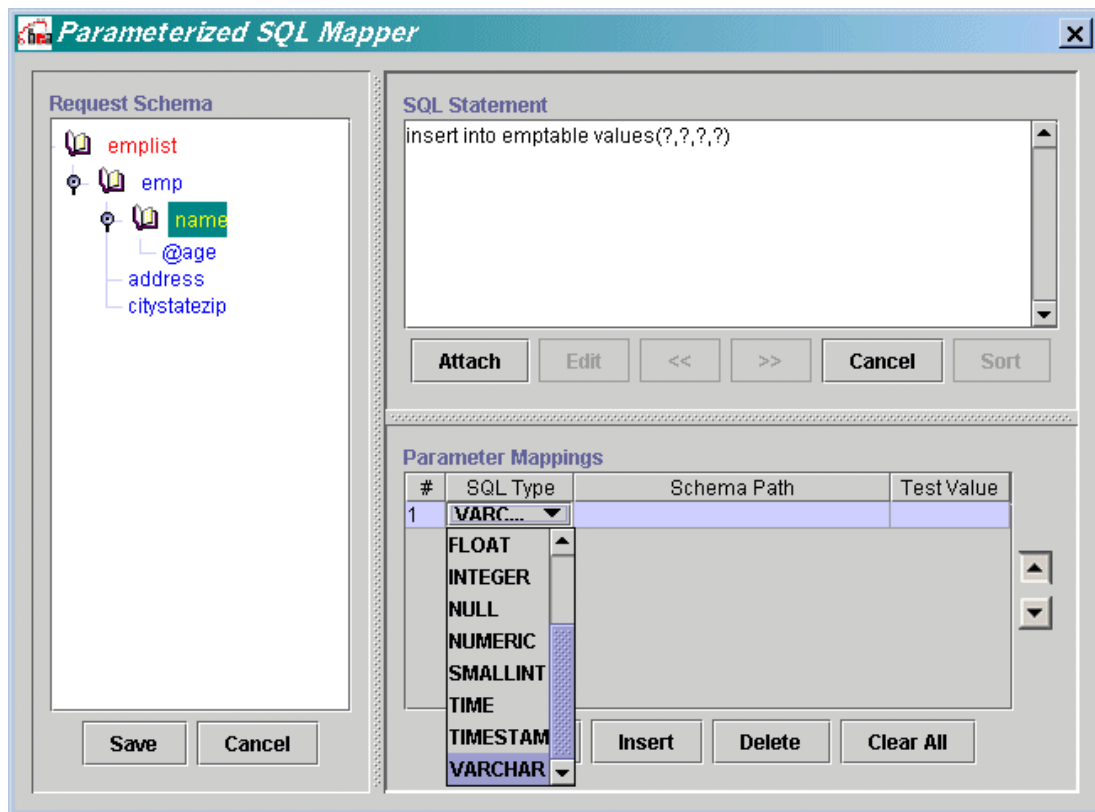
e. Select the SQL Type of the first parameter specified positionally from the Parameterized SQL Statement.

Figure 2-47   Parameterized SQL Mapper with Name Element Selected
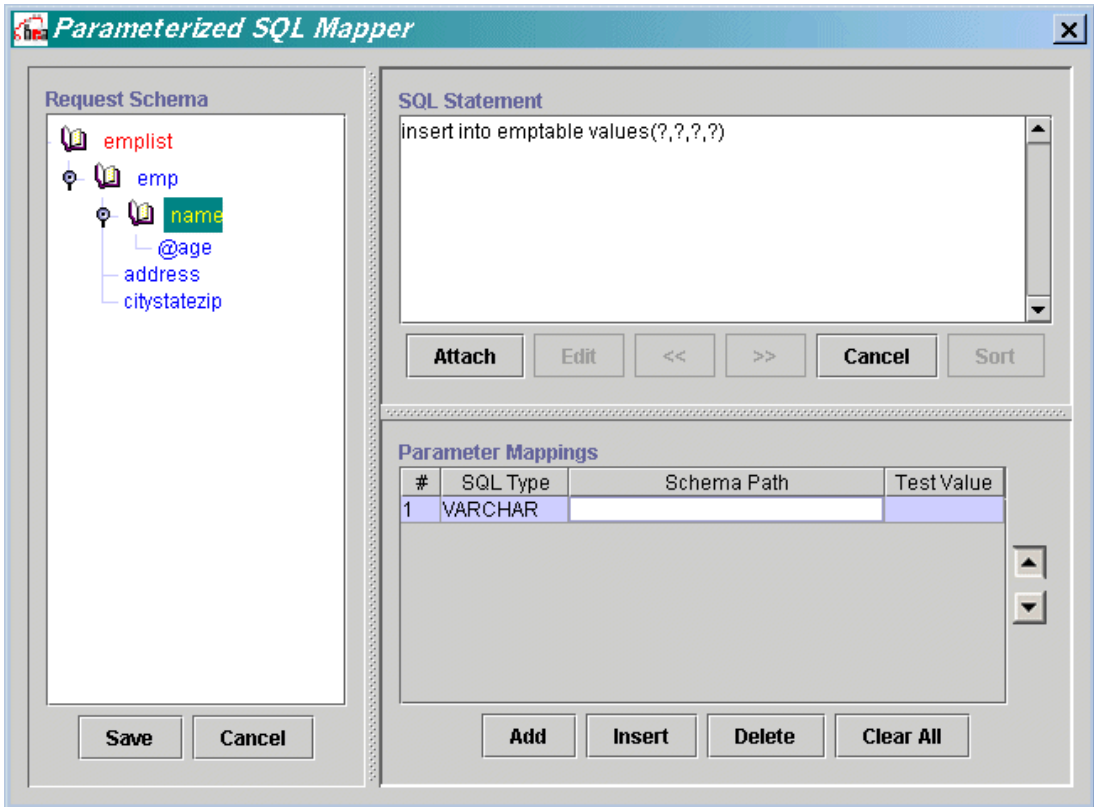
f. Double-click the value in the Request Schema tree that corresponds to the element name this parameter will map to, in this example, the Name element.

The Schema Path field is now updated with the element selected relative to the root node of the document, as shown in the following figure.

Figure 2-48   Parameterized SQL Mapper with Updated Element
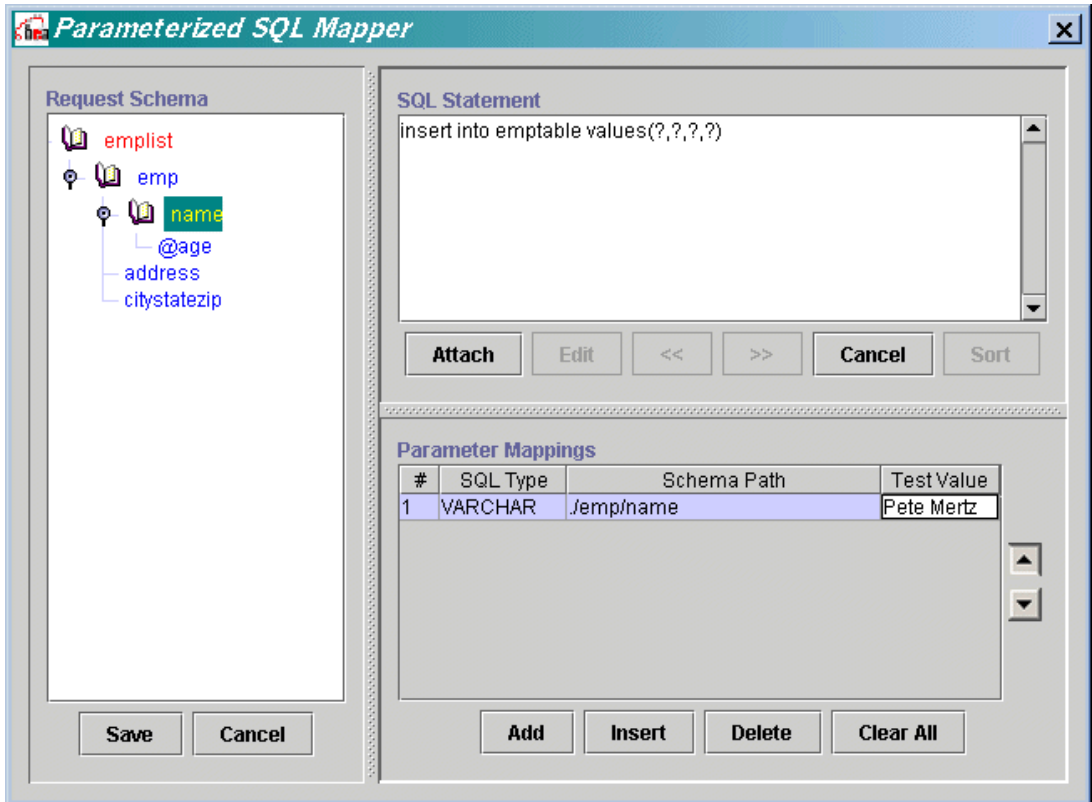
g. Click Test Value and enter a sample value used to verify this type.

h. Click Add to map the next parameter. Repeat this step until all parameters are mapped (in this case, four).

Note: In this example, it is possible to map a parameter to an attribute by selecting the @age entry from the Request Schema tree.

Figure 2-49   Parameterized SQL Mapper with @Age Entry Selected
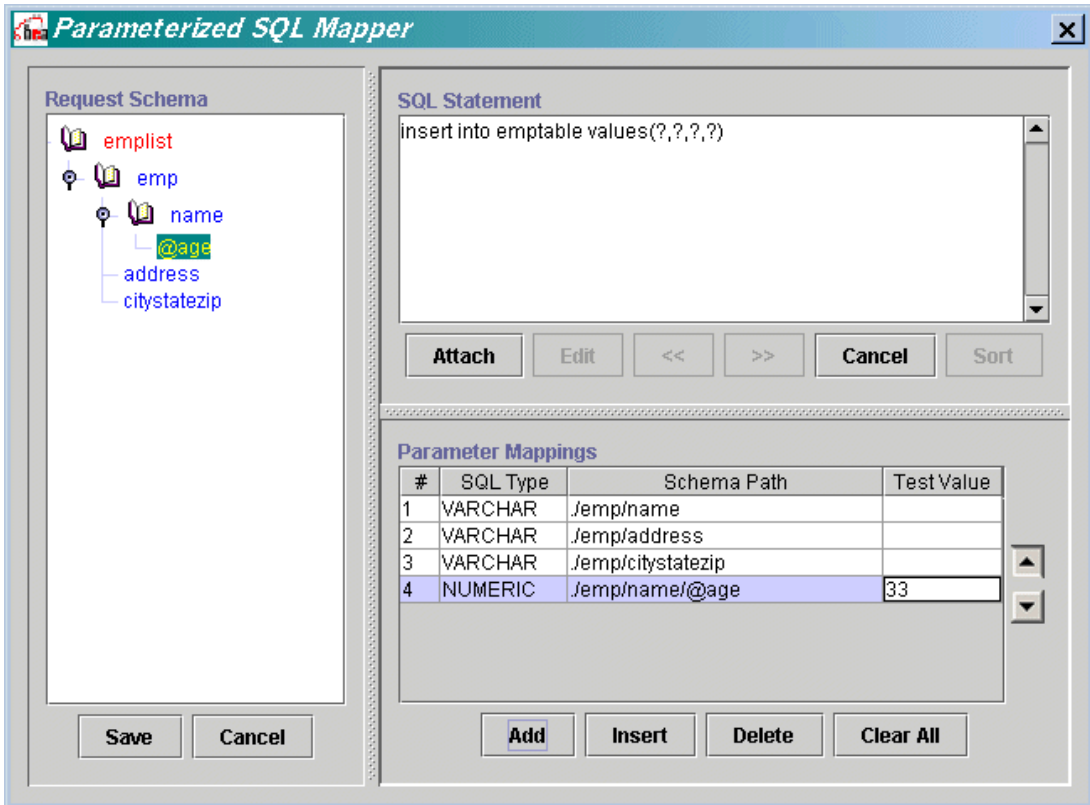
5. After all the parameters are mapped, click Attach.

A Details tab appears.

Figure 2-50   Detailed Parameterized SQL Statement
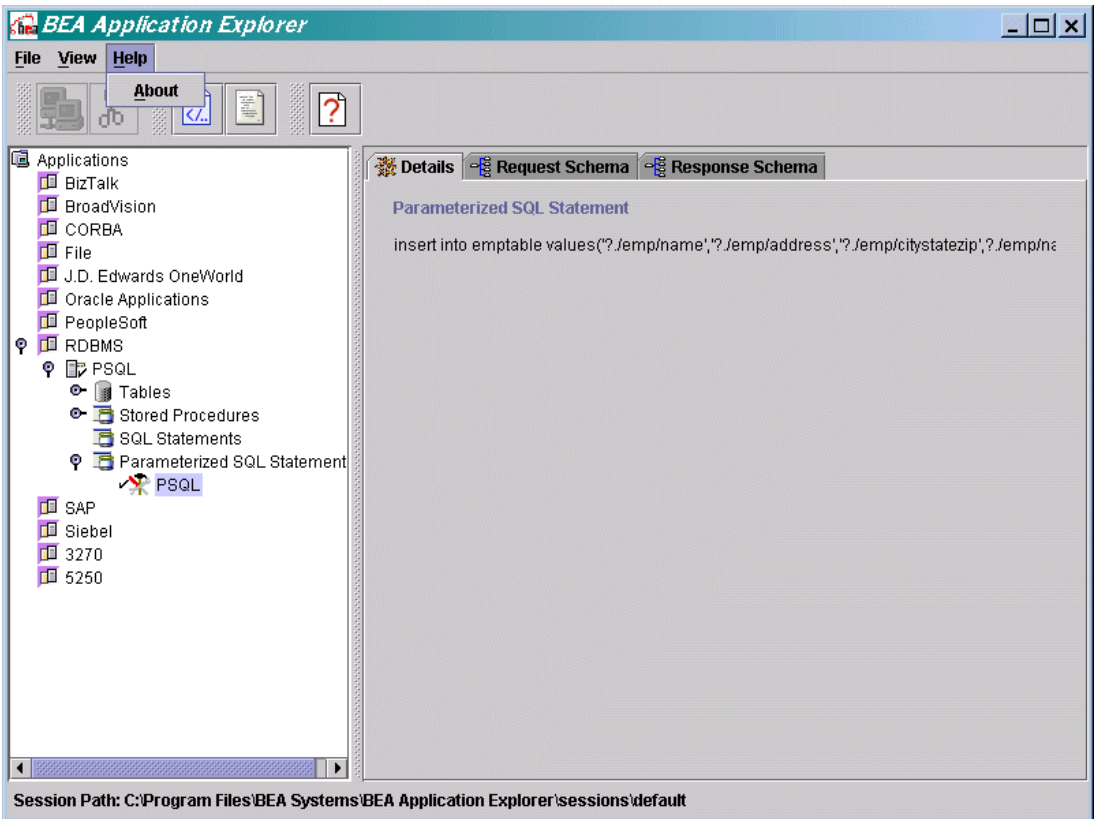
6. Click the Request Schema tab or the Response Schema tab to view the Request or Response schema, respectively.

   Note: For a further explanation on setting up and configuring a service for Parameterized SQL as well as a discussion of target nodes, see Chapter 3, "Defining an Application View."

Combining Parameterized SQL Feature with Stored Procedures

It is possible to combine the parameterized SQL feature and stored procedures. In effect, this is done to set up a parameterized SQL statement that calls a stored procedure.

For more information on how to combine the parameterized SQL feature and stored procedures, see "Generating Service Schemas Under the Parameterized SQL Statement Node" on page 2-40.

Generating Schemas for Stored Procedures

To create a service schema for a stored procedure for use with the BEA WebLogic Adapter for RDBMS, perform the following steps:

1. Start the BEA Application Explorer by choosing Start→Programs→BEA Application Explorer.

The BEA Application Explorer opens.

Figure 2-51   BEA Application Explorer - Create Service Schemas
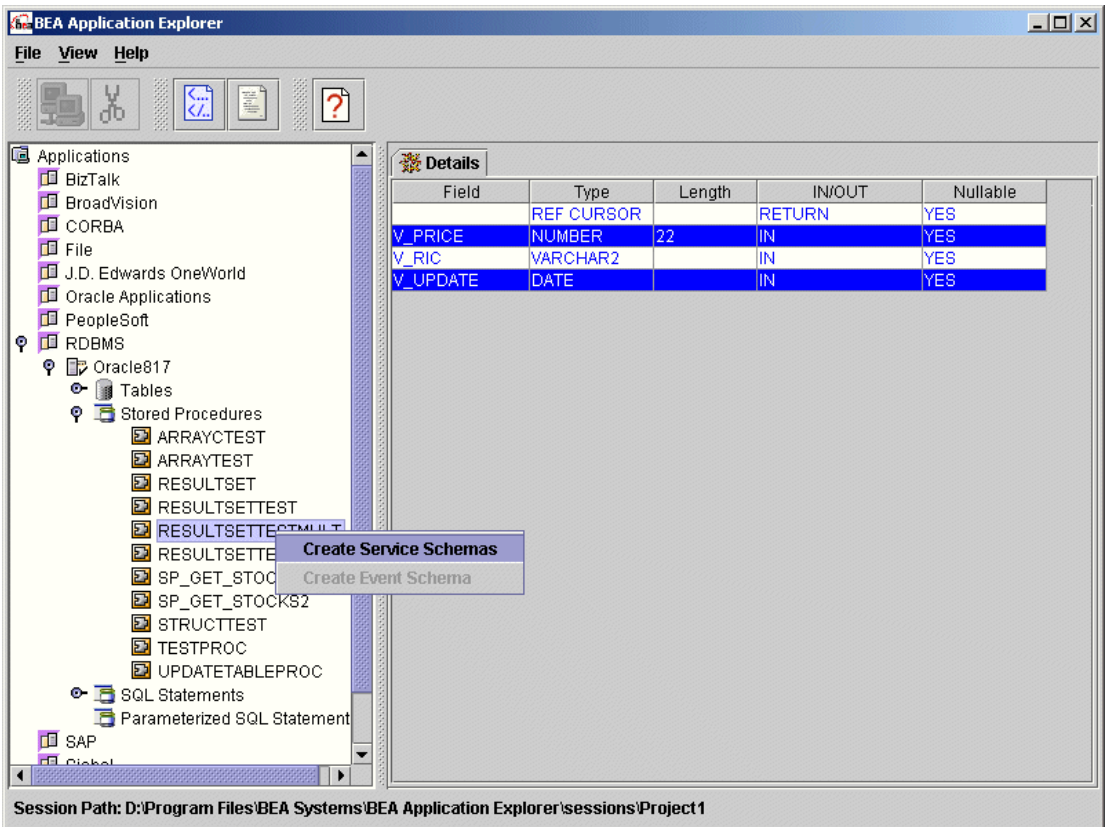
2. Right-click the stored procedure. Note that Create Event Schemas is not available.

3. Select Create Service Schemas.

The Create Service Schema window opens.

Figure 2-52 Create Service Schema Window
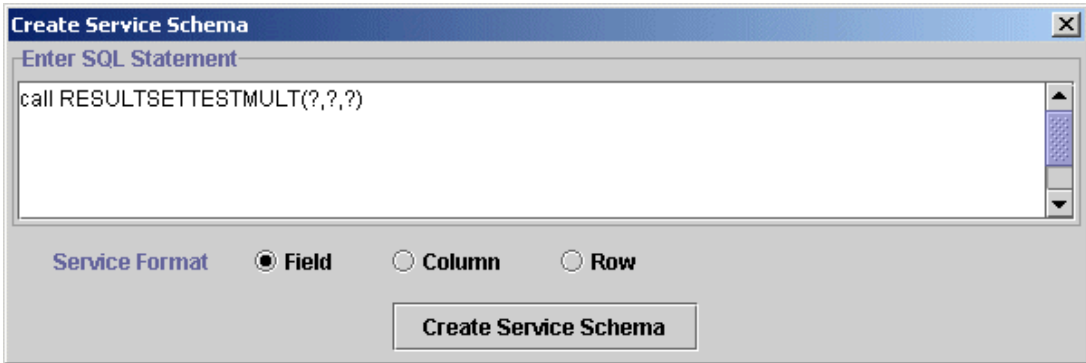
The Create Service Schema window shows all parameter markers denoted by a "?".

a. Replace each ? associated with an IN parameter with the value you want to call the stored procedure.

   Ignore all parameter markers (?'s) that represent OUT parameters.

   For example you can enter 40, 'TEST', '2002-07-19', as shown in the following figure.

Figure 2-53 Create Service Schema - Test SQL Value
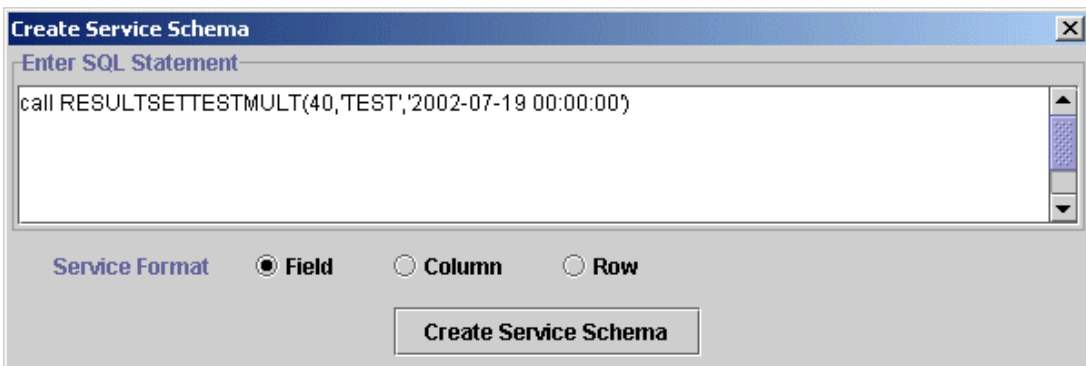
b. Select Field, Column, or Row.

This determines the service response format for this stored procedure.

The value selected must match the format value used to generate the application view service, or the schema of the request document does not validate.

c. To execute the stored procedure, click Create Service Schema.

The stored procedure is run with the parameters entered.

The right pane of the Application Explorer displays the Details, Request Schema, and Response Schema tabs as shown in the following figure.

