Hello World, Part 2: Rational Application Developer

Access a Cloudscape database without using JDBC

Skill Level: Introductory

Jane Fung (mailto:jcyfung@ca.ibm.com)
Software Developer
IBM Toronto Lab

13 Jun 2006

Welcome to the second tutorial in the "Hello, World" series, which provides high-level overviews of IBM software products. This tutorial introduces you to IBM Rational Application Developer and highlights some of its basic features. It includes practical exercises that show how to create a Java application, create and invoke a Web service, and create a Web application that can access a Cloudscape database using a Relational Record List.

Section 1. Before you start

About this series

This series is for novices who want high-level overviews of IBM software products. The modules are designed to introduce the products and draw your interest for further exploration. The exercises only cover the basic concepts, but are enough to get you started.

About this tutorial

This tutorial provides a high-level overview of Rational® Application Developer and gives you the opportunity to practice using it with hands-on exercises. Step-by-step instructions show how to develop applications, create Web services, create a JSP page, and access a database.

Objectives

After completing this tutorial, you should understand the basic functions of Rational Application Developer and be able to create applications, as demonstrated in the exercises, including Java™ applications, Web services, and Web applications.

Prerequisites

This tutorial is for beginner-level Java application developers with a general familiarity of the Java language.

To run the examples in this tutorial, install IBM Rational Application Developer for WebSphere Software V6.0.

To view the demos included in this tutorial, JavaScript must be enabled in your browser and Macromedia Flash Player 6 or higher must be installed. You can download the latest Flash Player at http://www.macromedia.com/go/getflashplayer/

Animated demos

If this is your first encounter with a developerWorks tutorial that includes demos, here are a few things you might want to know:

- Demos are an optional way to see the same steps described in the tutorial. To see an animated demo, click the Show me link. The demo opens in a new browser window.
- Each demo contains a navigation bar at the bottom of the screen. Use the navigation bar to to pause, exit, rewind, or fast forward portions of the demo.
- The demos are 800 x 600 pixels. If this is the maximum resolution of your screen or if your resolution is lower than this, you will have to scroll to see some areas of the demo.
- JavaScript must be enabled in your browser and Macromedia Flash Player 6 or higher must be installed.

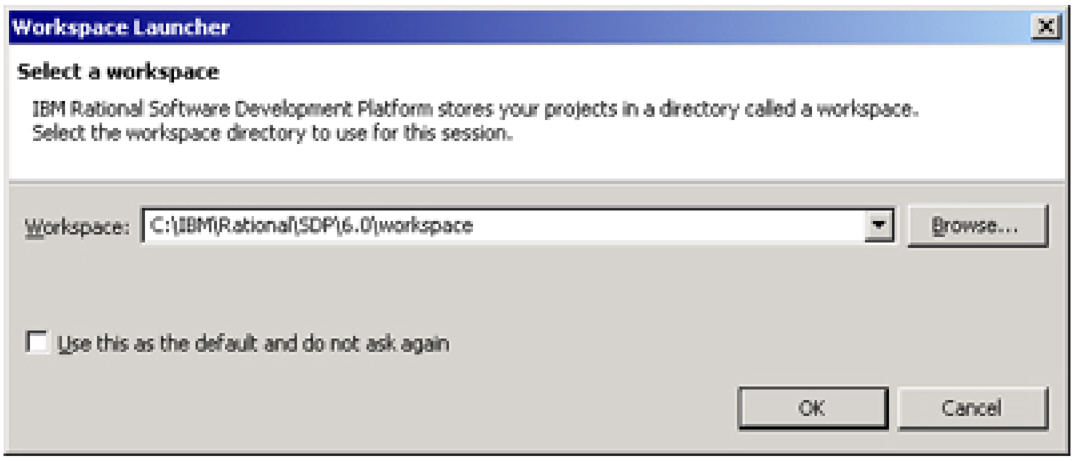
Section 2. Introduction

Rational Application Developer, hereafter Application Developer, is a successor to IBM WebSphere Studio Application Developer. Unlike its predecessor, Application Developer is based on the open-source Eclipse development platform. The *Eclipse* platform is an extensible development platform and application framework for building software. You can add plug-ins to the platform to include new functionality.

The Eclipse platform by itself comes with a Java Integrated Development Environment (IDE) for Java application development. It provides some common tools and features for Java developers. Built on top of the Eclipse platform, Application Developer adds a great number of capabilities including J2EE development, XML, Web service development, database development.

