A quick start: Develop and run SQLJ programs

Skill Level: Introductory

Yongli An (yongli@ca.ibm.com)
DB2 UDB Performance Engineer
IBM

02 Dec 2004

Learn how to quickly start using WebSphere Studio Application Developer and DB2 Universal Database to develop SQLJ applications following these very detailed steps using a simple servlet application. This tutorial also provides some basic information about these IBM products and SQLJ-related information.

Section 1. Before you start

About this tutorial

This tutorial is written for those who are interested in coding, testing, and running SQLJ applications using WebSphere Studio Application Developer and DB2 Universal Database. It is also for those who are interested in running their applications in a WebSphere Application Server and DB2 UDB environment. If you are looking for a quick start to have some practical experience, and you don't want to sift through the manuals, this tutorial is for you. For users who have DB2, and one or both of WebSphere Application Server and WebSphere Studio Application Developer installed, he can make use of the ZIP file that comes with this tutorial to get firsthand experience, by following the steps described in this tutorial.

Prerequisites

In order to do the hands-on exercise to get the quick taste of running SQLJ in WebSphere and DB2, the minimum requirements include having an installation of WebSphere Application Server V5.1, DB2 UDB V8.1 FixPack 4 and WebSphere Studio Application Developer V5.1.2. Each product has the trial version. Users can refer to the individual user manual and instructions that come with the downloaded package to install each product. This tutorial is based on the version of each product

for Windows 2000.

Here is the list of software I used:

- Microsoft Windows 2000, Service Pack 3
- WebSphere Application Server V5.1
- DB2 Universal Database V8.1 FixPack 4
- WebSphere Studio Application Developer V5.1.2

To run the examples or sample code in this tutorial, you'll need to download the souce file .

Section 2. Introduction

Introduction

SQLJ is a set of programming extensions that allow a programmer using the Java programming language to embed statements that provide SQL (Structured Query Language) database requests. The Java language has been getting increasingly popular. As a result, accessing data in a database from a Java program is getting used more and more widely. There are two standard methods to access relational data in a database: using SQLJ and JDBC. With the latest WebSphere Application Server V5 and DB2 UDB V8 supporting running SQLJ programs, and WebSphere Studio Application Developer V5 supporting developing SQLJ programs, users are now offered a set of products to support the usage of SQLJ. This tutorial uses a simple servlet application accessing a DB2 database through SQLJ to demonstrate how easily the user can start developing SQLJ applications using these IBM products. Along the way, some basic information about these IBM products is also introduced.

Section 3. JDBC drivers and DB2 UDB V8

JDBC drivers and DB2 UDB V8

This section discusses some general information about JDBC drivers, and the drivers that are supported in DB2 UDB V8.

JDBC drivers in general

In general, Java applications use JDBC for dynamic SQL and they use SQLJ for static SQL. However, because SQLJ can inter-operate with JDBC, an application program can use JDBC and SQLJ within the same unit of work.

DB2 Universal Database provides driver support for client applications and applets that are written in Java using JDBC, and for embedded SQL for Java (SQLJ). JDBC is an application programming interface (API) that Java applications use to access relational databases. DB2 UDB support for JDBC lets you write Java applications that access local DB2 data or remote relational data on a server that supports DRDA.

SQLJ provides support for embedded static SQL in Java applications. SQLJ was initially developed by IBM, Oracle, and Tandem to complement the dynamic SQL JDBC model with a static SQL model.

According to JDBC specifications, there are four types of JDBC driver architectures, as described below.

Type 1 : Type 1 includes drivers that implement the JDBC API as a mapping to another data access API, such as Open Database Connectivity (ODBC). Drivers of this type are generally dependent on a native library, which limits their portability. The JDBC-ODBC Bridge driver is an example of a type 1 driver.

Type 2 : Type 2 includes drivers that are written partly in the Java programming language and partly in native code. The drivers use a native client library specific to the data source to which they connect. Because of the native code, their portability is limited.

Type 3: Type 3 includes drivers that use a pure Java client and communicate with a server using a database-independent protocol. The server then communicates the client's requests to the data source.

Type 4: Type 4 includes drivers that are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source.

JDBC drivers in DB2 UDB V8

DB2 UDB V8 supports a type 2 driver and a driver that combines type 2 and type 4 JDBC implementations. DB2 UDB V8 also supports a type 3 driver, although this driver is deprecated. The JDBC drivers in previous releases of DB2 UDB for Linux, UNIX, and Windows were built on DB2 CLI (Call Level Interface). The DB2 UDB V8 type 2 and type 3 drivers continue to use the DB2 CLI interface to communicate with DB2 UDB servers. DB2 UDB V8 adds a new DB2 Universal JDBC Driver that is written completely in Java. Below are the drivers that are supported in DB2 UDB V8:

DB2 JDBC type 2 driver for Linux, UNIX, and Windows

The DB2 JDBC type 2 driver lets Java applications make calls to DB2 through JDBC. Calls to the DB2 JDBC type 2 driver are translated to Java native methods. The Java applications that use this driver must run on a DB2 client, through which JDBC requests flow to the DB2 server. DB2 Connect Version 8 must be installed before the DB2 JDBC application driver can be used to access DB2 UDB for iSeries data sources or data sources in the DB2 for OS/390 or z/OS environments. The DB2 JDBC type 2 driver supports most JDBC and SQLJ functions that are described in the JDBC 1.2 specification, and some of the methods that are described in the JDBC 2.0 specification.

DB2 JDBC type 3 driver for Linux, UNIX, and Windows (deprecated)

The DB2 JDBC type 3 driver, which is also known as the applet or net driver, consists of a JDBC client and a JDBC server. The DB2 JDBC applet driver can be loaded by a Web browser along with the applet, or the applet driver can be used in standalone Java applications. When the applet requests a connection to a DB2 database server, the applet driver opens a TCP/IP socket to the DB2 JDBC applet server on the machine where the Web server is running. After a connection is set up, the applet driver sends each of the subsequent database access requests from the applet to the JDBC server through the TCP/IP connection. The JDBC server then makes corresponding DB2 calls to perform the task. On completion, the JDBC server sends the results back to the JDBC client through the connection. The JDBC server process is db2jd.

DB2 Universal JDBC driver (type 2 and type 4)

The DB2 Universal JDBC Driver is a single driver that includes JDBC type 2 and JDBC type 4 behavior, as well as SQLJ support. When an application loads the DB2 Universal JDBC Driver, a single driver instance is loaded for type 2 and type 4 implementations. The application can make type 2 and type 4 connections using this single driver instance. The type 2 and type 4 connections can be made concurrently. DB2 Universal JDBC Driver type 2 driver behavior is referred to as DB2 Universal JDBC Type 2 Connectivity. DB2 Universal JDBC type 4 driver behavior is referred to as DB2 Universal JDBC Type 4 Connectivity. The Universal Driver is one of the biggest changes in DB2 UDB V8 and it is an architecture-neutral JDBC driver for distributed and local DB2 access.

The following tables list the JDBC interfaces and indicate which drivers supports them, the drivers and their supported platforms.

Table 1. JDBC drivers for DB2 UDB

JDBC driver name	Associated DB2 UDB
`DB2 Universal JDBC Drivercode`	DB2 UDB for Linux, UNIX, and Windows or DB2 UDB for z/OS
`JDBC/SQLJ 2.0 Driver for OS/390`	DB2 UDB for z/OS
`DB2 JDBC Type 2 driver for Linux, UNIX, and Windows`	DB2 UDB for Linux, UNIX, and Windows

Installing the DB2 Universal JDBC Driver

If you select JDBC support during the installation of any of the DB2 UDB for Linux, UNIX, and Windows products, the installation program performs some of the installation steps for the DB2 Universal JDBC Driver.

The DB2 UDB Version 8 for Windows installation program performs the following tasks:

- Installs the `db2jcc.jar` and `sqlj.zip` files and adds them to the system CLASSPATH
- Installs file `db2jcct2.dll`, which is required for Universal Type 2 Connectivity, in the sqllib\bin directory

---

Section 4. General information on SQLJ applications

In 1997, IBM, Oracle, and several other companies started to work together to define and advance the standardization of Java development for enterprise-level and server-side database applications. SQLJ was proposed as a standard and as a simpler and easier-to-use alternative to JDBC (Java Database Connectivity). In 1998 the SQLJ specification was accepted as ANSI Standard X3.135.10-1998.

Reasons to use SQLJ

SQLJ primarily is a productivity environment that gives Java developers a quick and easy way to use SQL directly in their Java applications without the tedium of having to do database programming. This means reduced coding effort. Now applications involving a very large quantity of data manipulation, such as financial, personnel, or inventory control, can now be written in Java rather than in C. To experienced SQL programmers, SQLJ means elimination of writing the actual JDBC calls, concentrating on Java application logic with the capability of using the same familiar SQL to access the database.