Figure 2-54   BEA Application Explorer - Schema Display
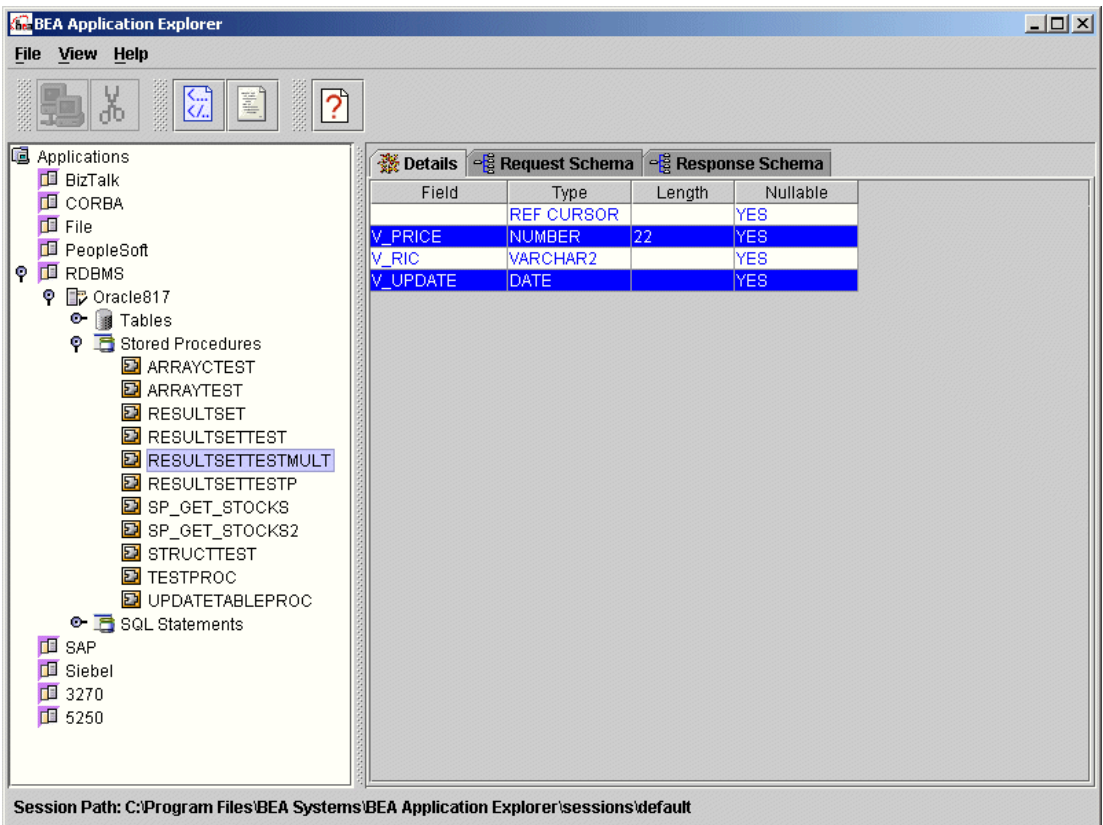
4. Click the Request Schema tab to see the request schema, as shown in the following figure.

Figure 2-55   BEA Application Explorer - Request Schema Tab
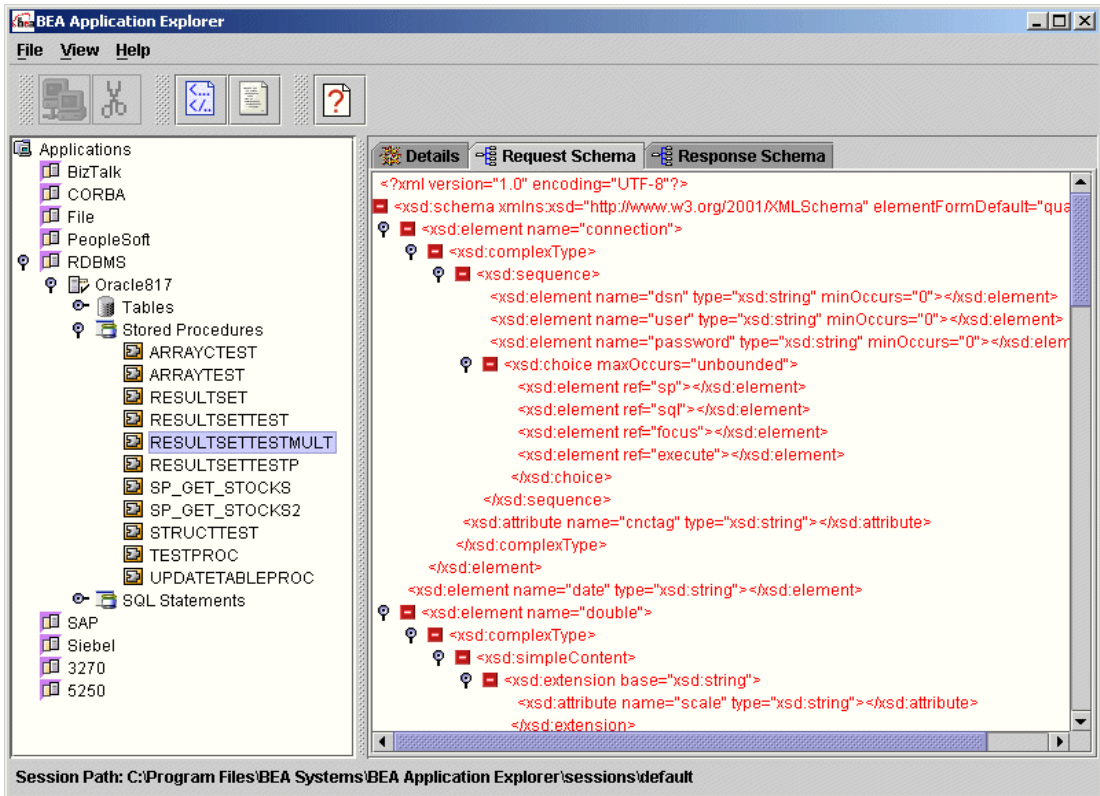
d. Click the Response Schema tab to see the response schema, as shown in the following figure.

Figure 2-56   BEA Application Explorer - Response Tab
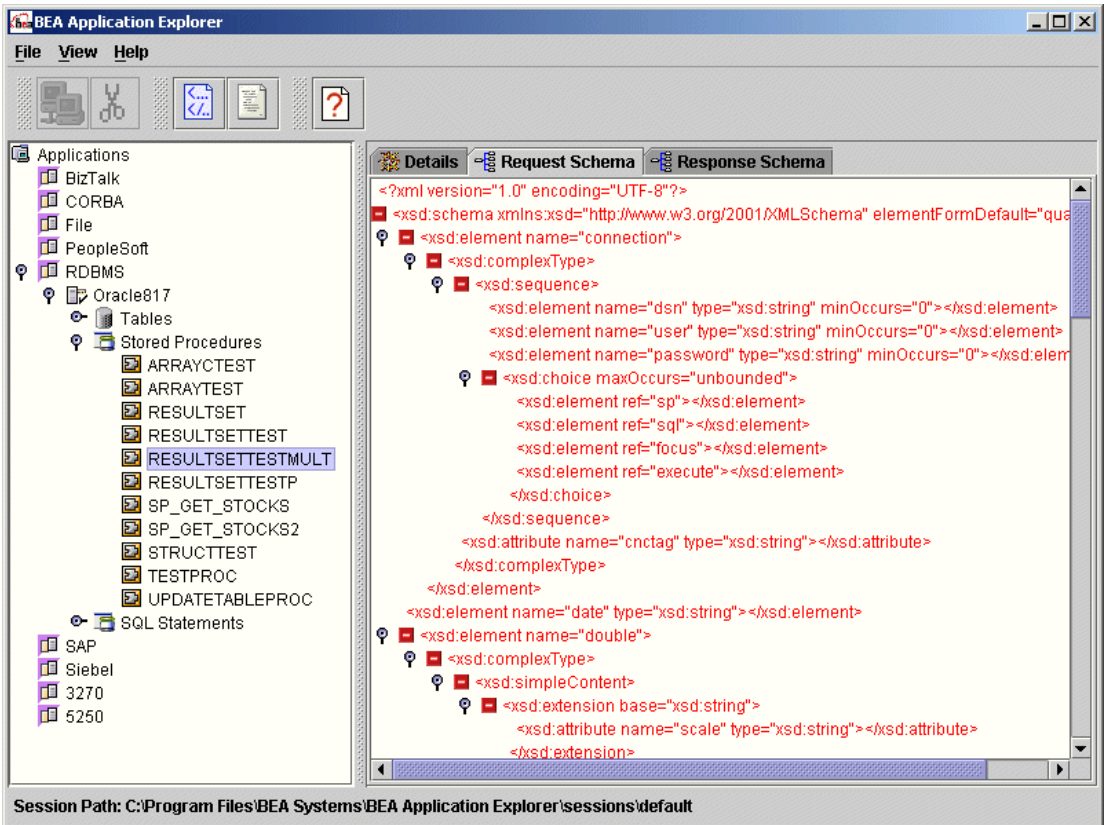
The schemas are now ready to use.

Removing Schemas

To remove a schema from an SQL Statement:

1. Start the BEA Application Explorer by choosing Start→Programs→BEA
   Application Explorer.

   The BEA Application Explorer opens.

Figure 2-57   BEA Application Explorer - Remove Schema Option
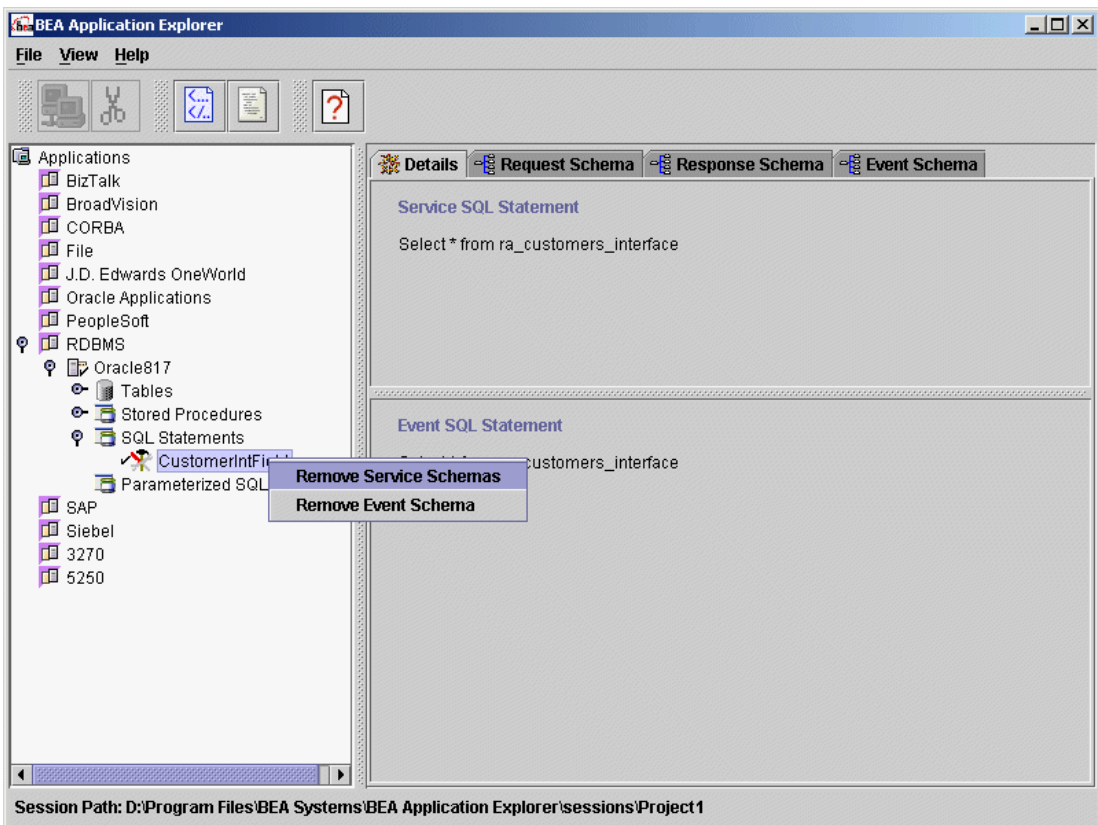
2. Right-click the desired SQL Statement.

3. Select a remove schema option.

   The schema(s) (service or event) is removed, and the manifest.xml file is updated.

   The right pane no longer displays the Request and Response Schema tabs for the schema after you select Remove Service Schemas as shown in the following figure.

Figure 2-58   BEA Application Explorer - Schema Removed
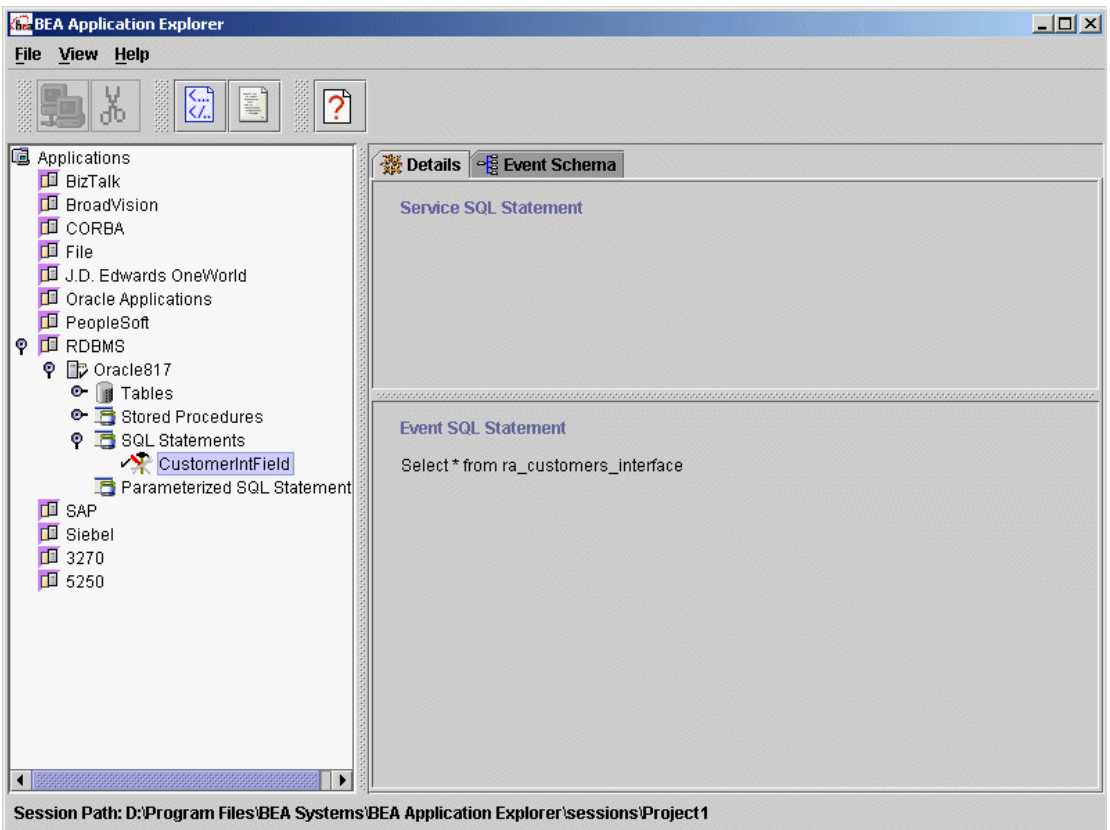
3   Defining an Application View

This section provides information on creating and deploying application views that include the BEA WebLogic Adapter for RDBMS. It includes the following topics:

- Defining a New Application View
- Adding a Service Adapter to an Application View
- Deploying an Application View
- Adding an Event Adapter to an Application View
- Handling Null Values
- Defining a Data Source

Defining a New Application View

When you define an application view, you create an XML-based interface between WebLogic Server and a particular relational database management system (RDBMS) within your enterprise. Once you create the application view, a business analyst can create business processes that use the application view. For any adapter, you can create any number of application views, each with any number of services and events.

Note: This procedure shows how to install and configure an Oracle implementation of the RDBMS event and service adapter. When configuring other JDBC drivers for other RDBMSs, the parameters you enter are different.

To define an application view, perform the following steps:

1. Navigate your browser to the Application View Console - Logon screen. The Application View Console can be found at `http://host:port/wlai`, here, *host* is the IP address or DNS name on which WebLogic Server is installed, and *port* is the socket on which the server is listening. The default port is 7001.

Figure 3-1   Application View Console - Logon Window
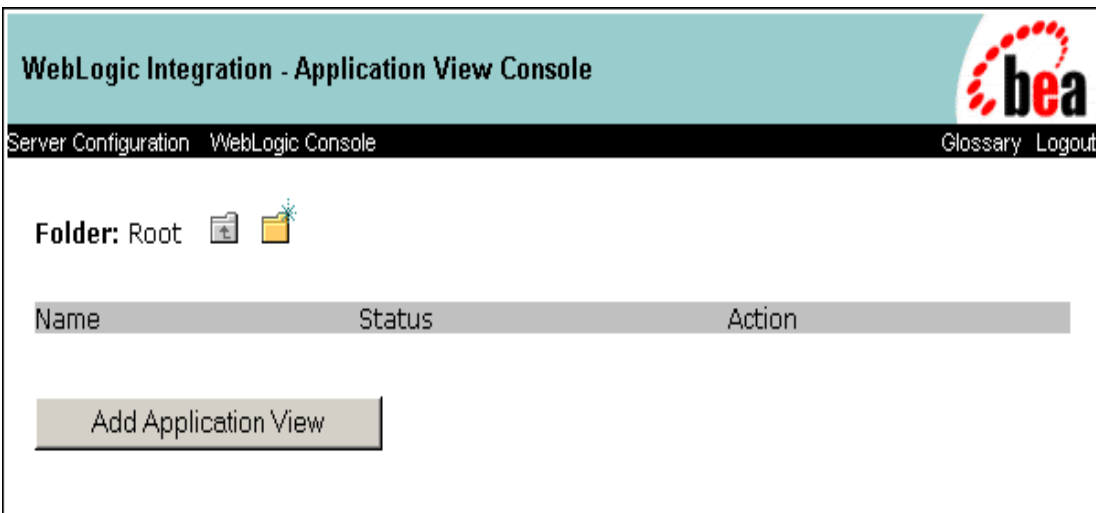
2. Enter a valid WebLogic user name and password and click Login. For more information, see "Logging On to the WebLogic Integration Application View Console" in "Defining an Application View" in *Using Application Integration*:

   - For WebLogic Integration 7.0, see
     `http://edocs.bea.com/wli/docs70/aiuser/2usrdef.htm`

   - For WebLogic Integration 2.1, see
     `http://edocs.bea.com/wlintegration/v2_1sp/aiuser/2usrdef.htm`

   Note: If the user name is not `system`, it must be included in the `adapter` group. For more information on adding the administrative server user name to the `adapter` group, see the *BEA WebLogic Adapter for RDBMS Installation and Configuration Guide*.

Figure 3-2   Application View Console Window
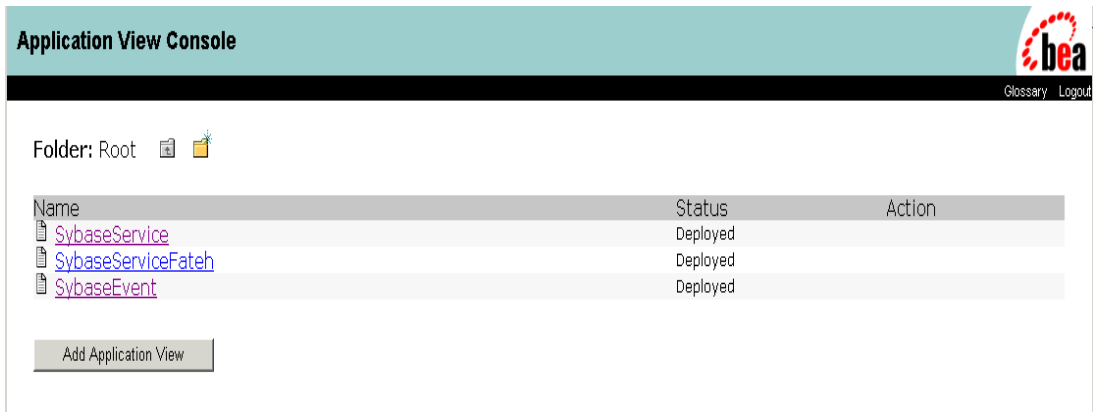
3. Click Add Application View to create an application view for the adapter. An application view enables a set of business processes for this adapter's target enterprise information system (EIS) application. For more information, see "Defining an Application View" in *Using Application Integration*:

    ● For WebLogic Integration 7.0, see
       `http://edocs.bea.com/wli/docs70/aiuser/2usrdef.htm`

    ● For WebLogic Integration 2.1, see
       `http://edocs.bea.com/wlintegration/v2_1sp/aiuser/2usrdef.htm`

Figure 3-3 Define New Application View Window

4. In the Application View Name field, enter a name. The name should describe the set of functions performed by this application. Each application view name must be unique to its adapter. Valid characters include a-z, A-Z, 0-9, and the _ (underscore) character.

5. In the Description field, enter any relevant notes. These notes are displayed when you use this application view to create workflows in WebLogic Integration Studio.

6. From the Associated Adapter drop-down list, select BEA_RDBMS_1_0 and click OK to create this application view. For important information on the BEA_RDBMS_1_0.ear file, see the *BEA WebLogic Adapter for RDBMS Installation and Configuration Guide.*

7. Click OK.

Figure 3-4   Configure Connection Parameters Window
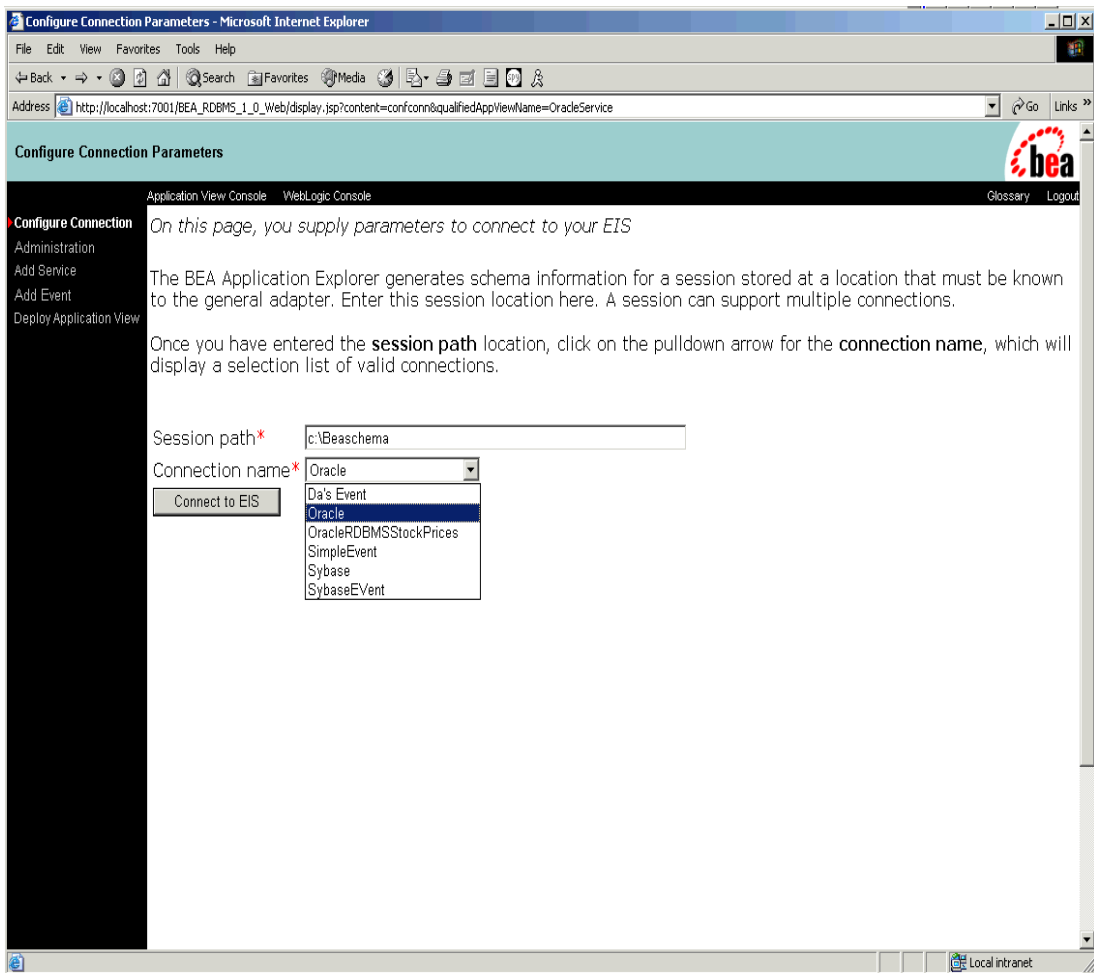
8. In the Session path field, enter the location of the working directory established for the BEA Application Explorer. For more information, see Chapter 2, "Using the BEA Application Explorer With an RDBMS."