Select a workspace location, as shown in Figure 1, when you first start Application Developer. A *workspace* can be any directory location where your work is stored. If you are working on multiple projects at the same time, use different workspaces for each project to ensure a clear code separation.

Figure 1. The window for selecting a workspace location



Workspace Launcher

Select a workspace

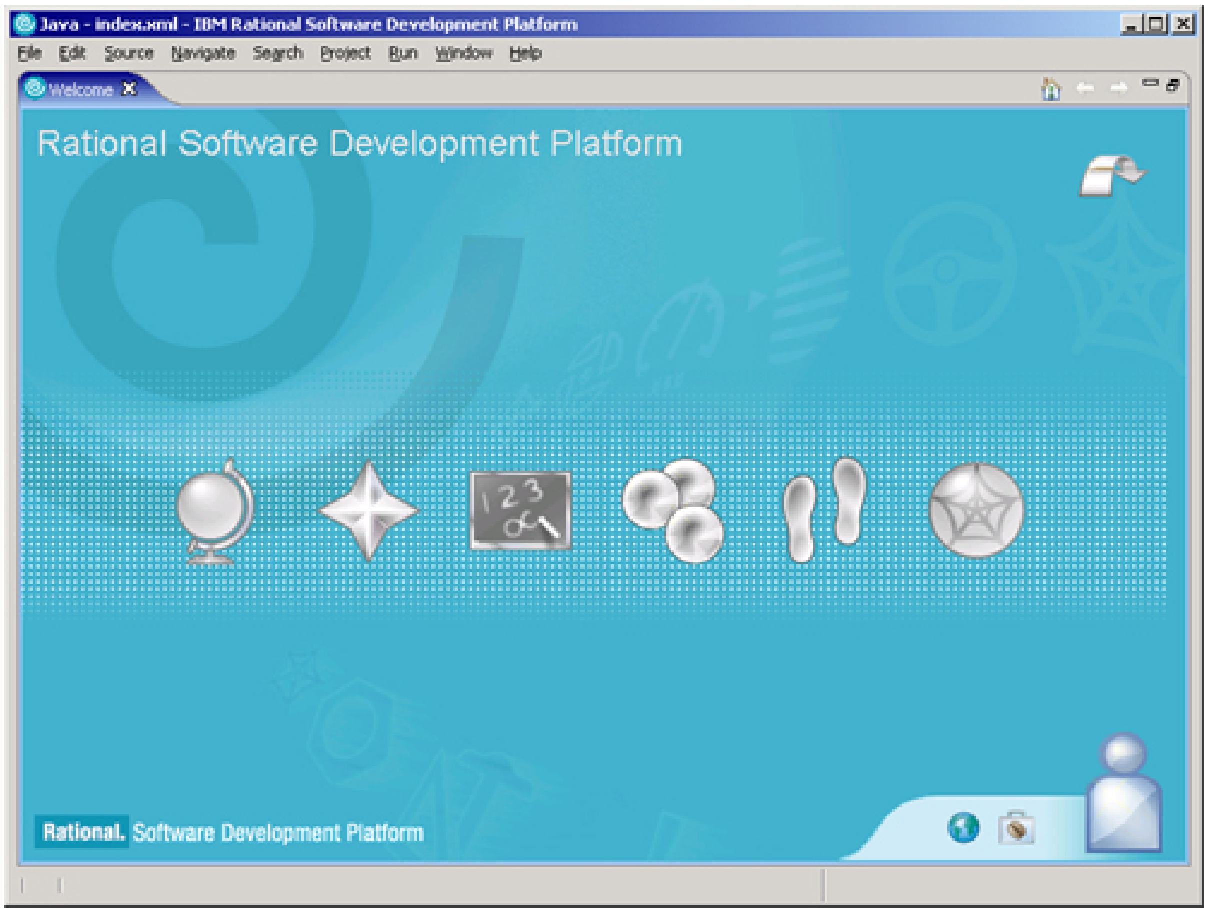
IBM Rational Software Development Platform stores your projects in a directory called a workspace.
Select the workspace directory to use for this session.

Workspace: C:\IBM\Rational\SDP\6.0\workspace

☐ Use this as the default and do not ask again

After you have chosen a workspace, the Welcome page displays, as shown in Figure 2. It offers quick links to tutorials and samples. Click on the Overview icon to take a guided tour of Application Developer.