SQLJ is built on top of JDBC, using embedded SQL to access the database. As discussed in Section 3, all the supported JDBC drivers in DB2 UDB V8 will be available to SQLJ application running against DB2 UDB.

Furthermore, the main difference between using JDBC and SQLJ to access DB2 is that JDBC always uses dynamic SQL. SQLJ can use dynamic SQL, but through the customization process can also be made to use static SQL statements. This normally means better run-time performance. The customization process can also be used to check the syntax of the SQL statements being used therefore reduces

the possibility of run-time errors.

Extra steps when using SQLJ

Writing a SQLJ application has much in common with writing a SQL application in any other language. In general, you need to do the following things:

- Import the Java packages that contain SQLJ and JDBC methods
- Declare variables for sending data to or retrieving data from DB2 tables
- Connect to a data source
- Execute SQL statements
- Handle SQL errors and warnings
- Disconnect from the data source

Although the tasks that you need to perform are similar to those in other languages, the way that you execute those tasks, and the order in which you execute those tasks, is somewhat different.

In particular, in order to realize the benefit of static SQL in SQLJ applications, developing SQLJ applications involves the use of the DB2 SQLJ Translator command (sqlj) and the DB2 SQLJ Profile Customizer command (db2sqljcustomize). If you need to bind a previously customized profile, you also need to use the db2sqljbind command. Another command is called db2sqljprint, which is used to print out the contents of a customized profile in plain text. It's very useful to debug package-related problems. However, all these steps are done automatically or easily in the WebSphere Studio Application Developer (Application Developer) environment with the proper configuration and a couple of clicks.

sqlj command

The command sqlj is the DB2 sqlj translator. It translates an SQLJ source file into a Java source file and zero or more SQLJ serialized profiles. By default, the Java source file is also compiled.

Each serialized profile generated corresponds to a connection context class used in the SQLJ source code. The profiles are named as follows: pgmname_SJProfile0.ser, pgmname_SJProfile1.ser, pgmname_SJProfile2.ser, and so on, where the number represents the order in which the connection context classes are referenced in the source file.

db2sqljprint command

This command can be used to view the contents of a profile. Each profile must be customized separately with the db2sqljcustomize command.

db2sqljcustomize command

This is the DB2 SQLJ Profile Customizer. It processes an SQLJ profile containing embedded SQL statements. By default, four DB2 packages are created in the database, one for each isolation level. This utility augments the profile with DB2-specific information for use at run time, and should be run after the SQLJ application has been translated, but before the application is run.

One option that can be used in this command is *-bindoptions*, which specifies a list of bind options. There is a list of options supported. Check the DB2 documentation for details on supported options.

db2sqljbind command

This is the DB2 SQLJ profile binder command. It binds a previously customized SQLJ profile to a database. By default, four packages are created, one for each isolation level. If the *-singlepkgname* option is used when customizing, only a single package is created and the ISOLATION option must be used. This utility should be run after the SQLJ application has been customized.

Static or dynamic SQL: a quick way to find out

If your SQLJ application does not go through the customization step successfully, your SQLJ application will be still using dynamic SQL instead of static SQL. Here is one quick way to find out by running the DB2 snapshot for dynamic SQL command.

If not yet, first you need to turn on the monitor switch for statement:

```
db2 -v update monitor switches using statement on
```

Then use the following to get the snapshot after running your application at least once:

```
db2 -v get snapshot for dynamic sql
```

From the snapshot output for dynamic SQL, you will be able to recognize the SQL statements from your application (if they do appear). If none of them appears, then your application is using static SQL.

If any of them appears, it could mean that they were invoked dynamically before being customized to static package or the application is still running in dynamic mode. In this case, you need to capture another snapshot after running your application again (such as browsing a couple of pages), then compare the output with the previous output. If you have your application customized properly, then it should currently use the generated static package. In this case, the values of *Number of executions* and *Number of compilations* for those SQL statements used in your application should not change between these two snapshot outputs. Please see Section 9.7 for more detail in snapshot samples.

If any of them increases, then your application is still running in dynamic mode. If

you want to run them in static mode, make sure you customize the application properly. Refer to section 7.2 for customization steps in Application Developer or section 9.8 for customizing and binding the application in WebSphere Application Server environment.

Section 5. The sample SQLJ application

For illustration purposes, this article uses a very simple SQLJ application, which is based on a SQLJ sample program shipped with DB2 in the *samples* directory. The main program is called GeneratePayroll.sqlj, which uses the *sample* database to generate a payroll report for all employees in one department based on the user input of a department name.

The original sample program GeneratePayroll.sqlj can be found in *sqllib**samples**java**sqlj*\.

The ported application has three major files that are based on the original sample program. The application (including the source code for the three files) is available with this paper for download in a ZIP file. The three files are *GeneratePayRollServlet.java, Payroll.java* and *DatabaseAccessUsingSQLJ.sqlj*.

The ZIP file can be imported into your Application Developer development environment for further changes or testing.

Section 6. Prepare the DB2 environment

Assuming that you already have a DB2 environment installed, the following steps are needed to make sure you have the right environment to run this sample SQLJ application, and make sure port number 50000 is not being used by any other applications other than this DB2 instance by checking the file in `C:\winnt\system32\drivers\etc\services`.

```
db2 -v update dbm cfg using svcename 50000
db2 -v update dbm cfg using authentication server
db2set DB2COMM=tcpip
db2stop
db2start
```

If the DB2 sample database is not created yet, run **db2sampl** to create it. To confirm you have the sample database created correctly, issue the following command and check the output:

```
db2 -v "connect to sample"
   Database Connection Information

 Database server        = DB2/NT 8.1.5
 SQL authorization ID   = YONGLI
 Local database alias   = SAMPLE

db2 -v "select * from department"

select * from department

DEPTNO DEPTNAME                     MGRNO  ADMRDEPT LOCATION
------ ---------------------------- ------ -------- ----------------
A00    SPIFFY COMPUTER SERVICE DIV.  000010 A00      -
B01    PLANNING                     000020 A00      -
C01    INFORMATION CENTER           000030 A00      -
D01    DEVELOPMENT CENTER           -      A00      -
D11    MANUFACTURING SYSTEMS        000060 D01      -
D21    ADMINISTRATION SYSTEMS       000070 D01      -
E01    SUPPORT SERVICES             000050 A00      -
E11    OPERATIONS                   000090 E01      -
E21    SOFTWARE SUPPORT             000100 E01      -

9 record(s) selected.
```

Section 7. A simple servlet app using Application Developer

WebSphere Studio Application Developer provides the tools you need to create, develop, test, and manage all of the resources involved with building Web and enterprise-scale J2EE and Web services applications. Customizable perspectives (views of the development resources and an organization of the desktop) let Web developers, Java programmers, EJB developers, and administrators share the same development tool. The WebSphere test environment, which is available in Application Developer, can be used to test your applications directly from the development environment. It provides all the function of the full run-time environment, but eliminates dependencies on network connections. After testing your application locally, you can use the server tools to publish the application, either locally or to another machine.

For the purpose of developing and testing the simple servlet application, this tutorial covers only a very small portion of the product usage. The following sections will walk you through all the steps for successfully testing the sample application locally within the Application Developer environment.

