Chapter 9 of *WebSphere Application Server V7 Administration and Configuration Guide,* SG24-7615

WebSphere Application Server V7: Accessing Databases from WebSphere

When an application or WebSphere® component requires access to a database, that database must be defined to WebSphere as a data source. Two basic definitions are required:

- A JDBC provider definition defines an existing database provider, including the type of database access that it provides and the location of the database vendor code that provides the implementation.
- A data source definition defines which JDBC provider to use, the name and location of the database, and other connection properties.

In this chapter, we discuss the various considerations for accessing databases from WebSphere.

We cover the following topics:

JDBC resources

The JDBC API provides a programming interface for data access of relational databases from the Java™ programming language. WebSphere Application Server V7 supports the following JDBC APIs:

- JDBC 4.0 **(New in V7)**
- JDBC 3.0
- JDBC 2.1 and Optional Package API (2.0)

In the following sections, we explain how to create and configure data source objects for use by JDBC applications. This method is the recommended method to connect to a database and the only method if you intend to use connection pooling and distributed transactions.

Note: DB2® for z/OS® local JDBC Provider (RRS) Version 6.1 is not supported in WebSphere Application Server V7.0. If you use this provider, you need to migrate IBM® JCC Driver or DB2 Universal JDBC Driver.

The following database platforms are supported for JDBC:

- DB2
- Oracle
- Sybase
- Informix®
- SQL Server
- IBM Cloudscape and IBM Derby (test and development only)
- Third-party vendor JDBC data source using SQL99 standards

JDBC providers and data sources

A *data source* represents a real-world data source, such as a relational database. When a data source object is registered with a JNDI naming service, an application can retrieve it from the naming service and use it to make a connection to the data source that it represents.

Information about the data source and how to locate it, such as its name, the server on which it resides, its port number, and so on, is stored in the form of *properties* on the DataSource object. Storing this information in this manner makes an application more portable because it does not need to hard code a driver name, which often includes the name of a particular vendor. It also makes maintaining the code easier because if, for example, the data source is moved to a different server, all that needs to be done is to update the relevant property in the data source. None of the code using that data source needs to be touched.

After a data source is registered with an application server's JNDI name space, application programmers can use it to make a connection to the data source that it represents.

The connection usually is a *pooled connection*. In other words, when the application closes the connection, the connection is returned to a connection pool, rather than being destroyed.

Data source *classes* and JDBC *drivers* are implemented by the data source vendor. By configuring a JDBC provider, you provide information about the set of classes that are used to implement the data source and the database driver. Also, you provide the environment settings for the DataSource object. A driver can be written purely in the Java programming language or in a mixture of the Java programming language and the Java Native Interface (JNI) native methods.

In the next sections, we describe how to create and configure data source objects, as well as how to configure the connection pools used to serve connections from the data source.

WebSphere support for data sources

The programming model for accessing a data source is as follows:

1. An application retrieves a DataSource object from the JNDI naming space.
2. After the DataSource object is obtained, the application code calls the `getConnection()` request on the data source to get a Connection object. The connection is obtained from a pool of connections.
3. After the connection is acquired, the application sends SQL queries or updates to the database.

In addition to the data source support for Java EE 5, J2EE 1.3, and J2EE 1.4 applications, support is also provided for J2EE 1.2 data sources. The two types of support differ in how connections are handled. However, from an application point of view, they look the same.

Data source support

The primary data source support is intended for J2EE 1.3 and J2EE 1.4, and Java EE 5 applications. Connection pooling is provided by two components, a JCA Connection Manager, and a relational resource adapter. See Figure 1.

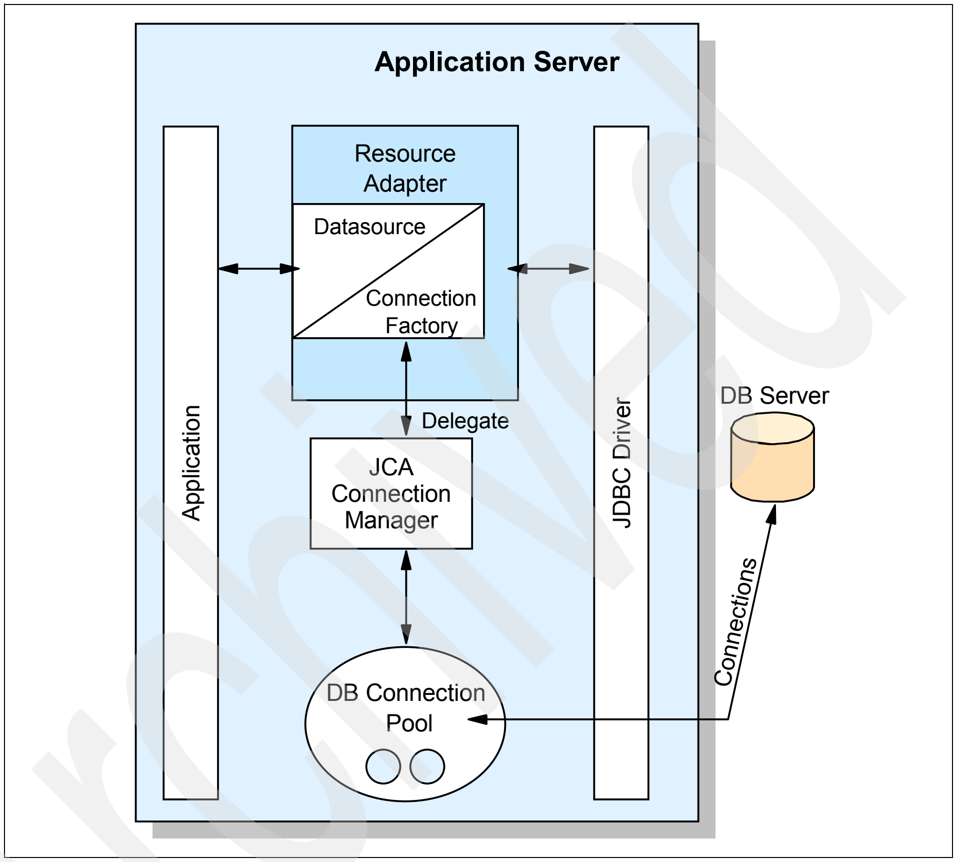


Figure 1 Resource adapter in J2EE connector architecture

The JCA Connection Manager provides connection pooling, local transactions, and security support.

The relational resource adapter provides JDBC wrappers and the JCA CCI implementation that allows BMP, JDBC applications, and CMP beans to access the database.

Figure 2 shows the relational resource adapter model.

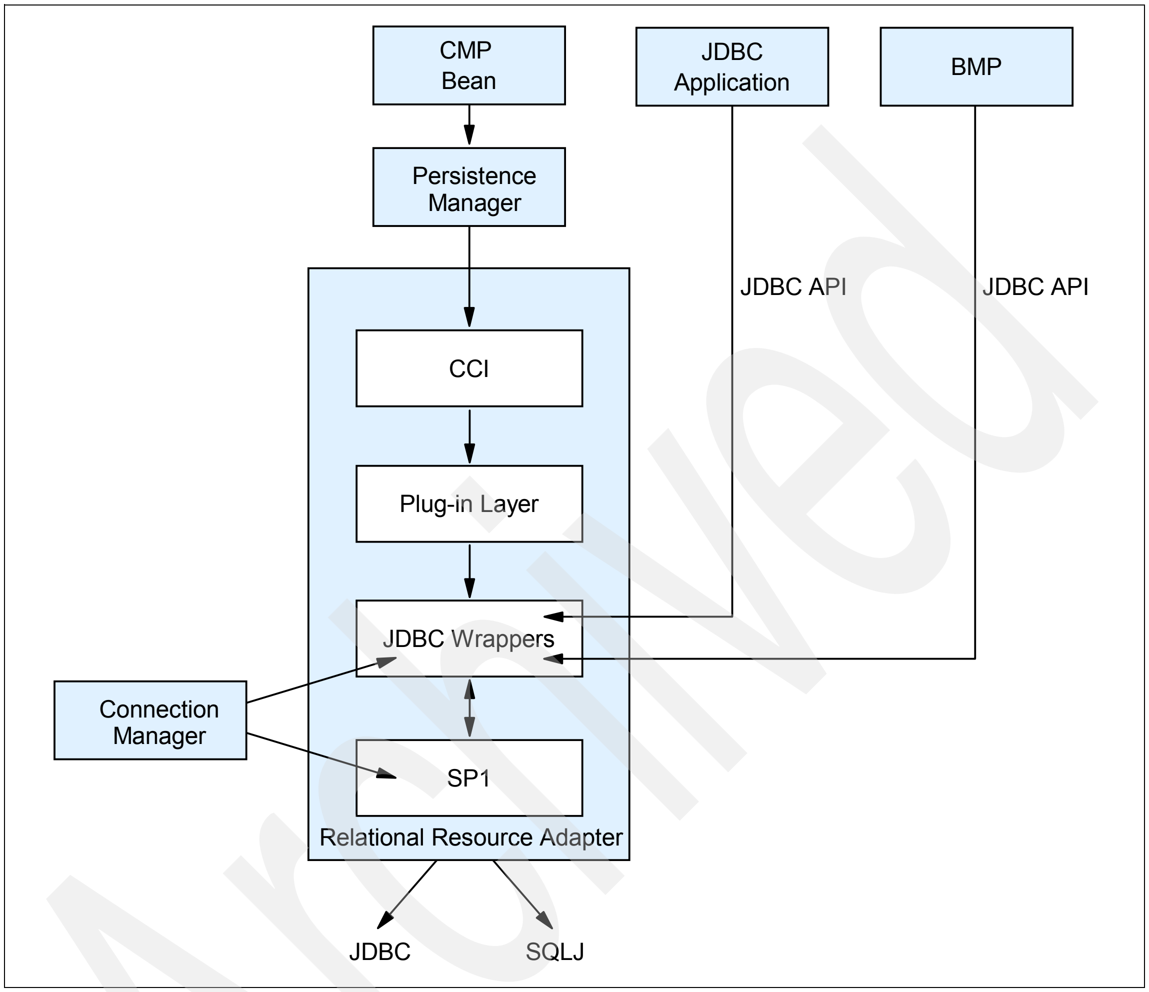


Figure 2 Persistence resource adapter model

WebSphere Application Server has a Persistence Resource Adapter that provides relational persistence services to EJB beans as well as providing database access to BMP and JDBC applications. The Persistence Resource Adapter has two components:

- The Persistence Manager, which supports the EJB CMP persistence model
- The Relational Resource Adapter

The Persistence Resource Adapter code is included in the following Java packages:

- The `com.ibm.ws.rsadapter.cci` package contains CCI implementation and JDBC wrappers.
- The `com.ibm.ws.rsadapter.spi` package contains SPI implementation.
- The `com.ibm.ws.rsadapter.jdbc` package contains all the JDBC wrappers.
- The `com.ibm.websphere.rsadapter` package contains DataStoreHelper, WSCallerHelper, and DataAccessFunctionSet.

The Relational Resource Adapter is the Persistence Manager's vehicle to handle data access to and from the back-end store, providing relational persistence services to EJB beans. The implementation is based on the J2EE Connector (JCA) specification and implements the JCA CCI and SPI interfaces.

When an application uses a data source, the data source uses the JCA connector architecture to get to the relational database.

For an EJB, the sequence is as follows:

1. An EJB performs a JNDI lookup of a data source connection factory and issues a `getConnection()` request.
2. The connection factory delegates the request to a connection manager.
3. The connection manager looks for an instance of a connection pool in the application server. If no connection pool is available, then the manager uses the ManagedConnectionFactory to create a physical, or nonpooled, connection.

Version 4 data source

WebSphere Application Server V4 provided its own JDBC connection manager to handle connection pooling and JDBC access. This support is included with WebSphere Application Server V7.0 to provide support for J2EE 1.2 applications. If an application chooses to use a Version 4 data source, the application has the same connection behavior as in Version 4 of the application server.

Use the Version 4 data source for the following purposes:

- J2EE 1.2 applications

  All EJB beans, JDBC applications, or Version 2.2 servlets must use the Version 4 data source.

- EJB 1.x modules with 1.1 deployment descriptor

  All of these modules must use the Version 4 data source.

Steps in defining access to a database

The following steps are involved in creating a data source:

1. Verify that connection to the database server is supported by WebSphere Application Server. See:

   http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg27012369

2. Ensure that the database has been created and can be accessed by the systems that will use it.
3. Ensure that the JDBC provider classes are available on the systems that will access the database. If you are not sure which classes are required, consult the documentation for the provider.
4. Create an authentication alias that contains the user ID and password that will be used to access the database.
5. Create a JDBC provider.

   The JDBC provider gives the classpath of the data source implementation class and the supporting classes for database connectivity. This is vendor-specific.

   The information center provides information about JDBC driver support and requirements. To determine if your provider is supported, refer to the JDBC Provider Summary article at:

   http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/udat_minreq.html

   New in V7 for DB2: The DB2 Using IBM JCC Driver is a one-phase commit JCC provider for DB2 that uses the IBM Data Server Driver for JDBC and SQLJ. The DB2 Using IBM JCC Driver is the next generation of the DB2 Universal JCC driver. Data sources that you create with this provider support only one-phase commit processing, unless you use the type 2 JDBC driver with the application server for z/OS. If you run the application server on z/OS with the type 2 driver, the driver uses RRS and supports two-phase commit processing. This driver provides some JDBC 4.0 capabilities.

6. Create a data source.

   The JDBC data source encapsulates the database-specific connection settings. You can create many data sources that use the same JDBC provider.