Figure 2. Rational Software Development Platform Welcome page


Java - index.xml - IBM Rational Software Development Platform
File Edit Source Navigate Search Project Run Window Help
Welcome
Rational Software Development Platform
Rational. Software Development Platform

Close the Welcome page. The J2EE perspective displays. A *perspective* is a consolidation of tools and views that a developer needs. Application Developer offers a number of perspectives tailored for different types of developers, such as Web, Data, J2EE, and Debug, and Java programming.

Application Developer supports the development of many different types of applications, including:

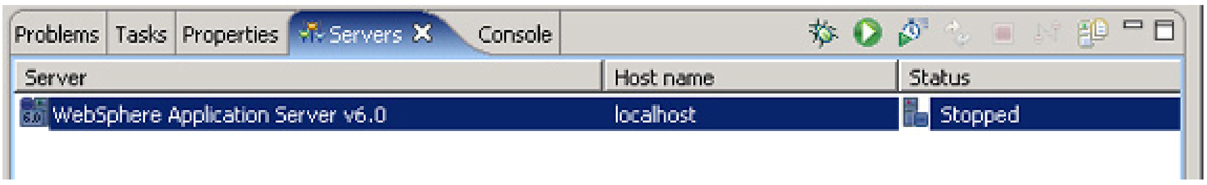
- Java
- Web
- EJB
- J2EE
- Database
- Web Service JMS
- SQLJ
- Portal
- Visual Modeling (allows you to visualize your source code in sequence diagram, topic diagram, and so forth)

You can develop any of the preceding applications, test, and debug right on

Application Developer. Directly deploy and test J2EE applications on the embedded WebSphere Application Server. *WebSphere Application Server* is a J2EE server that can support running J2EE enterprise applications. Application Developer comes with an integrated WebSphere Application Server.

By default, Application Developer ships with a WebSphere Application Server for unit testing. Figure 3 shows the server view, from which you can start and stop the WebSphere Application Server.

Figure 3. The server view



Server	Host name	Status
WebSphere Application Server v6.0	localhost	Stopped

Section 3. How does Application Developer fit into SOA?

Application Developer is a robust and powerful development tool. It fits very well with the Service Oriented Architecture (SOA). *SOA* is an architectural style for building distributed systems that deliver application functionality as services to be used by end-user applications or for building other services. It enables customers to create sophisticated applications and solutions swiftly and easily by assembling from new and existing services. Each business function in a company can be implemented as a service, which can then be integrated with other services to fulfill the company's business requirements.

SOA leverages open standards to represent business function as a service. Each service becomes a building block to create enterprise applications. Services in SOA can be implemented in multiple programming languages on different platforms. They can interact with each other because they are exposed using a common interface. One example of an SOA service is Web services.

Web services combine many widespread technologies and open standards to help enable the integration of heterogeneous systems. Through Web services, an SOA can be implemented, with new and existing applications as functional building blocks accessible over standard Internet protocols that are independent from platforms and programming languages. Application Developer lets you create Web services that can be used in a service-oriented architecture.

Application Developer supports the creation of Web services using a top-down approach (starts with a WSDL file to generate the Web service implementation) or a bottom-up approach (starts with a Java bean or EJB implementation to generate a Web service). It provides wizards to quickly create Web services, Web services clients, and publish Web services externally.

Section 4. Creating a Java application

This exercise steps you through the creation of a Java application that prints the line, *Hello, World!*

1. Start Application Developer, if you haven't already done so: from the Windows Start menu select **Programs > IBM Rational > IBM Rational Application Developer v6.0 > Rational Application Developer**.
2. A window displays and requests the workspace directory. Click **OK** to accept the default.