Create the servlet demo using Application Developer

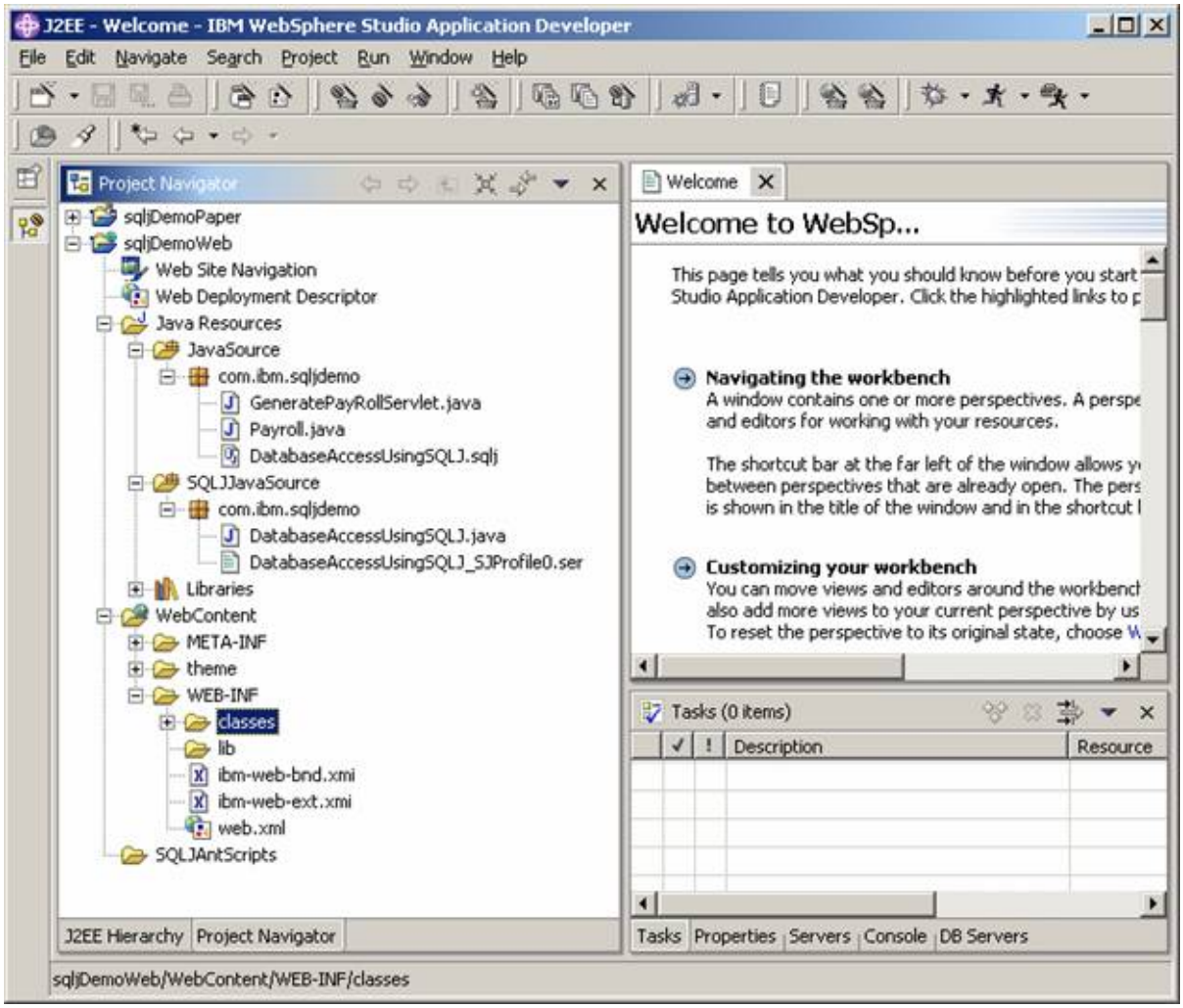
Focusing more on SQLJ-specific details of building a SQLJ servlet, this tutorial will not go through the details of creating the Java files in Application Developer from scratch. Instead, the zip file of the projects that can be downloaded is imported into the Application Developer environment. In WebSphere Studio Application Developer Version 5.1.2, there is a new feature called *Project Interchange*, which is the proper way to interchange Application Developer projects.

Here are the steps to import the zip file using the Project Interchange into Application Developer before proceeding to other sections.

- Download the complete source for the application used in this tutorial
- Start Application Developer in your Windows system
- Click **File -> Import**
- Highlight **Project Interchange**, then click **Next**
- Click **Browse** to locate the zip file sqljDemoPaper.zip, then click **Select All**
- Click **Finish**

Figure 1 below shows what you will have after clicking the Project Navigator tab in the left pane of the Application Developer window.

Figure 1: A view of Application Developer after importing the zip file



Build and customize the SQLJ servlet application

If you right-click on project **sqljDemoWeb**, you will see the option "Remove SQLJ support." This means the SQLJ support has been enabled before I created the zip file. Normally when you create a new project that has SQLJ files, you have to manually go through the process of enabling SQLJ support for that project. The steps are very simple:

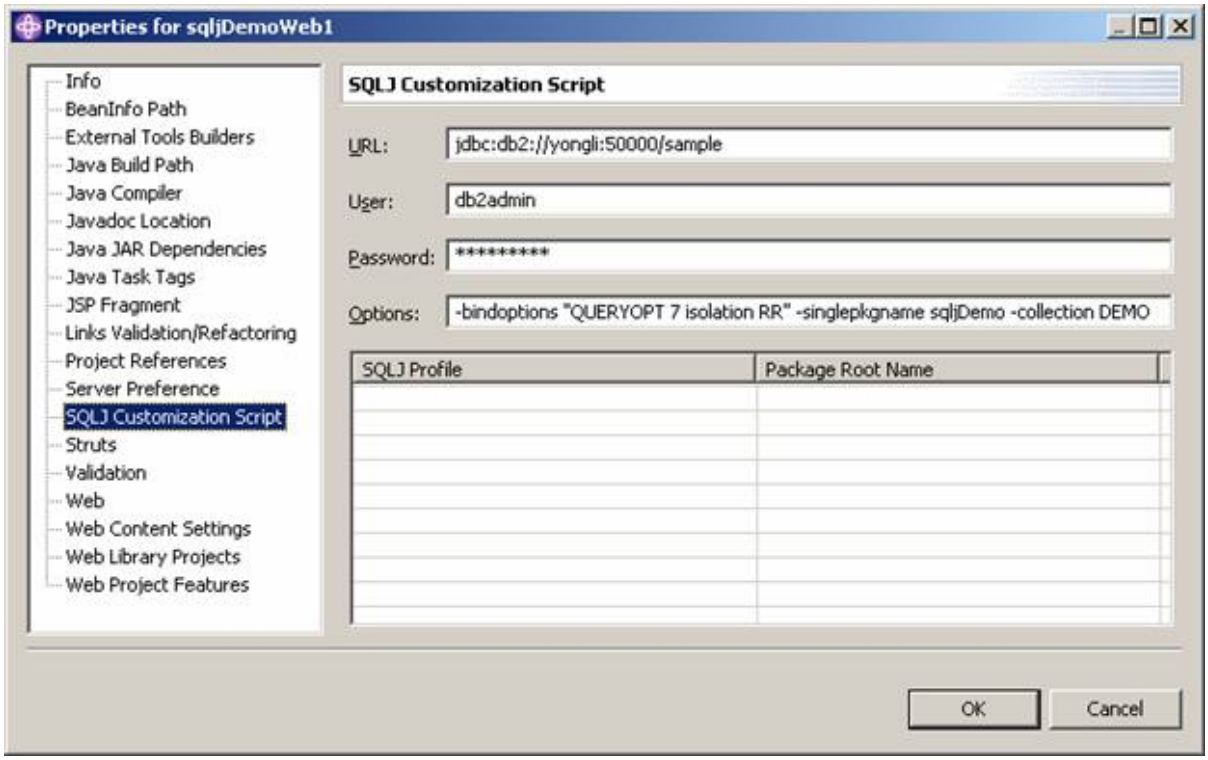
- Right-click on the project name **sqljDemoWeb**
- Click on **Add SQLJ Support**
- Click **Finish**

After adding the SQLJ support, any new changes in your SQLJ file will automatically trigger the translation and compilation of your SQLJ file.

The next step is to configure the properties and generate the customization script:

- Right-click on project **sqjDemoWeb** in the **project Navigator** tree
- Click on option **Property** at the bottom
- Click on **SQLJ Customization Script**
- Fill in the window accordingly, as shown in Figure 2 below, for generating the package. The window below shows the sample values for my environment. Please change the values of URL, User, and Password properly, according to your environment. You can just leave the *Options* value unchanged.