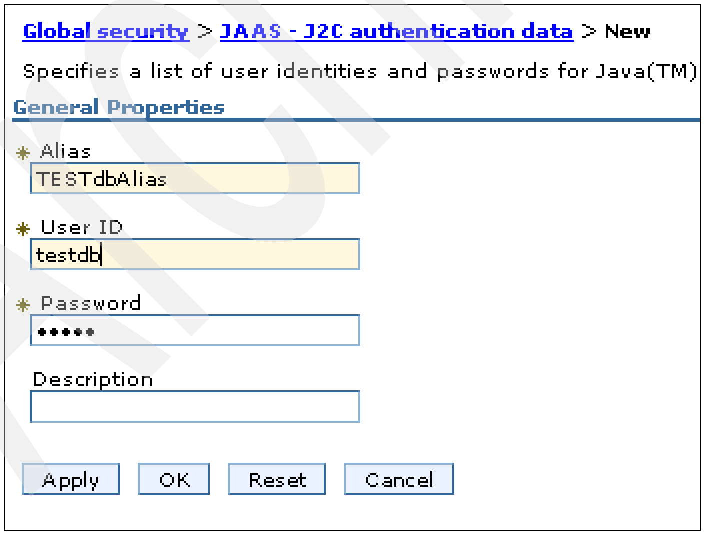
7. Save the changes to the master repository and synchronize with the nodes involved.
8. Test the connection to the data source.
9. Review and adjust the connection pool settings (this should be done on a periodic basis).

Creating an authentication alias

The examples in this chapter assume that the database is password protected and that the user ID and password will be defined at run time.

To create a J2C authentication alias that contains the user ID and password that is required to access the database, follow these steps:

1. Select **Security** → **Global security**.
2. In the Authentication area, expand Java Authentication and Authorization Server, and click **J2C authentication data**.
3. Click **New**.
4. Enter an alias name, user ID, and password, as shown in Figure 3. The alias name will be used later when you create a resource to identify this as the authentication alias to use. The user ID and password must be valid for the database system and have authority to the database.


Global security > JAAS - J2C authentication data > New

Specifies a list of user identities and passwords for Java(TM)

General Properties

* Alias: TESTdbAlias

* User ID: testdb

* Password: •••••

Description:

Figure 3 Define an authentication alias

5. Click **OK**.

Example: Connecting to an IBM DB2 database

In this section, we illustrate how to configure a JDBC provider using a DB2 provider as an example.

Creating the JDBC provider

To create a JDBC provider, complete the following steps from the administrative console:

1. Ensure that the implementation classes for the provider are available to the system. The class files will need to be located on each system where the application servers will run.
2. In the administrative console, expand **Resources** → **JDBC** from the navigation tree.
3. Click **JDBC Providers**.
4. Select the scope. (Although you can select **All scopes** to view all resources, you must select a specific scope to create a resource.)

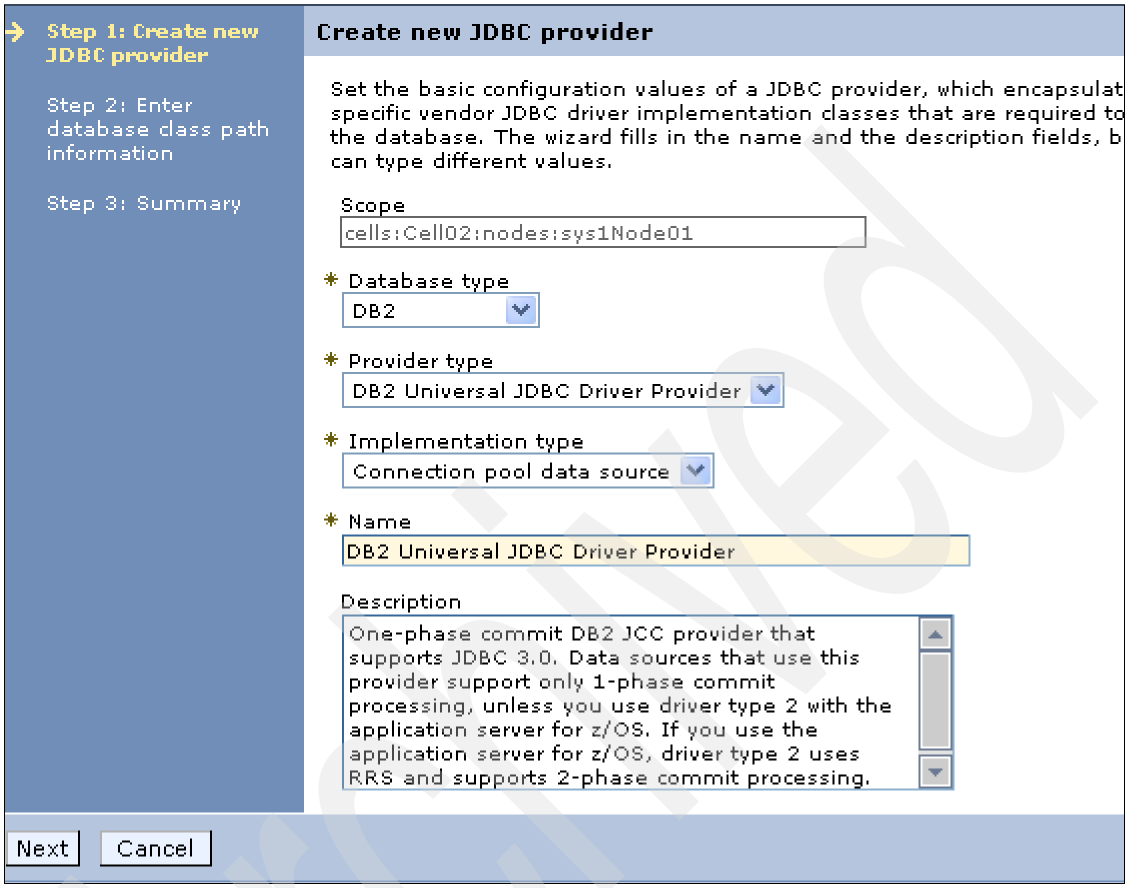
   Note: JDBC resources are created at a specific scope level. The data source scope level is inherited from the JDBC provider. For example, if we create a JDBC provider at the node level and then create a data source using that JDBC provider, the data source inherits:

   - The JDBC provider settings, such as classpath, implementation class, and so on
   - The JDBC provider scope level

     In this example, if the scope were set to node-level, all application servers running on that node register the data source in their name space.

   The administrative console now shows all the JDBC providers that are created at that scope level.
5. Select **New** to start the wizard and to create a new JDBC provider.

6. In step 1 of the wizard, define the type of provider you will use. See Figure 4.


Step 1: Create new JDBC provider
Step 2: Enter database class path information
Step 3: Summary

Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulat
specific vendor JDBC driver implementation classes that are required to
the database. The wizard fills in the name and the description fields, b
can type different values.

Scope
cells:Cell02:nodes:sys1Node01

* Database type
DB2

* Provider type
DB2 Universal JDBC Driver Provider

* Implementation type
Connection pool data source

* Name
DB2 Universal JDBC Driver Provider

Description
One-phase commit DB2 JCC provider that supports JDBC 3.0. Data sources that use this provider support only 1-phase commit processing, unless you use driver type 2 with the application server for z/OS. If you use the application server for z/OS, driver type 2 uses RRS and supports 2-phase commit processing.

Figure 4 Define a new JDBC provider: Window 1

Specify the following information

– Database type

  Select the vendor-specific database type. If the database type you need is not in the list, select **User-defined**, and consult the vendor documentation for the specific properties that are required.

– Provider type

  Select from a predefined list of supported provider types, based on the database type that you select.

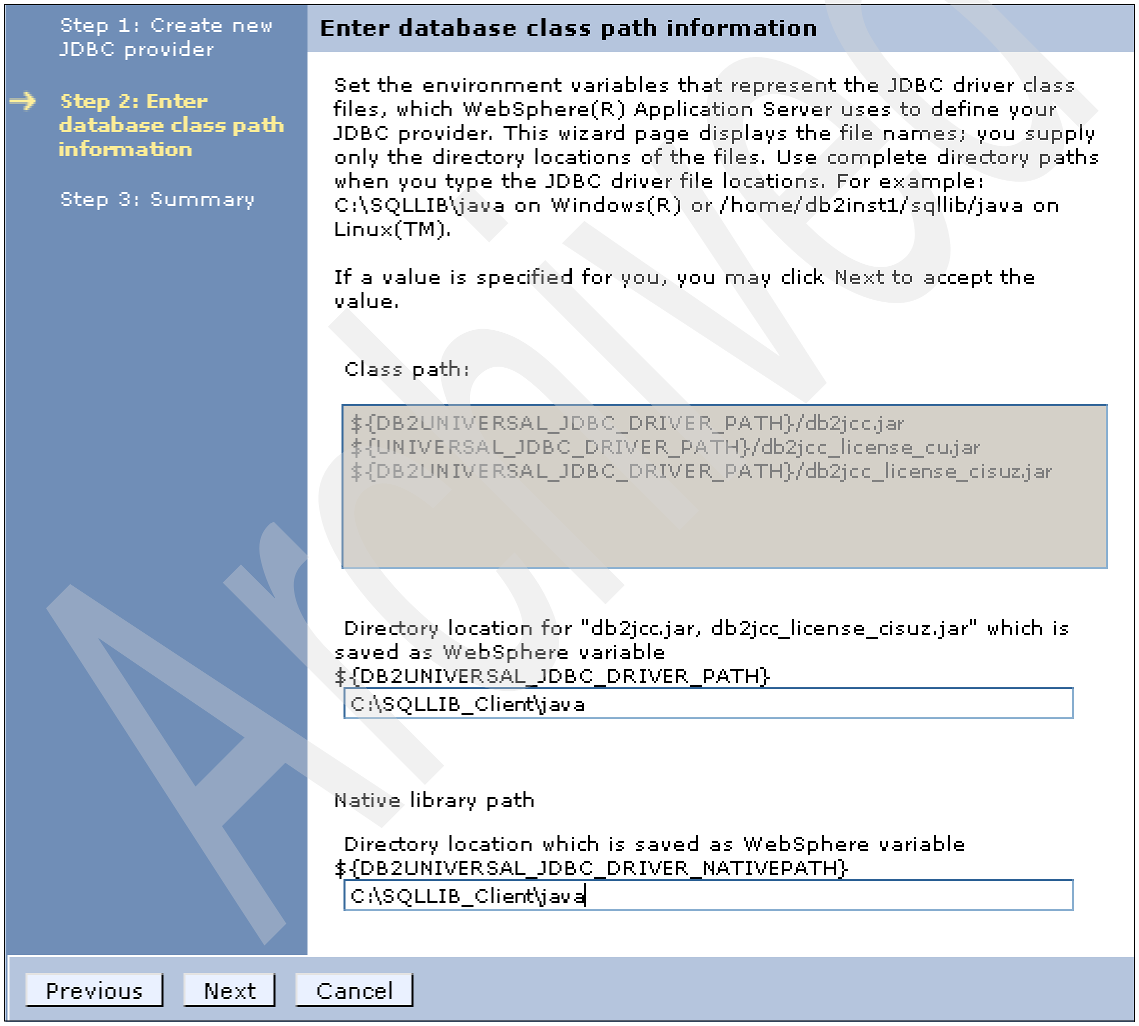
- Implementation type

  Select from the implementation types for the provider type that you selected.
- Name

  Specify a Name for this driver.

Click **Next**.

7. The settings page for your JDBC database class path opens. Figure 5 shows the configuration page for a Universal JDBC Provider.



Step 1: Create new JDBC provider

→ **Step 2: Enter database class path information**

Step 3: Summary

Enter database class path information

Set the environment variables that represent the JDBC driver class files, which WebSphere(R) Application Server uses to define your JDBC provider. This wizard page displays the file names; you supply only the directory locations of the files. Use complete directory paths when you type the JDBC driver file locations. For example: C:\SQLLIB\java on Windows(R) or /home/db2inst1/sqllib/java on Linux(TM).

If a value is specified for you, you may click Next to accept the value.

Class path:

```
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar
```

Directory location for "db2jcc.jar, db2jcc_license_cisuz.jar" which is saved as WebSphere variable ${DB2UNIVERSAL_JDBC_DRIVER_PATH}

C:\SQLLIB_Client\java

Native library path

Directory location which is saved as WebSphere variable ${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}

C:\SQLLIB_Client\java

Figure 5 Define a new JDBC provider: Window 2

Enter the JDBC provider properties:

– Classpath

  This field is a list of paths or JAR file names that together form the location for the resource provider classes. This field is pre-set using variable names that are specific to each type of provider. If you are creating a user-defined provider, specify the entries by pressing Enter between each entry.

The remaining properties are dependent upon the type of provider. They represent the variables that are used in the classpath and their value. If you enter a value for a variable on this panel, the corresponding variables are populated automatically with these values. Conversely, if the variables are already defined, these fields are populated with the variables.

You can view or modify the variables by selecting **Environment** → **WebSphere Variables** in the navigation menu.

Because this example is for DB2, the following fields are available:

– Library path

  This field specifies the values for the global variable UNIVERSAL_JDBC_DRIVER_PATH, which indicates the classpath jar's location.

– Native Library Path

  This field is an optional path to any native libraries. Entries are required if the JBDC provider chosen uses non-Java, or native, libraries. The global variable for this is UNIVERSAL_JDBC_DRIVER_NATIVEPATH.

8. After verifying the settings, click **Finish** to enable the links to create data sources under the Additional Properties section.

Tip: To make a data source available on multiple nodes using different directory structures, complete the following steps using the administrative console:

1. Define the JDBC provider and data source at the cell scope. Use WebSphere environment variables for the classpath and native path.
2. Define the variables at the node scope for each node to specify the driver location for the node.

   For example, ${DRIVER_PATH} can be used for the classpath in the provider definition. You can then define a variable called ${DRIVER_PATH} at the cell scope to act as a default driver location. Then you can override that variable on any node by defining ${DRIVER_PATH} at the node scope. The node-level definition takes precedence over the cell-level definition.