9. In the Connection name field, select the name of the connection used for creating schemas. The Application Explorer creates this folder for you.

10. Click Connect to EIS.

11. Click Continue.

The Application View Administration window is displayed, as shown in the following figure:

Figure 3-5   Application View Administration Window
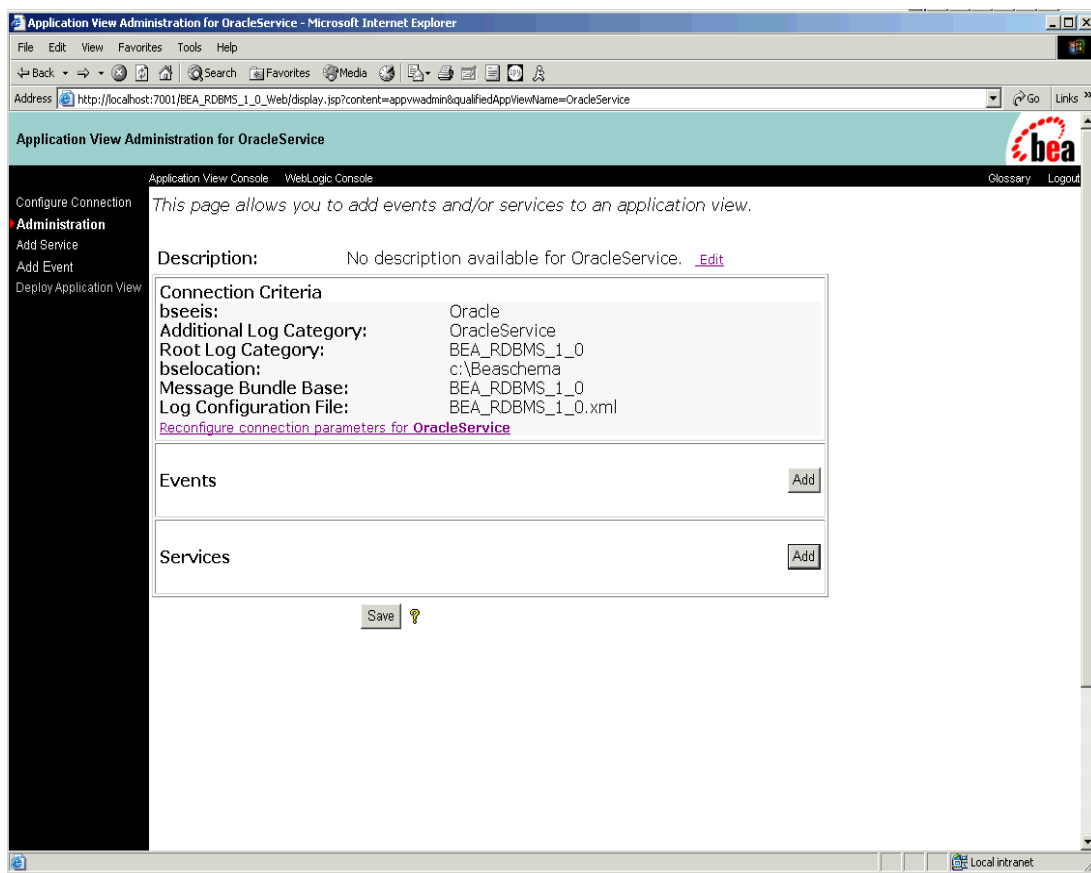
At this point you can create event(s) and/or service(s).

- For more information on adding services, see "Adding a Service Adapter to an Application View" on page 3-7.

- For more information on adding events, see "Adding an Event Adapter to an Application View" on page 3-37.

Adding a Service Adapter to an Application View

After you create and configure an application view as described in "Defining a New Application View" on page 3-1, add services that support the application's functions.

For this example, create a DBMS service adapter. Information about other service types is presented throughout this section.

1. While the Application View window is open, click Administration. The Application View Administration window is displayed, as shown in the following figure.

Figure 3-6   Application View Administration Window

2. Click Add in the Services row.

Figure 3-7   Add Service Window

3. Enter the following parameter information in the appropriate field:

Table 3-1 Service Properties

Parameter	Definition
Unique Service Name* (*Required)	This name must be unique to its application view. Valid characters include a-z, A-Z, 0-9, and the _ (underscore) character.
Select	Choose DBMS as the listener.
user* (*Required)	The RDBMS Application's user ID authorized to access the Oracle Applications system.
Password	A password associated with the specified user ID.

Table 3-1 Service Properties (Continued)

Parameter	Definition
Isolation Level* (*Required) **Note:** Before setting the isolation level for a service, check with your database administrator.	Choose one of the following: ■ **asis.** Does not specifically set an isolation level for the connection. ■ **readUncommitted.** Does not prevent any read violation. ■ **readCommitted.** Only data that has been committed by a transaction can be read by other transactions. This level prohibits a transaction from reading a row with uncommitted changes in it. This setting prevents a dirty read, but allows non-repeatable reads and phantom reads. ■ **repeatableRead.** Only data that has been committed by a transaction can be read by other transactions, and multiple reads yield the same result as long as the data has not been committed. This setting prevents dirty reads and non-repeatable reads, but not phantom reads. ■ **Serializable.** This setting is the highest isolation level, stipulating that all transactions run serially to achieve maximum data integrity. This yields the slowest performance and least concurrency. This setting prevents Dirty Reads, Non-repeatable reads, and Phantom reads. **Note:** For definitions of Dirty Read, Phantom Read, and Non-repeatable read, and for more information on isolation levels, see "Transaction Isolation Levels" on page 3-12.

format* (*Required)	Choose one of the following:
	
	■ **row.** The data that is produced is returned on a single line (per record) enclosed in <row> tags.
	
	■ **column.** The data produced is returned field by field, and each field is enclosed in <column> tags. The column tag has an attribute whose value is the name of the field; for example,
	
	`<row>`
	`<column name="ID">1000</column>`
	`<column name="First_Name">Scott</column>`
	`</row>`
	
	■ **field.** The data produced is returned field by field, and each field is enclosed in a tag that bears the field name; for example,
	
	`<row>`
	`<ID>1000></column>`
	`<FIRST_NAME>Scott</column>`
	`</row>`

Table 3-1 Service Properties (Continued)

Parameter	Definition
DRIVER* (*Required)	The name of the JDBC Driver. For example, the Oracle JDBC driver is oracle.jdbc.driver.OracleDriver.
URL* (*Required)	The address (URL) for the connection to the RDBMS. For example, an Oracle URL is `jdbc:oracle:thin:@Oracle.ibi.com:app`
Data_Source_Name	The Data Source JNDI name for the JDBC connection pool to use for connecting to the RDBMS system. If a value is present, the adapter will use the connection pool to connect to the RDBMS. If no value is specified, connection will use the Driver, URL, UserId and Password specified in the service. **Note:** For more information on setting up a Data Source, see "Defining a Data Source" on page 51.
DataSourceType	Choose one of the following: ■ **ConnectionPoolDataSource.** The connection name specified in the Data_Source_Name field is used as a JNDI context for a WebLogic Integration connection pool. ■ **XADatasource.** The connection name specified in the Data_Source_Name field is used as a JNDI context for an XADatasource whose transactions participate in the WebLogic Integration XA transaction. If the Data_Source_Name field is left blank, the adapter uses the user ID, password, and URL to establish a plain JDBC 2.0 connection with the database. If the Data Source_Name field is populated, the user ID, password, and URL are ignored.
schema	From the drop-down list, select the name of the schema that contains connection and other related information about the service you are adding.

4. Click Add.

Figure 3-8 Application View Administration Window (Example Oracle Service)
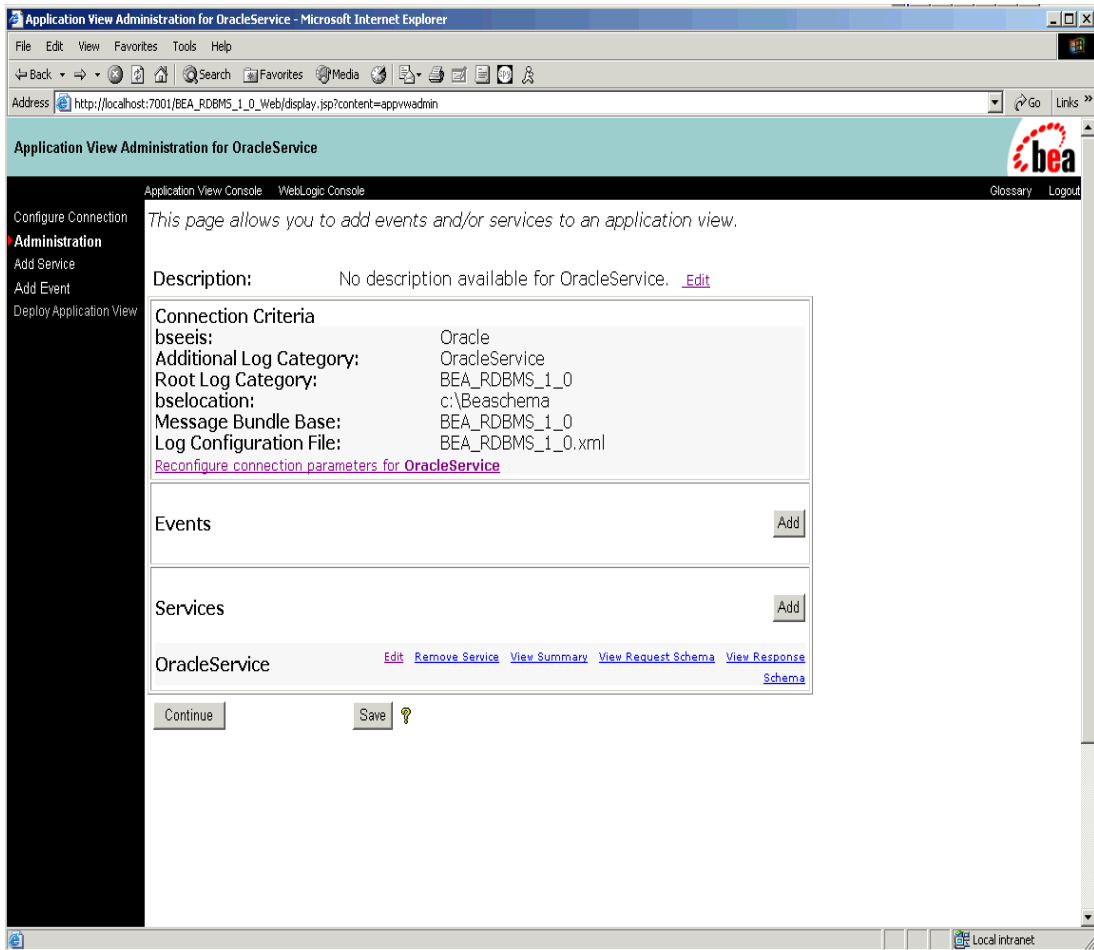
5. If you are finished adding services or events, click the Continue button to deploy the application view.

    For more information on deploying and testing application views, see "Deploying an Application View" on page 3-15.

Transaction Isolation Levels

Transaction isolation levels, as defined in the ANSI SQL specification, are supported by the JDBC standard. As such, the BEA WebLogic Adapter for RDBMS supports transaction isolation to the level of the underlying RDBMS and its JDBC driver. For more information on isolation levels, refer to the ANSI SQL specification.

Isolation levels manage the level of interference between transactions in a multi-user database system. In an ideal world, all transactions would be serializable, meaning that the same results would be produced whether transactions are run concurrently or in series. Unfortunately, a high level of transaction isolation has significant performance implications, and the level of isolation should be set according to the transaction needs.

The settings available for isolation levels for the BEA WebLogic Adapter for RDBMS are as follows:

Note: For more information about where to establish isolation levels, see "Adding a Service Adapter to an Application View" on page 3-7.

Table 3-2 Isolation Level Settings

Isolation Level	Description
as is	This level does not specifically set an isolation level for the connection
Read Uncommitted	Data that has been updated but not yet committed by a transaction may be read by other transactions. This level does not prevent any read violation. It allows a row changed by one transaction to be read by another transaction before any changes in that row have been committed. If any of the changes are rolled back, the second transaction retrieves an invalid row.
Read Committed	Only data that has been committed by a transaction can be read by other transactions. This level prohibits a transaction from reading a row with uncommitted changes in it.
Repeatable Read	Only data that has been committed by a transaction can be read by other transactions, and multiple reads yield the same result as long as the data has not been committed.

Table 3-2 Isolation Level Settings (Continued)

Isolation Level	Description
Serializable	This is the highest possible isolation level and ensures a transaction's exclusive read-write access to data. It includes the conditions of ReadCommitted and RepeatableRead. This setting is the highest isolation level, stipulating that all transactions run serially to achieve maximum data integrity. This yields the slowest performance and least concurrency.

Three problems affect the multi-user database system:

- Dirty Reads

- Non-repeatable Reads

- Phantom Reads

Dirty Reads are common. In the Dirty Read, transaction A updates a row in a database, but has not yet committed the update. Transaction B then reads the updated row. Due to a problem in the first transaction, such as the failure of a second update in transaction A, the original update is not committed and a rollback is issued. Transaction B reads a record that was incorrect (Dirty).

Dirty Reads are prevented by selecting Read Committed, Repeatable Read, or Serializable as an isolation level.

Another problem is the Non-Repeatable Read. In this case a transaction, transaction A, reads a row from a database and continues on with its processing. After the initial read by transaction A, another transaction, transaction B, updates the same row in the database. Transaction A then rereads the row, but the row has changed, hence the read is non-repeatable.

Non-Repeatable Read is prevented by selecting Repeatable Read or Serializable as an isolation level.

In the case of a Phantom Read, a transaction, transaction A, reads a set of rows from a database and continues with its processing. After the initial read by transaction A, another transaction, transaction B, adds (or deletes) records that transaction A would have received. If transaction A then re-reads the database, there are additional (or missing) Phantom records.

Phantom Read is addressed only by selecting Serializable as an isolation level.

For information on how to set isolation levels, see "Transaction Isolation Levels" on page 3-12.

Transaction Management

The BEA WebLogic Adapter for RDBMS supports transaction management in WebLogic Integration. Transaction management enables the BEA WebLogic Adapter for RDBMS to commit or rollback action, based on successful or unsuccessful actions respectively, of subsequent tasks in the workflow.

To participate in transactions, the agent joins the transaction in progress by identifying a class that exposes the XDTx interface. This class is identified to and joins the transaction via the storeTx() method, using a transaction identifier that is passed into the agent via the getTID() method. The class itself implements commit, rollback, prepare, and canPrepare methods. The Java hashCode() method associates like classes in the transaction manager.

The BEA WebLogic Adapter for RDBMS delegates the transaction control to the transaction class. In the adapter, if the service adapter is determined to be running in a transaction, getTID() returns a transaction id. Then, the agent does not commit or close the connection but delegates this processing to the transaction class. If the service adapter is not in a transaction, getTID() returns null. The agent performs the commit() and close() prior to returning control to the workflow.

Note that the transaction class is invoked by the system after the BEA WebLogic Adapter for RDBMS returns from the processing. Additional work on an RDBMS must be performed by another task.

For more information on transaction management, see "Understanding the BPM Transaction Model" in *Programming BPM Client Applications*:

- For WebLogic Integration 7.0, see
  `http://edocs.bea.com/wli/docs70/devclient/trans.htm`

- For WebLogic Integration 2.1, see
  `http://edocs.bea.com/wlintegration/v2_1sp/devclient/trans.htm`

Deploying an Application View

You may deploy an application view when you have added at least one event or service to it. You must deploy an application view before you can test its services and/or events or use the application view in the WebLogic Server environment. Application view deployment places relevant metadata about its services and events into a run-time metadata repository. Deployment makes the application view available to other WebLogic Server clients. This means business processes can interact with the application view, and you can test the application view's services and events. To deploy an application view, perform the following steps:

1. With the application view open, click Administration.

   The Application Administration window is displayed.

2. To deploy the application view, click Deploy Application View. You can click the Save button and deploy the application view at a later time.

   The Deploy Application View window is displayed, as follows:

   Note: To enable workflow functionality or other authorized clients to asynchronously call the services (if any) of this application view, select the Enable Asynchronous Service Invocation check box.

Figure 3-9   Deploy Application View Window (ORACLE Service to Server)

3. Click Deploy. The Summary for Application View Window opens.

Figure 3-10   Summary for Application View Window (Example Oracle Service)
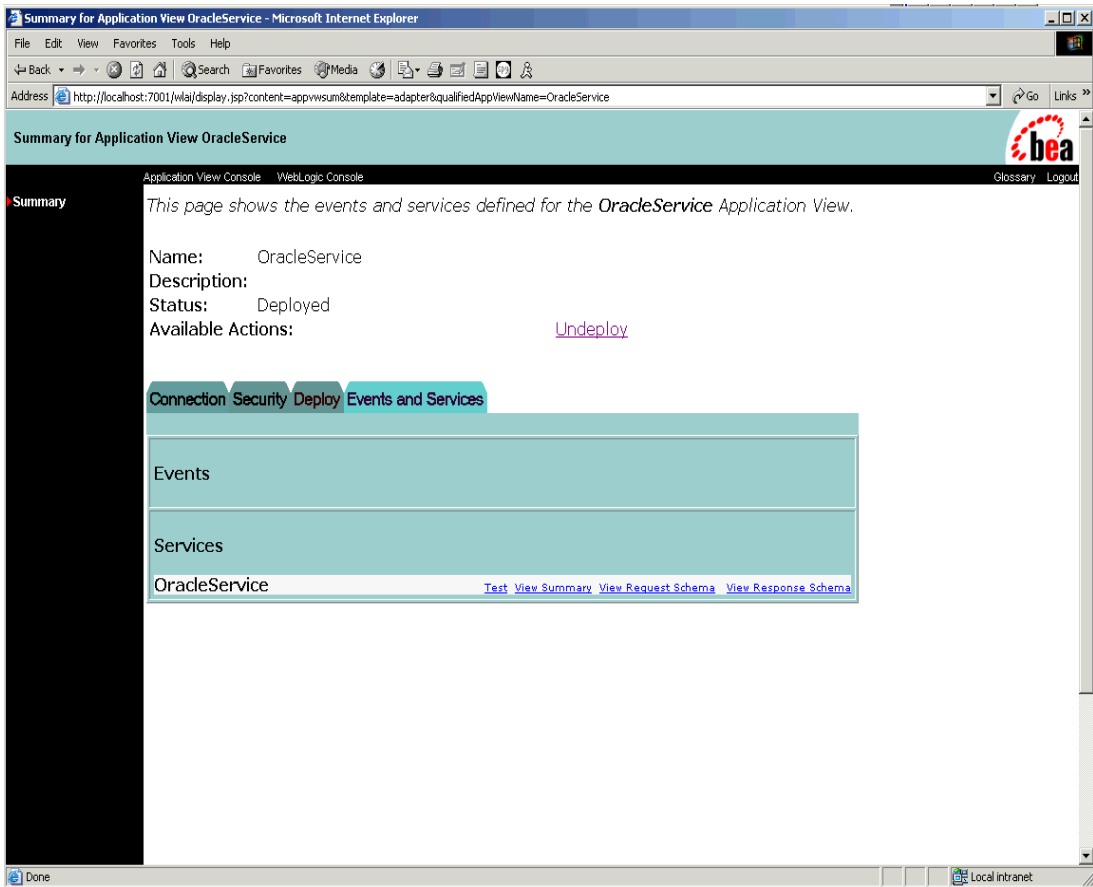
> After you create and deploy an application view that contains services or events, test the application view services. Testing evaluates whether or not the application view service or event interacts properly with the target adapter.

4. To test the application view service, find the service or event in the current Services or Events area and click the Test link.

The Test window is displayed.