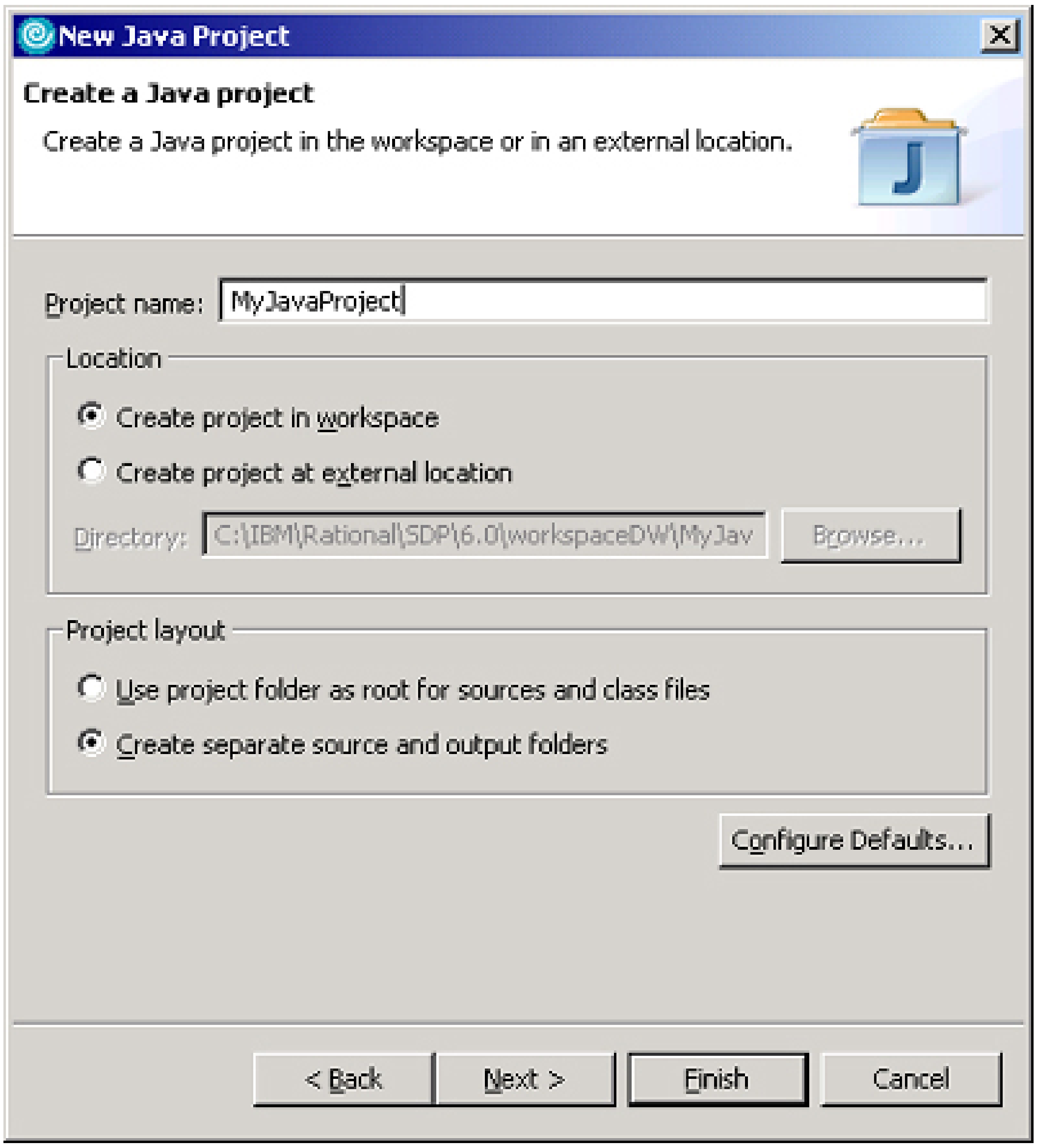
Creating a new Java project

Create a Java project named `MyJavaProject`, which has source and binary folders to store the Java and class files, respectively:

Would you like to see these steps demonstrated for you?

Show me

1. From the workbench, select **File > New > Project**.
2. Select **Java > Java Project > Next**. If you do not see Java, select the Show All Wizards check box. Select **OK** if you are prompted about enabling Java development capability.
3. Enter `MyJavaProject` as the project name.
4. Select the Create separate source and output folders radio button and click **Finish**.
5. Click **Yes** if you are asked to change to the Java Perspective.
Figure 4. New Java Project window


New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Project name: MyJavaProject