Creating the data source

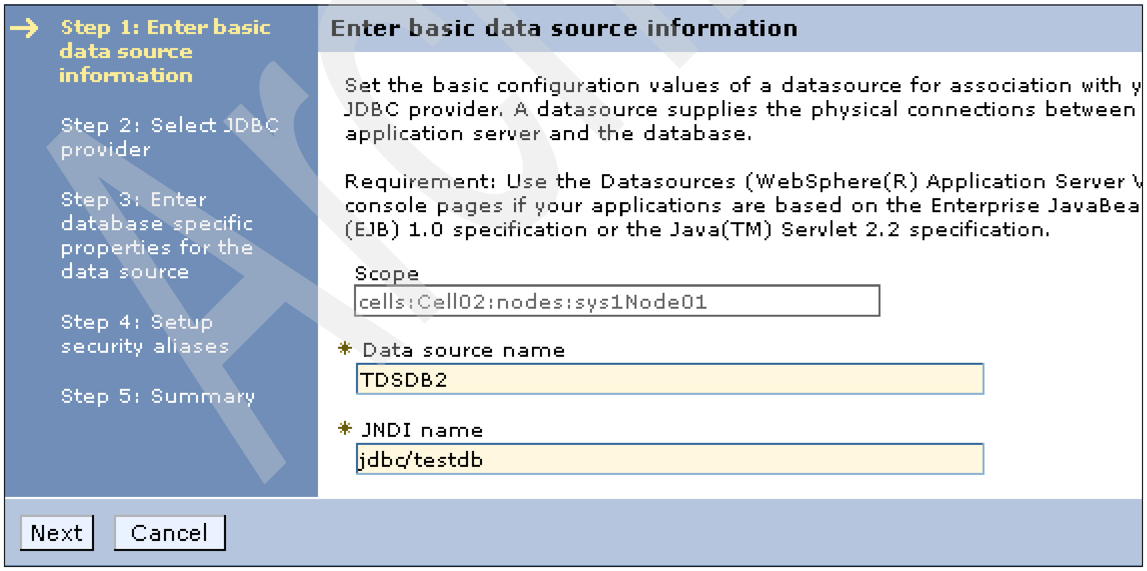
Data sources are associated with a specific JDBC provider and can be viewed or created from the JDBC provider configuration page. You have two options when creating a data source, depending on the J2EE support of the application. Here we discuss creating or modifying data sources for Java EE5, J2EE 1.3, and J2EE 1.4 applications. For information about using data sources with J2EE 1.2 applications, see the topic, *Data sources (Version 4)* in the information center.

The administrative console provides a wizard that helps you create a data source. Keep in mind, however, that although the wizard provides a good way to establish connections quickly, it also establishes default-sized connection pool settings that you need to tune properly before production.

To create a data source, complete the following steps:

1. Expand **Resources** → **JDBC** in the navigation tree, and select **Data sources**.
2. Select the scope. Although you can select **All** to view all resources, you must select a specific scope to create a resource.

   The scope determines which applications can use this data source. We recommend that you select the narrowest scope that is required, while also ensuring that the applications that require the resource can access it.
3. Click **New** to create a new data source and to start a wizard (Figure 6).


Step 1: Enter basic data source information
Step 2: Select JDBC provider
Step 3: Enter database specific properties for the data source
Step 4: Setup security aliases
Step 5: Summary
Next
Cancel

Enter basic data source information

Set the basic configuration values of a datasource for association with y
JDBC provider. A datasource supplies the physical connections between
application server and the database.

Requirement: Use the Datasources (WebSphere(R) Application Server V
console pages if your applications are based on the Enterprise JavaBea
(EJB) 1.0 specification or the Java(TM) Servlet 2.2 specification.

Scope
cells:Cell02:nodes:sys1Node01

* Data source name
TDSDB2

* JNDI name
jdbc/testdb

Figure 6 Data source general properties

Specify the following information:

– Data source name

  This field is a name by which to administer the data source. Use a name that is suggestive of the database name or function.

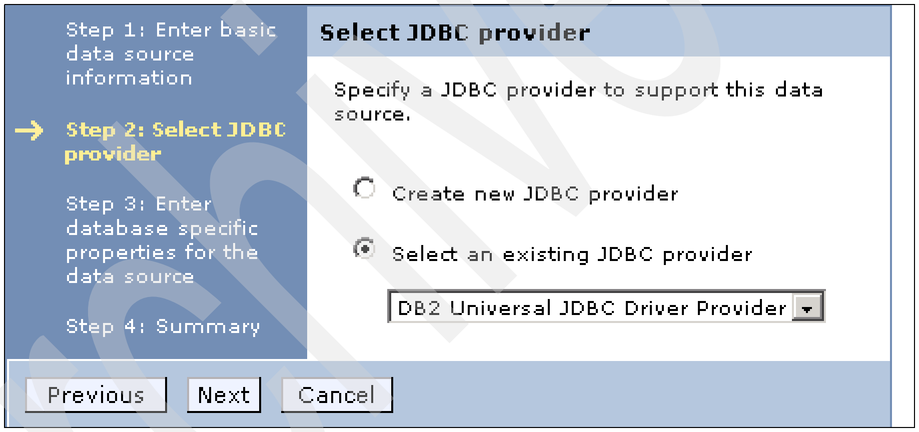
– JNDI name

  This field refers to the data source's name as registered in the application server's name space.

  When installing an application that contains modules with JDBC resource references, the resources need to be bound to the JNDI name of the resources; for example, `jdbc/<`*`database_name`*`>`.

  Click **Next**.

4. Now you need to specify database specific properties, as shown on the right of Figure 7. Click **Next**.



Step 1: Enter basic data source information
Step 2: Select JDBC provider
Step 3: Enter database specific properties for the data source
Step 4: Summary
Select JDBC provider
Previous
Next
Cancel

Specify a JDBC provider to support this data source.

○ Create new JDBC provider

● Select an existing JDBC provider

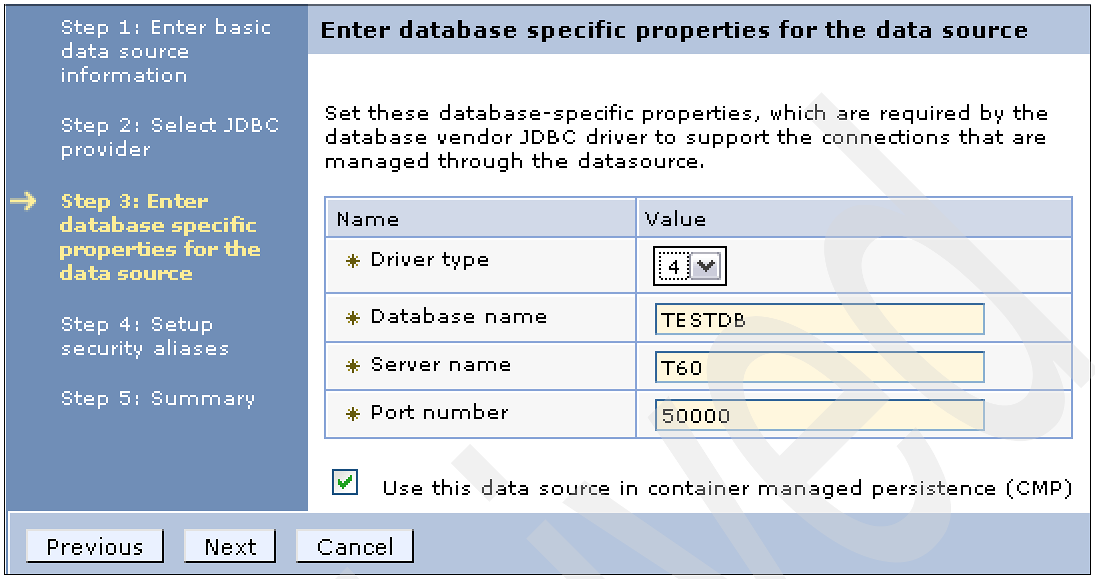
DB2 Universal JDBC Driver Provider

Figure 7 Select a JDBC provider

This window allows you to select a JDBC provider or to create a new one. If you create a new JDBC provider, you will be routed through the windows seen earlier in "Creating the JDBC provider" on page 9. If you select an existing JDBC provider, continue with the next step here.

In this case, we select an existing JDBC provider and click **Next**.

The entries shown in Figure 8 are specific to the JDBC driver and data source type, which show the properties for the Universal data source.


Step 1: Enter basic data source information
Step 2: Select JDBC provider
Step 3: Enter database specific properties for the data source
Step 4: Setup security aliases
Step 5: Summary
Previous
Next
Cancel

Enter database specific properties for the data source

Set these database-specific properties, which are required by the database vendor JDBC driver to support the connections that are managed through the datasource.

Name	Value
* Driver type	4
* Database name	TESTDB
* Server name	T60
* Port number	50000

✓ Use this data source in container managed persistence (CMP)

Figure 8 Database-specific properties

Specify the following information:

– Driver type

  The type of JDBC Driver (2 or 4) used to access the database. To determine the best type of driver to use for your circumstances, consult the documentation for the specific driver that you use.

  In general, however, use type 2 for databases on the same system as the application server and type 4 for remote databases.

– Database Name

  The name of the database (or the cataloged alias).

– Server name and port

  The database server name and its listening port (the default for DB2 is 50000).

– Container managed persistence (CMP)

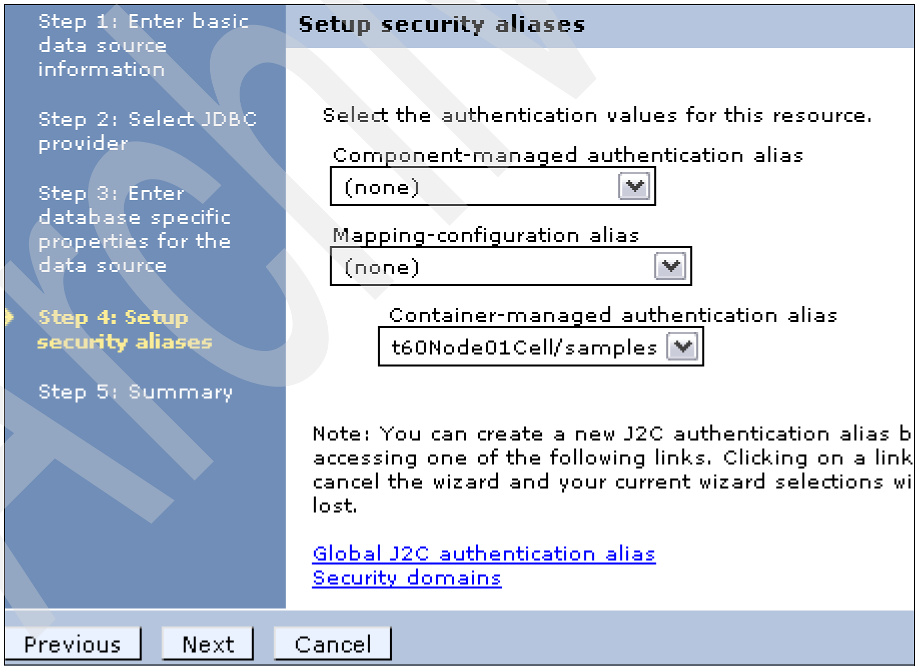
  This field specifies if the data source is to be used for container managed persistence of EJB beans.

Deep-dive: Selecting the “Use this data source in container managed persistence (CMP)” option causes a CMP connection factory that corresponds to this data source to be created for the relational resource adapter. The name of the connector factory that is created is <*datasourcename*>_CF and the connector factory is registered in JNDI under the entry eis/<*jndi_name*>_CMP.

To view the properties of the just created connection factory, select **Resources** → **Resource Adapters** → **Resource Adapters**. Enable the **Show built-in resources** check box in the preferences. Select **WebSphere Relational Resource Adapter** → **CMP Connection Factories**. Be sure to set the scope so that it is the same scope as that for the data source.

Click **Next**.

5. The next step allows you to select or define a J2C authentication alias for the database. The authentication alias simply contains the user ID and password required to access the database (Figure 9).



Step 1: Enter basic data source information
Step 2: Select JDBC provider
Step 3: Enter database specific properties for the data source
Step 4: Setup security aliases
Step 5: Summary
Previous
Next
Cancel

Setup security aliases

Select the authentication values for this resource.

Component-managed authentication alias: (none)

Mapping-configuration alias: (none)

Container-managed authentication alias: t60Node01Cell/samples

Note: You can create a new J2C authentication alias b accessing one of the following links. Clicking on a link cancel the wizard and your current wizard selections wi lost.

Global J2C authentication alias
Security domains

Figure 9 Specify the authentication alias

Click **Next**.

6. A summary of the options that you chose displays. Click **Next** to create the data source.

The new data source is listed in the table of resources. You can test the new connection by checking the box to the left of the data source and clicking **Test Connection**. You can view or modify settings for the new data source by clicking the name in the resources list.

Example: Connecting to an Oracle database

This example illustrates a connection to an Oracle Express 10g database.

