Develop your applications using SQLJ

Create Java and J2EE apps that use SQLJ with DB2 UDB V8.1 and Application Developer V5.1.2

Owen Cline

December 16, 2004

This "how-to" article explains how you can use SQLJ in your Java™ applications as well as your J2EE™ applications. It covers SQLJ syntax, accessing data sources, setting up WebSphere® Application Developer projects to support SQLJ, and creating and invoking DB2® stored procedures using SQLJ.

Overview of SQLJ

SQL statements include queries (SELECT), data-manipulation statements (INSERT, UPDATE, DELETE), and others that operate on data stored in tables in relational databases. SQLJ is a convention for embedding those SQL statements in Java, in a way that allows programming tools to perform translation-time (compile-time) analysis of SQL in Java, for syntax checking of SQL statements, for type checking to determine that the data exchanged between Java and SQL have compatible types and proper type conversions, and for schema checking to assure that the SQL constructs are well-formed and valid in the database schema where they will be executed. Embedded SQL statements are said to be "static" because they are textually evident in the Java program, and can therefore be compiled (the usual term is "pre-compiled") when the containing Java program is compiled. We propose SQLJ as a convenient and efficient standard for tightly integrated Java/SQL programs.

Brief history of SQLJ

As Connie Tsui explained in her article referenced below, an informal and open group of database vendors started to meet regularly in April 1997 to exchange ideas about how to use static SQL statements and constructs in the Java programming language. The key participants included IBM®, Oracle, Compaq, Informix®, Sybase, Cloudburst™ and Sun Microsystems. The group named the specification they were working on JSQL. After the discovery that JSQL was a trademarked term, JSQL was renamed to SQLJ. In December 1997, Oracle provided a reference implementation of embedded SQL in Java to the other members. This reference implementation could run on any platform that supported the JDK 1.1, and it was vendor-neutral. In December 1998, the specification for embedding SQL in Java was fully developed and accepted as ANSI standard Database Language - SQL, Part 10 Object Language Bindings (SQL/OLB).
ANSI x3.135.10-1998. This specification has commonly been referred to as Part 0 of the SQLJ specifications. It is now known as SQL/OLB (Object Language Bindings).

**SQLJ versus JDBC**

**SQLJ - The open sesame of Java database applications**

The SQLJ standard now provides SQL-based database access from Java apps. SQLJ - The open sesame of Java database applications

Here are some of the advantages that SQLJ offers over coding directly in JDBC:

SQLJ programs require fewer lines of code than JDBC programs. They are shorter, and hence easier to debug. SQLJ can perform syntactic and semantic checking on the code, using database connections at compile time. SQLJ provides strong type-checking of query results and other return parameters, while JDBC values are passed to and from SQL without having been checked at compile time.

SQLJ provides a simplified way of processing SQL statements. Instead of having to write separate method calls to bind each input parameter and retrieve each select list item, you can write one SQL statement that uses Java host variables. SQLJ takes care of the binding for you.

However, JDBC provides finer-grained control over the execution of SQL statements and offers true dynamic SQL capability. If your application requires dynamic capability (discovery of database or instance metadata at runtime), then you should use JDBC.

**Setting up WebSphere Studio Application Developer projects for SQLJ**

If you are planning to incorporate SQLJ code in any WebSphere Studio Application Developer (Application Developer) project, you will need to enable your project for SQLJ. To do that, right-click your project and select **Add SQLJ Support**. This will bring up the SQLJ Wizard shown next.
Figure 1. Enabling Application Developer projects for SQLJ Support

SQLJ and Java applications

From the Application Developer main menu, select **File -> New -> Other** and then select **Java and Java Project**. Click **Next**.

Figure 2. Create a Java project

Enter a project and click **Next**.
Figure 3. Enter a Java project name

Select **Finish** to create the project.

Figure 4. Enter Java project settings

Before the Java application will run successfully, you need to add the DB2 JDBC driver, `db2java.zip`. Right-click your Java project and select **Properties**.
Figure 5. Edit Java project properties

Next, select **Java Build path** and select the **Libraries** tab. And then select **Add External JARs**.