Figure 3-11   Test Service Window (Oracle)
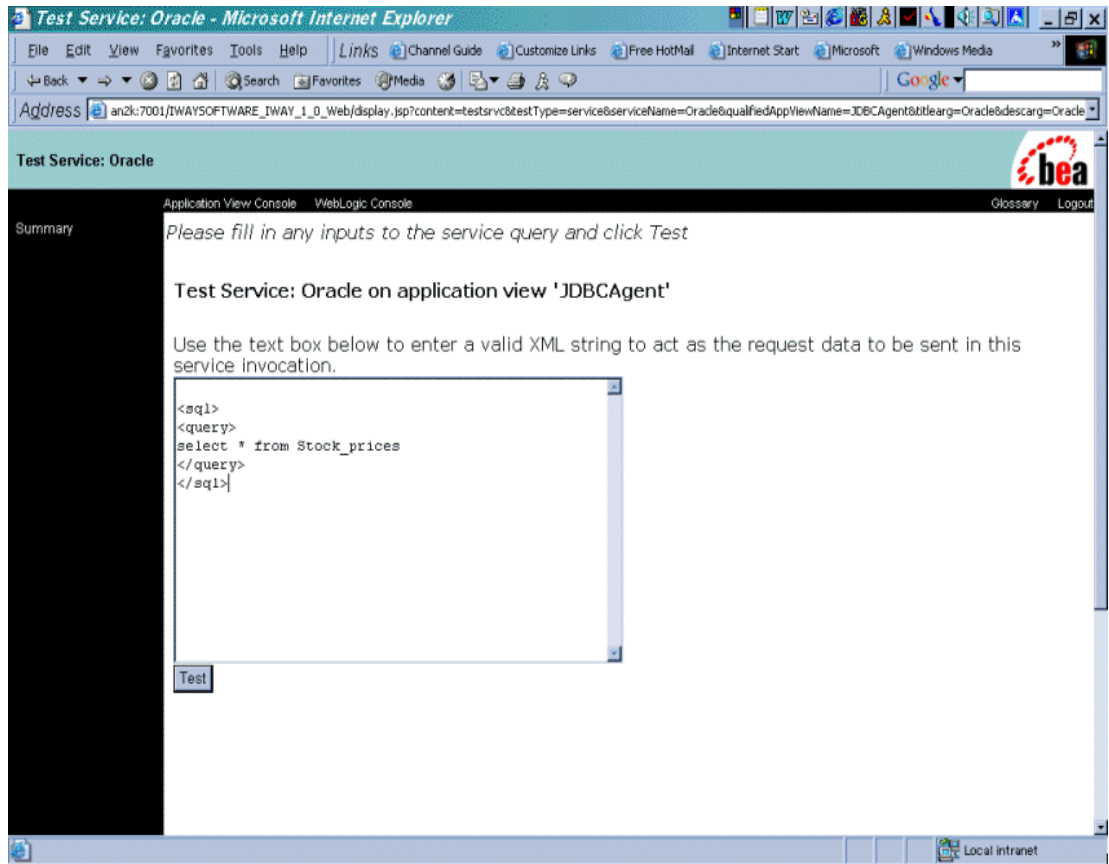
5. Enter the appropriate XML for the RDBMS service adapter.

   The format for the XML is:

   ```
   <connection>
      <sql>
         <query>your sql statement here</query>
      </sql>
      <sql>
         <query>your subsequent sql statement here</query>
      </sql>
   </connection>
   ```

6. Click Test. The test request is executed and the Test Results window is displayed.

Figure 3-12   Test Results Window (Oracle)
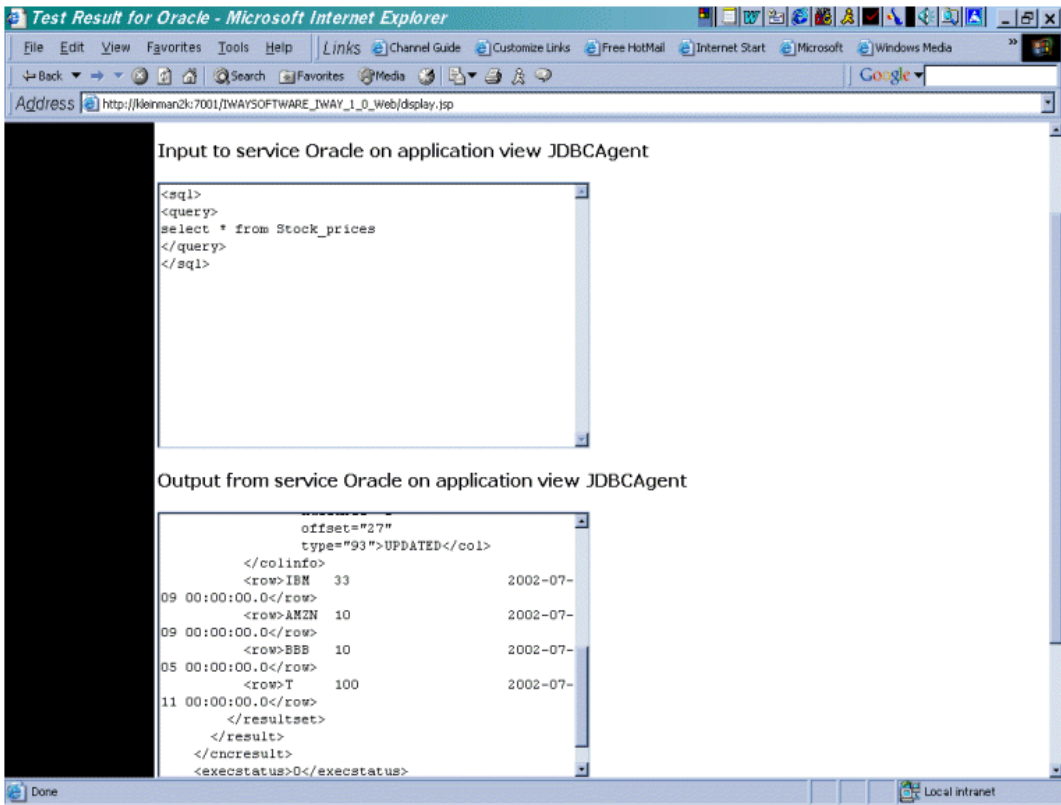
If the test is successful, the Test Result window displays the input XML and the result set. This confirms that the application view service is successfully deployed. You can now employ the service in business process workflows or write custom code. For more information, see "Using Application Views in the Studio" in *Using Application Integration*:

- For WebLogic Integration 7.0, see
  `http://edocs.bea.com/wli/docs70/aiuser/3usruse.htm`

- For WebLogic Integration 2.1, see
  `http://edocs.bea.com/wlintegration/v2_1sp/aiuser/3usruse.htm`

If the test fails, the Test Result window displays a timed out message.

Example: Issuing an SQL Query Request

The Request document in the following example connects to a data source (using the connect information specified in the service configuration) and issues a simple SQL query. This document also specifies that the Field Agent executes the request:

```
<sql>
   <query>
select * from stock_prices
   </query>
</sql>
```

Example: Issuing a Stored Procedure Request

The service adapter can execute stored procedures against Sybase, Oracle, DB2, Informix, and MS SQL Server. Because DBMS manufacturers produce JDBC agents with varying degrees of functionality, stored procedures can be executed in several different ways within the service adapter. Depending on the DBMS being selected and the return of a result set, the listener type selected from the Add Service window can change.

This example consists of two parts: stored procedures for use with Sybase and stored procedures for use with Oracle, which itself contains a number of smaller examples.

Note: Oracle subprograms (stored procedures) can be called whether they reside as functions, stored procedures, or stored procedures located within packages.

Stored Procedures for Use with Sybase

Stored procedures returning result sets are called from the same services as those that do not return result sets. To set this up, add a service as you normally would and select the DBMS Listener from the Add Service window.

The format for calling the procedure is as follows:

```
<sql>
 <query>
   exec stored_procedure_name( @param_variable1 = value1,
   @param_variable2 = value2,… etc)
 </query>
<sql>
```

The elements of the procedure call are defined as follows:

- *stored_procedure_name* is the name of the stored procedure.

- *@param_variable1* is the name of the first parameter.

- *value1* is the value of the first parameter.

The value of @param_variable$n$ is the name of the nth parameter to be passed and value$n$ is the value of the nth parameter to be passed. The following is an example:

```
<sql>
  <query>
    exec PROC2(@l_name='REIS')
  </query>
</sql>
```

Stored Procedures for Use with Oracle

For Oracle stored procedures, a separate listener must be added to the Oracle service that selects the Stored Procedure Results Listener.

To add a stored procedure service:

1. Add a service by following the steps in "Adding a Service Adapter to an Application View" on page 3-7.

2. In the Add Service window, configure the parameters as shown in the following figure and accompanying table.

Figure 3-13   Add Stored Procedure Service Window

Table 3-3 Add Stored Procedure Parameter and Value Definitions

Parameter	Definition
Unique Service Name* (*Required)	This name must be unique to its application view. Valid characters include a-z, A-Z, 0-9, and the _ (underscore) character.
Select	Choose StoredProcedureResults.
userid* (*Required)	The RDBMS Application's user ID authorized to access the Oracle Applications system.

Table 3-3 Add Stored Procedure Parameter and Value Definitions (Continued)

Parameter	Definition
Password	A password associated with the specified user ID.
Driver* (*Required)	The name of the JDBC Driver. For example, the Oracle JDBC driver is `oracle.jdbc.driver.OracleDriver`.
URL* (*Required)	The address (URL) for the connection to the RDBMS. For example, an Oracle URL is `jdbc:oracle:thin:@Oracle.ibi.com:app`

format*
(*Required)
	Choose one of the following:
	
	■ **row.** The data that is produced is returned on a single line (per record) enclosed in `<row>` tags.
	
	■ **column.** The data produced is returned field by field, and each field is enclosed in `<column>` tags. The column tag has an attribute whose value is the name of the field; for example,
	
	`<row>`
	`<column name="ID">1000</column>`
	`<column name="First_Name">Scott</column>`
	`</row>`
	
	■ **field.** The data produced is returned field by field, and each field is enclosed in a tag that bears the field name; for example,
	
	`<row>`
	`<ID>1000></column>`
	`<FIRST_NAME>Scott</column>`
	`</row>`

Data_Source_Name	The Data Source JNDI name for the JDBC connection pool to use for connecting to the RDBMS system. If a value is present, the adapter will use the connection pool to connect to the RDBMS. If no value is specified, connection will use the Driver, URL, UserId and Password specified in the service. **Note:** For more information on setting up a Data Source, see "Defining a Data Source" on page 51.

Table 3-3 Add Stored Procedure Parameter and Value Definitions (Continued)

Parameter	Definition
DataSourceType	Choose one of the following: ■ **ConnectionPoolDataSource.** The connection name specified in the Data_Source_Name field is used as a JNDI context for a WebLogic Integration connection pool. ■ **XADatasource.** The connection name specified in the Data_Source_Name field is used as a JNDI context for an XADatasource whose transactions participate in the WebLogic Integration XA transaction. If the Data_Source_Name field is left blank, the adapter uses the user ID, password, and URL to establish a plain JDBC 2.0 connection with the database. If the Data Source_Name field is populated, the user ID, password, and URL are ignored.
schema	From the drop-down list, select the name of the schema that contains connection and other related information about the service you are adding.

Format for XML Request to Execute a Stored Procedure

The format for the XML request to execute a stored procedure for Oracle is as follows:

```
<sp>
    <proc>Stored_Procedure_Name</proc>
    <parm>parmvalue1</parm>
    <parm>parmvalue2</parm>
            .
            .
    <parm>parmvalueN</parm>
</sp>
```

The elements of the request are defined as follows:

■ *Stored_Procedure_Name* is the name of the PL/SQL stored procedure or function.

■ *parmvalueN* is the *N*th positional (in or in/out) parameter of the PL/SQL stored procedure or function.

Because all parameters are positional, they must be included in the XML Request. If a parameter is to be omitted, an empty XML value must be used; that is, `<parmvalue2></parmvalue2>`.

In/Out/In-Out parameters can be mixed positionally in the signature of the Oracle Stored Procedure code. When calling the procedure from the XML request only in and in-out parameters must be specified in the order in which they fall in the stored procedure code.

Format for XML Request to Execute Multiple Stored Procedures

The format for the XML request to execute multiple stored procedures for Oracle is as follows:

```
<connection>
  <sp>
    <proc> Stored_Procedure_Name</proc>
            <parm> parmvalue1</parm>
            <parm> parmvalue2</parm>
            .
            .
            .
            <parm>parmvalueN</parm>
  </sp>
</connection>
```

The elements of the request are defined as follows:

- *Stored_Procedure_Name* is the name of the PL/SQL stored procedure or function.

- *parmvalueN* is the *N*th positional (in or in/out) parameter of the PL/SQL stored procedure or function.

Format of the Output XML from a Stored Procedure

The format for the output XML from a stored procedure is as follows:

```
<response>
  <result format="std">
  <parameter>OutValue</parameter>
  <parameter name="parmvalue_name1">OutValue1</parameter>
  <parameter name="parmvalue_name2">OutValue2</parameter>
    .
    .
  <parameter name="parmvalue_nameN">OutValueN</parameter>
 </result>
</response>
```

The elements of output XML are as follows:

- parmvalue_name*N* is the name of the *N*th positional out parameter.

- *OutValue* is the returned result set or returned value of a stored procedure function. When this returned value is a scalar type, it is enclosed in `<parameter></parameter>` tags.

- OutValue*N* is the value or result associated with the *N*th positional out parameter of the PL/SQL stored procedure or function. If the out parameter is returning a scalar value, the value will be enclosed in `<parameter name="parmvaluex_name">val</parameter>` tags. If the out parameter being returned is a result set, it will be enclosed in `<resultset><colinfo><row>` tags.

Example 1: XML Request with One Input Parameter (String)

```
<sp>
        <proc>PROCINOUT</proc>
        <parm>Test Input Parm</parm>
</sp>
```

Example 2: XML Request with Three Input Parameters (Integer, String, Date)

```
<sp>
        <proc>RESULTSETTESTMULT</proc>
        <parm>100</parm>
        <parm>Test String</parm>
        <parm>2001-09-31 00:00:00</parm>
</sp>
```

Example 3: Response XML from Function with Return Value

```
<?xml version="1.0"?>
<eda>
  <response>
    <timestamp>2002-10-10T20:29:20Z</timestamp>
    <cncresult>
      <result format="field">
        <parameter name="RETURN">tested</parameter>
      </result>
    </cncresult>
    <execstatus>0</execstatus>
  </response>
</eda>
```

Example 4: Response XML from Stored Procedure with One Return Value and One Out Parameter

```
<?xml version="1.0"?>
<eda>
  <response>
    <timestamp>2002-10-10T20:27:23Z</timestamp>
    <cncresult>
      <result format="field">
        <parameter name="RETURN">returned</parameter>
        <parameter name="Y">tested</parameter>
      </result>
    </cncresult>
    <execstatus>0</execstatus>
```

```
            </response>
        </eda>
```

Example 5: Response from Request That Is Calling a Stored Procedure (Function) That Returns One Parameter and Has One Out Parameter

PL/SQL:

```
CREATE OR REPLACE function funcout (yParm out char)

return char is

begin yParm := 'tested';

return 'returned'; end;
```

XML Request:

```
<sp>
        <proc>FUNCOUT</proc>
</sp>
```

XML Response:

```
<response>
  <result format="std">
    <parameter>returned</parameter>
    <parameter name="YPARM">tested<parameter>
  </result>
</response>
```

Example 6: PL/SQL Stored Procedure That Has One Input Parameter and Returns 2 Result Sets Through Two Out Parameters

Note: For the following example returning a result set (cursor variable), an object type REF CURSOR must be declared. There are multiple ways of doing this. It can be done as an object type or declared in a package.

The following code snippet when executed will create a package with the appropriate type for the example:

```
CREATE OR REPLACE PACKAGE types
AS
    TYPE ref_cursor IS REF CURSOR;
END;
```

Stored procedure PL/SQL:

```
CREATE OR REPLACE procedure sp_get_stocks6(v_price IN NUMBER,
stock_cursor OUT types.ref_cursor, stock_cursor2 OUT
types.ref_cursor) IS

BEGIN
OPEN stock_cursor FOR SELECT ric,price,updated FROM stock_prices
WHERE price < v_price;

OPEN stock_cursor2 FOR SELECT ric,price,updated FROM stock_prices
WHERE price > v_price/2;

END;
```

XML Request:

```
<sp>
        <proc>SP_GET_STOCKS6</proc>
         <parm>25</parm>
</sp>


XML Response Output:

<?xml version="1.0"?>
<response>
  <result format="std">
    <resultset>
      <colinfo>
        <col length="6"
             offset="0"
             type="12">RIC</col>
        <col length="21"
             nullable="1"
             offset="6"
             type="2">PRICE</col>
        <col length="7"
             nullable="1"
             offset="27"
             type="93">UPDATED</col>
      </colinfo>
```

```
                 <row>ttt   3             2002-08-20 00:00:00.0</row>
                 <row>l     3             2002-08-08 00:00:00.0</row>
                 <row>yyyy  0             2001-10-01 14:15:11.0</row>
             </resultset>
             <resultset>
               <colinfo>
                 <col length="6"
                      offset="0"
                      type="12">RIC</col>
                 <col length="21"
                      nullable="1"
                      offset="6"
                      type="2">PRICE</col>
                 <col length="7"
                      nullable="1"
                      offset="27"
                      type="93">UPDATED</col>
               </colinfo>
               <row>TEST  40            2002-07-19 00:00:00.0</row>
               <row>ATTT  30            2002-03-04 00:00:00.0</row>
               <row>PPOO  30            2002-03-03 00:00:00.0</row>
             </resultset>
           </result>
         </response>
```

Execution of Stored Procedure Within a Test Service Window

The following figure shows a stored procedure within a test service window:

Figure 3-14   Execution of Stored Procedure Within a Test Service Window
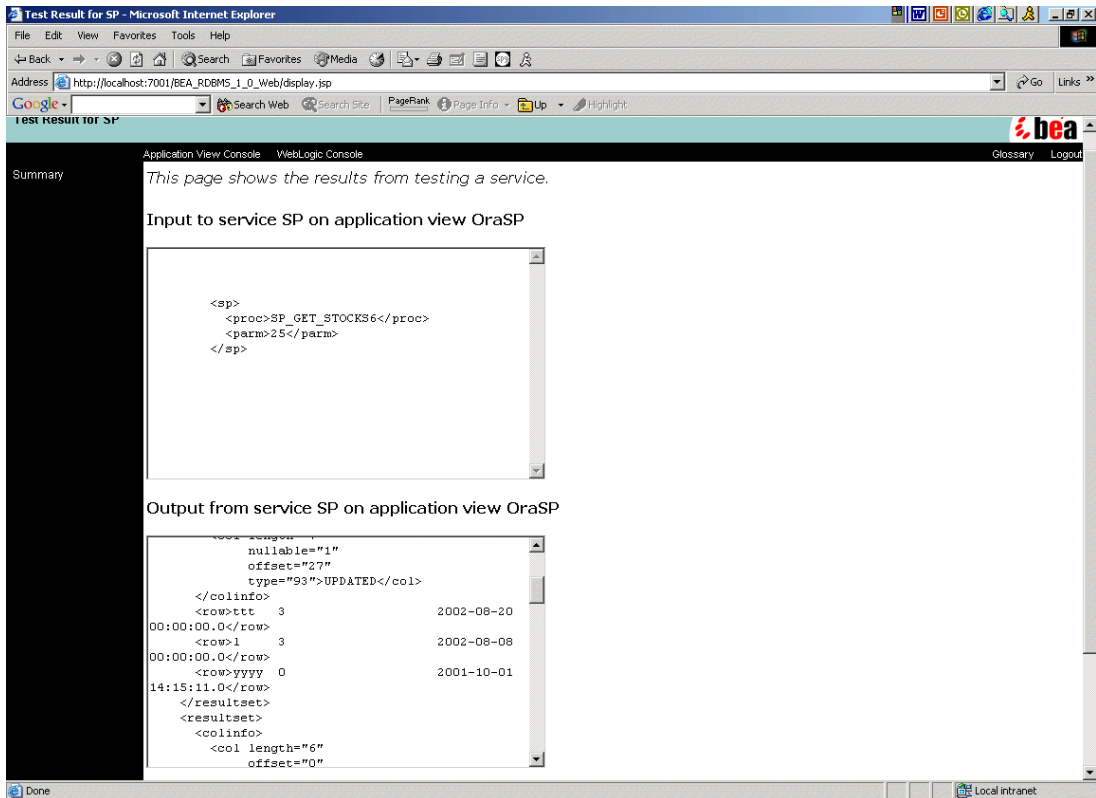
The following are the data types supported for parameters in Oracle stored procedures.

Table 3-4 Supported Data Types for Oracle Stored Procedure Parameters

Data Type	Supported Types
All Scalar Types	BINARY INTEGER DEC DECIMAL DOUBLE PRECISION FLOAT INT INTEGER NATURAL NATURALN1 NUMBER NUMERIC PLS INTEGER POSITIVE POSITIVEN REAL SIGNTYPE SMALLINT CHAR CHARACTER LONG NCHAR NVARCHAR2 STRING VARCHAR2 VARCHAR BOOLEAN DATE
Data Types for Output Parameters	All scalar types (as listed above) Result sets (REF_CURSOR) **Note:** Result sets that are defined in PL/SQL type definitions are not supported at this time.
Composite Type Structures	This data type is not supported at this time.

Working with Parameterized SQL

The Parameterized SQL option is designed to receive an XML structure and place specific values inside a predefined parameterized SQL statement. XML structure can become quite complex and can contain identical XML tag names under different nodes. To more easily identify the proper node "starting" point for parameter insertion, a second parameter, called Target_Nodes, is provided.

Figure 3-15   Add Parameterized SQL Service

Table 3-5 Add Parameterized SQL Parameters

Parameter	Definition
Unique Service Name * (*Required)	This name must be unique to its application view. Valid characters include a-z, A-Z, 0-9, and the _ (underscore) character.
Select	Select Parameterized SQL from the drop-down list.

format* (*Required)	Choose one of the following:
	■ **row.** The data that is produced is returned on a single line (per record) enclosed in `<row>` tags.
	■ **column.** The data produced is returned field by field, and each field is enclosed in `<column>` tags. The column tag has an attribute whose value is the name of the field; for example,
	`<row>`
	`<column name="ID">1000</column>`
	`<column name="First_Name">Scott</column>`
	`</row>`
	■ **field.** The data produced is returned field by field, and each field is enclosed in a tag that bears the field name; for example,
	`<row>`
	`<ID>1000></column>`
	`<FIRST_NAME>Scott</column>`
	`</row>`

Propname	Name of Properties Group (default is "NULL")

Table 3-5 Add Parameterized SQL Parameters (Continued)

Parameter	Definition
Isolation Level* (*Required) **Note:** Before setting the isolation level for a service, check with your database administrator.	Choose one of the following: ■ **asis.** Does not specifically set an isolation level for the connection. ■ **readUncommitted.** Does not prevent any read violation. ■ **readCommitted.** Only data that has been committed by a transaction can be read by other transactions. This level prohibits a transaction from reading a row with uncommitted changes in it. This setting prevents a dirty read, but allows non-repeatable reads and phantom reads. ■ **repeatableRead.** Only data that has been committed by a transaction can be read by other transactions, and multiple reads yield the same result as long as the data has not been committed. This setting prevents dirty reads and non-repeatable reads, but not phantom reads. ■ **Serializable.** This setting is the highest isolation level, stipulating that all transactions run serially to achieve maximum data integrity. This yields the slowest performance and least concurrency. This setting prevents Dirty Reads, Non-repeatable reads, and Phantom reads. **Note:** For definitions of Dirty Read, Phantom Read, and Non-repeatable read, and for more information on isolation levels, see "Transaction Isolation Levels" on page 3-12.
URL* (*Required)	The address (URL) for the connection to the RDBMS. For example, an Oracle URL is `jdbc:oracle:thin:@Oracle.ibi.com:app`
user* (*Required)	The RDBMS Application's user ID authorized to access the Oracle Applications system.
Password	A password associated with the specified user ID.
DRIVER* (*Required)	The name of the JDBC Driver. For example, the Oracle JDBC driver is `oracle.jdbc.driver.OracleDriver`.
Data Source Name	The Data Source JNDI name for the JDBC connection pool to use for connecting to the RDBMS system. If a value is present, the adapter will use the connection pool to connect to the RDBMS. If no value is specified, connection will use the Driver, URL, UserId and Password specified in the service. **Note:** For more information on setting up a Data Source, see "Defining a Data Source" on page 51.