Figure 2: Information for SQLJ Customization Script



- Click **OK** to exit, and you should see the file *sqlj.project.properties* generated under folder *SQLJAntScripts*
- Right-click on project *sqljDemoWeb*, and then click on **Generate SQLJ Customization Script**. You should see another file called *sqlj.customize.xml* is generated under the same folder

Once the package is generated, the package name should be called *sqljDemo*. Before we start to run the customization script to generate the static package for this simple application in DB2 server, let's check whether the package is already there or not by running the following commands:

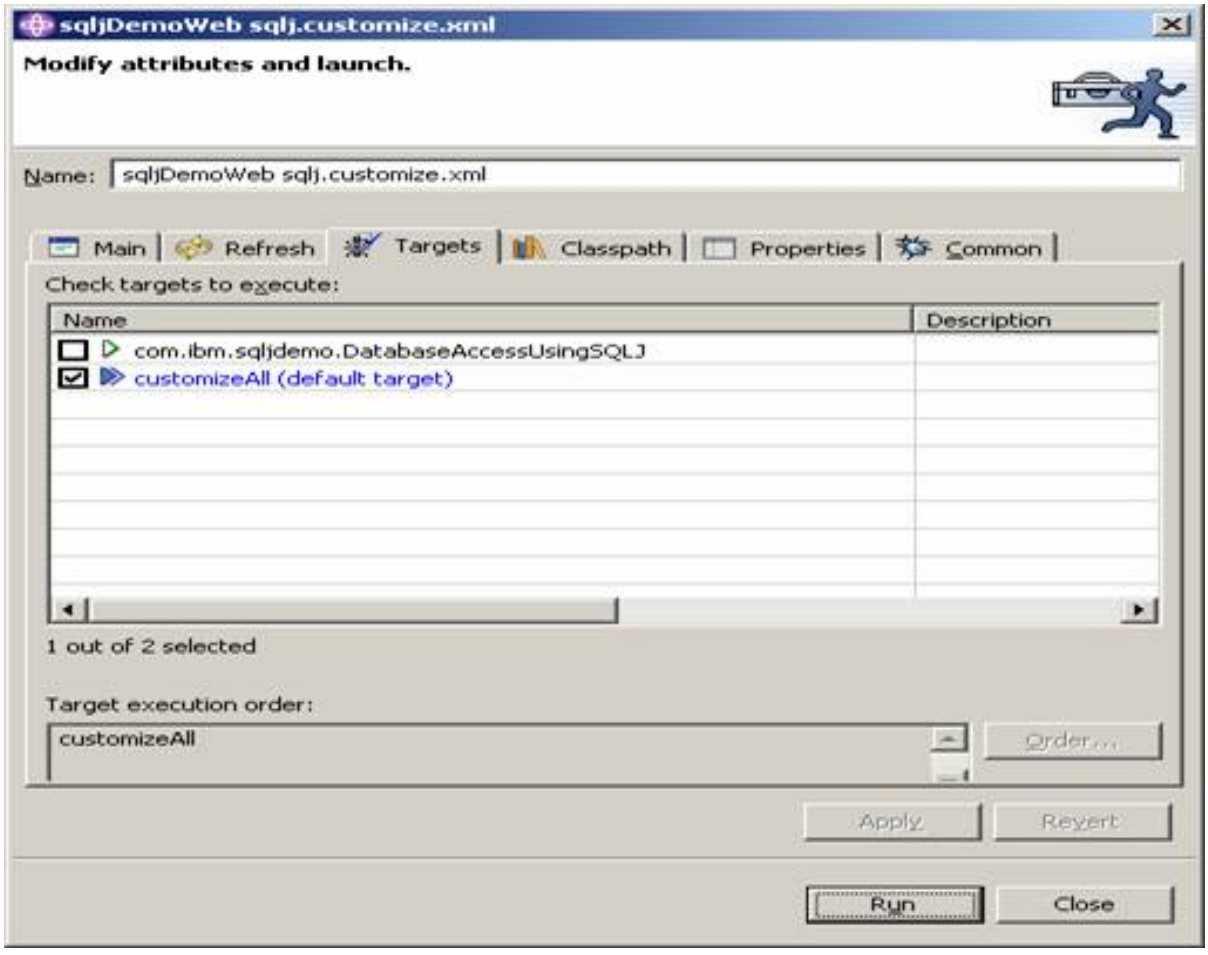
```
db2 -v "connect to sample"

db2 -v "select substr(pkgschema,1, 10) as pkgschema, substr
(pkgname,1 ,20)
```

```
as pkgname, isolation, blocking, last_bind_time  from syscat.packages
where pkgname
like '%sqlj%'"

KGSCHEMA  PKGNAME              ISOLATION BLOCKING LAST_BIND_TIME
----------      --------------------     ---------       --------  ----

0 record(s) selected.

db2 -v terminate
```

Clearly there is no package called *sqljDemo* in the sample database yet. In order to generate the package in the sample database, follow the steps below:

- Right-click on **sqlj.customize.xml**
- Choose **Run Ant**
- Click on **Run** in the window without changing anything

Figure 3: Window after clicking *Run Ant*



You will see something similar to the following in the Application Developer console:

```
Buildfile: D:\wsappdev51\workspace1\sqljDemoWeb\SQLJAntScripts\sqlj.customize.xml

com.ibm.sqljdemo.DatabaseAccessUsingSQLJ:
```

```
    [java] [ibm][db2][jcc][sqlj]
    [java] [ibm][db2][jcc][sqlj] Begin Customization
    [java] [ibm][db2][jcc][sqlj] Loading profile: com\ibm\sqljdemo\
    DatabaseAccessUsingSQLJ_SJProfile0
    [java] [ibm][db2][jcc][sqlj] Customization complete for profile ..\SQLJJavaSource\
    com\ibm\sqljdemo\DatabaseAccessUsingSQLJ_SJProfile0.ser
    [java] [ibm][db2][jcc][sqlj] Begin Bind
    [java] [ibm][db2][jcc][sqlj] Loading profile: com\ibm\sqljdemo\
    DatabaseAccessUsingSQLJ_SJProfile0
    [java] [ibm][db2][jcc][sqlj] User bind options: QUERYOPT 7 isolation RR
    [java] [ibm][db2][jcc][sqlj] Driver defaults(user may override): BLOCKING ALL
    VALIDATE BIND STATICREADONLY YES
    [java] [ibm][db2][jcc][sqlj] Fixed driver options: DATETIME ISO DYNAMICRULES BIND
    [java] [ibm][db2][jcc][sqlj]   Binding package sqljdemo
    [java] [ibm][db2][jcc][sqlj] Bind complete for com\ibm\sqljdemo\
    DatabaseAccessUsingSQLJ_SJProfile0
    [copy] Copying 1 file to D:\wsappdev51\workspace1\sqljDemoWeb\WebContent\
    WEB-INF\classes\com\ibm\sqljdemo

customizeAll:
BUILD SUCCESSFUL
Total time: 4 seconds
```

This indicates that the SQLJ profile has been successfully customized and bound. The static package *DEMO.sqljdemo* is also generated in the sample database on the DB2 server. You can check the package by issuing the following command:

```
db2 -v connect to sample

db2 -v "select substr(pkgschema,1, 10) as pkgschema, substr(pkgname,1 ,20) as pkgname,
isolation, blocking, last_bind_time  from syscat.packages  where pkgname like '%sqljDemo%'"

select substr(pkgschema,1, 10) as pkgschema, substr(pkgname,1 ,20) as pkgname, isolation,
blocking, last_bind_time  from syscat.packages  where pkgname like '%sqljdemo%'

PKGSCHEMA  PKGNAME              ISOLATION BLOCKING LAST_BIND_TIME
---------- -------------------- --------- -------- --------------------------
DEMO       sqljdemo             RR        B        2004-08-06-23.06.40.625000

1 record(s) selected.

db2 -v terminate
```

Export the application to an EAR file

In order to deploy the application into a WebSphere Application Server environment and run it, the application needs to be exported to an EAR file first.

- Click **File > Export**
- Highlight **Ear file** then click **Next**
- Choose the project name *sqljDemoPaper* in the pull-down list as the Enterprise Application project name
- Click **Browse** to choose the destination of the EAR file, and give it a proper file name, such as *sqljDemo.ear*
- If you like, check **Export source files** to allow the source code in the EAR file

- Click **Finish** to create the EAR file

Now the EAR file sqljDemo.ear is ready to be used for deploying into WebSphere Application Server later.

Section 8. Testing the application in Application Developer

The server tools in WebSphere Application Developer contain the complete run-time environment of WebSphere Application Server. This environment is called the WebSphere test environment. The test environment includes a local copy of the WebSphere Application Server run-time environment, where you can test Web projects, EJB projects, Java Application client projects, and Enterprise Application projects. When you install this product, you indicate which versions of the WebSphere test environment you wish to use for testing. One of the optional test environments included in Application Developer Version 5.1.2 is WebSphere Application Server, Version 5.1. In this tutorial, we are going to use this version of the test environment to test the sample application.

In order to test it directly in Application Developer, you should either already have an existing server or to create a new server. If this is the first time you are testing a file or project, you will be prompted to create a new server. The Server selection dialog box presents you with two options for deciding which server to run:

- **Use an existing server**. This list includes servers that may or may not have already been configured. If you choose a server that is not already configured, it will be configured automatically and may be restarted if the server is already running.
- **Create a new server**. This option will create a new server and server configuration before testing. If this is the first time you are testing a file or project, *Create a new server* is selected for you by default. When you select the type of server and click **Next**, a server will be automatically created for you and the server will start.

Once you have a server to use, then you have to configure the server to have the right data source and an authentication alias that are going to be used by this sample SQLJ application.