Figure 6. Add External JARs - db2java.zip

Here is how your libraries path should appear when they are done:
The next step is to create a Java class to hold your main method. To start this, right-click your Java project and then select **New -> Class**. Fill in the package name and class name. Also, make sure you check the box to create a main method. Click **Finish**.
Figure 8. Create Java main class

Finally, you will need to create your SQLJ file. From the Application Developer main menu, select File -> New -> Other and then select Data -> SQLJ and SQLJ File. Lastly, click Next. Application Developer will insure that this SQLJ file is transformed into a Java file.
Figure 9. Select SQLJ file wizard

Fill in the package name and the class name. Click Finish.

Figure 10. Create SQLJ file

At this point, you will need to add code to your Java main method and you will have to create a SQLJ method. To do that, let's define what we want to accomplish. We will create a SQLJ method, which reads an employee record given the employee number from the DB2 SAMPLE database. The SQLJ method will take as input the employee number (as a Java String) and pass that as
a parameter to a SQLJ Select statement. The Java main method will simply invoke the SQLJ method.

**Listing 1. Java class with main method**

```xml
<![[CDATA[

/*
 * Created on Nov 27, 2004
 */
package com.ibm.sqlj.main;

import com.ibm.sqlj.Select;

/**
 * @author Owen Cline
 */
public class SQLJJava {
    public static void main(String[] args) {
        Select select = new Select();
        select.selectEmployee("000110");
    }
}
]]>
```

**Listing 2. SQLJ class with SQLJ method**

```xml
<![[CDATA[

/*
 * Created on Nov 27, 2004
 */
package com.ibm.sqlj;

import java.sql.*;
import sqlj.runtime.ref.*;

/**
 * @author Owen Cline
 */
public class Select {
    // First, load the JDBC driver
    static {
        try {
            Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver").newInstance ();
        } catch (Exception e) {
            System.out.println ("\n Error loading DB2 Driver...\n");
            System.out.println (e);
            System.exit(1);
        }
    }
    public void selectEmployee(String empNo) {
        Connection con = null;
        DefaultContext ctx = null;
```
try {
    String firstName = null;
    String lastName = null;

    // use the DB2 SAMPLE database
    String url = "jdbc:db2:SAMPLE";

    // Get the connection
    con = DriverManager.getConnection(url);

    // Set the default context
    ctx = new DefaultContext(con);
    DefaultContext.setDefaultContext(ctx);

    // Lookup the employee given the employee number
    #sql { SELECT FIRSTNME, LASTNAME INTO :firstName, :lastName
        FROM EMPLOYEE
        WHERE EMPNO = :empNo }

    System.out.println ("Employee " + firstName + " " + lastName);

catch( Exception e )
{
    System.out.println (e);
}
} catch( Exception e )
{
    System.out.println (e);
} catch( Exception e )
{
    System.out.println (e);
}
}

At this point, you can run the Java application. Select your Java project and then select Run -> Run As -> Java Application from the Application Developer main menu. You should see the following message printed in the console "Employee VINCENZO LUCCHESSI".

**SQLJ and J2EE applications**

As you might imagine, you can use SQLJ in your J2EE application either in servlets, Session beans, BMP Entity beans and MDB beans. What we will do now is take the code we created above, which runs in a Java application, and port it to run in a Session bean.

To start, we need to create an EJB project. From the Application Developer main menu, select File -> New -> Project. Then select EJB and EJB Project. Click Next.
Figure 11. Create EJB Project

Make sure you select **Create 2.0 EJB Project** and then click **Next**.

Figure 12. Select an EJB Version

Now, enter the EJB Project name as **SQLJSession**. The EAR project should be **DefaultEAR**. Click **Next**.
For the Module Dependencies dialog, select **Finish** to create the EJB project and the EAR project.

At this point, we will want to create a Session Bean. From the Application Developer main menu, select **File -> New -> Enterprise Bean**. Click **Next**.
**Figure 15. Create a Session Bean - step 1**

Make sure the you have selected the Session bean radio button and that you have entered a Bean name. Click **Next**.

**Figure 16. Create a Session Bean - step 2**

Make sure that you check the **Local client view** checkbox so you get a local interface (they are faster if EJB container is locally deployed which it will be) and then click **Finish** to create the Session Bean.
Figure 17. Create a Session Bean - step 3

Now we will create a SQLJ file to contain our business logic. Right-click the EJB project `SQLJSession` and select **New -> Other** and then select **Data -> SQLJ** and **SQLJ File**. Click **Next**.
Figure 18. Create SQLJ File - step 1

Enter the package name and the file name and then select **Finish** to create the SQLJ file. Notice also that since we are creating this SQLJ file in a project that has not had SQLJ support added yet, that support will be added for us at this time.