Table 3-5 Add Parameterized SQL Parameters (Continued)

Parameter	Definition
DataSourceType	Choose one of the following: ■ **ConnectionPoolDataSource.** The connection name specified in the Data_Source_Name field is used as a JNDI context for a WebLogic Integration connection pool. ■ **XADatasource.** The connection name specified in the Data_Source_Name field is used as a JNDI context for an XADatasource whose transactions participate in the WebLogic Integration XA transaction. If the Data_Source_Name field is left blank, the adapter uses the user ID, password, and URL to establish a plain JDBC 2.0 connection with the database. If the Data Source_Name field is populated, the user ID, password, and URL are ignored.
Schema	From the drop-down list, select the name of the schema that contains connection and other related information about the service you are adding.

Adding an Event Adapter to an Application View

The event adapter supports the polling of relational tables for incoming data inserted by any process, trigger, or method. It captures any incoming data (to specified tables) and performs operations based on the contents of the rows. This highly configurable event adapter, when triggered, reads one or more rows from the table and creates an XML document representing the column data in each row. Standard business logic facilities are then applied to the constructed XML documents, which include transformation, validation, security management, and application processing. Each row in the table is deleted (optional) or updated if the business logic has properly completed.

Note: The event adapter requires the JDBC technology-based driver for the database being monitored. Please contact your DBMS vendor to obtain the appropriate JDBC driver.

After you create and configure an application view, you can add the event adapter. For information on creating an application view, see "Defining a New Application View" on page 3-1.

1. While the Application View Console is open, click Administration. The Application View Administration window displays, as shown in the following figure:

Figure 3-16   Application View Administration Window - Event Adapter
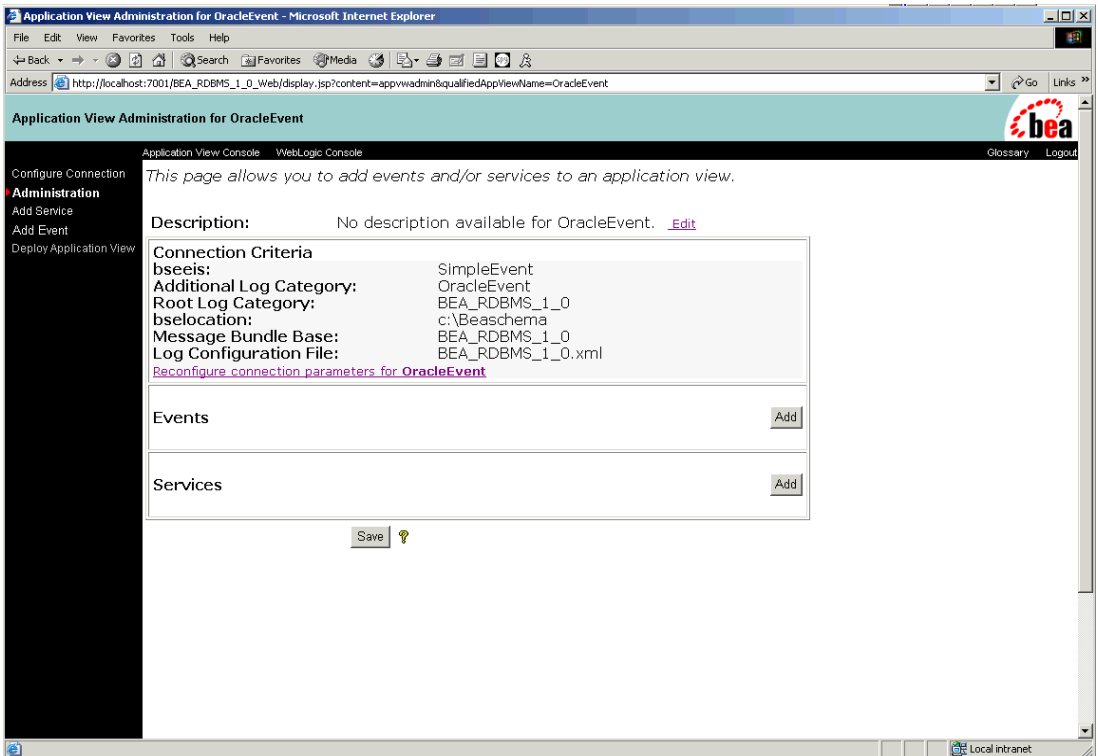
2. Click Add in the Events row.

    The Add Event window displays.

Figure 3-17   Add Event Window

Add Event	

Application View Console   WebLogic Console

Configure Connection
Administration
Add Service
Add Event
Deploy Application View

On this page, you add events to your application view.

Unique Event Name: * [   ]

RDBMS

encoding	ISO-8859-1
Driver*	oracle.jdbc.driver.OracleDriver
url*	jdbc:oracle:thin:@Oracle11i:1523:vis
User Name	USER001
Password	*****************************
Format*	field ▼
Maximum Rows	1
SQL Post Query	
Delete Keys	
Polling Interval	20
Data Source Name	

schema: sql0000_EventTest ▼

[Add]

3. Enter the parameter information required to poll on an RDBMS table:

Table 3-6 RDBMS Add Event Parameters

Parameter	Description
Unique Event Name* (*Required)	This name must be unique to its application view. Valid characters include a-z, A-Z, 0-9, and the _ (underscore) character.
encoding	Enter the following value: `ISO-8859-1`.
Driver* (*Required)	Vendor-specific JDBC driver for access to the database. This parameter requires a fully-qualified name.
url* (*Required)	A database URL (or JDBC URL) is a platform-independent way of addressing a database. A database/JDBC URL has the following form: `jdbc:[subprotocol]:[node]/[databaseName]`
User Name	Valid user name for access to the database.
Password	Valid password associated with the user name for access to the database.

Format *
(*Required)
	Choose one of the following:
	
	■ **column.** The data produced is returned field by field, and each field is enclosed in <column> tags. The column tag has an attribute whose value is the name of the field; for example,
	
	`<row>`
	`<column name="ID">1000</column>`
	`<column name="First_Name">Scott</column>`
	`</row>`
	
	■ **field.** The data produced is returned field by field, and each field is enclosed in a tag that bears the field name; for example,
	
	`<row>`
	`<ID>1000></column>`
	`<FIRST_NAME>Scott</column>`
	`</row>`

Table 3-6 RDBMS Add Event Parameters (Continued)

Parameter	Description
Maximum Rows	Number of data rows to be retrieved from the database table in a single operation. For example, if five were specified, then up to five rows are read and processed in a single operation. In most circumstances, you should not allow this parameter to exceed the number of parallel threads available for execution.
	The number of events created in a single polling interval is dependent on the setting for Maximum Rows, and the number of new rows added to the database since the last time the database was polled. For example, if Maximum Rows is set to 5, and 23 new rows are found to have been added when the database is polled, four events containing five rows, and one event containing three rows are created.
SQL Post-Query	SQL Query that is executed after the initial query request.
	If this parameter is not configured, the following command is executed:
	`DELETE field1,field2... from table_name`
	This parameter should not be configured if the RDBMS event adapter exit is configured.
	Two types of operators are available: `?fieldname` and `^fieldname`.
	■ The `?fieldname` will evaluate at run time to `?fieldname= value`.
	■ The `^fieldname` will evaluate at run time to value.
	A SQL Post query using the `?` can be used in an update statement as follows: update `tablename` where `?fieldname`.
	For example, `update stock_prices_temp where ?RIC`.
	A SQL Post Query using the `^` can be used in an insert statement as follows:
	Insert into `tablename values` (`^fieldname1, ^fieldname2, ^fieldname3`).
	For example, `Insert into stock_prices_temp values (^RIC, ^PRICE, ^UPDATED)`.
Delete Keys	Comma separated list of keys used in the DELETE statement. A delete operates on keys, so you should enter the table's key columns in this parameter.
Polling Interval	Interval in seconds at which the database is monitored for new rows. If this parameter is not configured, the default value is two seconds.

Table 3-6 RDBMS Add Event Parameters (Continued)

Parameter	Description
Data_Source_Name	The Data Source JNDI name for the JDBC connection pool to use for connecting to the RDBMS system. If a value is present, the adapter will use the connection pool to connect to the RDBMS. If no value is specified, connection will use the Driver, URL, UserId and Password specified in the service. **Note:** For more information on setting up a data source, see "Defining a Data Source" on page 3-51.
Schema	From the drop-down list, select the name of the schema that contains connection and other related information about the event you are adding.

Note: All events and related SQL must be entered through the BEA Application Explorer. For more information, see Chapter 2, "Using the BEA Application Explorer With an RDBMS."

Note that you specified the parameterized SQL statement when you generated the event schema; for more information about generating event schemas, see Chapter 2, "Using the BEA Application Explorer With an RDBMS."

4. Click Add to proceed with the adding event process.

The Application View Administration window is displayed:

Figure 3-18   Application View Administration Window - Event Adapter
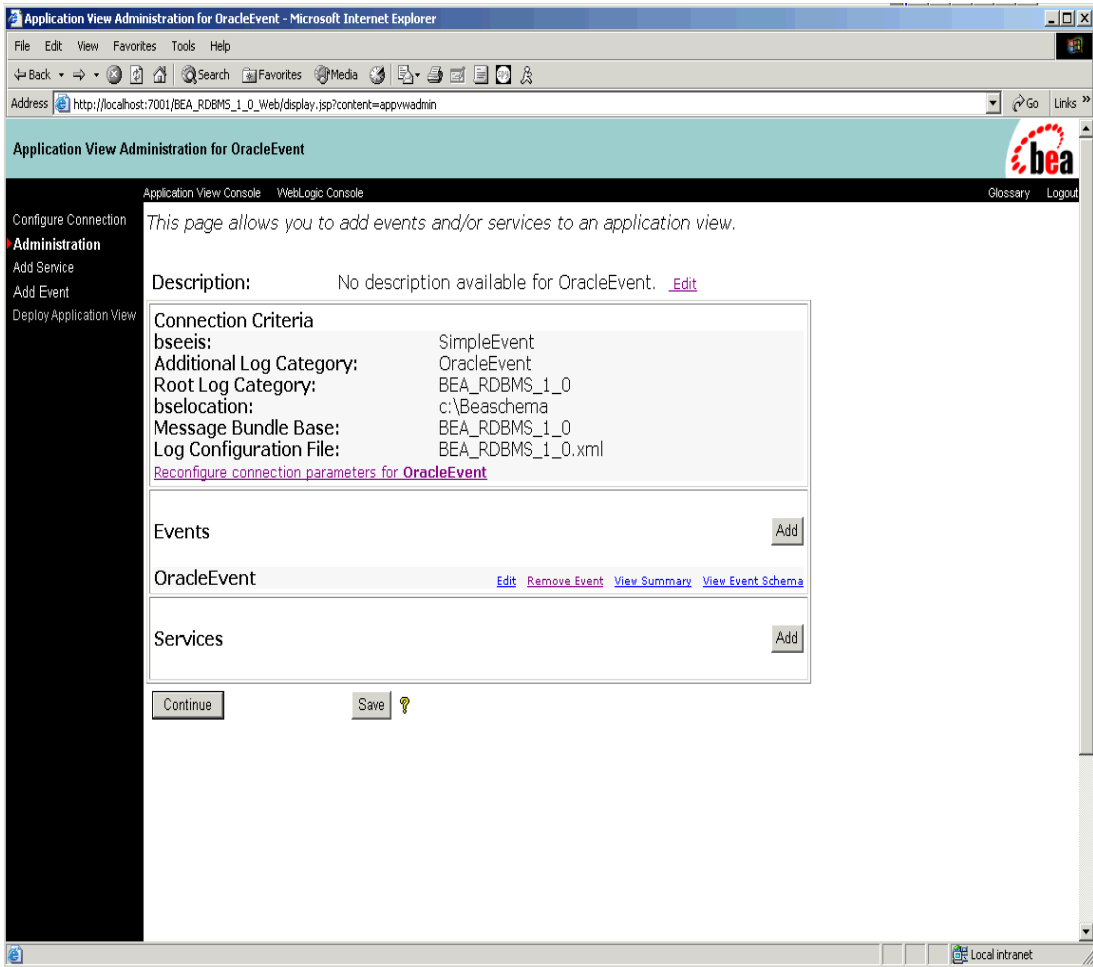
5. Click Continue.

Figure 3-19   Deploy Application View Oracle Events to Servers Window

6. Click Deploy to deploy the application view.

   For more information on deploying and testing application views, see
   "Deploying an Application View" on page 3-15.

Figure 3-20   Event Summary Window
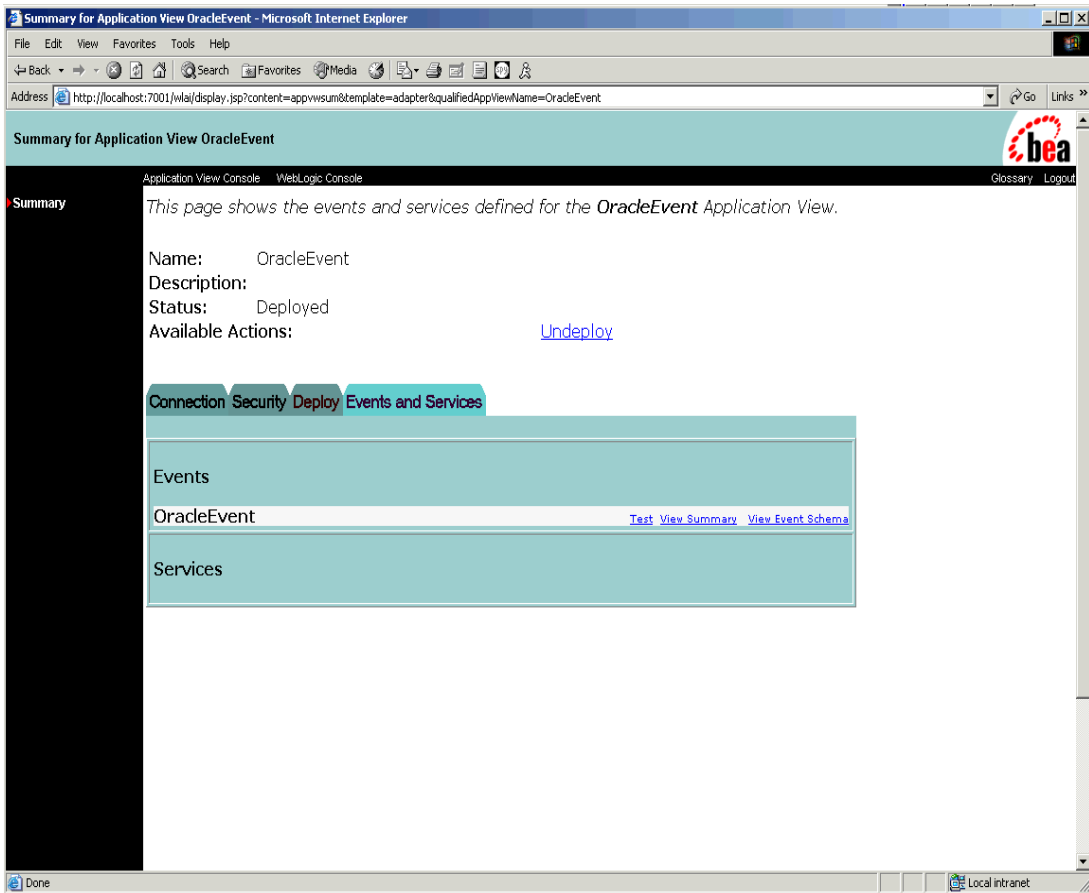
7. After you create and deploy an application view that contains events, test the application view event. Testing evaluates whether the application view event interacts properly with the RDBMS system.

8. To test an application view, find the event in the Current Events area, and click Test for that event.

Figure 3-21   Test Event Window
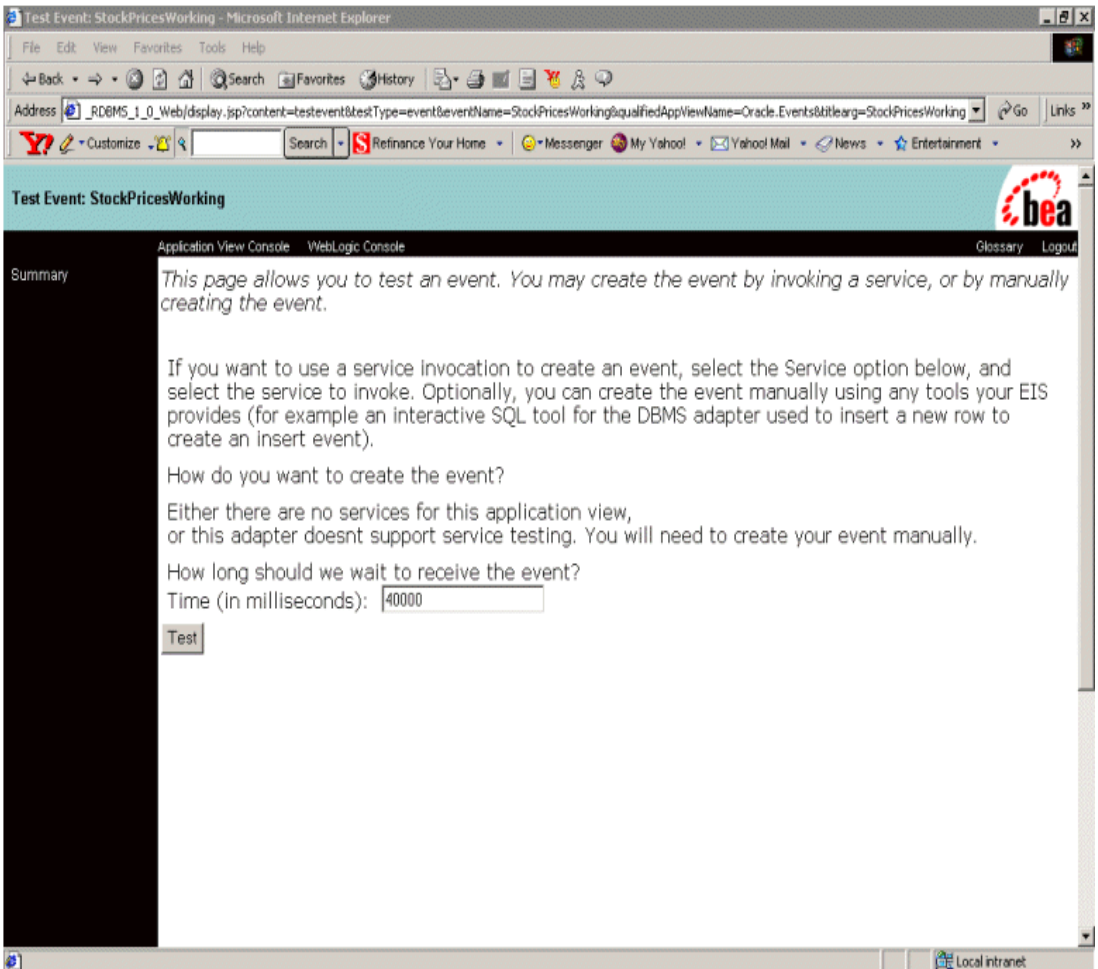
9. Enter a time, in milliseconds, for the test to wait for an incoming event. When you are ready to initiate the event, click the Test button.

Figure 3-22   Test Result Window
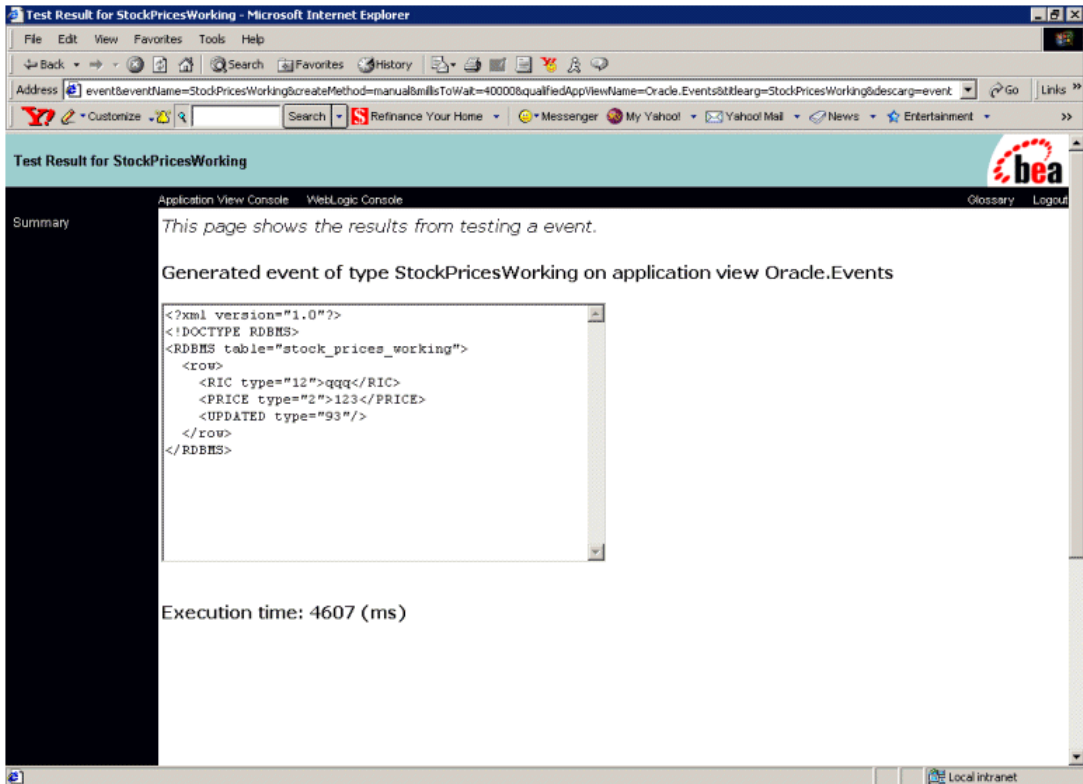
If the test is successful, the Test Result window displays the input XML and the result set. This confirms that the application view event is successfully deployed. You can now employ the event in business process workflows or write custom code. For more information, see "Using Application Views in the Studio" in *Using Application Integration*:

- For WebLogic Integration 7.0, see
  `http://edocs.bea.com/wli/docs70/aiuser/3usruse.htm`

- For WebLogic Integration 2.1, see
  `http://edocs.bea.com/wlintegration/v2_1sp/aiuser/3usruse.htm`

Note: If the test fails, the Test Result window displays a timed out message.