In this tutorial, we choose to create a server and server configuration explicitly for later use.

Create a server

A server identifies the run-time test environment that you can use to test your project. A server configuration contains the information that is required to set up and publish to a server. First we need to create a server and a server configuration.

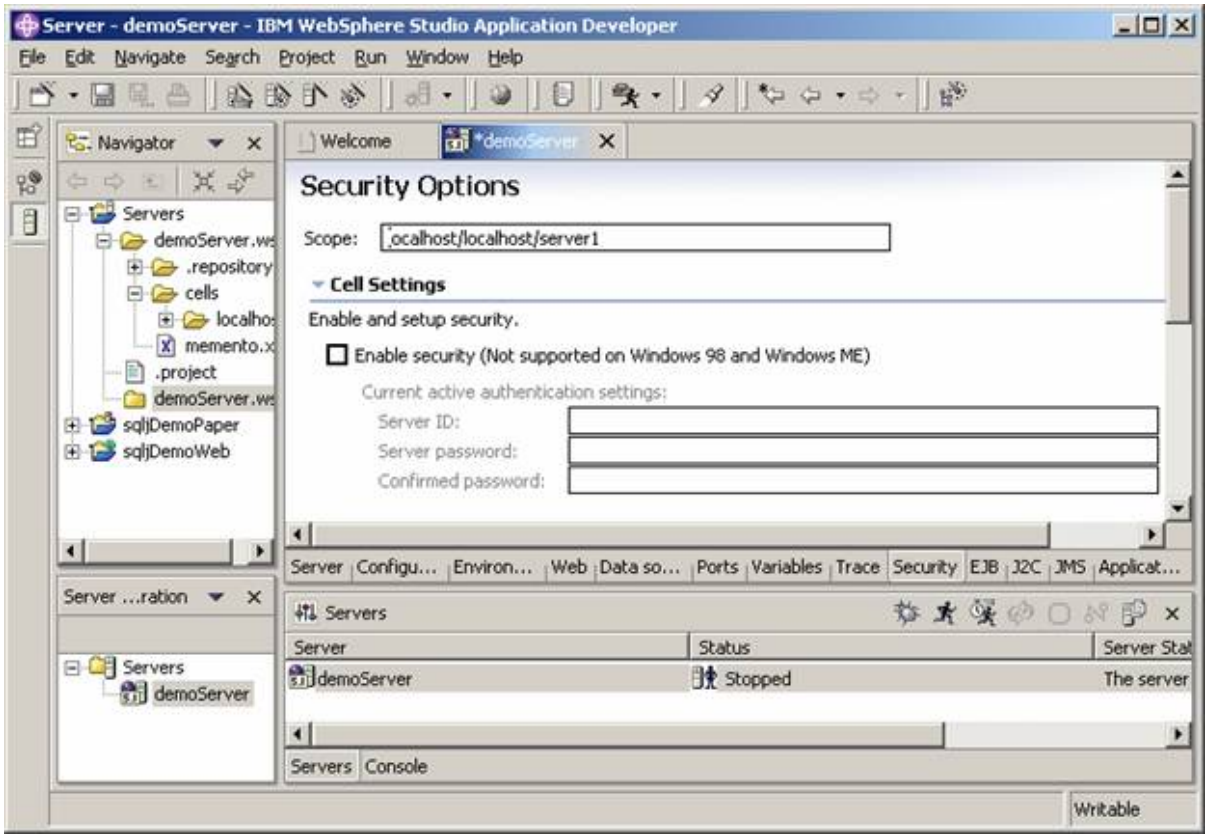
- On the Application Developer window, click **File -> New -> Other**
- Select **Server** in the left pane then select **Server and Server Configuration** in the right pane. Click **Next**
- The **Create a New Server and Server Configuration** wizard opens. This wizard creates a new server and configuration at the same time

Now follow the instructions in the wizard to specify the details of the server and server configuration that you want to create.

- Enter **demoServer** in the **Server name** box
- Choose **Test Environment** under WebSphere Version 5.1 in the **Server type** list
- The **Server configuration type** is automatically selected for you based on the choice of server type. In this case, the configuration type is **WebSphere V5.1 Server Configuration**
- Since there is no need to change anything else, click **Finish** to create the server and configuration

Now go to the Server Perspective, and double-click the WebSphere Server Instance file *demoServer.wsi*. This file can be found in the Navigation view, under the Servers project folder. Then select the **Security** tab to create an authentication alias. Figure 4 shows what you should see in the Application Developer window. If you like, you can now run `C:\Program Files\SQLLIB\java12\usejdbc2.bat` to enable extra features that are at JDBC 2.0.

Figure 4: Data source tab in the server instance window



Create the authentication alias in the server

Before creating a data source in the server, an authentication alias has to be created first to be used by the data source, which will be created in the next section.

- Click the **Add** button beside the JAAS authentication entries list table
- In the Add JAAS Authentication Entry dialog box, enter the following values:
  Alias: sqljTest_login
  User ID: xxyyzz (the user id you use to connect to your sample database)
  Password: ****** (the password for the id being used)
  Description: authentication alias for sqlj demo data source
- Click **OK** to create the alias

A JAAS authentication alias has been added to the server demoServer.

Create the data source in the server

Now let's configure the JDBC driver first before creating a data source in the server:

- Click **Data source** tab to switch to the data source window

- Click In the JDBC provider list, select **Default DB2 JDBC Provider**
- Click **edit** then select **${DB2_JDBC_DRIVER_PATH}/db2java.zip** and click **Remove**
- Select **Add External JARs** and navigate to the local copy of db2java.zip in **SQLLIB\java\db2java.zip** (default location is *C:\Program Files*\)
- Click **Open** to add it
- Repeat to add sqlj.zip, db2jcc_license_cisuz.jar, and db2jcc.jar into the class path
- Select the Implementation class name: **com.ibm.db2.jcc.DB2ConnectionPoolDataSource**. This means the Universal JDBC driver in DB2 is selected. If you like to try the legacy CLI JDBC driver, then you need to choose a different implementation class, such as **COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource**. In this tutorial, we use the Universal JDBC driver throughout the exercise
- Click **Finish**

Now create a data source:

- In the *Data source defined in the JDBC provider selected above* pane, click **Add**
- Select **Universal JDBC Provider**, select **Version 5.0 data source** then click **Next**
- In the *Modify Data Source* window, enter the following values, then click **Next**
  Name: sqljDemoDatasource
  JDNI name: jdbc/sqlj_test
  Description: DB2 Universal Driver Datasource for sqlj demo application
  Componet-managed authentication alias: sqljTest_login (from pull down list)
  Container-managed authentication alias: sqljTest_login (from pull down list)
- In the new window, go through the following resource properties and enter the right values:
  databaseName: sample
  driverType: 4
  serverName: yongli (or the server name where your db2 is running)
  portNumber: 50000 (the port number you used in section 6
  enableSQLJ: true
- Click **Finish**
- To save the changes to the server, type `Ctrl-s`

Now you have a data source created in the server demoServer. We are now ready

to test the application in the Application Devloper test environment.