Location

(●) Create project in workspace

( ) Create project at external location

Directory: C:\IBM\Rational\SDP\6.0\workspaceDW\MyJav

Project layout

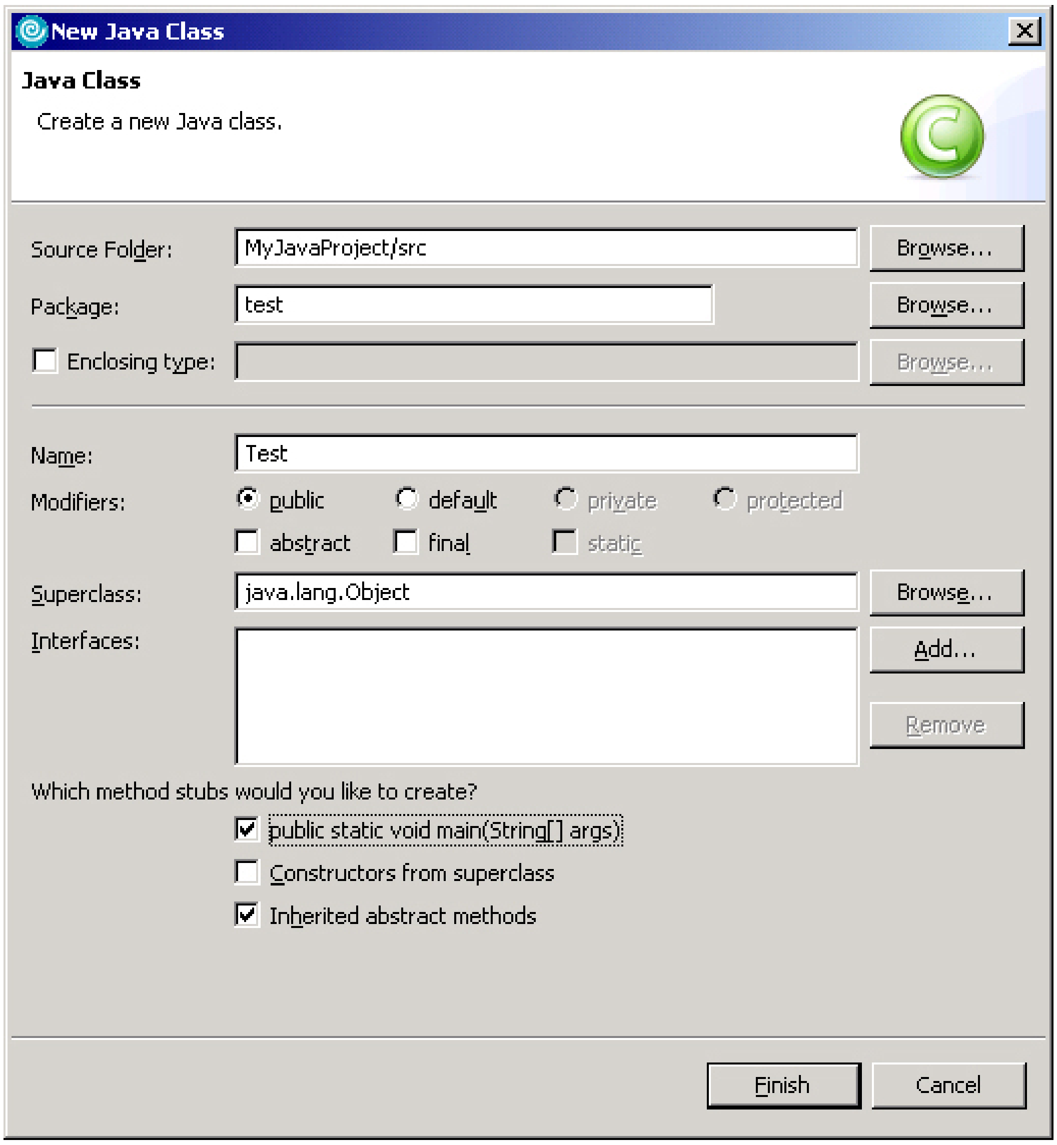
( ) Use project folder as root for sources and class files

(●) Create separate source and output folders

Creating a new Java class

Create a Java class named `Test` inside a test package, as shown in Figure 5:

1. In the Package Explorer view, right-click **MyJavaProject** and click **New > Class**.
2. Enter `test` as the package and `Test` as the file name.
3. Make sure **public static void main (String[] args)** is checked.
4. Click **Finish** and the Java editor opens.
   Figure 5. A new Java class



New Java Class

Java Class

Create a new Java class.

Source Folder: MyJavaProject/src — Browse...

Package: test — Browse...

☐ Enclosing type: — Browse...

Name: Test

Modifiers: [✓ public] default private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object — Browse...

Interfaces: — Add... Remove

Which method stubs would you like to create?