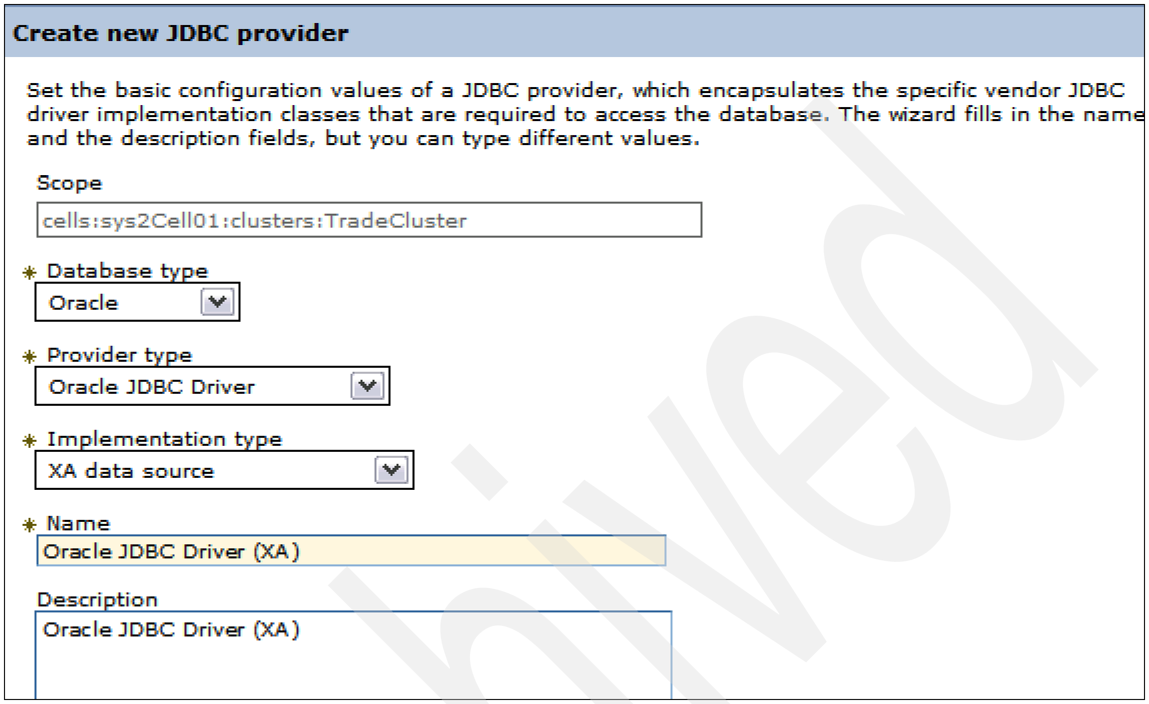
Ensure that the implementation classes for the provider are available to the system. The class files need to be located on each system where the application servers will run.

Creating the JDBC provider

Follow these steps to create the JDBC provider:

1. In the administrative console, expand **Resources** → **JDBC** from the navigation tree.
2. Click **JDBC Providers**.
3. Select the scope. (Although you can select All scopes to view all resources, you must select a specific scope to create a resource.)
4. Select **New** to start the wizard and to create a new JDBC provider.

5. In step 1 of the wizard, define the type of provider that you will use. See Figure 10.


Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulates the specific vendor JDBC driver implementation classes that are required to access the database. The wizard fills in the name and the description fields, but you can type different values.

Scope: cells:sys2Cell01:clusters:TradeCluster

* Database type: Oracle

* Provider type: Oracle JDBC Driver

* Implementation type: XA data source

* Name: Oracle JDBC Driver (XA)

Description: Oracle JDBC Driver (XA)

Figure 10 Define a new Oracle JDBC provider, Step 1

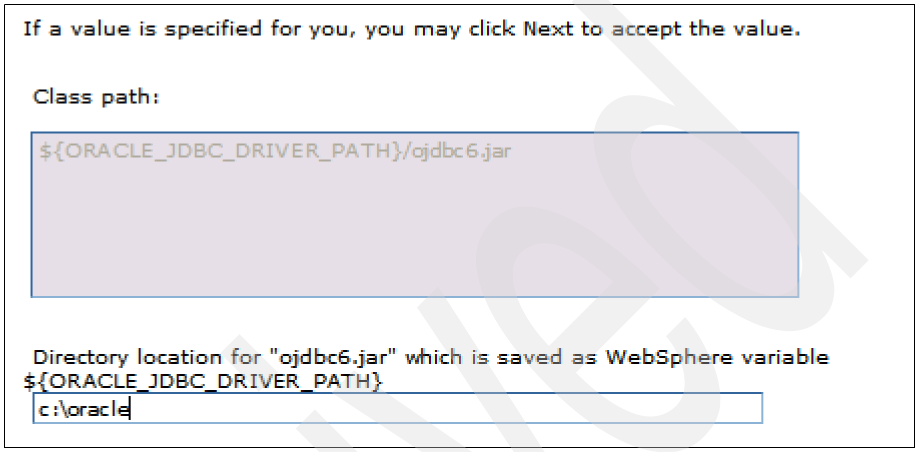
The database type is Oracle and provider type is Oracle JDBC driver.

Options of implementation type are XA data source or connection pool data source. XA data source types support two-phase commit transactions.

Click **Next**.

6. In the next panel (Figure 11), enter the directory location for the Oracle JDBC drivers.

   In this example, the `ojdbc6.jar` is assumed by the wizard. However, the database requires the `ojdbc14.jar` driver. For now, we complete the wizard to define the driver, and then alter the driver name in the configuration page.



If a value is specified for you, you may click Next to accept the value.

Class path:

${ORACLE_JDBC_DRIVER_PATH}/ojdbc6.jar

Directory location for "ojdbc6.jar" which is saved as WebSphere variable ${ORACLE_JDBC_DRIVER_PATH}

c:\oracle

Figure 11 Define a new Oracle JDBC provider, Step 2

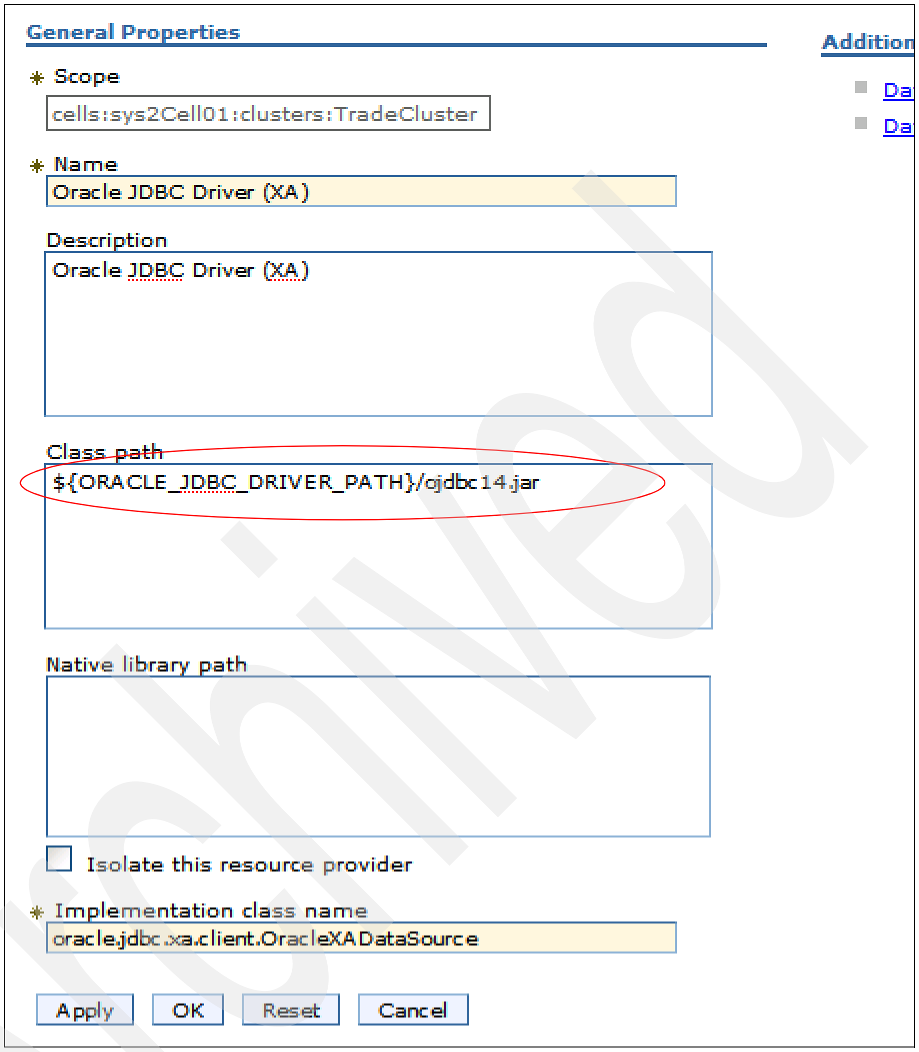
   If you have predefined the ORACLE_JDBC_DRIVER_PATH variable, the driver location is already entered. If you enter a value here, it is saved in the variable.

   Click **Next**.

7. Review the summary of the settings, and click **Finish**. The new JDBC provider displays in the list of providers.
8. Click the JDBC provider name to open the configuration page (Figure 12). Remember that the wizard assumes that the `ojdbc6.jar` driver is used.

   Change the class path field to point to the `ojdbc14.jar`.

   The implementation class name stays the same.


General Properties

Addition

Da

Da

* Scope

cells:sys2Cell01:clusters:TradeCluster

* Name

Oracle JDBC Driver (XA)

Description

Oracle JDBC Driver (XA)

Class path

${ORACLE_JDBC_DRIVER_PATH}/ojdbc14.jar

Native library path

Isolate this resource provider

* Implementation class name

oracle.jdbc.xa.client.OracleXADataSource

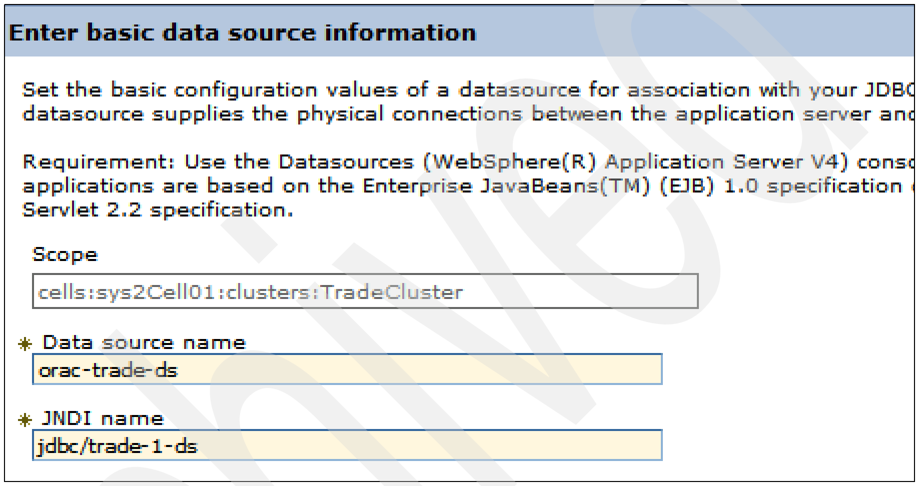
Figure 12 Configure the class path

Click **OK**.

Creating the data source

To create a data source, complete the following steps:

1. Expand **Resources** → **JDBC** in the navigation tree, and select **Data sources**.
2. Select the scope. Although you can select **All** to view all resources, you must select a specific scope to create a resource.
3. Click **New** to create a new data source and to start a wizard (Figure 13).



Enter basic data source information

Set the basic configuration values of a datasource for association with your JDB
datasource supplies the physical connections between the application server an

Requirement: Use the Datasources (WebSphere(R) Application Server V4) cons
applications are based on the Enterprise JavaBeans(TM) (EJB) 1.0 specification
Servlet 2.2 specification.

Scope
cells:sys2Cell01:clusters:TradeCluster

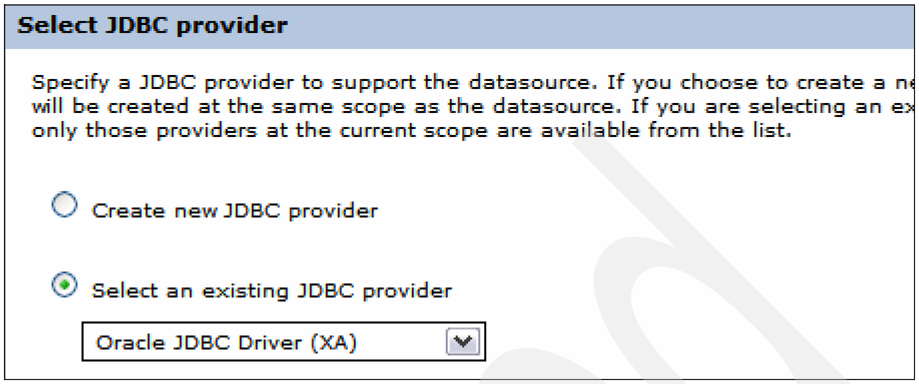
* Data source name
orac-trade-ds

* JNDI name
jdbc/trade-1-ds

Figure 13 Create a data source, Step 1

Enter a name for the new data source. This is used for administrative purposes. Enter the JNDI name that will be used to access the data source and click **Next**.

4. In the next panel (Figure 14), select the Oracle JDBC driver and click **Next**.



Select JDBC provider

Specify a JDBC provider to support the datasource. If you choose to create a ne
will be created at the same scope as the datasource. If you are selecting an ex
only those providers at the current scope are available from the list.

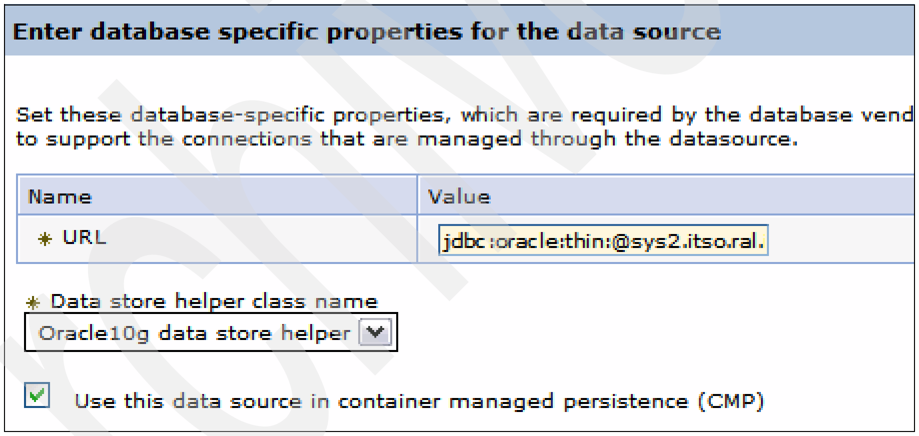
( ) Create new JDBC provider

(●) Select an existing JDBC provider

Oracle JDBC Driver (XA)

Figure 14 Create a data source, Step 2

5. Enter the properties for the database, as shown in Figure 15.



Enter database specific properties for the data source

Set these database-specific properties, which are required by the database vend
to support the connections that are managed through the datasource.