Handling Null Values

Relational databases support the notion of null values in a data field. On the event side, it is sometimes important to ascertain whether a field contains a null value, or is simply an empty string. On the service side, it is important to properly set null values when performing an insert or update.

Null Values in Events

For events, null values in the data being read are denoted in the result document by the attribute `null='y'`. The following table depicts the behavior of the BEA WebLogic Adapter for RDBMS in handling null values for events.

Table 3-7 Handling Null Values for Events

Database Value	Database Nullable	XML Value	XML Attribute
Spaces	Yes	Spaces	
Spaces	No	Spaces	
Null	Yes	None	`null='y'`
Null	No	N/A	N/A
Empty	Yes	None	
Empty	No	None	
Text	Yes	Text	
Text	No	Text	

The attribute "null" is used to indicate fields containing a null value. For the Field and Column formats, the null attribute is set to 'y' in the case of a nullable field containing a null value. The following is an example of XML, in the Field format, containing fields representing the valid combinations from the above table.

Listing 3-1   XML Generated by an Event

```
<RDBMS table="testnulls">
  <row>
    <SPACESNull type="1">  </SPACESNull>       spaces/nullable
    <SPACESNNull type="1">  </SPACESNNull>      spaces/not nullable
    <NULLFIELD type="1" null="y"/>              null/nullable
    <EmptyFieldNull type="1"/>                  empty/nullable
    <EmptyFieldNotNull type="1"/>               empty/not nullable
    <TextNull type="1">Text </TextNull>         text/nullable
    <TextNNull type="1">Text </TextNull>        text/not nullable
  </row>
</RDBMS>
```

The following is an example of XML, in the Column format, containing fields representing the valid combinations from the above table.

Listing 3-2   Column-Formatted XML Generated by an Event

```
<RDBMS table="testnulls">
  <row>
    <col name"SPACESNull" type="1">  </SPACESNull>       spaces/nullable
    <col name"SPACESNNull" type="1">  </SPACESNNull>      spaces/not nullable
    <col name"NULLFIELD" type="1" null="y"/>              null/nullable
    <col name"EmptyFieldNull" type="1"/>                  empty/nullable
    <col name"EmptyFieldNotNull" type="1"/>               empty/not nullable
    <col name"TextNull" type="1">Text </TextNull>         text/nullable
    <col name"TextNNull" type="1">Text </TextNull>        text/not nullable
  </row>
</RDBMS>
```

When the RDBMS event is set to produce row formatted event documents, the null attribute, on the <row> node, uses positional 0 and 1 values to designate the presence of a null. The following is an example of XML, in the row format, containing three fields, where two of the fields contain a null value. The colinfo node of the XML contains metadata about the individual columns. The nullable='1' indicates that the column is nullable. If the value was 0, then the column could not be null. In this example, the ADDRESS and CITY columns are nullable and contain a null value.

Listing 3-3   Columns Containing Null Values

```
<RDBMS table="aaa">
      <colinfo>
              <col length="35"
                     nullable="1"
                     offset="0"
                     type="12">NAME</col>
              <col length="35"
                     nullable="1"
                     offset="35"
                     type="12">ADDRESS</col>
              <col length="23"
                     nullable="1"
                     offset="70"
                     type="12">CITY</col>
      </colinfo>
      <row nulls="011">a
</RDBMS>
```

Null Values in Services

For services, you need to properly designate null values in order to insert those values into the RDBMS. The BEA WebLogic Adapter for RDBMS propagates null values into SQL under the following code.

```
<ttable>
   <NField/>
</ttable>
```

Here, the application view has a service that is defined with Target_Nodes set to

/ttable/

and SQL set to

INSERT INTO ANOTHER_TABLE VALUES('?ttable')

The resulting SQL sent to the RDBMS is:

INSERT INTO ANOTHER_TABLE VALUES(NULL)

For more information about using parameterized SQL with null values, see "Working with Parameterized SQL."

Defining a Data Source

WebLogic Server supports the establishment of connection pools and data sources. The connection pool contains named groups of JDBC connections that are created when the connection pool is registered, usually when starting up WebLogic Server. The BEA WebLogic Adapter for RDBMS borrows a connection from the pool, uses it, and then returns it to the pool by closing it. A data source object enables JDBC clients to obtain a DBMS connection. Each data source object points to a connection pool. The adapter uses the data source JNDI name to locate the appropriate connection pool.

Creating a Connection Pool

To create a connection pool, perform the following steps:

1. Navigate your browser to the WebLogic Server Administration Console. The WebLogic Server Administration Console can be found at the following URL: `http://`*`host:port`*`/console`.

   Here, *`host`* is the IP address or DNS name of the machine on which WebLogic Server is running, and *`port`* is the socket on which the server is listening.

   The WebLogic Server Administration window displays.

2. In the left pane, choose Services and then JDBC and then Connection Pools from the navigation tree.

   The console displays the JDBC Connection Pools window.

Figure 3-23   Connection Pools Window
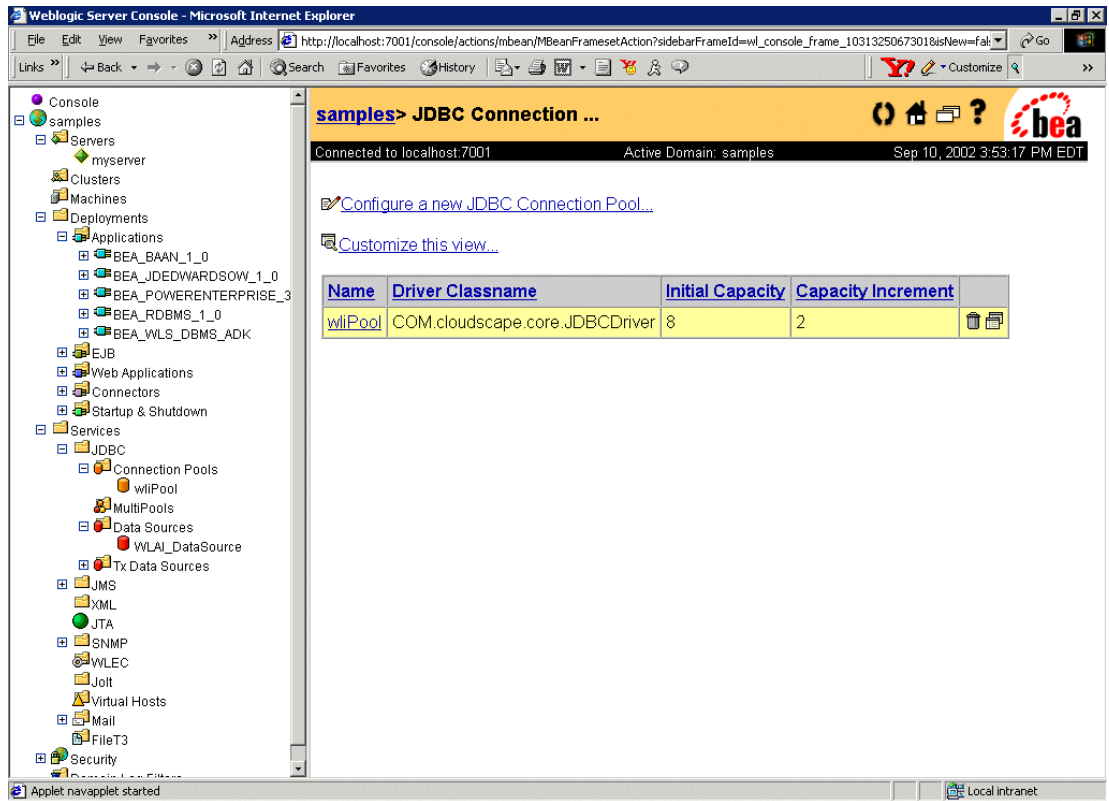
3. To create a new connection pool, click the Configure a new JDBC Connection Pool link.

The Create a new JDBCConnectionPool window opens.

Figure 3-24   Create a new JDBC Connection Pool Window
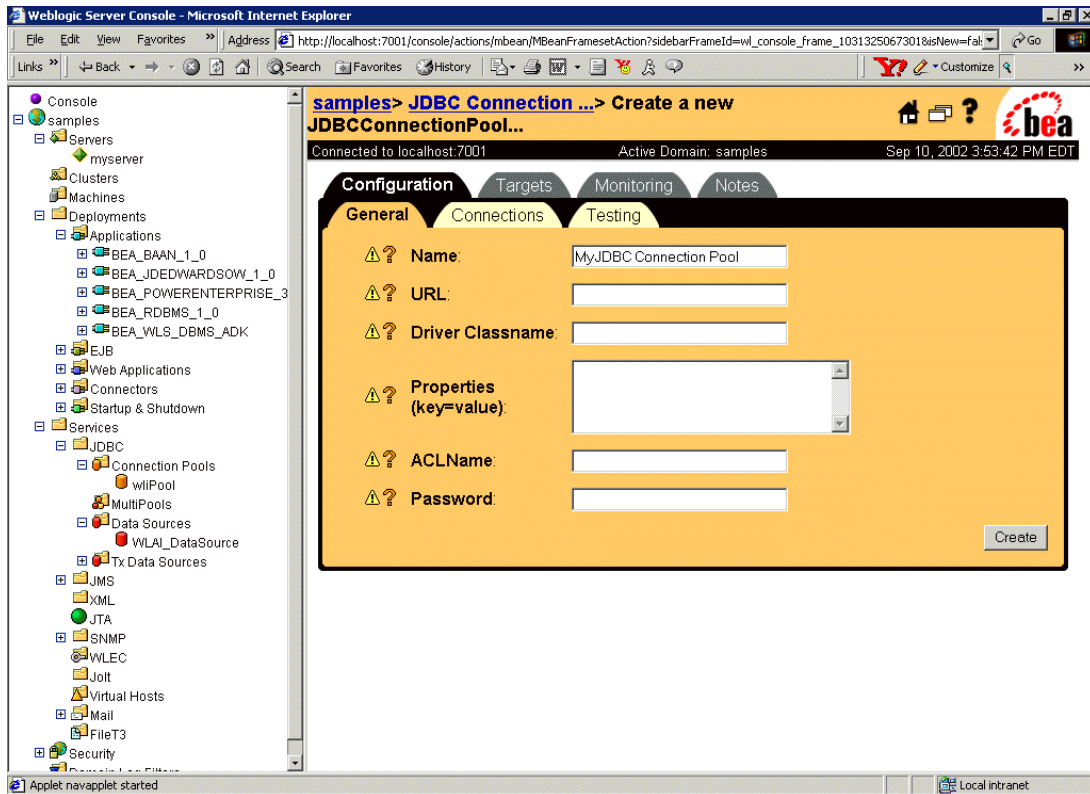
4. Enter the appropriate information into the JDBC connection pool fields.

5. Click Create.

The following figure shows a connection to an Oracle database.

Figure 3-25   JDBC Connection Pool Window with Data
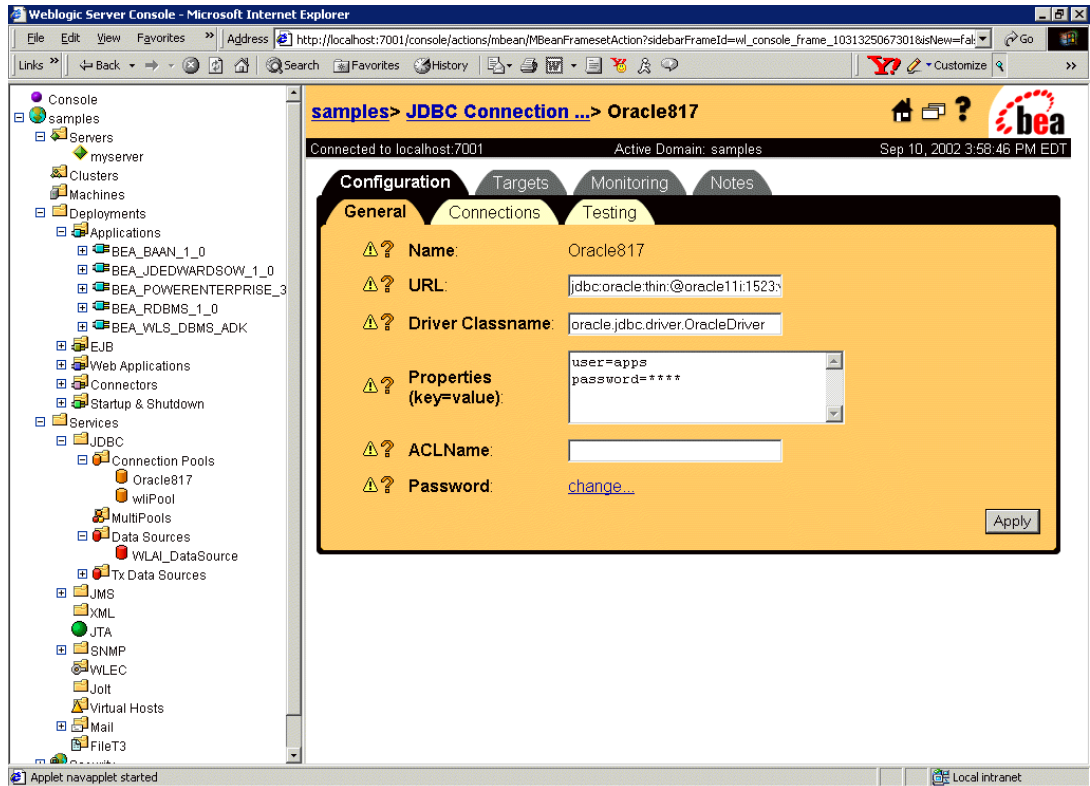
Creating a Data Source

To create a new data source, perform the following steps:

1. Navigate your browser to the WebLogic Server Administration Console. The WebLogic Server Administration Console can be found at the following URL: `http://host:port/console`.

   Here, `host` is the IP address or DNS name of the machine on which WebLogic Server is running, and `port` is the socket on which the server is listening.

   The WebLogic Server Administration window displays.

2. In the left pane, choose Services and then JDBC and then Data Sources from the navigation tree.

    The console displays the JDBC Data Sources window.

Figure 3-26   JDBC Data Sources Window
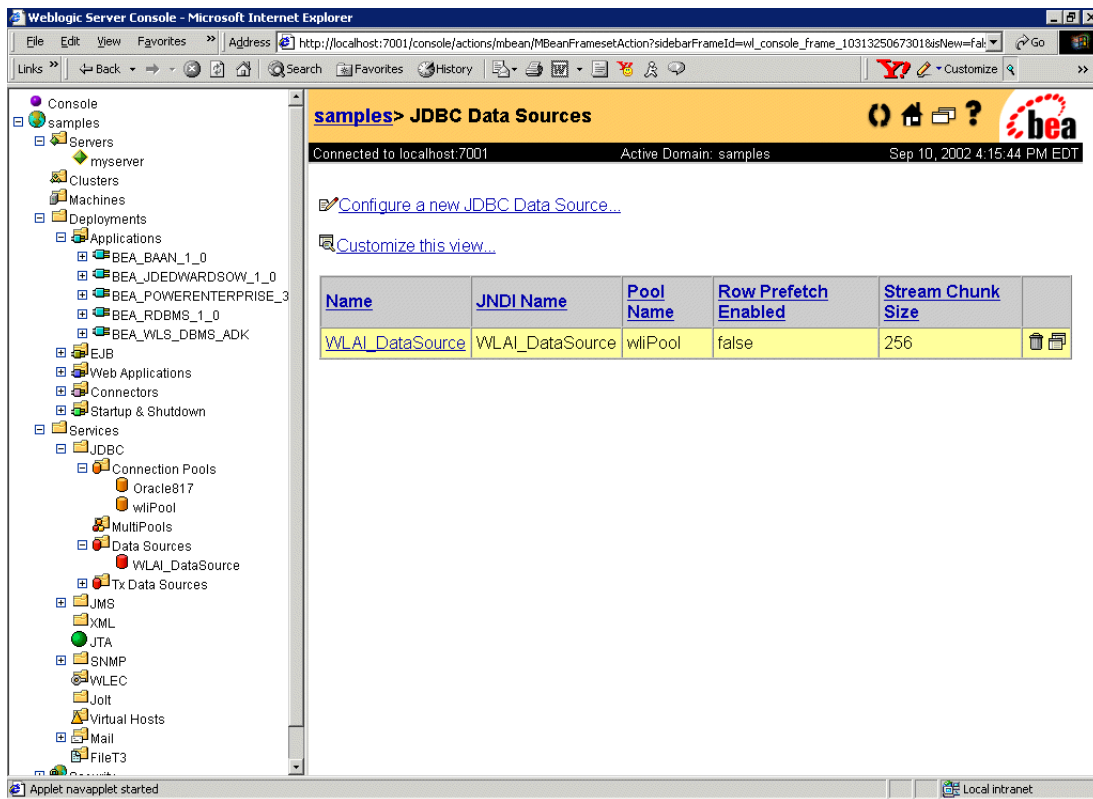
3. Click the Configure a new JDBC Data Source link.

The Create a new JDBCDataSource window opens.

Figure 3-27   Create a new JDBC Data Source Window
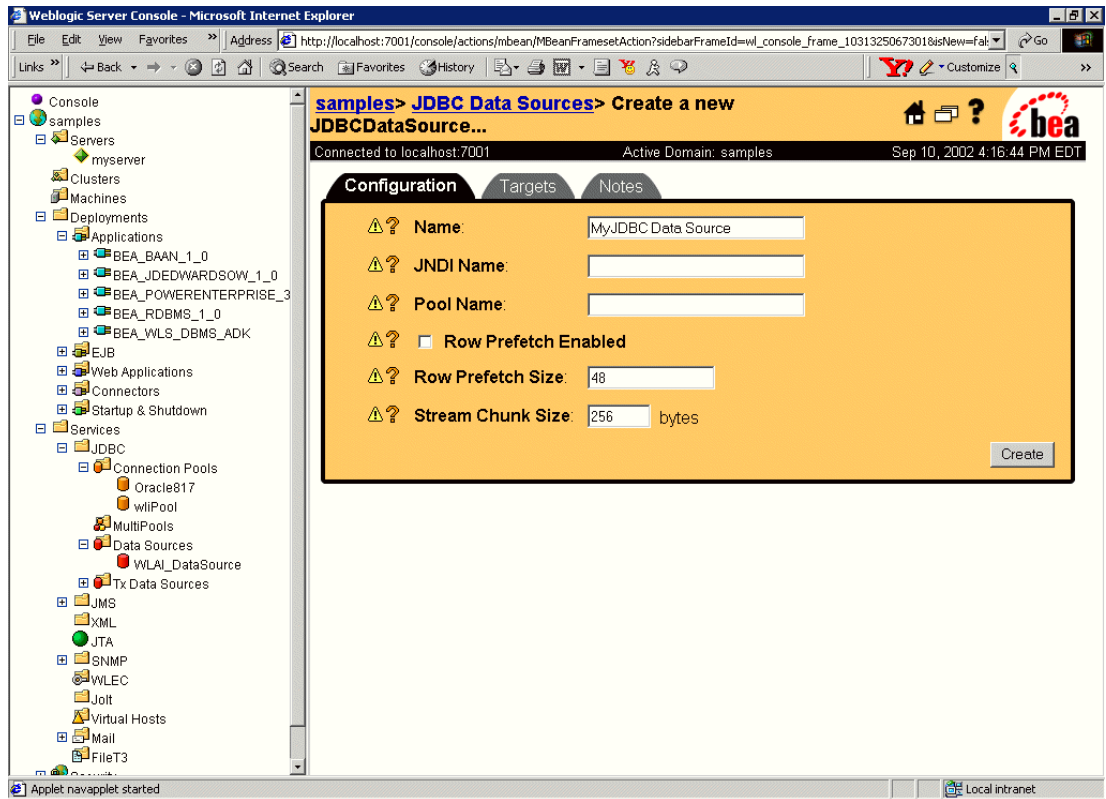
4. Enter the appropriate information into the JDBC data source fields.

5. Click Create.

The following figure shows a data source using the connection pool created in the previous example.

Figure 3-28   JDBC Data Source Window with Data
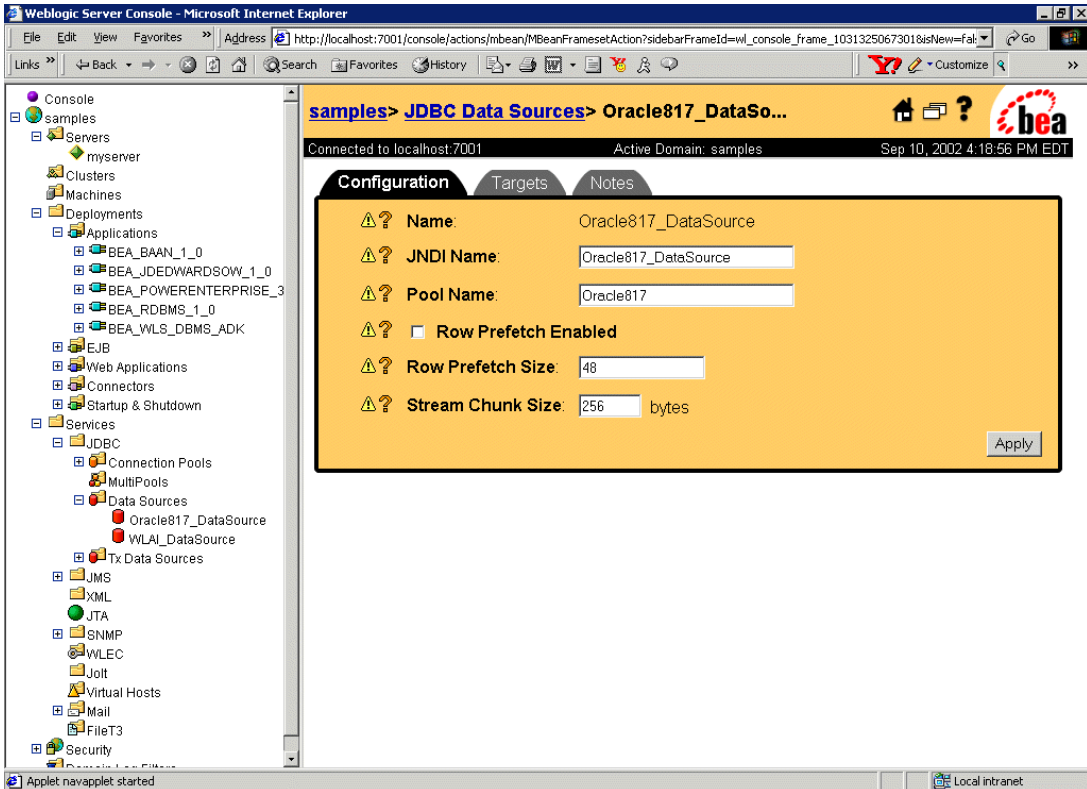
Once created, the Data Source JNDI Name can be specified in Application View Service. For more information on connection pools and data sources, see "JDBC Components-Connection Pools, Data Sources, and MultiPools," in "Managing JDBC Connectivity" in the WebLogic Server *Administration Guide*:

- For WebLogic Server 7.0, see
  `http://edocs.bea.com/wls/docs70/adminguide/jdbc.html`

- For WebLogic Server 6.1, see
  `http://edocs.bea.com/wls/docs61/adminguide/jdbc.html`

4 Service XML Examples

This section provides service samples that use the service adapter and includes the following examples:

- XML Schemas

- Select Statement

- Simple Insert Statement

- Delete Statement

- Multi-Select Statements

- Update Statement

- Stored Procedure

- Including Multiple SQL Statements in an XML Request

XML Schemas

These examples use the service adapter and are shown with business process management functionality.