[✓] public static void main(String[] args)

☐ Constructors from superclass

[✓] Inherited abstract methods

Finish Cancel

Modifying the Java class

1. In the Java editor, modify the Java class as Listing 1 shows in **bold**:
   Listing 1. Code for the test.Test class

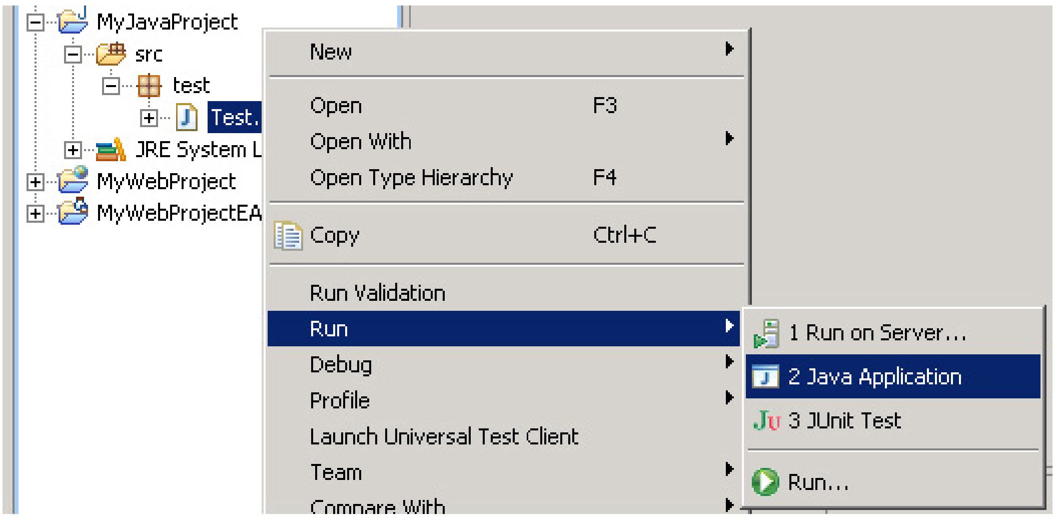
```
public class Test {

        public static void main(String[] args) {
                System.out.println ("Hello,World!!");
                System.out.println ("Hello, World Again!!");
        }
}
```

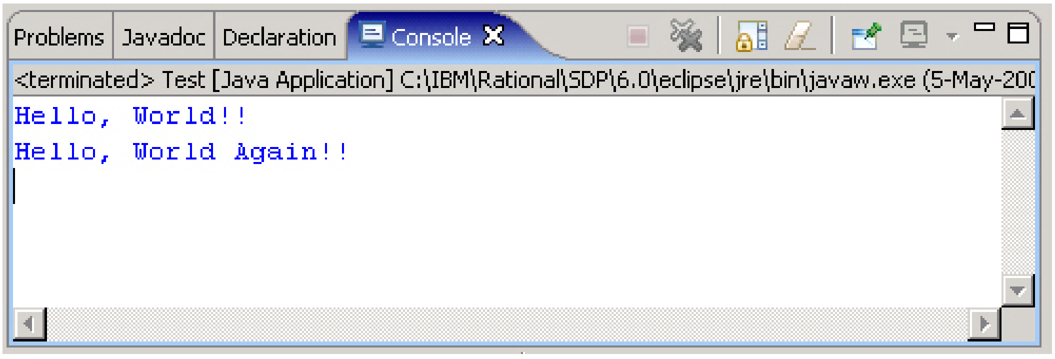
   As you type, you can use the Code Assist (Ctrl-Space) function to help you complete the keywords.

2. Save the file by pressing Ctrl-S.

Running the Java application

Now it's time to run your first application:

1. In the Package Explorer view, right-click the **Test** class and click **Run > Java Application**, as shown in Figure 6:
   Figure 6. Running a Java application
   

2. Switch to the Console view to see the result of the application, shown in Figure 7. If the Console view is not visible, you can switch to it by going to the **Windows menus > Show View > Console**.
   Figure 7. The Console view
   

```
<terminated> Test [Java Application] C:\IBM\Rational\SDP\6.0\eclipse\jre\bin\javaw.exe (5-May-200
Hello, World!!
Hello, World Again!!
```

   The output for running your Java application is displayed in the console. You see the two lines of text that was printed from the Java application.

Section 5. Creating and invoking a Web service

This exercise demonstrates how to create a Web service and invoke it from

JavaServer Pages (JSP) files. A Web service is based on the open standard interface, *Web Service Description Language* (WSDL). Each Web service is described by a *WSDL interface*. This interface contains the definitions for port types, operations, and message types for the Web service. The *port type* contains a collection of operations that this service supports. Each operation has input and output messages defined. In Java terms, a port type is like a class definition, an operation is like a method and the input and output messages are like the input parameter or return types to a method.

Creating a Web service is pretty straight forward. The Web Service wizard automatically generates everything for you. In this example, create a Java class that returns a hard-coded IBM stock price. Then use the Web Service wizard to make it into a Web service. In that process, a WSDL file is created and to be used later by the JSP client to invoke the Web service.