Name	Value
* URL	jdbc:oracle:thin:@sys2.itso.ral.

* Data store helper class name

Oracle10g data store helper

✓ Use this data source in container managed persistence (CMP)

Figure 15 Create a data source, Step 3

Specify the following information:

– The URL for the connection to the XE database is in the following format:

  jdbc:oracle:thin:@*host_name:port:service*

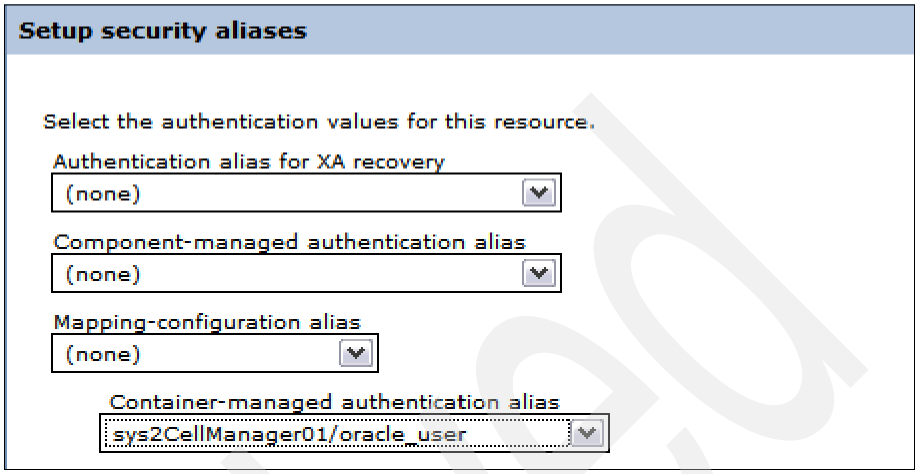
  In this case:

  `jdbc:oracle:thin:@sys2.itso.ral.ibm.com:1521:XE`

– Select the data store helper class name.

Click **Next**.

6. In the next panel (Figure 16), select the authentication alias that will provide the user ID and password required to access the database. Click **Next**.



Setup security aliases

Select the authentication values for this resource.

Authentication alias for XA recovery: (none)

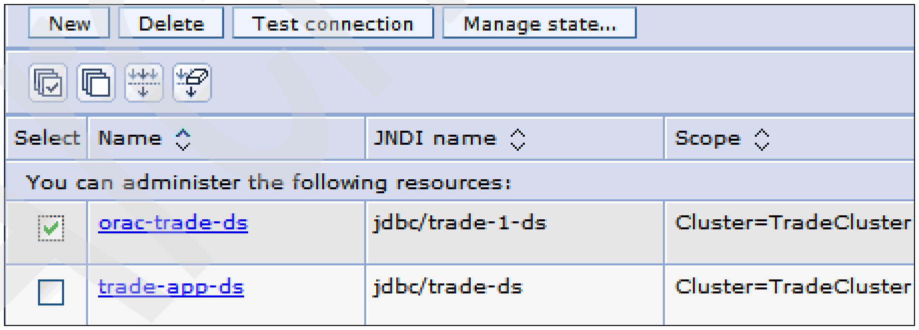
Component-managed authentication alias: (none)

Mapping-configuration alias: (none)

Container-managed authentication alias: sys2CellManager01/oracle_user

Figure 16 Create a data source, Step 4

7. Review the summary of your selections, and click **Finish**.
8. When the data source creation is complete, save the configuration and synchronize the changes with the nodes.
9. Test the new connection by selecting the new data source and clicking **Test connection** (Figure 17).



Select	Name	JNDI name	Scope
You can administer the following resources:			
✓	orac-trade-ds	jdbc/trade-1-ds	Cluster=TradeCluster
☐	trade-app-ds	jdbc/trade-ds	Cluster=TradeCluster

Figure 17 Test the connection

Tip: If you receive the following error, make sure that you have adjusted the JDBC provider to use the correct implementation JAR file (step 9):

```
DSRA8040I: Failed to connect to the DataSource.  Encountered "":
java.sql.SQLException: Invalid argument(s) in callDSRA0010E: SQL
State = 99999, Error Code = 17,068
```

Example: Connecting to an SQL Server database

This example illustrates a connection to a Microsoft® SQL Server Express 2005 database.

Ensure that the implementation classes for the provider are available to the system. The class files need to be located on each system where the application servers will run.

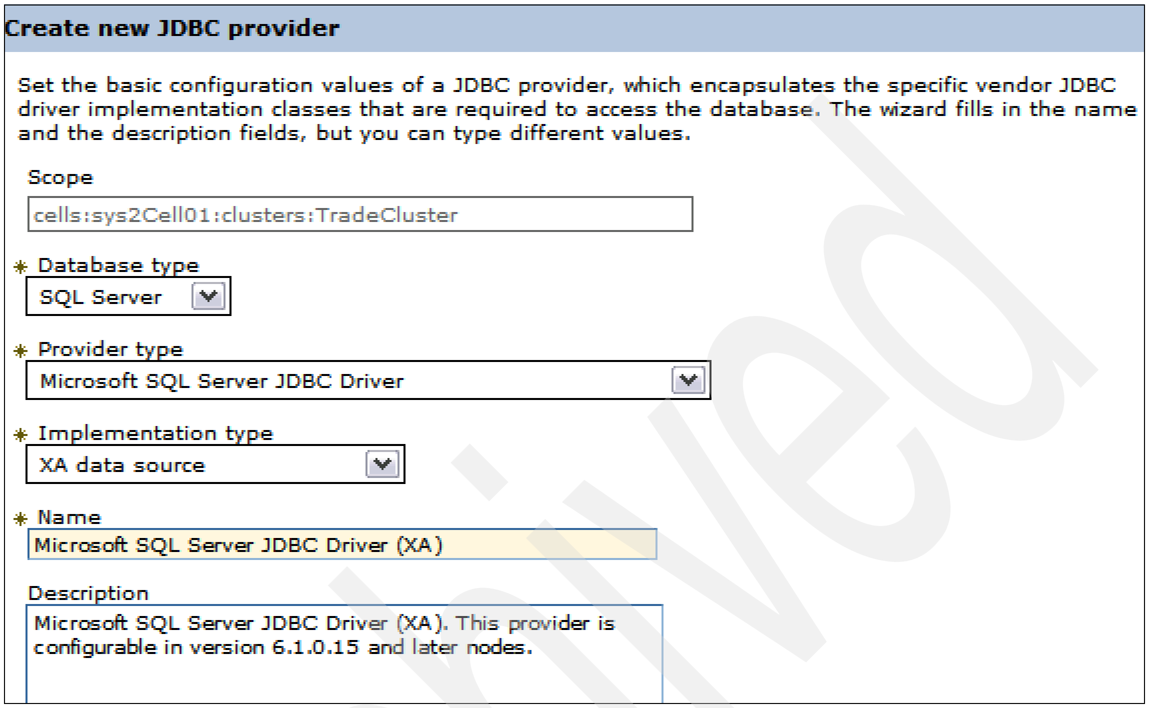
Although some JDBC drivers are bundled with WebSphere Application Server to facilitate quick connectivity, in general, JDBC drivers are provided by the database vendor. Information about the location and features of the JDBC provider is provided by the database vendor versus the WebSphere documentation.

Creating the JDBC provider

To create a JDBC provider:

1. In the administrative console, expand **Resources** → **JDBC** from the navigation tree.
2. Click **JDBC Providers**.
3. Select the scope. (Although you can select All scopes to view all resources, you must select a specific scope to create a resource.)
4. Select **New** to start the wizard to create a new JDBC provider.

5. In step 1 of the wizard, define the type of provider that you will use. See Figure 18.


Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulates the specific vendor JDBC driver implementation classes that are required to access the database. The wizard fills in the name and the description fields, but you can type different values.

Scope: cells:sys2Cell01:clusters:TradeCluster

* Database type: SQL Server

* Provider type: Microsoft SQL Server JDBC Driver

* Implementation type: XA data source

* Name: Microsoft SQL Server JDBC Driver (XA)

Description: Microsoft SQL Server JDBC Driver (XA). This provider is configurable in version 6.1.0.15 and later nodes.

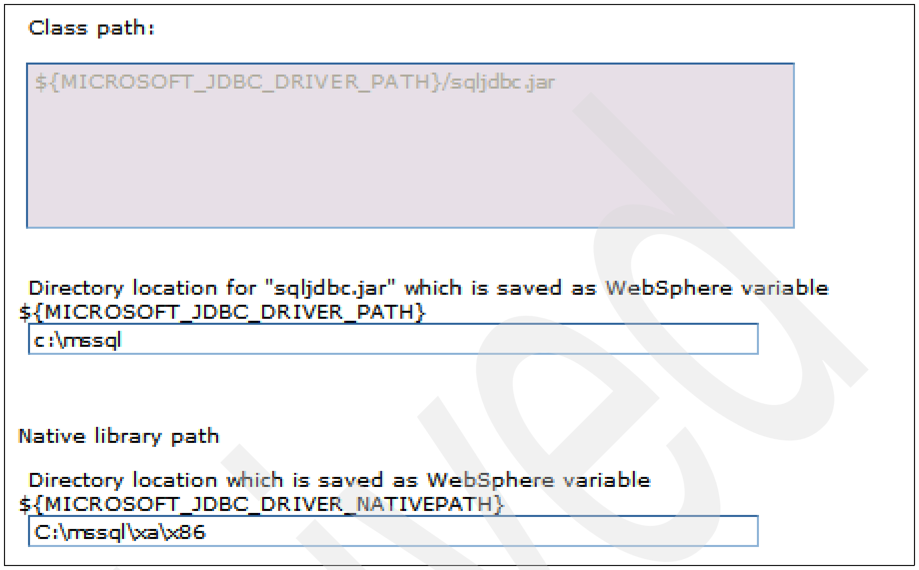
Figure 18 Define a new SQL Server JDBC provider, Window1

The database type is SQL Server and provider type is Microsoft SQL Server JDBC driver.

Options of implementation type are XA data source or connection pool data source. XA data source types support two-phase commit transactions.

Click **Next**.

6. In the next panel (Figure 19), enter the directory location for the SQL Server JDBC drivers.


Class path:

${MICROSOFT_JDBC_DRIVER_PATH}/sqljdbc.jar

Directory location for "sqljdbc.jar" which is saved as WebSphere variable ${MICROSOFT_JDBC_DRIVER_PATH}

c:\mssql

Native library path

Directory location which is saved as WebSphere variable ${MICROSOFT_JDBC_DRIVER_NATIVEPATH}

C:\mssql\xa\x86

Figure 19 Define a new SQL Server JDBC provider, Window 2

If you have predefined the variables used here, the driver locations are entered already. If you enter a value here, it is saved in the appropriate variable.

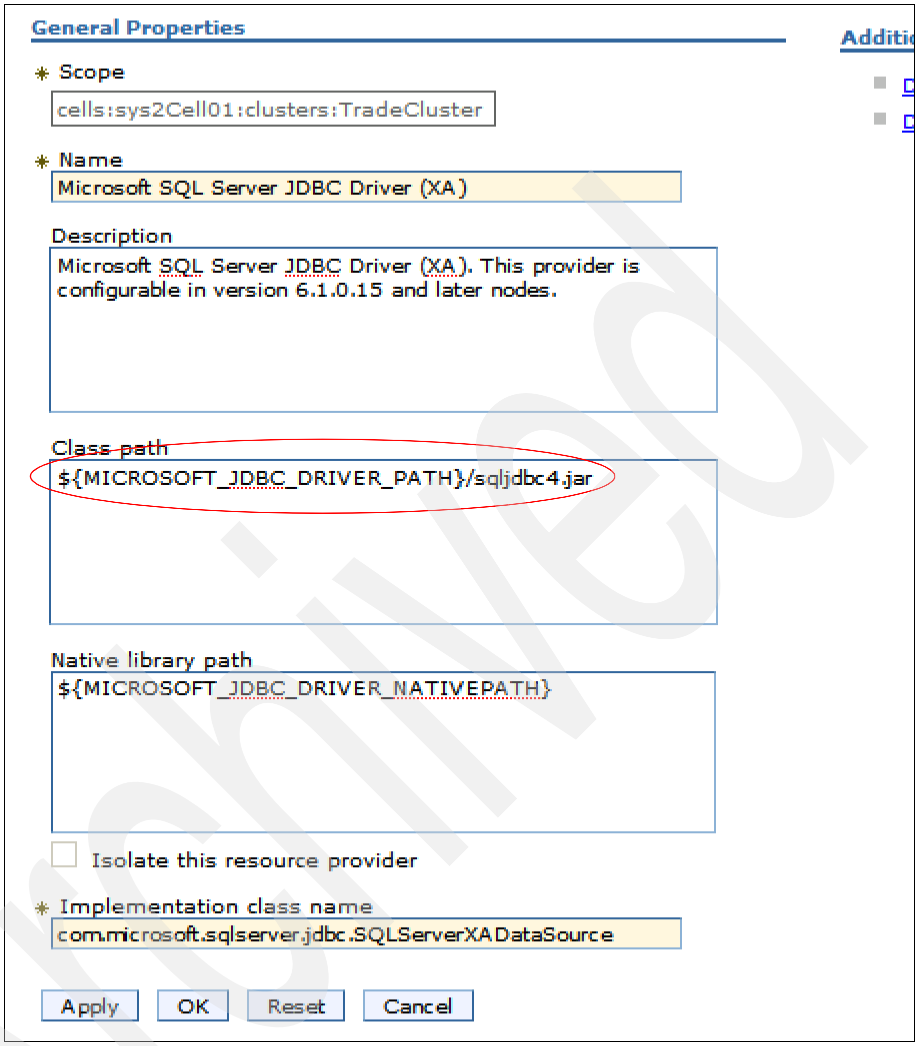
Click **Next**.