Test the application in Application Developer

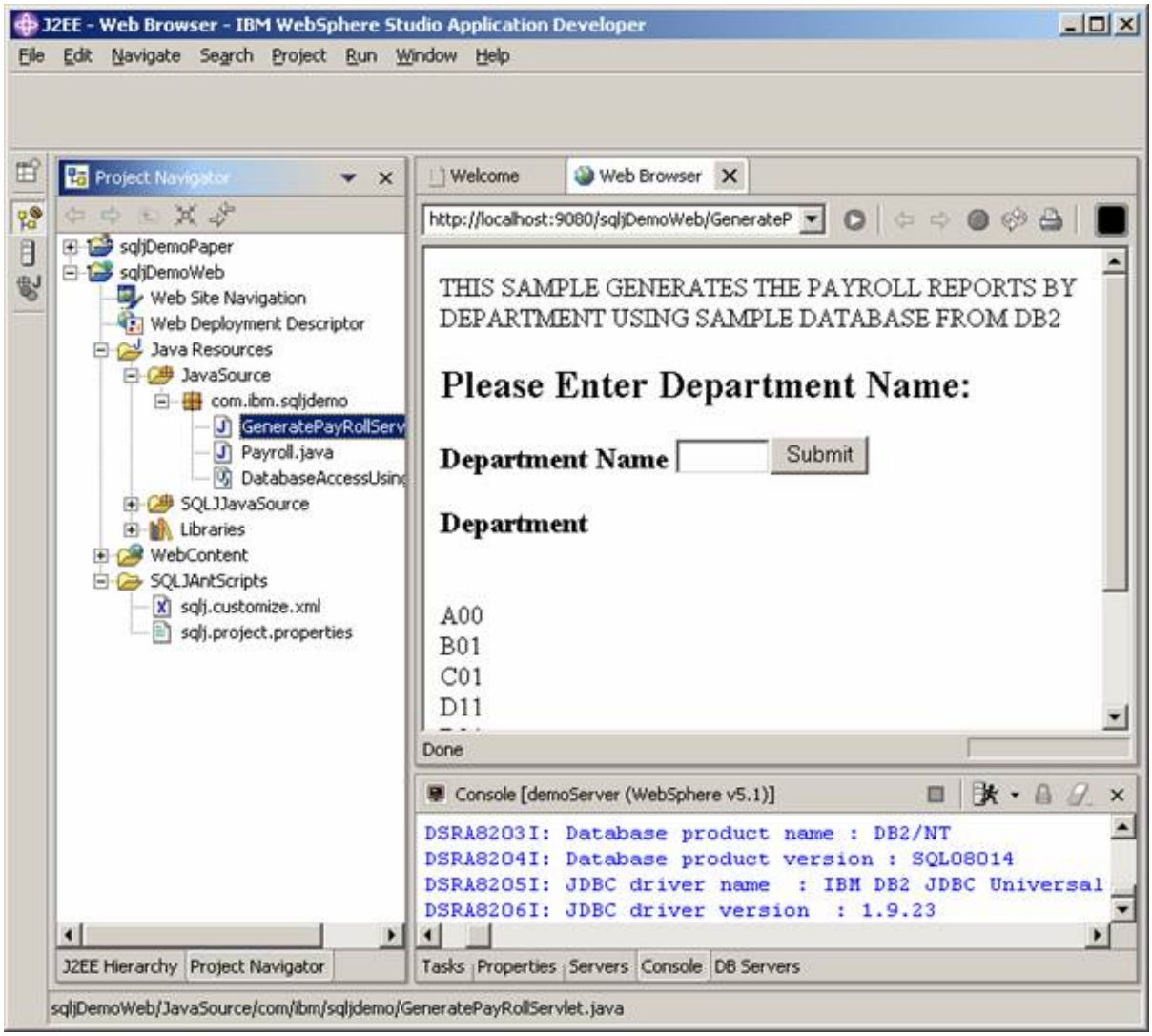
Now it's ready to run the sample application in the Application Developer test environment.

- Expand the Web project **sqljDemoWeb -> Java Resources -> JavaSource -> com.ibm.sqljdemo**
- Right-click on the servlet file **GeneratePayRollServlet.java**
- Select **Run on server**, then the Server selection wizard opens
- Check **Use an existing server radio button** and select the server demoServer that we just created
- Click **Finish**. The server tools automatically do the following tasks for you:
    1. By default the *Automatically publish before starting servers* check box is checked on the Server preference page (**Window -> Preferences -> Server**), the server tools check to see if your project and files on the server are synchronized. If they are not, the project and the files are automatically updated on the server when the server is restarted.
    2. Starts the server (it may takes a couple of minutes)
    3. Displays the file in the Web Browser, as shown in Figure 5
- Enter different department names and click the **Submit** button to see the payroll report for each department

The initial page of the sample servlet is shown in Figure 5, below.

Congratulations! You have successfully gone through the process of developing the sample servlet using SQLJ and tested it in Applicatin Developer.

Figure 5: the servlet intial page in Application Developer test environment



Section 9. Running in a WebSphere Application Server environment

The previous sections have gone through the building and testing cycle for a sample servlet application using SQLJ to access a DB2 database. Now it's time to deploy and run this sample application in a WebSphere Application Server. This section will describe all the steps for deploying, configuring, managing, and testing the sample application in such an environment.

You will manage configurations and states of application servers, applications, and other resources in the WebSphere administrative domain through the administrative console.

Log in to the WebSphere Administrative Console

This section shows how to log in to the WebSphere Administrative Console. Whenever applicable, the instructions in other following sections assume that you have already done the login and are in a session of the administrative console.

- The WebSphere Application Server should be started, if it is not already, by running the following under *$WAS_INSTALL_PATH/bin*. For example, under *C:\WebSphere\AppSever\bin*, you run the following: `startServer.bat server1`
- Check file *SystemOut.log* under *$WAS_INSTALLPATH/logs/server1*, and confirm the server is started successfully. You should see a message like the following:

```
[4/28/04 11:13:15:859 CDT] 3ef72e70 WsServer      A WSVR0001I: Server server1
open for e-business
```

- Open a browser pointing to **http://yourWASHostName:9090/admin**
- Log in by providing a user name, then a browser will open the first page of the administrative console

This page will give access to different options of managing and configuring the application and the WebSphere Application Server.

Create a J2C Authentication Data Entry

First, one J2C Authentication data entry needs to be created. This entry will be used when creating the data source in the next section.

- In the left pane, expand **Security -> JAAS Configuration -> J2C Authentication Data**, then click the **New** button in the right pane
- Enter **Alias** as **sqljTest_login**, and enter the proper user ID and password in order to connect to the database *sample* on the DB2 server successfully, then click **Ok** button
- Click the **Save** button at the top of the window, then click the **Save** button in the *Save* window

Now the J2C Authentication Data Entry *sqljSample_login* is created and saved successfully.

Create a data source for the application

To create a data source for the SQLJ demo application, please follow the steps

below:

- In the left pane of the administrative console, expand **Resources -> JDBC Providers**, then click on the **New** button to create a new JDBC provider
- Choose **DB2 Universal JDBC Driver Provider** in the pull-down list for JDBC Providers, and then click the **OK** button. This is the place where you make different choices of JDBC drivers
- In the Configuration window, replace **${DB2UNIVERSAL_JDBC_DRIVER_PATH}/** with the real path in the *Classpath* box based on your DB2 installation to avoid possible problems. For example, use a real path like **C:\SQLLIB\java**
- Click the **Apply** button to apply the changes
- Click on **Data Sources** to open the Data Sources window
- Click the **New** button to start creating a new data source
- Enter **sqjDemoDataSource** for the data source name
- Enter **jdbc/sqlj_test** in the **JNDI Name** box
- Check **Container managed persistence**
- Edit the Description box with some meaningful description
- Choose the Authentication Alias **sqljTest_login** that you just created in Create a J2C Authentication Data Entry from the pull-down list for both *Component-managed Authentication Alias* and *Container-managed Authentication Alias*
- Choose **DefaultPrincipalMapping** from the pull-down list for *Mapping-Configuration Alias*
- Click the **Ok** button and save the changes by clicking the **Save** button at the top of the window

Now the data source named *sqlj_test* is created.

Configure the data source

The data source that was just created has to be configured by following the steps shown below in order to be used by the sample SQLJ application.

- In the left pane, expand **Resources -> JDBC providers**, looking for the provider that was just created. In our case, it's called *DB2 Universal JDBC Driver Provider*.
- Click on **DB2 Universal JDBC Driver Provider**, and scroll to the bottom of the window

- Click on **Data Sources**, and then click on the data source that was just created. In this case, it's called *sqljDemoDataSource*.
- Click on **sqljDemoDataSource**, and scroll down to the bottom of the window, where we can configure the Connection Pool and some Custom Properties. We will leave the connection pool setting to its default.
- Click on **Custom Properties**, and enter "sample" for databaseName, then click **OK**
- Click on **driverType** then enter **4** for Universal JDBC driver type 4, or enter **2** for type 2. Click **OK**
- If you choose type 4 in the previous step, then you have to supply the serverName by clicking on serverName. Enter the machine name where your DB2 is running and click **OK**.
- If you choose type 4, then you also need to provide the port number by clicking on portNumber and enter **50000**, which has to be the same value you used when you prepared DB2 in Section 6, then click **OK**
- Click the **Save** option at the top of the window then click the **Save** button to save all the changes

If you would like to test the connection on the data source just created, follow the steps below:

- Follow the first three steps above to reach the page of data source called *sqljDemoDataSource*
- Click on **sqljDemoDataSource**, then a button called *Test Connection* should be seen in the Configuration page
- Click on the **Test Connection** button. if successful, you should see an informational message at the top of the page, like the following:

```
Message(s)
Test Connection for datasource sqljDemoDataSource on server server1
at node pizza44 was successful.
```

Now you have everything needed to run the sample application in the WebSphere Application Server except the application itself. The next section has the simple steps for installing the sample application into WebSphere Application Server.

Install the application into WebSphere Application Server

The application is installed to the WebSphere Application Server through the administrative console.