To invoke a Web service that is hosted on your local or on a remote server, obtain a WSDL file from the Web service provider and then generate a proxy in the tooling. Use the proxy to invoke the Web service.

Web services can be posted to a public registry called the *Universal Description, Discovery and Integration* (UDDI). *UDDI* provides a centralized repository for Web services. Web service providers can publish their Web services, including links to WSDL documents, into a UDDI repository for users to discover and subsequently use.

Would you like to see these steps demonstrated for you?

Show me

Creating a new Web project

Create a new Web project that only contains the Java class and the generated Web service. A *Web project* is basically a Web component in the J2EE specification. You create the JSP client in another Web project later.

1. From the workbench, select **File > New > Project**.
2. Select **Web > Dynamic Web Project > Next**. Select **OK** if prompted to enable "Web Development" capability.
3. Type `MyDemoWebServiceProject` as the project name. Click **Finish**. Select **Yes** if prompted to switch to the Web perspective.

Creating a Java class

Create a Java class that acts as a back end of the Web service. This Java class does not really do anything; however, in reality, a Web service back-end could potentially do anything, such as processing information through an Enterprise Java Bean (EJB), connecting to database, or even invoking another Web service.

1. In the Package Explorer view, expand **MyDemoWebServiceProject** and right-click on **Java Resources** and click **New** > **Class**.
2. Enter `demo` as the package and `DemoWebService` as the file name.
3. Make sure the public static void main (String[] args) box is not checked.
4. Click **Finish** and the Java editor opens.
5. Add the code shown in **bold** in Listing 2 and modify the code as follows:
   Listing 2. Code for the demo.DemoWebService class

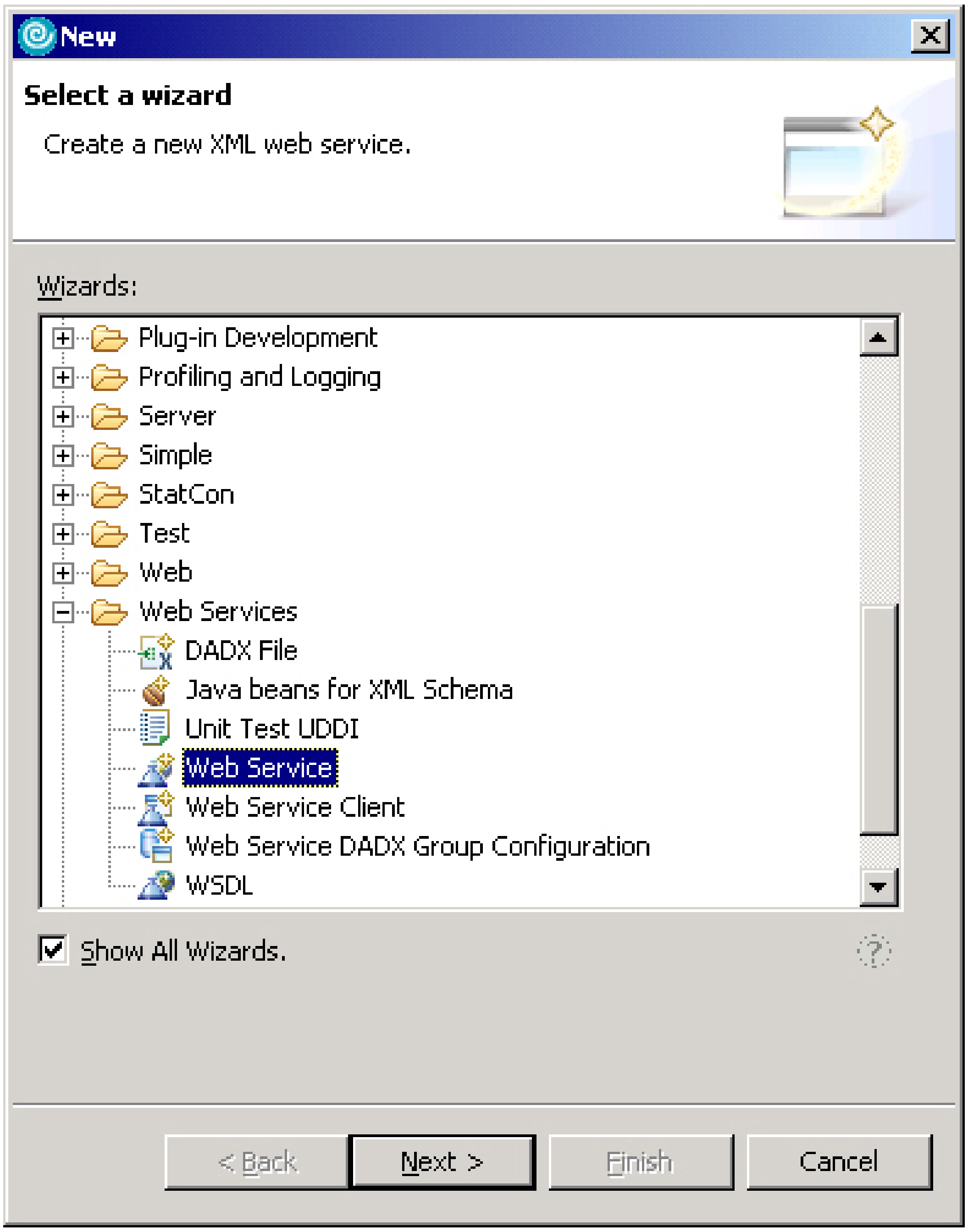
```
package demo;

public class DemoWebService {
        public double getQuote (String symbol){
                if (symbol.equals ("IBM")){
                        return 120;
                }
                return 0;
        }
}
```