The input XML schema for these examples is:

Listing 4-1   Input XML Schema

```
<?xml version="1.0" encoding="utf-16"?>
<xs:schema id="NewDataSet" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="sql">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="query" type="xs:string" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="NewDataSet" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="sql" />
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The output XML schema for these examples is:

Listing 4-2   Output XML Schema

```
<?xml version="1.0" encoding="utf-16"?>
<xs:schema id="eda" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="eda" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
```

```
<xs:element name="response">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="timestamp" type="xs:string" minOccurs="0" />
      <xs:element name="execstatus" type="xs:string" minOccurs="0" />
      <xs:element name="cncresult" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
          <xs:element name="result" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="resultset" minOccurs="0"
maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="colinfo" minOccurs="0"
maxOccurs="unbounded">
                        <xs:complexType>
                          <xs:sequence>
                            <xs:element name="col" nillable="true"
minOccurs="0" maxOccurs="unbounded">
                              <xs:complexType>
                          <xs:simpleContent msdata:ColumnName="col_Text"
msdata:Ordinal="3">
                                  <xs:extension base="xs:string">
                                    <xs:attribute name="length"
type="xs:string" />
                                    <xs:attribute name="offset"
type="xs:string" />
                              <xs:attribute name="type" type="xs:string" />
                                    <xs:attribute name="nullable"
type="xs:string" />
                                  </xs:extension>
                                </xs:simpleContent>
                              </xs:complexType>
                            </xs:element>
                          </xs:sequence>
                        </xs:complexType>
                      </xs:element>
                      <xs:element name="row" nillable="true" minOccurs="0"
maxOccurs="unbounded">
                        <xs:complexType>
                         <xs:simpleContent msdata:ColumnName="row_Text"
msdata:Ordinal="0">
                            <xs:extension base="xs:string">
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
```

```
                                 </xs:sequence>
                               </xs:complexType>
                             </xs:element>
                           </xs:sequence>
                           <xs:attribute name="format" type="xs:string" />
                         </xs:complexType>
                       </xs:element>
                     </xs:sequence>
                   </xs:complexType>
                 </xs:element>
               </xs:sequence>
             </xs:complexType>
           </xs:element>
         </xs:choice>
       </xs:complexType>
     </xs:element>
</xs:schema>
```

Select Statement

The following example shows a Select statement:

```
Select * from Stock_prices;
```

Figure 4-1   Input as Seen in WebLogic Integration Studio Window

Figure 4-2   Generated Output


```
View XML                                                          ×

<?xml version="1.0" encoding="UTF-8"?>
<eda>
  <response>
    <timestamp>2002-07-16T24:51:33Z</timestamp>
    <cncresult>
      <result format="std">
        <resultset>
          <colinfo>
            <col length="6" offset="0" type="12">RIC</col>
            <col length="21" nullable="1" offset="6" type="2">PRICE</col>
            <col length="7" nullable="1" offset="27" type="93">UPDATED</col>
          </colinfo>
          <row>IBM    44                2002-07-09 00:00:00.0</row>
          <row>AMZN   10                2002-07-09 00:00:00.0</row>
          <row>BBB    10                2002-07-05 00:00:00.0</row>
          <row>T      100               2002-07-11 00:00:00.0</row>
        </resultset></result>
    </cncresult>
    <execstatus>13</execstatus>
  </response></eda>

                            Close
```
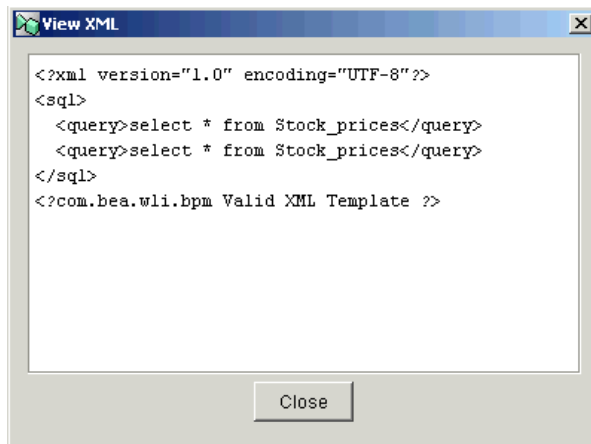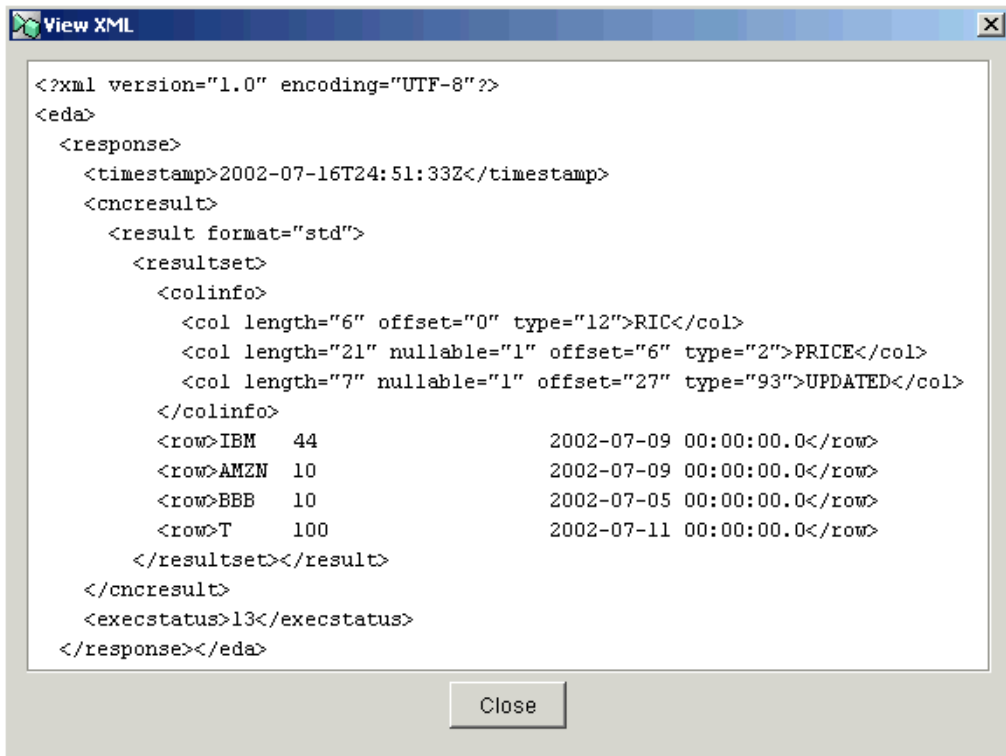
Simple Insert Statement

The following example shows a simple Insert statement:

```
Insert into stock_prices values('UPS',30,'16-JUL-2002')
```

Figure 4-3   Input as Seen in WebLogic Integration Studio Window

```
View XML

<?xml version="1.0" encoding="UTF-8"?>
<sql>
  <query>insert into stock_prices values(&apos;UPS&apos;,30,&apos;16-JUL-2002&apos;)</query>
</sql>

                                    Close
```

Figure 4-4   Generated Output - WebLogic Integration Studio Window

```
View XML

<?xml version="1.0" encoding="UTF-8"?>
<eda>
  <response>
    <timestamp>2002-07-16T24:57:02Z</timestamp>
    <cncresult>
      <result format="std">
        <resultset>
          <colinfo>
            <col length="6" offset="0" type="12">RIC</col>
            <col length="21" nullable="1" offset="6" type="2">PRICE</col>
            <col length="7" nullable="1" offset="27" type="93">UPDATED</col>
          </colinfo>
          <row>IBM    44               2002-07-09 00:00:00.0</row>
          <row>AMZN   10               2002-07-09 00:00:00.0</row>
          <row>BBB    10               2002-07-05 00:00:00.0</row>
          <row>T      100              2002-07-11 00:00:00.0</row>
          <row>UPS    30               2002-07-16 00:00:00.0</row>
        </resultset></result>
    </cncresult>
    <execstatus>14</execstatus>
  </response></eda>
```
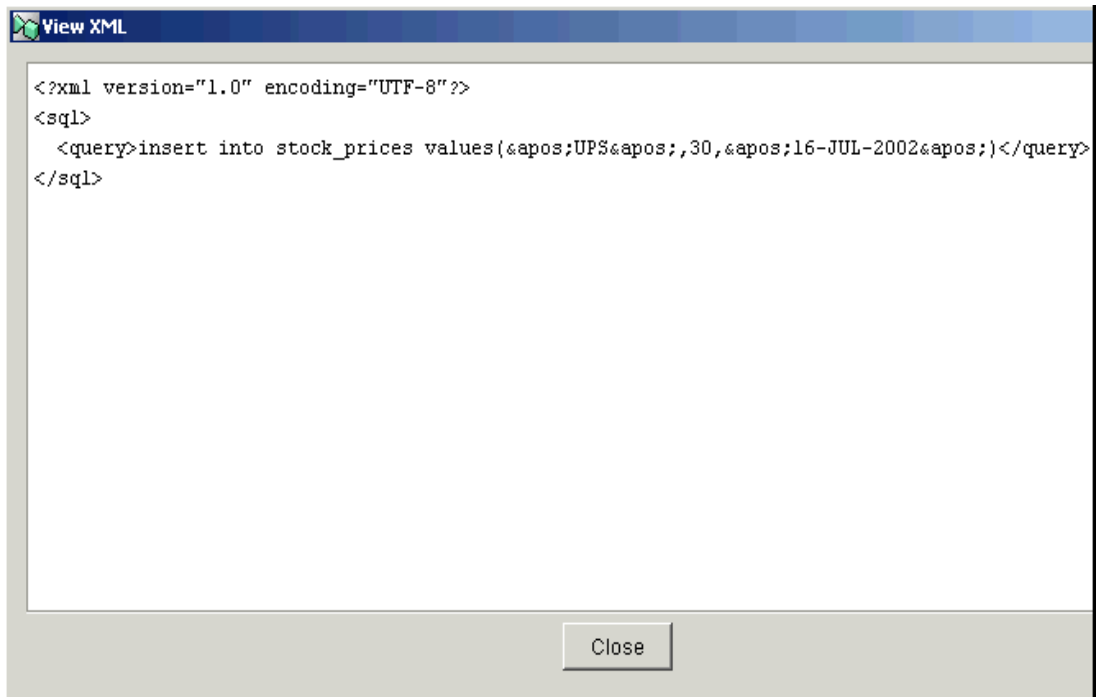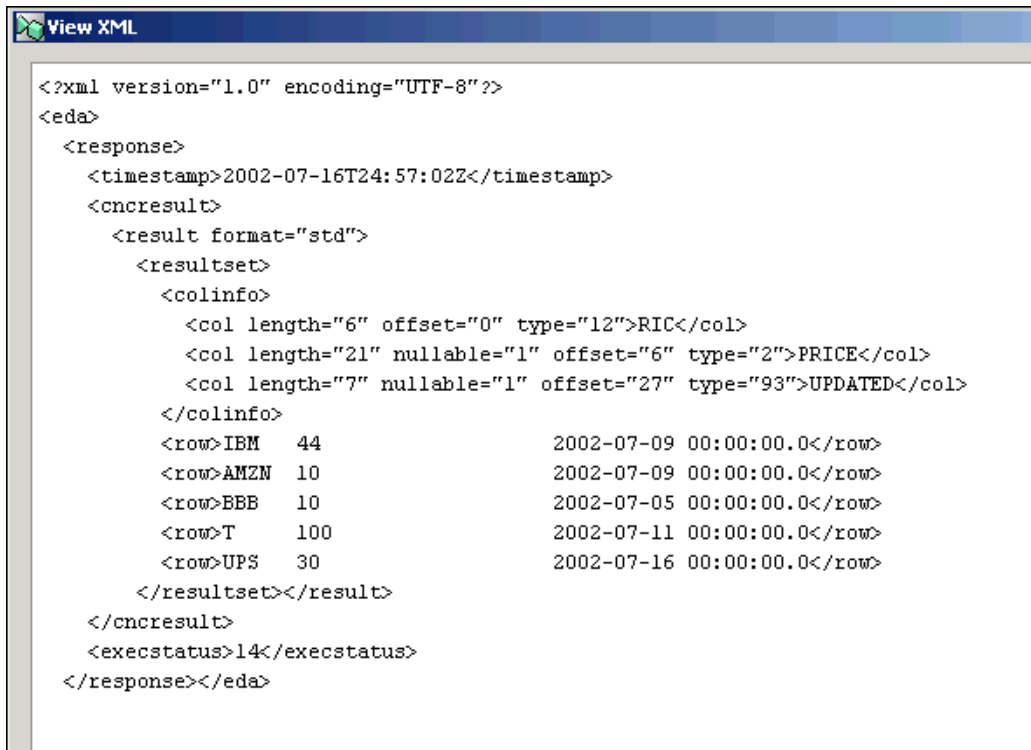
Delete Statement

The following example shows a Delete statement:

```
delete from stock_prices where RIC='UPS'
```

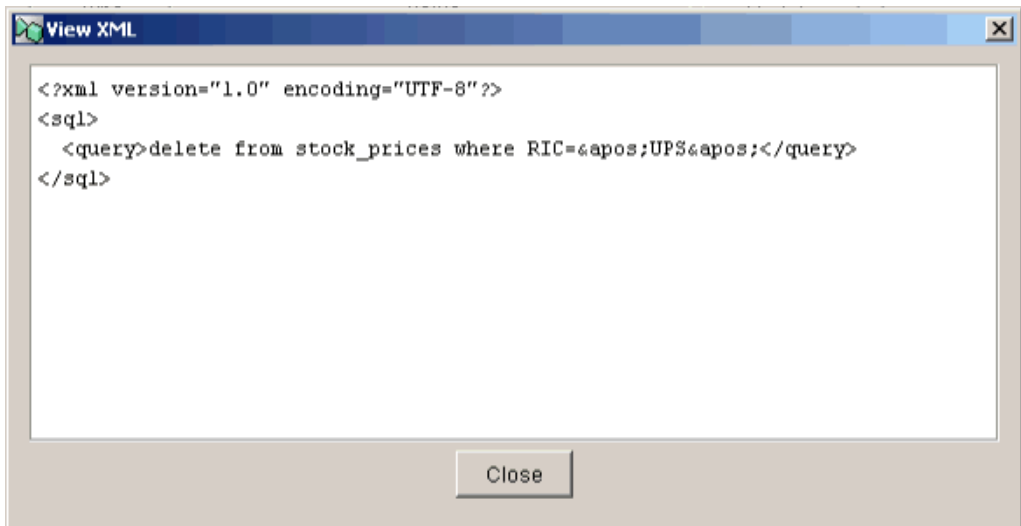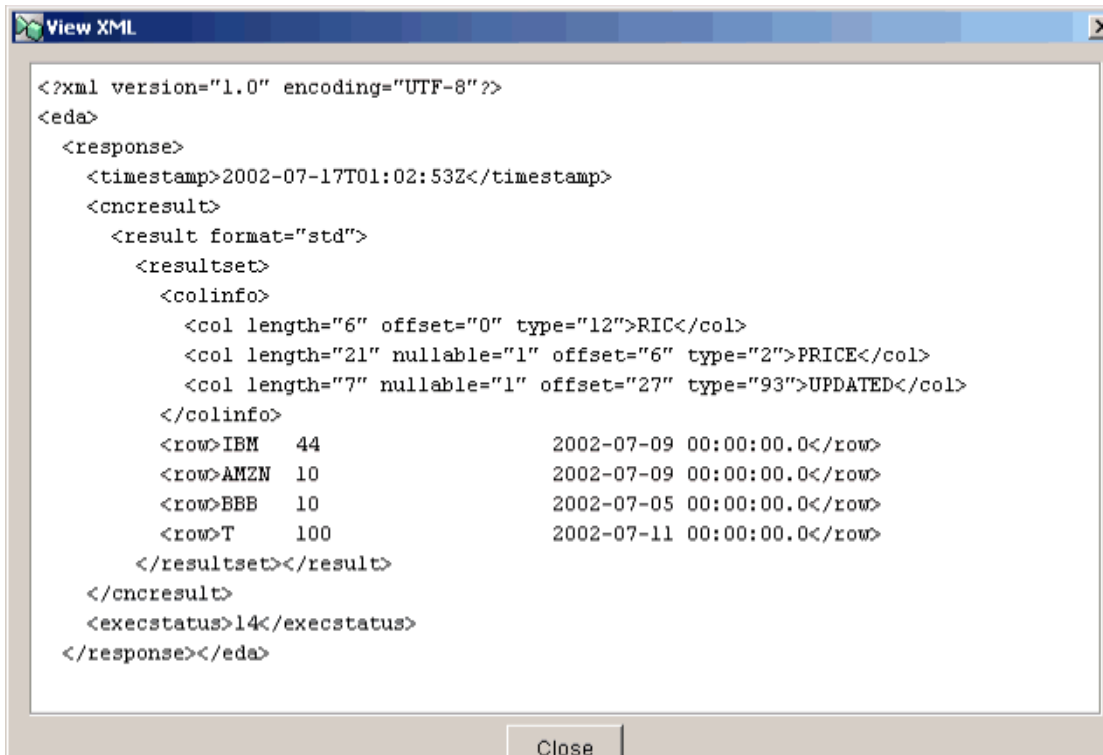**Figure 4-5   Input as Seen in WebLogic Integration Studio Window**

Figure 4-6   Generated Output - WebLogic Integration Studio Window

```
View XML                                                              ×

<?xml version="1.0" encoding="UTF-8"?>
<eda>
  <response>
    <timestamp>2002-07-17T01:02:53Z</timestamp>
    <cncresult>
      <result format="std">
        <resultset>
          <colinfo>
            <col length="6" offset="0" type="12">RIC</col>
            <col length="21" nullable="1" offset="6" type="2">PRICE</col>
            <col length="7" nullable="1" offset="27" type="93">UPDATED</col>
          </colinfo>
          <row>IBM    44                2002-07-09 00:00:00.0</row>
          <row>AMZN   10                2002-07-09 00:00:00.0</row>
          <row>BBB    10                2002-07-05 00:00:00.0</row>
          <row>T      100               2002-07-11 00:00:00.0</row>
        </resultset></result>
    </cncresult>
    <execstatus>14</execstatus>
  </response></eda>

                            Close
```
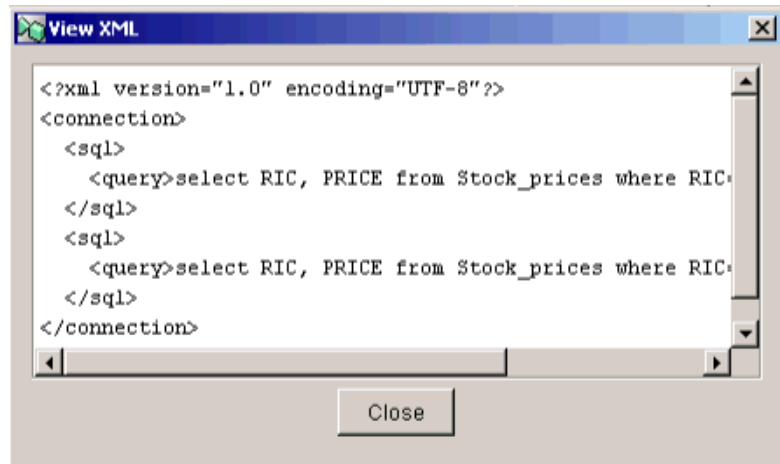
Multi-Select Statements

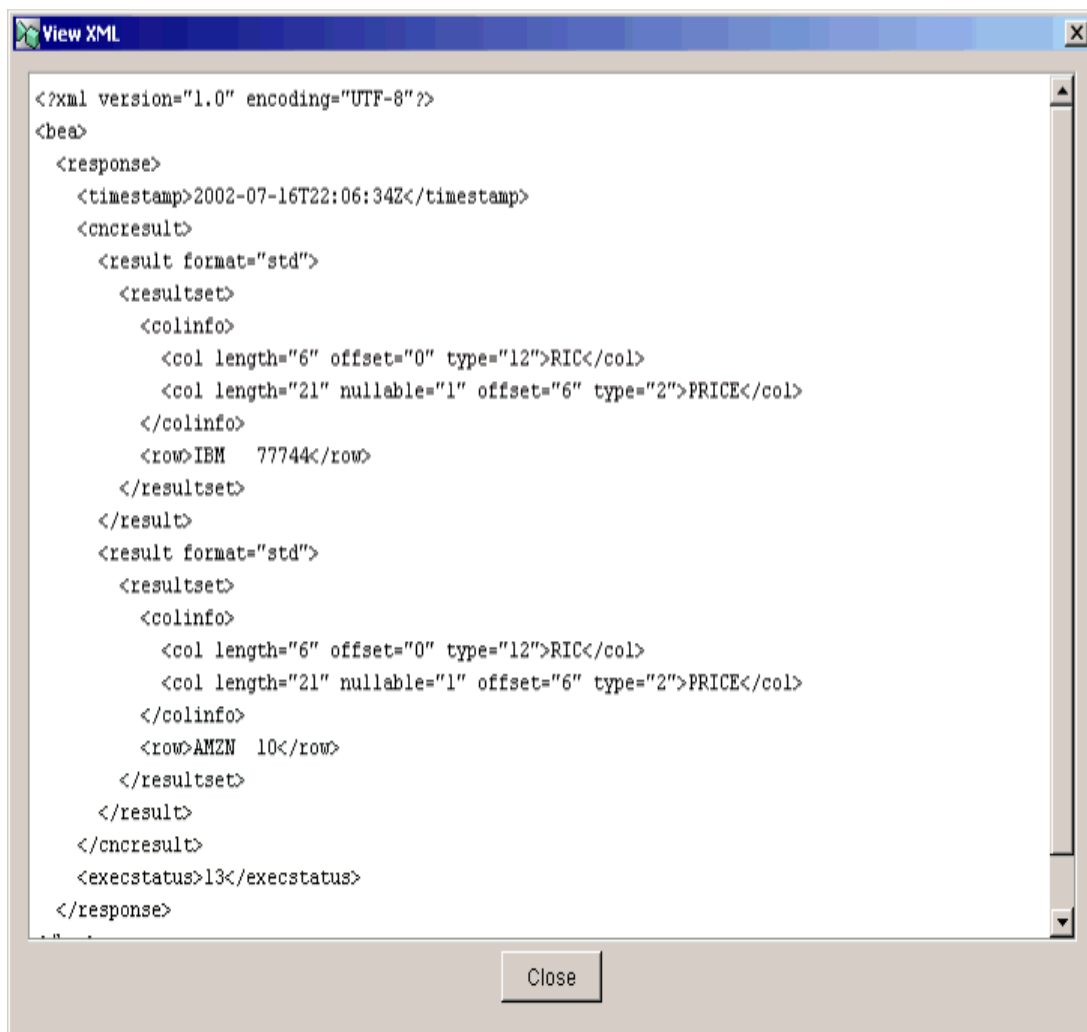The following example shows a multi-select statement:

```
Select * from Stock_prices where RIC='IBM';
```

Figure 4-7   Input as Seen in WebLogic Integration Studio Window

```
Select * from Stock_prices where RIC='AMZN';
```

Figure 4-8   Generated Output - WebLogic Integration Studio Window

Update Statement

The following example shows an Update statement:

```
update Stock_prices set PRICE=44 where RIC='IBM';
```

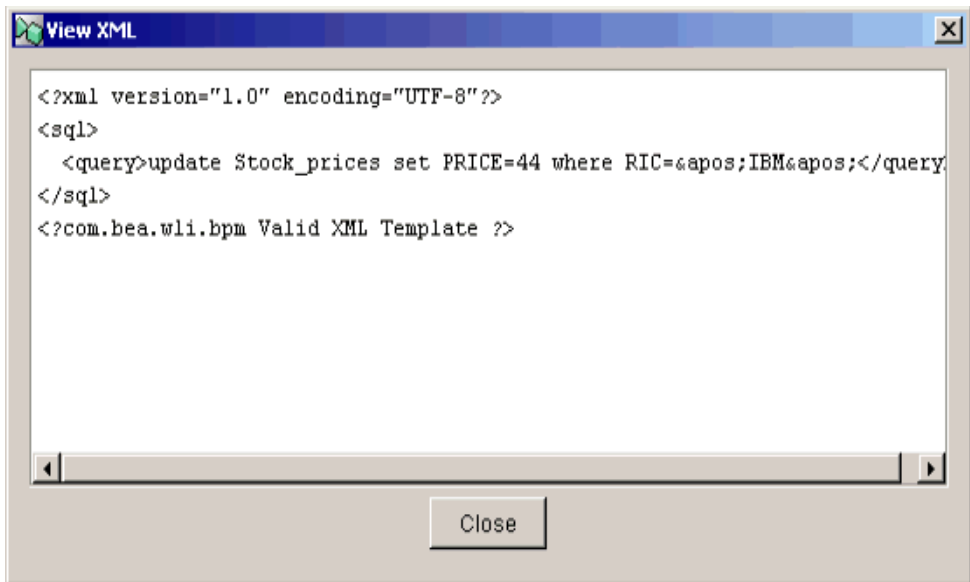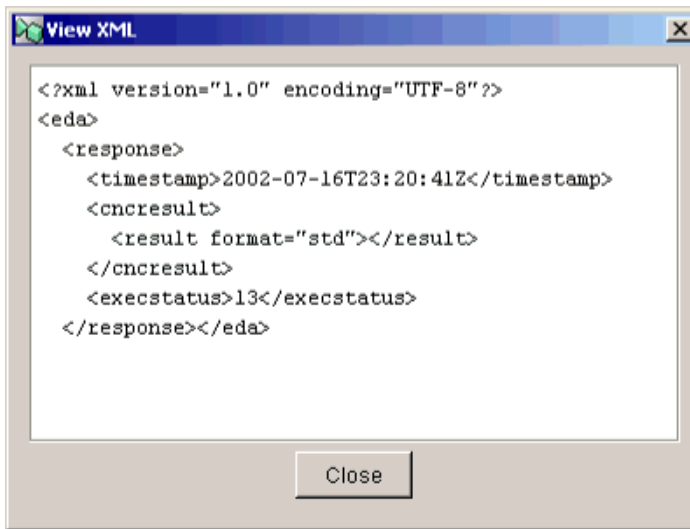**Figure 4-9   Input as Seen in WebLogic Integration Studio Window**

```
<?xml version="1.0" encoding="UTF-8"?>
<sql>
  <query>update Stock_prices set PRICE=44 where RIC=&apos;IBM&apos;</query
</sql>
<?com.bea.wli.bpm Valid XML Template ?>
```

Figure 4-10   Generated Output - WebLogic Integration Studio Window

Stored Procedure

Note: For the following example returning a result set (cursor variable), an object type REF CURSOR must be declared. There are multiple ways of doing this. It can be done as an object type or declared in a package.

The following code snippet when executed will create a package with the appropriate type for the example:

```
CREATE OR REPLACE PACKAGE types
AS
    TYPE ref_cursor IS REF CURSOR;
END;
```

The following example shows an Oracle stored procedure:

Listing 4-3   Oracle Stored Procedure

```
CREATE OR REPLACE FUNCTION sp_get_stocks(v_price IN NUMBER)
    RETURN types.ref_cursor
AS
    stock_cursor types.ref_cursor;
BEGIN
    OPEN stock_cursor FOR
        SELECT ric,price,updated FROM stock_prices
        WHERE price < v_price;

    RETURN stock_cursor;
END;
```

Listing 4-4   Oracle Input Schema

```
<?xml version="1.0" encoding="utf-16"?>
<xsd:schema id="NewDataSet" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xsd:element name="sp">
    <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="proc" type="xsd:string" minOccurs="0" />
       <xsd:element name="param" type="xsd:string" minOccurs="0" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="NewDataSet" msdata:IsDataSet="true">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element ref="sp" />
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 4-5 Oracle Output Schema

```
<?xml version="1.0" encoding="utf-16"?>
<xsd:schema id="eda" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xsd:element name="eda" msdata:IsDataSet="true">
<xsd:complexType>
<xsd:choice maxOccurs="unbounded">
<xsd:element name="response">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="timestamp" type="xsd:string" minOccurs="0" />
<xsd:element name="execstatus" type="xsd:string" minOccurs="0" />
<xsd:element name="cncresult" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="result" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="resultset" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="colinfo" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="col" nillable="true" minOccurs="0"
maxOccurs="unbounded">
<xsd:complexType>
<xsd:simpleContent msdata:ColumnName="col_Text"
msdata:Ordinal="3">
<xsd:extension base="xsd:string">
<xsd:attribute name="length" type="xsd:string" />
<xsd:attribute name="offset" type="xsd:string" />
<xsd:attribute name="type" type="xsd:string" />
<xsd:attribute name="nullable" type="xsd:string" />
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="row" nillable="true" minOccurs="0"
maxOccurs="unbounded">
<xsd:complexType>
<xsd:simpleContent msdata:ColumnName="row_Text"
```

```
msdata:Ordinal="0">
<xsd:extension base="xsd:string">
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="format" type="xsd:string" />
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:choice>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```
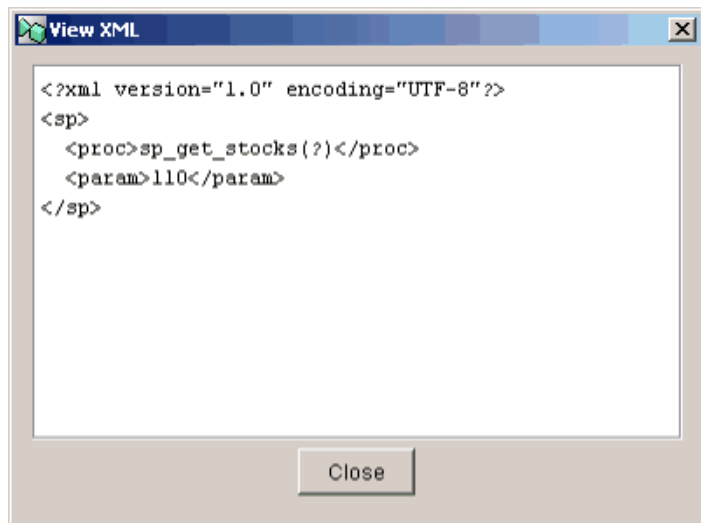
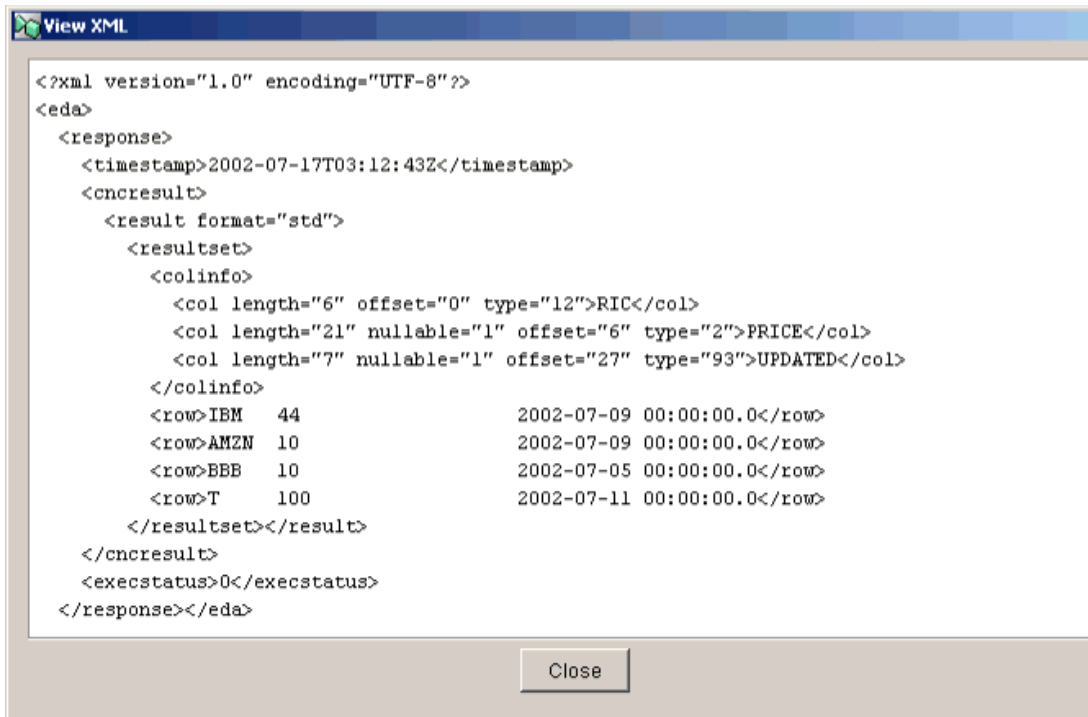**Figure 4-11   Input as Seen in WebLogic Integration Studio Window**

Figure 4-12 Generated Output - WebLogic Integration Studio Window

Including Multiple SQL Statements in an XML Request

The following example illustrates how to include more than one SQL statement in an XML request that you send to the service adapter. The format is similar to the format for a single SQL statement, with the addition of a `<connection>` tag.

The format for a request having multiple SQL statements is:

Listing 4-6 XML Format for Request with Multiple SQL Statements

```
<connection>
    <sql>
        <query>
            SQL STATEMENT1
```

```
        </query>
     </sql>
     <sql>
        <query>
            SQL STATEMENT2
        </query>
     </sql>
      .
      .
      .
     <sql>
        <query>
            SQL STATEMENTn
        </query>
     </sql>
</connection>
```

For example, you can include two statements that insert two records into a
Stock_Prices table as follows:

Listing 4-7   Sample Request with Multiple SQL Statements

```
<connection>
   <sql>
      <query>
          insert into stock_prices values('TICK1',30,'16-JUL-2002')

      </query>
   </sql>
   <sql>
      <query>
          insert into stock_prices values('TICK2',120,'16-JUL-2002')

      </query>
   </sql>
</connection>
```

The following figure shows the request running in the WebLogic Integration Test Tool:

Figure 4-13   XML Request with Multiple SQL Statements - Input and Output Windows

Input to service OracleSimpleService on application view OracleSimpleService

```
<connection>
<sql>
<query>
insert into stock_prices_temp values
('TICK1',30,'16-JUL-2002')
</query>
</sql>
<sql>
<query>
insert into stock_prices_temp values
('TICK2',120,'16-JUL-2002')
</query>
</sql>
</connection>
```

Output from service OracleSimpleService on application view OracleSimpleService

```
<?xml version="1.0"?>
<eda>
  <response>
    <timestamp>2002-07-30T18:32:24Z</timestamp>
    <cncresult>
      <result format="std"/>
      <result format="std"/>
    </cncresult>
    <execstatus>0</execstatus>
  </response>
</eda>
```

Execution time: 741 (ms)

5 Event Adapter Examples

This section provides examples that use the event adapter. It includes the following examples:

- Simple SQL Listener

- Setting up a Non-Destructive Read in the SQL Adapter

- Specifying Delete Keys in the SQL Adapter

Simple SQL Listener

In this example, the properties of a simple SQL listener are described. It is assumed that you are familiar with the process of setting up an application view and deploying an adapter event.

The SQL adapter polls a specified RDBMS table for data. In this example, the table is STOCK_PRICES. If found, the data is formatted in an XML response. The record in the RDBMS is then deleted from the table.

Listing 5-1   Sample Schema for Simple Event Adapter

```
SQL> describe stock_prices

Name                                     Null?    Type
---------------------------------------- -------- -----------
RIC                                      NOT NULL VARCHAR2(6)
PRICE                                             NUMBER(7,2)
UPDATED                                           DATE
```

The following is the Manifest for this example:

Listing 5-2   Sample Manifest for Simple Event Adapter

```
<manifest><connection>
<user>EDARPK</user>
<password> </password>
<driver>oracle.jdbc.driver.OracleDriver</driver>
<url>jdbc:oracle:thin:@Oracle11i.ibi.com:1523:vis</url>
</connection>
          <schemaref name="oracle">
               <request root="sql" file="StockPricesTemp.xsd" />
              <response root="sql" file="StockPricesTemp.xsd" />
          </schemaref>
</manifest>
```

The following is a schema of an example SQL listener event
(StockPricesTemp.xsd):

Listing 5-3   Schema of StockPricesTemp.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
      <xsd:element name="RDBMS">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="row"/>
            </xsd:sequence>
            <xsd:attribute name="table" type="xsd:string"
```

```
use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PRICE">
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:extension base="xsd:byte">
                    <xsd:attribute name="type" type="xsd:byte"
use="required"/>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RIC">
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                    <xsd:attribute name="type" type="xsd:byte"
use="required"/>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UPDATED">
        <xsd:complexType>
            <xsd:attribute name="type" type="xsd:byte"
use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="row">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="RIC"/>
                <xsd:element ref="PRICE"/>
                <xsd:element ref="UPDATED"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Figure 5-1   Add Event Window

The SQL listener properties for this JSP page are as follows:

- **Encoding.** ISO-8859-1

- **Driver.** JDBC Driver. For example,

  `oracle.jdbc.driver.OracleDriver`

- **URL.** URL of RDBMS. For example,

  `jdbc:oracle:thin:@Oracle11i.ibi.com:1523:vis`

- **User Name.** User name that is registered with the back-end RDBMS.

- **Password.** The Oracle Applications user ID authorized to access the Oracle Applications system.

- **Table.** STOCK_PRICES

- **Maximum Rows.** 1

- **SQL Query.** (empty)

- **SQL Post Query.** (empty)

- **Delete Keys.** (empty)

- **Polling Interval.** Interval in seconds.

Setting up a Non-Destructive Read in the SQL Adapter

The following example describes how to detect incoming data into a table without a destructive read. This example requires two tables that are named trans_event and last_trans.

Trans_event is the table that has the incoming data that you want to detect. Last_trans is a table that contains the last value of the primary key read from the trans_event table. Last_trans is to contain a single row, single value and must be set up prior to configuring the BEA WebLogic Adapter for RDBMS. The last_trans table field must have the same name as the primary key in the trans_event table. This key must be unique and sortable.

The table schemas for this example are:

Listing 5-4   Sample Schema for Non-Destructive Read

```
SQL> describe trans_event

Name                                      Null?     Type
---------------------------------------- -------- --------------

EVENT_ID                                 NOT NULL NUMBER(38)
LAST_NAME                                          VARCHAR2(50)
TRANS_ID                                           CHAR(2)

SQL> describe last_trans
Name                                      Null?     Type
---------------------------------------- ------- -------------
EVENT_ID                                           NUMBER
```

The last_trans single field value must be seeded with the starting value of the primary key.

The SQL adapter generates XML response documents for each record found in the trans_event table with a primary key greater than the value found in the last_trans table.

To set up the SQL listener for this example, you must establish an application view.

Figure 5-2   Edit Event Window

The SQL listener properties for this JSP page are as follows:

- **Encoding.** ISO-8859-1

- **Driver.** JDBC Driver. For example,

    `oracle.jdbc.driver.OracleDriver`

- **URL.** URL of RDBMS. For example,

    `jdbc:oracle:thin:@Oracle11i.ibi.com:1523:vis`

- **User Name.** User name that is registered with the back-end RDBMS.

- **Password.** Password of the user name.

- **Table.** Table name containing the value of the highest primary key processed. For example: LAST_TRANS

- **Maximum Rows.** 1

- **SQL Query.** SELECT * FROM TRANS_EVENT WHERE EVENT_ID>(select EVENT_ID from LAST_TRANS.EVENT_ID)

- **SQL Post Query.** UPDATE LAST_TRANS SET ?EVENT_ID

- **Delete Keys.** (empty)

- **Polling Interval.** Interval in seconds.

The following is the manifest for this example:

Listing 5-5   Sample Manifest for Non-Destructive Read

```
<manifest><connection>
<user>EDARPK2</user>
<password></password>
<driver>oracle.jdbc.driver.OracleDriver</driver>
<url>jdbc:oracle:thin:@Oracle11i.ibi.com:1523:vis</url>
</connection>
            <schemaref name="oracle">
                    <event root="RDBMS" file="EventComplex.xsd" />
            </schemaref>
</manifest>
```

The following is a schema of an example response (EventComplex.xsd):

Listing 5-6   Schema of EventComplex.xsd

```
<?xml version="1.0" enc
oding="UTF-8"?>
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xsd:element name="EVENT_ID">
        <xsd:complexType>
            <xsd:attribute name="type" type="xsd:byte"
```

```
use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="LAST_NAME">
        <xsd:complexType>
            <xsd:attribute name="type" type="xsd:byte"
use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="OracleSQL">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="row"/>
            </xsd:sequence>
            <xsd:attribute name="table" type="xsd:string"
use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="TRANS_ID">
        <xsd:complexType>
            <xsd:attribute name="type" type="xsd:byte"
use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="row">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="EVENT_ID"/>
                <xsd:element ref="LAST_NAME"/>
                <xsd:element ref="TRANS_ID"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Specifying Delete Keys in the SQL Adapter

During a destructive read in the SQL adapter, the deletion of the record from the database is accomplished through the adapter's dynamic creation of a delete statement containing all of the keys from the record along with the row values.

For the STOCK_PRICES example, this would be as follows:

```
Delete from STOCK_PRICES where
    RIC="value"
    PRICES=value
    UPDATED=date value.
```

This would be inefficient (and can be avoided) if the record contains a primary key. Under the delete keys in the SQL listener properties, you can specify the keys to use when the SQL adapter dynamically creates and executes the delete statement.

In the example table STOCK_PRICES, the primary key is the RIC field. For the adapter to dynamically create a delete statement like the following:

```
Delete from STOCK_PRICES where RIC= "value"
```

you must specify the RIC field in the SQL listener Properties window (as seen below in the Delete Keys field) when setting up the SQL listener for this event.

Figure 5-3   Add Event Window

The SQL listener properties for this JSP page are as follows:

- **Encoding.** ISO-8859-1

- **Driver.** JDBC Driver. For example,

  ```
  oracle.jdbc.driver.OracleDriver
  ```

- **URL.** URL of RDBMS. For example,

  `jdbc:oracle:thin:@Oracle11i.ibi.com:1523:vis`

- **User Name.** User name that is registered with the back-end RDBMS.

- **Password.** Password of the user name.

- **Table.** `STOCK_PRICES`

- **Maximum Rows.** 1

- **SQL Query.** (empty)

- **SQL Post Query.** (empty)

- **Delete Keys.** (RIC)

- **Polling Interval.** Interval in seconds.

The schema for the `STOCK_PRICES` table (in this example) is as follows:

Listing 5-7   Sample Schema for Specifying Delete Keys

```
SQL> describe stock_prices
Name                                    Null?    Type
--------------------------------------- -------- ---------------

RIC                                     NOT NULL VARCHAR2(6)
PRICE                                            NUMBER(7,2)
UPDATED                                          DATE
```

The following is the manifest for this example:

Listing 5-8   Sample Manifest for Specifying Delete Keys

```
<manifest><connection>
<user>EDARPK</user>
<password></password>
<driver>oracle.jdbc.driver.OracleDriver</driver>
<url>jdbc:oracle:thin:@Oracle11i.ibi.com:1523:vis</url>
</connection>
            <schemaref name="OracleRDBMSStockPrices">
                  <event root="RDBMS" file="StockPricesTemp.xsd" />
            </schemaref>
</manifest>
```

The following is a schema of an example response (`StockPricesTemp.xsd`):

Listing 5-9   Schema of StockPricesTemp.xsd

```
<?xml version="1.0" encoding="UTF-8"?><xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
     <xsd:element name="RDBMS">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="row"/>
             </xsd:sequence>
            <xsd:attribute name="table" type="xsd:string"
use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PRICE">
        <xsd:complexType>
            <xsd:simpleContent>
               <xsd:extension base="xsd:byte">
                   <xsd:attribute name="type" type="xsd:byte"
use="required"/>
               </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RIC">
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:extension base="xsd:string">
```

```
                        <xsd:attribute name="type" type="xsd:byte"
use="required"/>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UPDATED">
        <xsd:complexType>
            <xsd:attribute name="type" type="xsd:byte"
use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="row">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="RIC"/>
                <xsd:element ref="PRICE"/>
                <xsd:element ref="UPDATED"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```