- In the left pane of the console, expand **Applications** and click on **Install**

New Application

- Click the **Browse** button to find the exported EAR file, *sqljDemo.ear* , in your local file system (this file was exported in Section Export the application to an EAR file)
- Click **Next** on all the panels until you see the window with the *Finish* button, then click **Finish**
- Click **Save To Master Configuration**, and click the **Save** button to save the installation

Now you have the application installed into the WebSphere Application Server. It's time to do a test run.

A test run with the application

Before testing the application, you have to start the application first by following the steps below:

- Expand **Applications** then click on **Enterprise Application** in the left pane
- If the application *sqljDemo*, which was just installed, is not started yet, check the box beside it, then click the **start** button
- You should see the following message shown at the top of the page:

```
 Message(s)
Application sqljDemo on server server1 and node pizza44 started successfully
```

Now open another browser if you would like, and in its address box enter the following:

```
http://pizza44:9080/sqljDemoWeb/GeneratePayRollServlet
```

Please note that the hostname pizza44 in the URL has to be changed to match your environment.

Press **Enter**, and you will see the following page that shows the application is running correctly. The application now is ready to interactively take a department name in the list and give you back a payroll report on that department.

Figure 6: Home page of the application

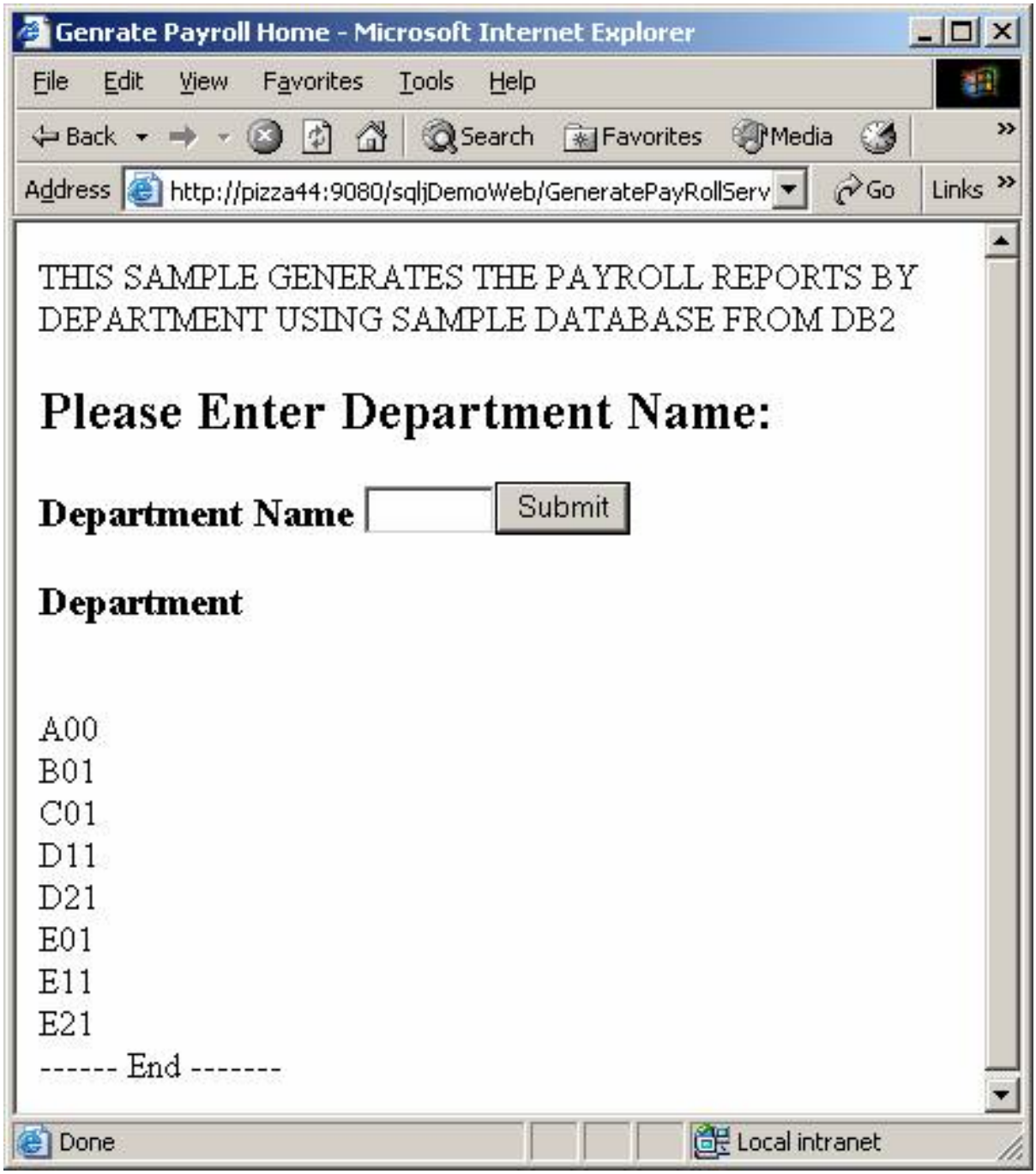
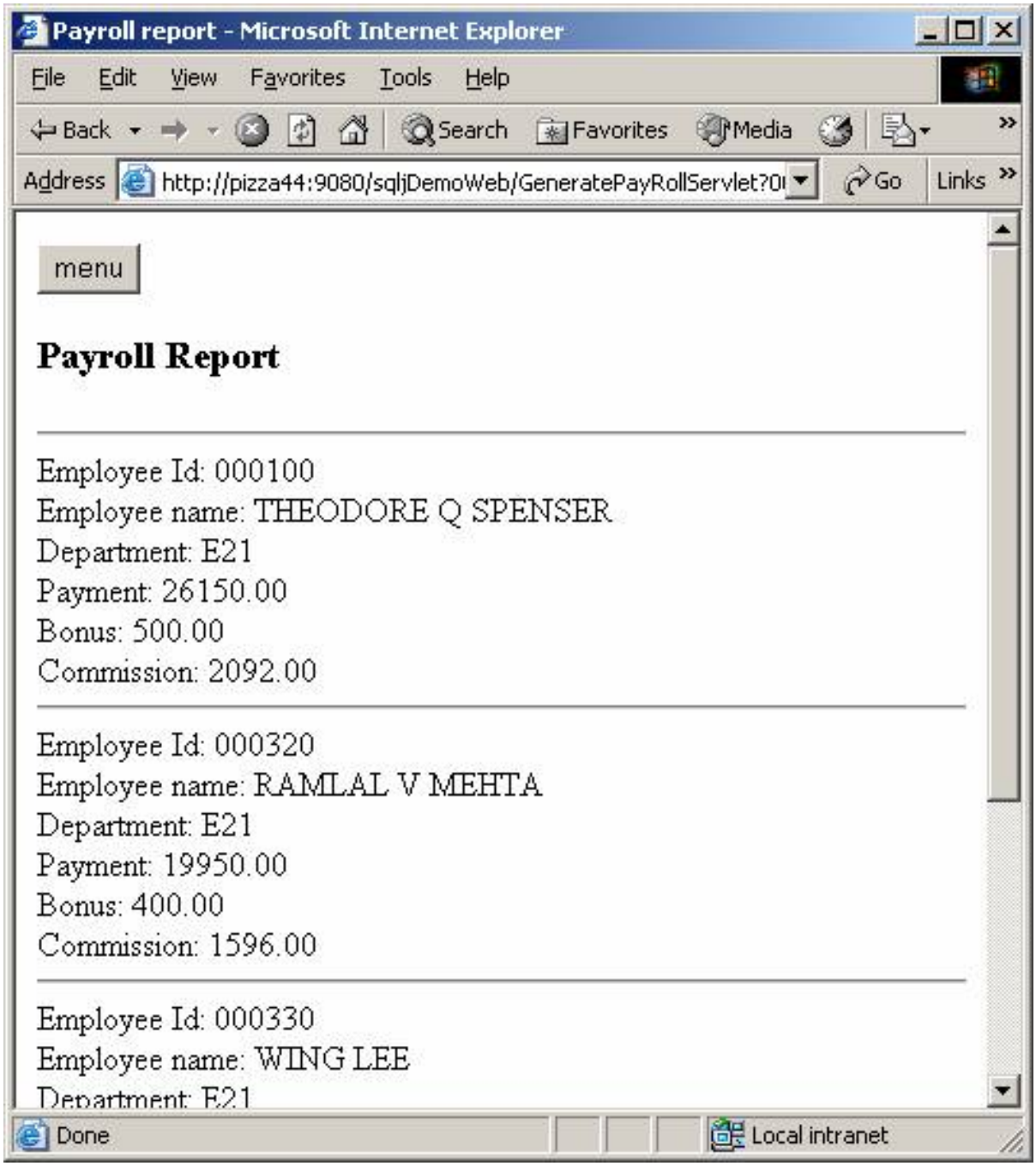


Figure 7 shows a sample payroll report once you enter a department name in the page above and click the **Submit** button. Then you can click the **Menu** button to go back the initial page where you can try another department name.