7. Review the summary of the settings, and click **Finish**. The new JDBC provider displays in the list of providers.

8. Click the JDBC provider name to open the configuration page (Figure 20). Remember that the wizard assumes that the `sqljdbc.jar` driver is used in the class path.

   Change the class path field to point to the `sqljdbc4.jar`.

   The implementation class name stays the same.


General Properties

Additi

* Scope
cells:sys2Cell01:clusters:TradeCluster

* Name
Microsoft SQL Server JDBC Driver (XA)

Description
Microsoft SQL Server JDBC Driver (XA). This provider is configurable in version 6.1.0.15 and later nodes.

Class path
${MICROSOFT_JDBC_DRIVER_PATH}/sqljdbc4.jar

Native library path
${MICROSOFT_JDBC_DRIVER_NATIVEPATH}

☐ Isolate this resource provider

* Implementation class name
com.microsoft.sqlserver.jdbc.SQLServerXADataSource

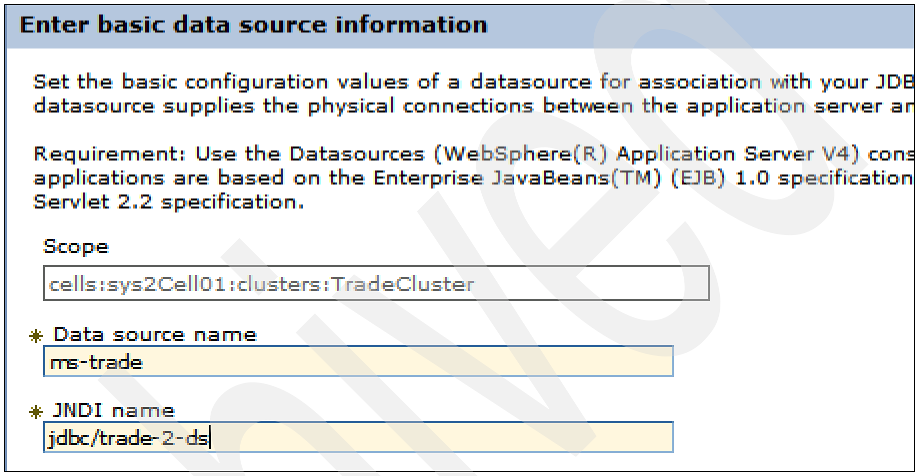
Figure 20   Configure the class path

Click **OK**.

Creating the data source

To create a data source, complete the following steps:

1. Expand **Resources** → **JDBC** in the navigation tree and select **Data sources**.
2. Select the scope. Although you can select **All** to view all resources, you must select a specific scope to create a resource.
3. Click **New** to create a new data source and to start a wizard (Figure 21).



Enter basic data source information

Set the basic configuration values of a datasource for association with your JDB
datasource supplies the physical connections between the application server ar

Requirement: Use the Datasources (WebSphere(R) Application Server V4) cons
applications are based on the Enterprise JavaBeans(TM) (EJB) 1.0 specification
Servlet 2.2 specification.

Scope
cells:sys2Cell01:clusters:TradeCluster

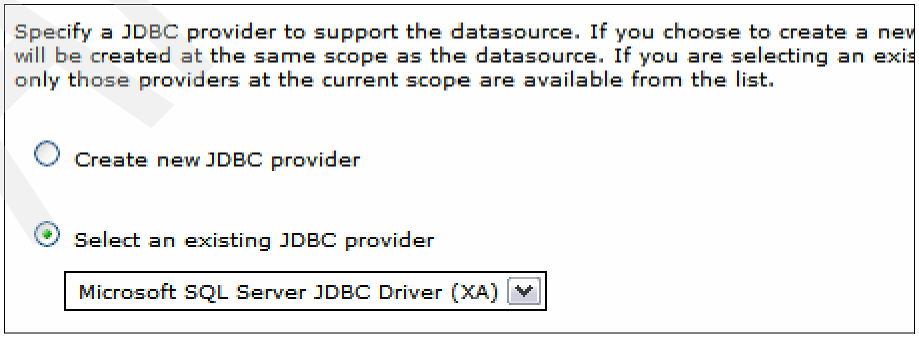
* Data source name
ms-trade

* JNDI name
jdbc/trade-2-ds

Figure 21 Create a data source, Step 1

Enter a name for the new data source. This name is used for administrative purposes. Enter the JNDI name that will be used to access the data source, and click **Next**.

4. In the next panel (Figure 22), select the Microsoft SQL Server JDBC driver, and click **Next**.



Specify a JDBC provider to support the datasource. If you choose to create a new
will be created at the same scope as the datasource. If you are selecting an exis
only those providers at the current scope are available from the list.

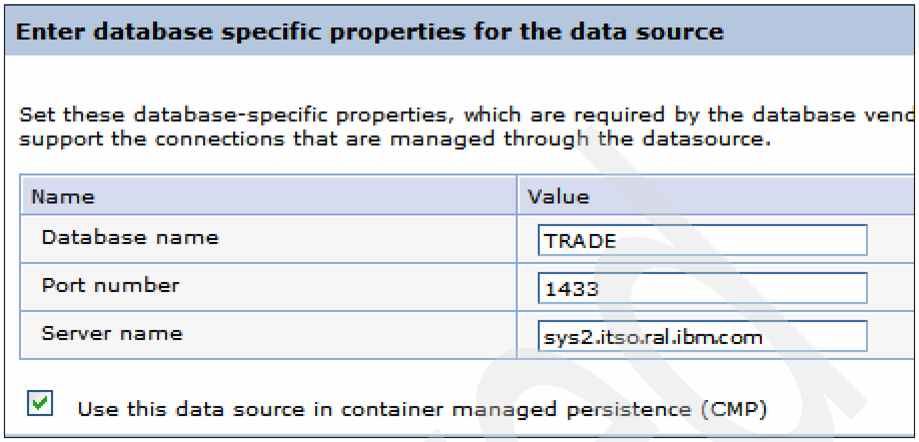
○ Create new JDBC provider

● Select an existing JDBC provider

Microsoft SQL Server JDBC Driver (XA)

Figure 22 Create a data source, Step 2

5. Enter the properties for the database (Figure 23).



Enter database specific properties for the data source

Set these database-specific properties, which are required by the database vend
support the connections that are managed through the datasource.

Name	Value
Database name	TRADE
Port number	1433
Server name	sys2.itso.ral.ibm.com

✓ Use this data source in container managed persistence (CMP)

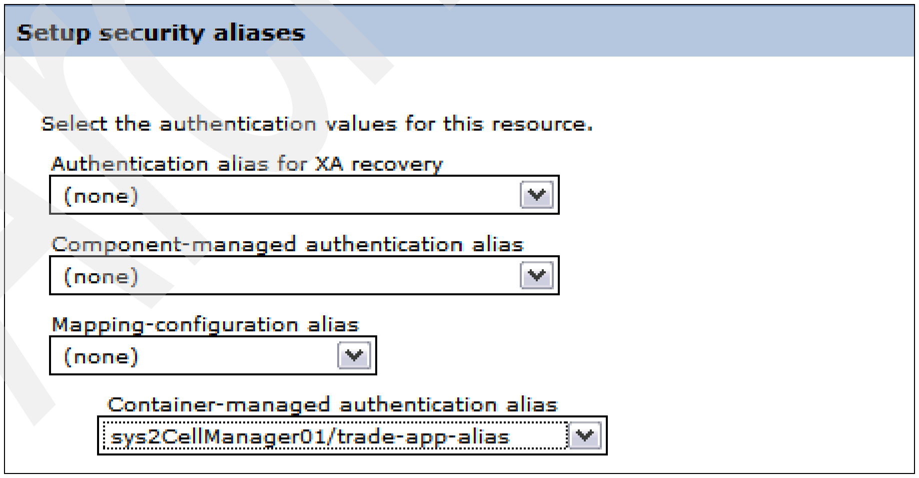
Figure 23 Create a data source, Step 3

Specify the following information:

– Enter the database name.
– Enter the port number the database server listens on.
– Enter the host name of the SQL Server installation.

Click **Next**.

6. In the next panel (Figure 24), select the authentication alias that provides the user ID and password that are required to access the database. Click **Next**.



Setup security aliases

Select the authentication values for this resource.

Authentication alias for XA recovery: (none)

Component-managed authentication alias: (none)

Mapping-configuration alias: (none)

Container-managed authentication alias: sys2CellManager01/trade-app-alias

Figure 24 Create a data source, Step 4

7. Review the summary of your selections, and click **Finish**.
8. When the data source creation is complete, save the configuration, and synchronize the changes with the nodes.
9. Test the new connection by selecting the new data source and clicking **Test connection**.

Example: Connecting to an Informix Dynamic Server database

This example illustrates a connection to an Informix Dynamic Server (IDS) database using the Informix JDBC driver.

Before starting the configuration, ensure that the implementation classes for the provider are available to the system. The class files need to be located on each system where the application servers will run.

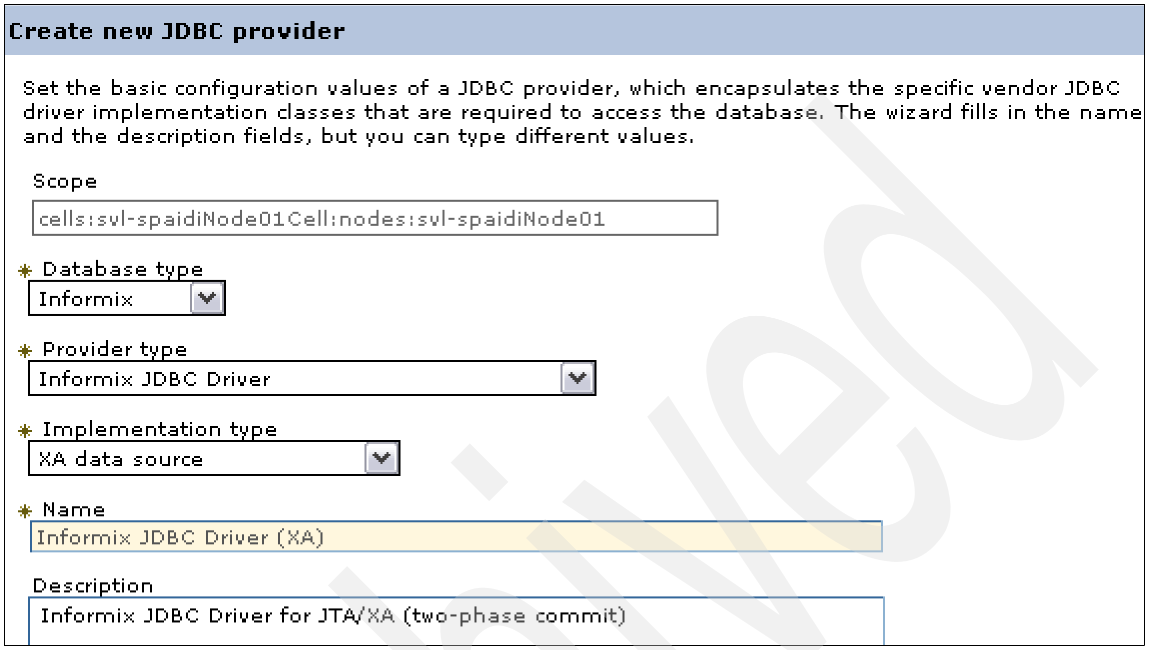
Also make sure that an authentication alias is created for the user ID that will be used to connect to the database. For more information, see "Creating an authentication alias" on page 8.

Creating the JDBC provider

Follow these steps to create the JDBC provider:

1. In the administrative console, expand **Resources** → **JDBC** from the navigation tree.
2. Click **JDBC Providers**.
3. Select the scope. (Although you can select **All scopes** to view all resources, you must select a specific scope to create a resource.)
4. Select **New** to start the wizard to create a new JDBC provider.

5. In step 1 of the wizard, define the type of provider that you will use. See Figure 25.



Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulates the specific vendor JDBC driver implementation classes that are required to access the database. The wizard fills in the name and the description fields, but you can type different values.

Scope: cells:svl-spaidiNode01Cell:nodes:svl-spaidiNode01

* Database type: Informix

* Provider type: Informix JDBC Driver

* Implementation type: XA data source

* Name: Informix JDBC Driver (XA)

Description: Informix JDBC Driver for JTA/XA (two-phase commit)

Figure 25 Define a new Informix JDBC provider, Step 1

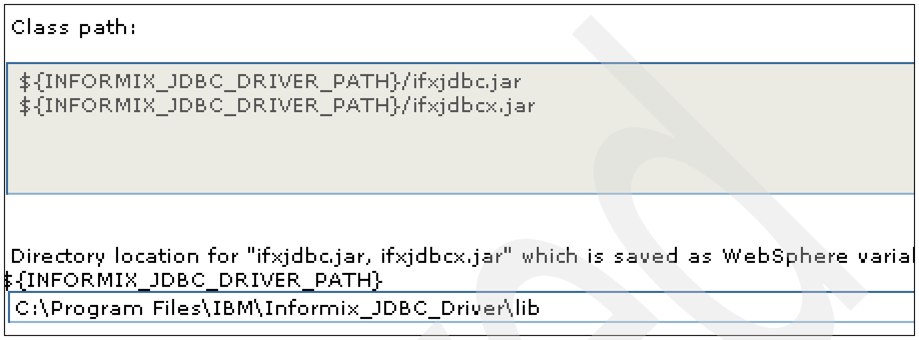
The database type is Informix and provider type is Informix JDBC driver.

Options of implementation type are XA data source or connection pool data source. XA data source types support two-phase commit transactions.

Click **Next**.