Go ahead and use the above steps to create a second SQLJ file named **SessionBeanSelectUsingDefaultDatasource** to demonstrate later how you could use the defaultDataSource. Now, we can add our business logic. In the SQLJ files we created as follows:
Figure 19. Create SQLJ File - step 2

Listing 3. SessionBeanSelect.sqlj

```java
package com.ibm.sqlj;
import java.sql.*;
import sqlj.runtime.ref.*;
import javax.sql.DataSource;

/**
 * @author Owen Cline
 *
 */
public class SessionBeanSelect {
    // Setup datasource to use. Notice that I am not using a global JNDI name
    // but instead using a Resource Reference which points to the global JNDI name.
    // This is a best practice.
    #sql public static context Ctx with (dataSource="java:comp/env/sqljDS");

    public void selectEmployee(String empNo) {
        String firstName = null;
        String lastName = null;

        try {
            // Create context
            Ctx conCtx = new Ctx();

            // Lookup the employee given the employee number
            #sql [conCtx] { SELECT FIRSTNME, LASTNAME INTO :firstName, :lastName
                FROM EMPLOYEE
                WHERE EMPNO = :empNo
             };

            System.out.println("SessionBeanSelect-Employee " + firstName + " " + lastName);
            conCtx.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```
Listing 4. SessionBeanSelectUsingDefaultDatasource.sqlj

```sql
package com.ibm.sqlj;
import java.sql.*;
import sqlj.runtime.ref.*;
import javax.sql.DataSource;

/**
 * @author Owen Cline
 */
public class SessionBeanSelectUsingDefaultDatasource {

    public void selectEmployee(String empNo) {
        String firstName = null;
        String lastName = null;

        try {
            // Lookup the employee given the employee number
            #sql { SELECT FIRSTNME, LASTNAME INTO :firstName, :lastName
                FROM EMPLOYEE
                WHERE EMPNO = :empNo } ;

            System.out.println("SessionBeanSelectUsingDefaultDatasource-Employee "+ firstName + "  " + lastName);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Now, we have to add bean methods to invoke our business logic in the SQLJ files. Edit the SQLJSessionBeanBean.java file and add the two methods: selectEmployee and selectEmployeeUsingDefaultDatasource as shown in the following listing. Then, don't forget to promote these two methods to the local interface. Lastly, make sure you generate the deployment and RMIC Code for the EJB project before continuing.

Listing 5. SQLJSessionBeanBean.java

```java
package com.ibm.sqlj.ejb.session;
import com.ibm.sqlj.SessionBeanSelect;
import com.ibm.sqlj.SessionBeanSelectUsingDefaultDatasource;

/**
 * Bean implementation class for Enterprise Bean: SQLJSessionBean
 */
```
To eventually test our Session Bean, we will have to create a Server Configuration. Switch to the Server Perspective and right-click in the Server Configuration view. Select New -> Server and Server Configuration. Enter a server name of TestServer and click Finish to create it.
Once it's created, right-click the server (again in the "Server Configuration" view) and select **Add and Remove Projects**. In the "Add and Remove Projects" dialog box (not shown here), add the DefaultEAR project and click **Finish**.

The last thing we will need to do is create the DB2 data sources that our Session Bean will use. Double-click the **Test Server** in the Server Configuration view, which will invoke the server configuration editor. Select the **Security** tab and then select the **Add** button next to the JAAS Authentication Entries dialog box shown next. Enter a name for the entry as well as a valid user name and password to access the database. Click **OK**.