Figure 7: A sample payroll report page from the application



Find out the dynamic or static SQLJ

As we explained in an above section, your SQLJ application could run in two possible modes: static SQLJ or dynamic SQLJ. Below is a sample of the snapshot output as explained in Section 4.3. If you get more snapshots every time you click the **Submit** button in the servlet page, you will notice that the *Number of Executions* counter increases. This means the application is running in dynamic mode. To get the benefit of static SQLJ, we need to do one more step, as shown in next section, to bind the application properly and have the needed static package generated in the sample database.

```
db2 get snapshot for dynamic sql on sample

     Dynamic SQL Snapshot Result

Database name                       = SAMPLE

Database path                       = C:\DB2\NODE0000\SQL00001\

Number of executions                = 4
Number of compilations              = 1
Worst preparation time (ms)         = 4
Best preparation time (ms)          = 4
Internal rows deleted               = 0
Internal rows inserted              = 0
Rows read                           = 128
Internal rows updated               = 0
Rows written                        = 0
Statement sorts                     = 0
Buffer pool data logical reads      = Not Collected
Buffer pool data physical reads     = Not Collected
Buffer pool temporary data logical reads   = Not Collected
Buffer pool temporary data physical reads  = Not Collected
Buffer pool index logical reads     = Not Collected
Buffer pool index physical reads    = Not Collected
Buffer pool temporary index logical reads  = Not Collected
Buffer pool temporary index physical reads = Not Collected
Total execution time (sec.ms)       = 0.002775
Total user cpu time (sec.ms)        = 0.000000
Total system cpu time (sec.ms)      = 0.000000
Statement text                      = SELECT distinct workdept FROM EMPLOYEE


Number of executions                = 4
Number of compilations              = 1
Worst preparation time (ms)         = 0
Best preparation time (ms)          = 0
Internal rows deleted               = 0
Internal rows inserted              = 0
Rows read                           = 128
Internal rows updated               = 0
Rows written                        = 0
Statement sorts                     = 0
Buffer pool data logical reads      = Not Collected
Buffer pool data physical reads     = Not Collected
Buffer pool temporary data logical reads   = Not Collected
Buffer pool temporary data physical reads  = Not Collected
Buffer pool index logical reads     = Not Collected
Buffer pool index physical reads    = Not Collected
Buffer pool temporary index logical reads  = Not Collected
Buffer pool temporary index physical reads = Not Collected
Total execution time (sec.ms)       = 0.000557
Total user cpu time (sec.ms)        = 0.000000
Total system cpu time (sec.ms)      = 0.000000
Statement text                      = SELECT empno, firstnme, midinit, lastname, workdept,
salary, bonus, comm FROM EMPLYEE WHERE workdept= ?
```

By running the same query to check the package from DB2 catalog tables, it's found that indeed no package for the sample SQLJ application is generated. In order to run this sample application in static mode, you have to take following steps to generate the package in the sample database for the application.

```
db2 -v connect to sample
db2 -v "select substr(pkgschema,1, 10) as pkgschema, substr(pkgname,1 ,20) as pkgname,
isolation, blocking, last_bind_time  from syscat.packages  where pkgname like '%sqljDemo%'"
PKGSCHEMA  PKGNAME             ISOLATION BLOCKING LAST_BIND_TIME
---------- ------------------- --------- -------- --------------------------
  0 record(s) selected.
```

Binding the application after installing the application in WAS

This section shows, from the command line in the WebSphere Application Environment, how to accomplish what has been done in Section Build and customize the SQLJ servlet application in Application Developer- binding the application to generate the DB2 package in the sample database so that the application will run static SQLJ.

Assuming that your WebSphere installation path is under the following path:

```
 \WebSphere\AppServer
```

the first step is to locate the file *DatabaseAccessUsingSQLJ_SJProfile0.ser*. It should be found in the path that is similar to the one below, where pizza44 is the machine name where you run WebSphere Application Server, and *sqljDemo.ear* is the name of the EAR file being installed.

```
C:\WebSphere\AppServer\installedApps\pizza44\sqljDemo.ear\sqljDemoWeb.war\
WEB-INF\classes\com\ibm\sqljdemo
```

After going to the right directory shown above, then run command **db2slqjcustomize** to customize and bind the application into the sample database. Note that you have to replace "******" with a real password.

```
C:\WebSphere\AppServer\installedApps\pizza44\sqljDemo.ear\sqljDemoWeb.war\
WEB-INF\classes\com\ibm\sqljdemo>db2sqljcustomize -bindoptions "QUERYOPT 7 isolation RR"
-singlepkgname sqljdemo -collection DEMO -url jdbc:db2://pizza44:50000/sample -user tpcc
-password ******** DatabaseAccessUsingSQLJ_SJProfile0.ser

[ibm][db2][jcc][sqlj]
[ibm][db2][jcc][sqlj] Begin Customization
[ibm][db2][jcc][sqlj] Loading profile: DatabaseAccessUsingSQLJ_SJProfile0
[ibm][db2][jcc][sqlj] Customization complete for profile
DatabaseAccessUsingSQLJ_SJProfile0.ser
[ibm][db2][jcc][sqlj] Begin Bind
[ibm][db2][jcc][sqlj] Loading profile: DatabaseAccessUsingSQLJ_SJProfile0
[ibm][db2][jcc][sqlj] User bind options: QUERYOPT 7 isolation RR
[ibm][db2][jcc][sqlj] Driver defaults(user may override): BLOCKING ALL VALIDATE BIND
STATICREADONLY YES
[ibm][db2][jcc][sqlj] Fixed driver options: DATETIME ISO DYNAMICRULES BIND
[ibm][db2][jcc][sqlj]   Binding package sqljdemo
[ibm][db2][jcc][sqlj] Bind complete for DatabaseAccessUsingSQLJ_SJProfile0
```

Want to check the package from DB2 catalog table again? Now you should be able to see *DEMO.sqljdemo* from the query on the DB2 catalog table, as shown below. Also, you should be able to confirm that now the application is running static SQLJ by following the steps in Section Static or dynamic SQL: a quick way to find out , and Find out the dynamic or static SQLJ.

```
db2 select substr(pkgschema,1, 10) as pkgschema, substr(pkgname,1 ,20) as pkgname,
isolation, blocking, last_bind_time  from syscat.packages  where pkgname like '%sqlj%'

PKGSCHEMA  PKGNAME              ISOLATION BLOCKING LAST_BIND_TIME
```

```
---------- -------------------- --------- -------- --------------------------
DEMO       sqljdemo             RR        B        2004-08-24-17.24.19.359000

  1 record(s) selected.
```

Section 10. Summary and resources

Summary

This tutorial discussed the advantages of using SQLJ accessing data in a database through a Java application. It also introduces a full set of IBM products supporting for developing, testing, and running SQLJ applications in a very efficient fashion. These products include the latest WebSphere Application Server, WebSphere Studio Application Developer, and IBM Universal Database Version 8. This tutorial walks you through all the detailed steps to demonstrate how you can leverage this set of products to your business need. It covers why you need to use SQLJ and how to use SQLJ through a sample Java application. It should serve you as a quick head start and lead you to explore the power of SQLJ from these IBM products.

Resources

Learn

- DB2 SQLJ samples
- Connie Tsui, Considering SQLJ for Your DB2 V8 Java Applications
- Grant Hutchison, Developing Enterprise Java Applications Using DB2 Version 8
- Yongli An and Peter Shum, DB2 Tuning Tips for OLTP Applications
- Yongli An, Tony Lau and Peter Shum, A Scalability Study for WebSphere Application Server and DB2 Universal Database

Get products and technologies

- Download the complete source for the examples used in this tutorial.
- DB2 UDB V8.1 download
- WebSphere Studio Application Developer V5.1.2 trial download
- WebSphere Application Server, Version 5.1 Trial Program

Discuss

- Participate in the discussion forum for this content.

About the author

Yongli An
Yongli An is a DB2 UDB Performance Engineer, IBM Certified Solutions Expert - DB2 UDB 7.1 Database Administration for UNIX, Windows and OS/2, and DB2 7.1 Family Application Development. Yongli is experienced in TPC-C, WebSphere benchmarking, and fine tuning DB2 UDB for customers to achieve optimal performance. His current focus is DB2 performance for WebSphere Advanced Server and e-business applications.