6. In the next panel (Figure 26), enter the directory location for the Informix JDBC drivers.

   In this example, the `ifxjdbc.jar` and `ifxjdbcx.jar` are assumed by the wizard.



```
Class path:

${INFORMIX_JDBC_DRIVER_PATH}/ifxjdbc.jar
${INFORMIX_JDBC_DRIVER_PATH}/ifxjdbcx.jar

Directory location for "ifxjdbc.jar, ifxjdbcx.jar" which is saved as WebSphere varial
${INFORMIX_JDBC_DRIVER_PATH}
C:\Program Files\IBM\Informix_JDBC_Driver\lib
```

Figure 26 Define a new Informix JDBC provider, Step 2

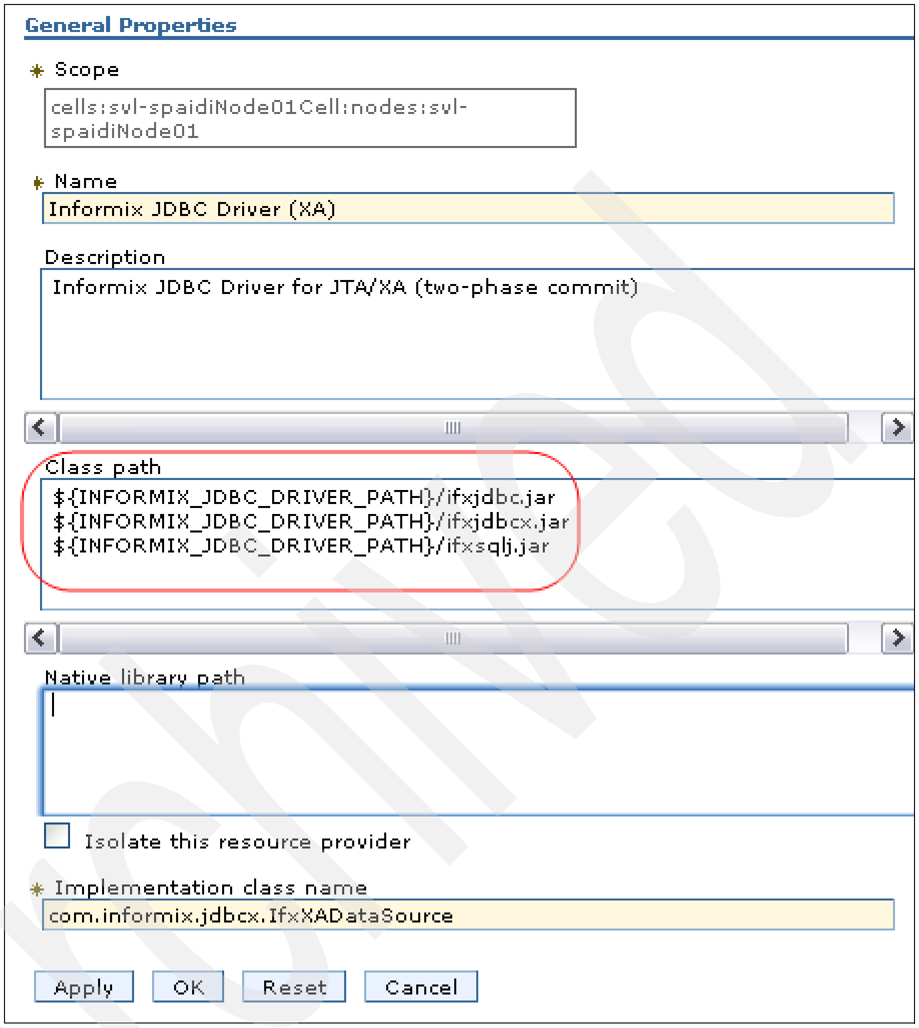
   If you have predefined the INFORMIX_JDBC_DRIVER_PATH variable, the driver location is already entered. If you enter a value here, it is saved in the variable.

   Click **Next**.

7. Review the summary of the settings, and click **Finish**. The new JDBC provider displays in the list of providers.

8. Click the JDBC provider name to open the configuration page (Figure 27). If you plan to use SQLJ for queries, change the class path field to add the `ifxsqlj.jar` file.

   The implementation class name stays the same.


General Properties

* Scope
cells:svl-spaidiNode01Cell:nodes:svl-spaidiNode01

* Name
Informix JDBC Driver (XA)

Description
Informix JDBC Driver for JTA/XA (two-phase commit)

Class path
```
${INFORMIX_JDBC_DRIVER_PATH}/ifxjdbc.jar
${INFORMIX_JDBC_DRIVER_PATH}/ifxjdbcx.jar
${INFORMIX_JDBC_DRIVER_PATH}/ifxsqlj.jar
```

Native library path

☐ Isolate this resource provider

* Implementation class name
com.informix.jdbcx.IfxXADataSource

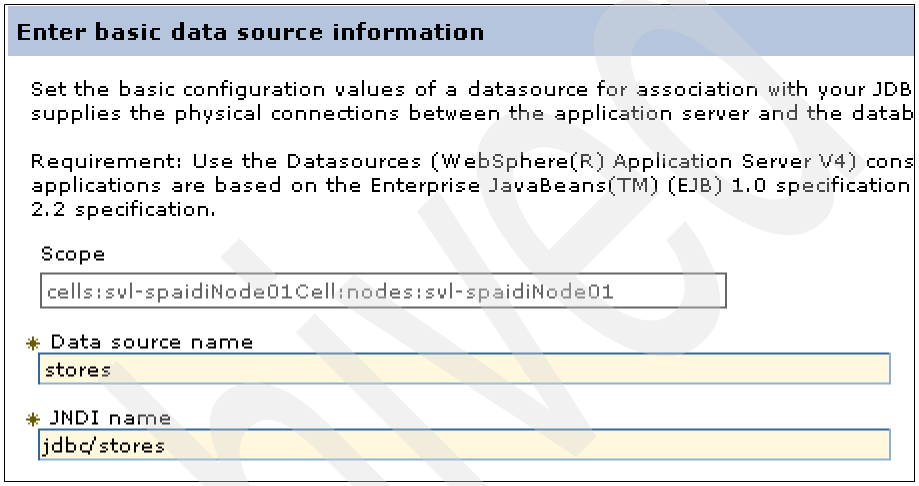
Figure 27 Configure the class path

Click **OK**.

Creating the data source

To create a data source, complete the following steps:

1. Expand **Resources** → **JDBC** in the navigation tree, and click **Data sources**.
2. Select the scope on the right. Although you can select **All scopes** to view all resources, you must select a specific scope to create a resource.
3. Click **New** to create a new data source and to start a wizard (Figure 28).



Enter basic data source information

Set the basic configuration values of a datasource for association with your JDB
supplies the physical connections between the application server and the datab

Requirement: Use the Datasources (WebSphere(R) Application Server V4) cons
applications are based on the Enterprise JavaBeans(TM) (EJB) 1.0 specification
2.2 specification.

Scope
cells:svl-spaidiNode01Cell:nodes:svl-spaidiNode01

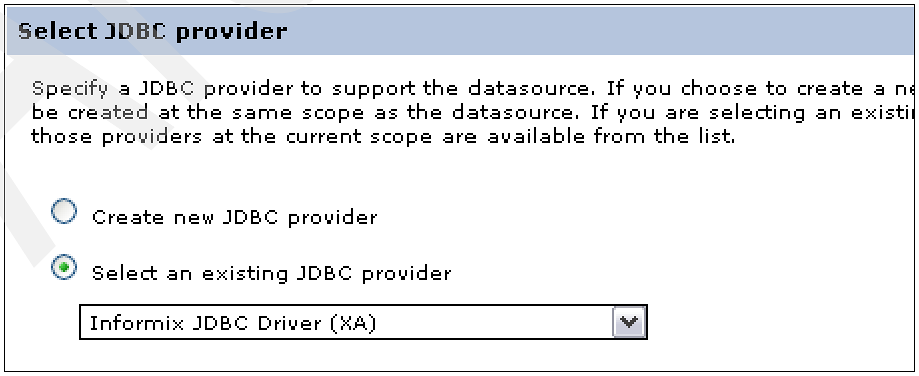
* Data source name
stores

* JNDI name
jdbc/stores

Figure 28 Create a data source, Step 1

Enter a name for the new data source. This name is used for administrative purposes. Enter the JNDI name that will be used to access the data source, and click **Next**.

4. In the next panel (Figure 29), select the Informix JDBC driver, and click **Next**.



Select JDBC provider

Specify a JDBC provider to support the datasource. If you choose to create a ne
be created at the same scope as the datasource. If you are selecting an existi
those providers at the current scope are available from the list.

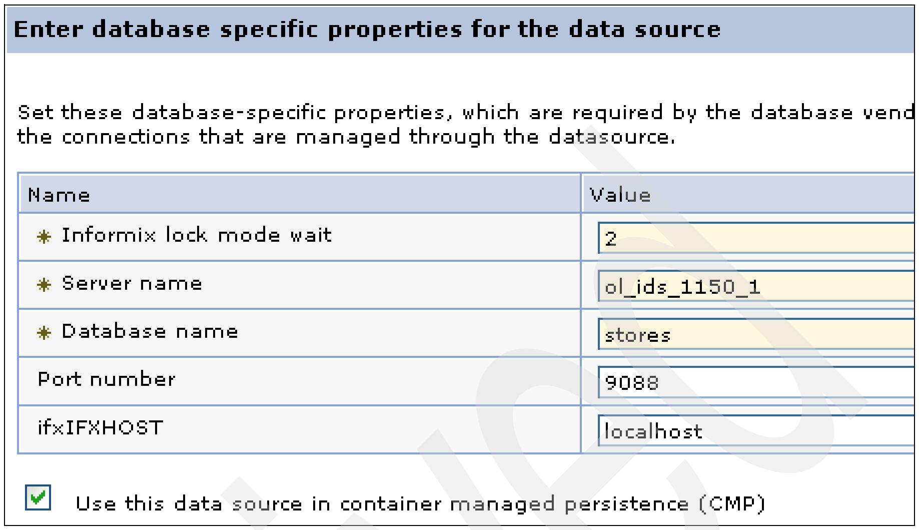
○ Create new JDBC provider

[✓ Select an existing JDBC provider]

Informix JDBC Driver (XA)

Figure 29 Create a data source, Step 2

5. Enter the properties for the database, as shown in Figure 30.



Enter database specific properties for the data source

Set these database-specific properties, which are required by the database vend
the connections that are managed through the datasource.

Name	Value
* Informix lock mode wait	2
* Server name	ol_ids_1150_1
* Database name	stores
Port number	9088
ifxIFXHOST	localhost

✓ Use this data source in container managed persistence (CMP)

Figure 30 Create a data source, Step 3

Specify the following information:

– Enter the Informix lock mode wait. Default is 2.

– Enter the server name. This name is the INFORMIXSERVER value, that is the Informix instance name.

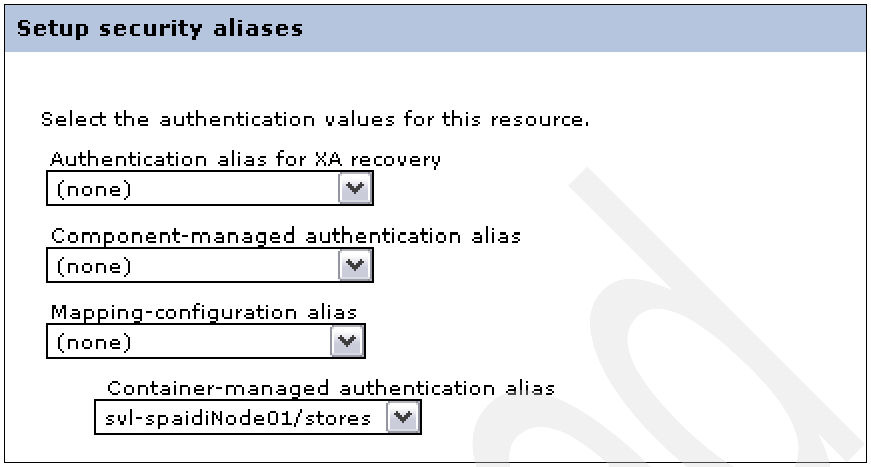
– Enter the database name.

– Enter the port number. This number is the olsoctcp protocol port number. Check your SQLHOSTS file on UNIX®, Linux®, or Windows® registry on Windows for the correct value to enter.

– Enter the ifxIFXHOST name. This name is host name or the IP Address of the host that is running your Informix instance.

Click **Next**.

6. In the next panel (Figure 31), select the authentication alias that provides the user ID and password required to access the database. In this step, we assume that an authentication alias is created already. Click **Next**.



Setup security aliases

Select the authentication values for this resource.

Authentication alias for XA recovery: (none)

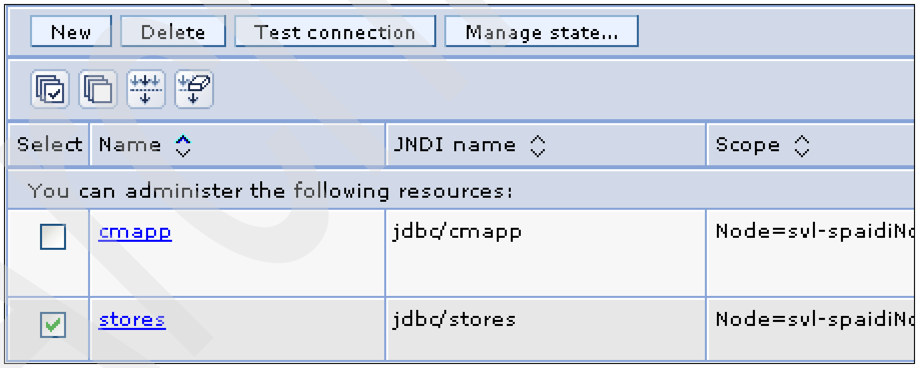
Component-managed authentication alias: (none)

Mapping-configuration alias: (none)

Container-managed authentication alias: svl-spaidiNode01/stores

Figure 31   Create a data source, Step 4

7. Review the summary of your selections, and click **Finish**.
8. When the data source creation is complete, save the configuration, and synchronize the changes with the nodes.
9. Test the new connection by selecting the new data source and clicking **Test connection** (Figure 32).