**Figure 21. Add JAAS Authentication Entry**

Now, select the **Data source** tab. Select the **Default DB2 JDBC Provider** entry in the JDBC Provider list. Select the **Add** button next to the Data source defined in the JDBC provider selected above table. Make sure that the **DB2 JDBC Provider** is selected as well as the **Version 5.0 data source** radio button. Click **Next**.
Figure 22. Select the type of Data source to create

Enter the data source name as `sqljDS` and the JNDI name as `jdbc/sqljDS`. Select the JAAS Authentication Entry that you created above for both the "Component-managed authentication alias" and the "Container-managed authentication alias". Click Finish.
Figure 23. Enter Data source parameters for sqljDS

<table>
<thead>
<tr>
<th><strong>Modify Data Source</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name:</strong> *</td>
<td>sqljDS</td>
</tr>
<tr>
<td><strong>JNDI name:</strong> *</td>
<td>jdbc/sqljDS</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>New JDBC DataSource</td>
</tr>
<tr>
<td><strong>Category:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Statement cache size:</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>Data source helper class name:</strong></td>
<td>[wss://webSphere:8080/DB2DataStoreJDBC]</td>
</tr>
<tr>
<td><strong>Connection timeout:</strong></td>
<td>1800</td>
</tr>
<tr>
<td><strong>Maximum connections:</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>Minimum connections:</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Reap time:</strong></td>
<td>1800</td>
</tr>
<tr>
<td><strong>Unused timeout:</strong></td>
<td>1800</td>
</tr>
<tr>
<td><strong>Aged timeout:</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Purge policy:</strong></td>
<td>EntryPool</td>
</tr>
<tr>
<td><strong>Component-managed authentication alias:</strong></td>
<td>SampleDB</td>
</tr>
<tr>
<td><strong>Container-managed authentication alias:</strong></td>
<td>SampleDB</td>
</tr>
<tr>
<td><strong>Use the data source in container managed persistence (CMP)</strong></td>
<td></td>
</tr>
</tbody>
</table>

* Required field.

Repeat these steps to create another data source with its name as `defaultDataSource` and its JNDI name as `jdbc/defaultDataSource`. Finally, save the server configuration.
At this point, we need to create a Resource Reference so our session bean can look up the sqljDS
data source. So, switch to the J2EE Perspective in Application Developer. While looking at the
J2EE Hierarchy view, double-click the SQLJSession EJB project to invoke the EJB Deployment
Descriptor editor. Select the References tab. Select the SQLJSessionBean and select the Add
button. Select the Resource Reference radio button and click Next.
Enter sqljDS as the name. Select avax.sql.DataSource as the type. Select Container for Authentication. Leave the Sharing Scope as Shareable. Click Finish.

Highlight the newly created resource reference and then enter jdbc/sqljDS as the JNDI name. Save the EJB Deployment Descriptor.
Figure 27. Enter the JNDI name for the Resource Reference

At this point, you should be able to test the Session Bean methods. First, while in the Server Perspective, start the TestServer by right-clicking it in the Servers view and then select **Start**. Next, invoke the Universal Test Client by right-clicking the TestServer in the Servers view and selecting **Run universal test client**. From the universal test client, select the JNDI Explorer and expand the "jdbc" menu to reveal the two data sources: sqljDS and defaultDataSource. Next, completely expand the Local EJB beans menu until you can select the **SQLJSessionBeanLocalHome**. From the EJB reference menu, completely expand it until you can select the **SQLJSessionBeanLocal.create()** method and then select the **Invoke followed by the Work with Object** buttons.
Figure 28. Invoke the `SQLJSessionBeanLocal.create()` method

Now, you will see the two methods to invoke: `selectEmployee` and `selectEmployeeUsingDefaultDatasource`. Select the `selectEmployee` method. Enter **000110** for the `empNo` value and select the **Invoke** button. Check the Console window to verify that 

"[11/28/04 16:56:05:176 PST] 6d2f338b SystemOut O SessionBeanSelect-Employee VINCENZO LUCCHESI" is displayed. Finally, try the `selectEmployeeUsingDefaultDatasource` method by yourself.

Figure 29. Invoke the `selectEmployee()` method
Summary

In summary, SQLJ is a very useful way to provide a persistence framework for both Java and J2EE applications where dynamic SQL is not required. Although it is not covered in this article, it is still possible to use SQLJ to develop Stored Procedures.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>code samples</td>
<td>SQLJArticle_sourcecode.zip (HTTP</td>
<td>FTP)</td>
</tr>
</tbody>
</table>
Related topics

- Download the source code used in this article.
- "Considering SQLJ for Your DB2 V8 Java Applications" (developerWorks, February 2003)
- "Building DB2 Stored Procedure Using WSAD V5" (Bibhas Bhattacharya)

© Copyright IBM Corporation 2004
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)