Select	Name	JNDI name	Scope
You can administer the following resources:			
☐	cmapp	jdbc/cmapp	Node=svl-spaidiN
✓	stores	jdbc/stores	Node=svl-spaidiN

Figure 32   Test the connection

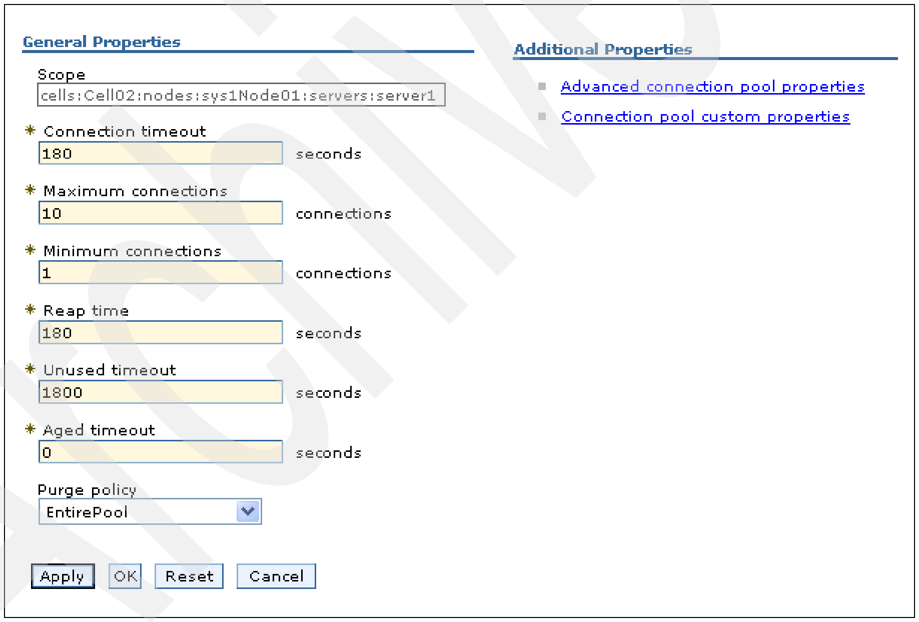
Configuring connection pooling properties

Performance of an application that connects to a database can be greatly affected by the availability of connections to the database and how those connections affect the performance of the database itself. There are no simple rules that tell you how to configure the connection pool properties. Your configuration is highly dependent on application, network, and database characteristics. You need to coordinate the values that you specify in WebSphere closely with the database administrator.

Remember to include all resources in capacity planning. If 10 applications all connect to a database using separate connection pools of 10 maximum connections, this means that there is a theoretical possibility of 100 concurrent connections to the database. Make sure that the database server has sufficient memory and processing capacity to support this requirement.

To access the connection pool properties:

1. Navigate to **Resources** → **JDBC** → **Data sources**, and click the data source name.
2. Select **Connection pool properties** in the Additional Properties section, which opens the panel shown in Figure 33.


General Properties

Scope: cells:Cell02:nodes:sys1Node01:servers:server1

* Connection timeout: 180 seconds

* Maximum connections: 10 connections

* Minimum connections: 1 connections

* Reap time: 180 seconds

* Unused timeout: 1800 seconds

* Aged timeout: 0 seconds

Purge policy: EntirePool

Apply
OK
Reset
Cancel
Additional Properties
Advanced connection pool properties
Connection pool custom properties

Figure 33 Data source connection pool properties

Specify the following information:

- Connection Timeout

  Specify the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown. This can occur when the pool is at its maximum (Max Connections) and all of the connections are in use by other applications for the duration of the wait.

  For example, if Connection Timeout is set to 300 and the maximum number of connections is reached, the Pool Manager waits for 300 seconds for an available physical connection. If a physical connection is not available within this time, the Pool Manager throws a ConnectionWaitTimeoutException.

  Tip: If Connection Timeout is set to 0, the pool manager waits as long as necessary until a connection is allocated.

- Max Connections

  Specify the maximum number of physical connections that can be created in this pool.

  These connections are the physical connections to the back-end database. After this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use is returned to the pool, or a ConnectionWaitTimeoutException is thrown.

  For example, if Max Connections is set to *5*, and there are five physical connections in use, the Pool Manager waits for the amount of time specified in Connection Timeout for a physical connection to become free. If, after that time, there are still no free connections, the Pool Manager throws a ConnectionWaitTimeoutException to the application.

- Min Connections

  Specify the minimum number of physical connections to be maintained. Until this number is reached, the pool maintenance thread does not discard any physical connections. However, no attempt is made to bring the number of connections up to this number.

  For example, if Min Connections is set to *3*, and one physical connection is created, that connection is not discarded by the Unused Timeout thread. By the same token, the thread does not automatically create two additional physical connections to reach the Min Connections setting.

Tip: Set Min Connections to zero (0) if the following conditions are true:

- ► You have a firewall between the application server and database server.
- ► Your systems are not busy 24/7.

– Reap Time

Specify the interval, in seconds, between runs of the pool maintenance thread.

For example, if Reap Time is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the Unused Timeout and Aged Timeout settings. The smaller the interval you set, the greater the accuracy. When the pool maintenance thread runs, it discards any connections that have been unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in Min Connections. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

Tip: If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

– Unused Timeout

Specify the interval in seconds after which an unused or idle connection is discarded.

Tips:

- ► Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections not in use exceeds the Min Connections setting.
- ► Make sure that the database server's timeout for connections exceeds the Unused timeout property specified here. Long lived connections are normal and desirable for performance.

For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded. Note that accuracy of this timeout, as well as performance, is affected by the Reap Time value. See the Reap Time bullet for more information.

- Aged Timeout

  Specify the interval in seconds before a physical connection is discarded, regardless of recent usage activity.

  Setting Aged Timeout to *0* allows active physical connections to remain in the pool indefinitely. For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. Note that accuracy of this timeout, as well as performance, is affected by the Reap Time value. See Reap Time for more information.

  Tip: Set the Aged Timeout value higher than the Reap Timeout value for optimal performance.

- Purge Policy

  Specify how to purge connections when a stale connection or fatal connection error is detected.

  Valid values are EntirePool and FailingConnectionOnly. If you choose EntirePool, all physical connections in the pool are destroyed when a stale connection is detected. If you choose FailingConnectionOnly, the pool attempts to destroy only the stale connection. The other connections remain in the pool. Final destruction of connections that are in use at the time of the error might be delayed. However, those connections are never returned to the pool.

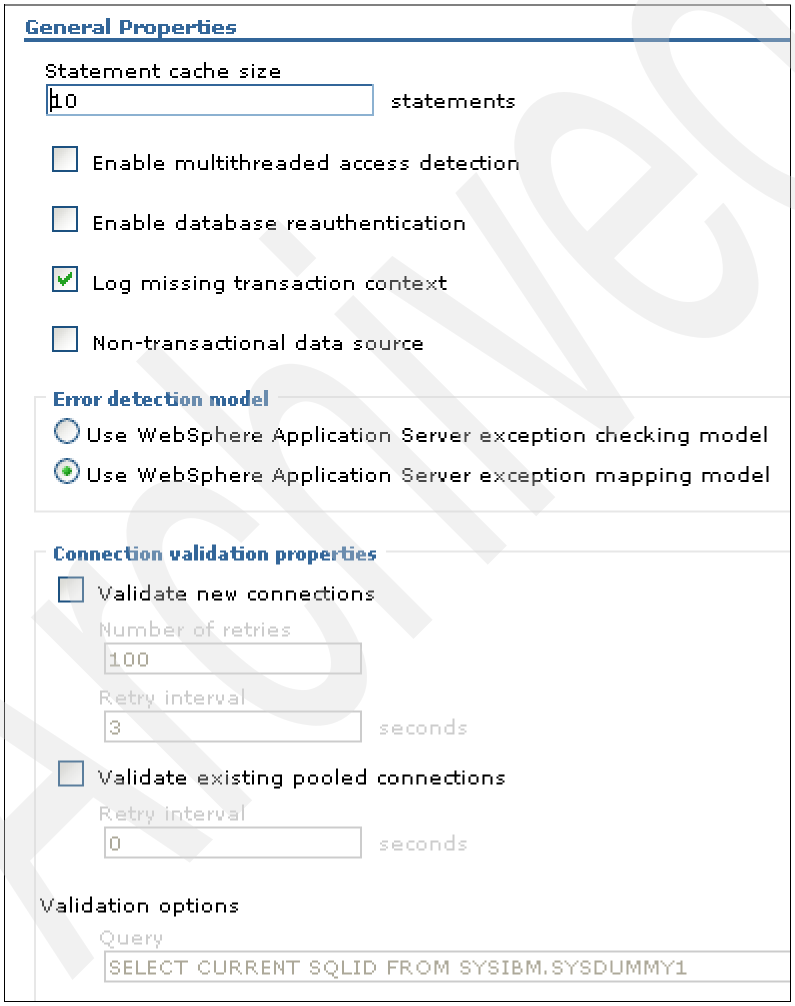
  Tip: Many applications do not handle a StaleConnectionException in the code. We recommend that you test to ensure that your applications handle them.

Selecting the **Advanced connection pool properties** link allows you to modify the properties additional connection pool properties. These properties require advanced knowledge of how connection pooling works and how your system performs. For information about these settings, see the *Connection pool advanced settings* topic in the information center.

WebSphere Application Server data source properties

You can set the properties that apply to the WebSphere Application Server connection, rather than to the database connection. To access the connection pool properties, navigate to **Resources** → **JDBC** → **Data sources**, and click the data source name. Select **WebSphere Application Server data source properties** in the Additional Properties section. See Figure 33 on page 37.

Clicking the link gives you the window shown in Figure 34.



General Properties

Statement cache size
10 statements

[ ] Enable multithreaded access detection

[ ] Enable database reauthentication

[✓] Log missing transaction context

[ ] Non-transactional data source

Error detection model
○ Use WebSphere Application Server exception checking model
● Use WebSphere Application Server exception mapping model

Connection validation properties
[ ] Validate new connections
Number of retries
100
Retry interval
3 seconds
[ ] Validate existing pooled connections
Retry interval
0 seconds

Validation options
Query
SELECT CURRENT SQLID FROM SYSIBM.SYSDUMMY1

Figure 34 WebSphere data source custom properties

Specify the following information:

- Statement Cache Size

  Specify the number of prepared statements that are cached per connection. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to execute the given SQL statement multiple times. The WebSphere Application Server data source optimizes the processing of prepared statements.

  In general, the more statements your application has, the larger the cache should be. For example, if the application has five SQL statements, set the statement cache size to 5, so that each connection has five statements.

  Tip: This setting is vital to performance of the database and will most likely require tuning to suit the specific application. Our general experience is that the default is not high enough for best performance.

- Enable multi-threaded access detection

  If you enable this feature, the application server detects the existence of access by multiple threads.

- Enable database reauthentication

  Connection pool searches do not include the user name and password. If you enable this feature, a connection can still be retrieved from the pool, but you must extend the DataStoreHelper class to provide implementation of the doConnectionSetupPerTransaction() method where the reauthentication takes place.

  Connection reauthentication can help improve performance by reducing the overhead of opening and closing connections, particularly for applications that always request connections with different user names and passwords.

- Log missing transaction context

  Specifies whether the container issues an entry to the activity log when an application obtains a connection without a transaction context.

- Non-transactional data source

  Setting the flag to true will cause the Application Server to never enlist the connections from the datasource in global or local transactions. Applications must explicitly call setAutoCommit(false) on the connection if they want to start a local transaction on the connection, and they must commit or rollback the transaction that they started. Note: this property should rarely be set to true, however the Java Persistence API (JPA) requires both JTA and non-JTA datasources.

- Error detection model

  The error detection model has been expanded and the data source has a configuration option that you can use to select the exception mapping model or the exception checking model for error detection.

- Connection validation properties

  There two properties and you can choose both. If you check the Validate new connections box, the application server tries to connect to database. If you select this property, you can specify how often, in seconds (interval).

  If you check the Validate existing pooled connections box, the application server retries to make a connection. If you select this property, you can specify retry interval for the server reroute. The pretest SQL string is sent to the database to test the connection.

  > **New in V7:** Connection validation by SQL query is deprecated in WebSphere Application Server V7.0. You can use validation by JDBC Driver instead. If you use the property of validation by JDBC driver, you need JDBC 4.0 or greater version. If you do not have JDBC 4.0, you have to update JDBC driver, the first.

- Advanced DB2 features

  – Optimize for get/use/close/connection pattern with heterogeneous pooling

    If you check this property, the heterogeneous pooling feature allows you to extend the data source definition. You can specify retry interval for client reroute, how often to retry, alternate server name or names for the DB2 server, port number, JNDI name. Details are described in “Extended DB2 data source” on page 44.

  – DB2 automatic client reroute options

    (New in V7) Client reroute for DB2 allows you to provide an alternate server location, in case the connection to the database server fails. If you decide to use client reroute with the persistence option, the alternate server information will persist across Java Virtual Machines (JVMs). In the event of an application server crash, the alternate server information will not be lost when the application server is restored and attempts to connect to the database.

Extended DB2 data source

(New in V7) The DB2 Universal JDBC Driver and DB2 using IBM JCC Driver support extends a DB2 data source with what is known as heterogeneous pooling. The extended DB2 data source configures a WebSphere DB2 data source with a set of core data source properties. An application can define one or more non-core set of data source properties and associate each with a different resource-reference that points to the main WebSphere DB2 data source.

The benefit of using an extended DB2 data source is that it allows applications to share the same WebSphere connection pool even though each application can have its own set of data source properties.

The sharing leads to a reduction of the number of open connections, and it pushes to reduce resource consumption on both the client side (WebSphere) and the server side (database layer).

For more information, refer to the following article in the information center:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.express.doc/info/exp/ae/tdat_heteropool.html

Notices

This document REDP-4577-01 was created or updated on March 26, 2010.

Send us your comments in one of the following ways:
- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an e-mail to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099, 2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

IBM®

Redpaper™

Trademarks