6. Click **Save**.

Creating a Web service

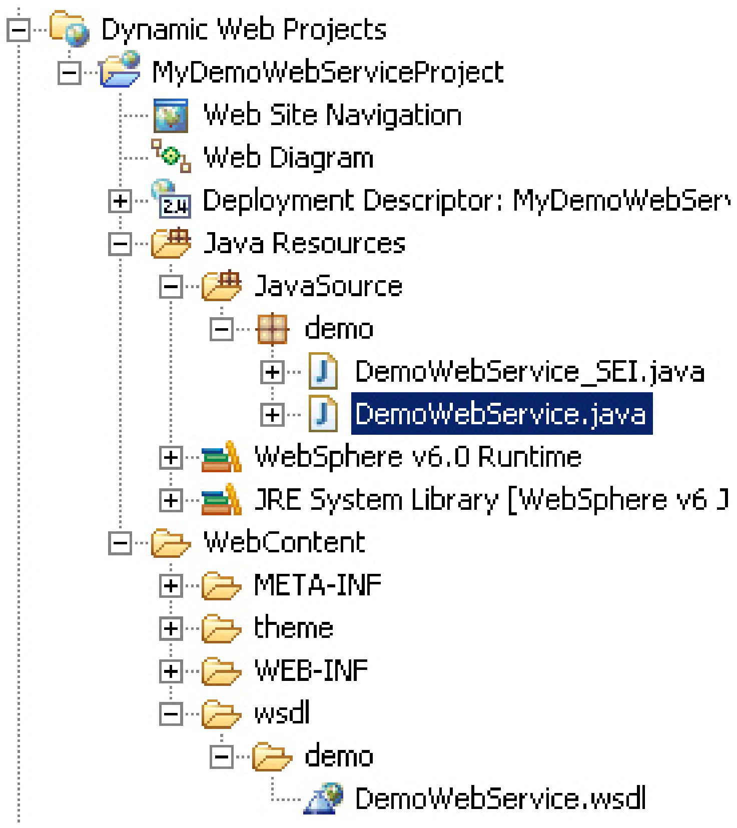
Creating a Web service is extremely simple. To create a Java Web service:

1. Right-click on **DemoWebService.java Java class > New > Other**.
2. In the New Wizard window, select the check box **Show All Wizards** to see all the available wizards for the capabilities that are not activated.
3. Expand **Web Services** and select **Web Service**. Click **Next**. Click **OK** if you are asked to enable the "Web Services Development" capability.
4. By default **Java Bean Web Service** should be selected as the Web service type.
5. Click **Finish**. The Web Service wizard starts the WebSphere Application Server and deploys the generated Web service to the server. Now the Web service is already up and running.
   Figure 8. Creating a Web Service



When the wizard is complete, you have successfully created your first Web service. You might see some new artifacts were created. They are DemoWebService_SEI.java and the DemoWebService.wsdl. Use the WSDL file in the next step when invoking this Web service.

Figure 9. Demo Web service



Creating a client Web project

Create another Web project that hosts the JSP client, which acts as a client to invoke the Web service:

1. From the workbench, select **File > New > Project**.
2. Select **Web > Dynamic Web Project > Next**. Select **OK** if prompted to enable Web Development capability.
3. Type `MyWebServiceClientProject` as the project name. Click **Finish**. Select **Yes** if prompted to switch to the Web perspective.

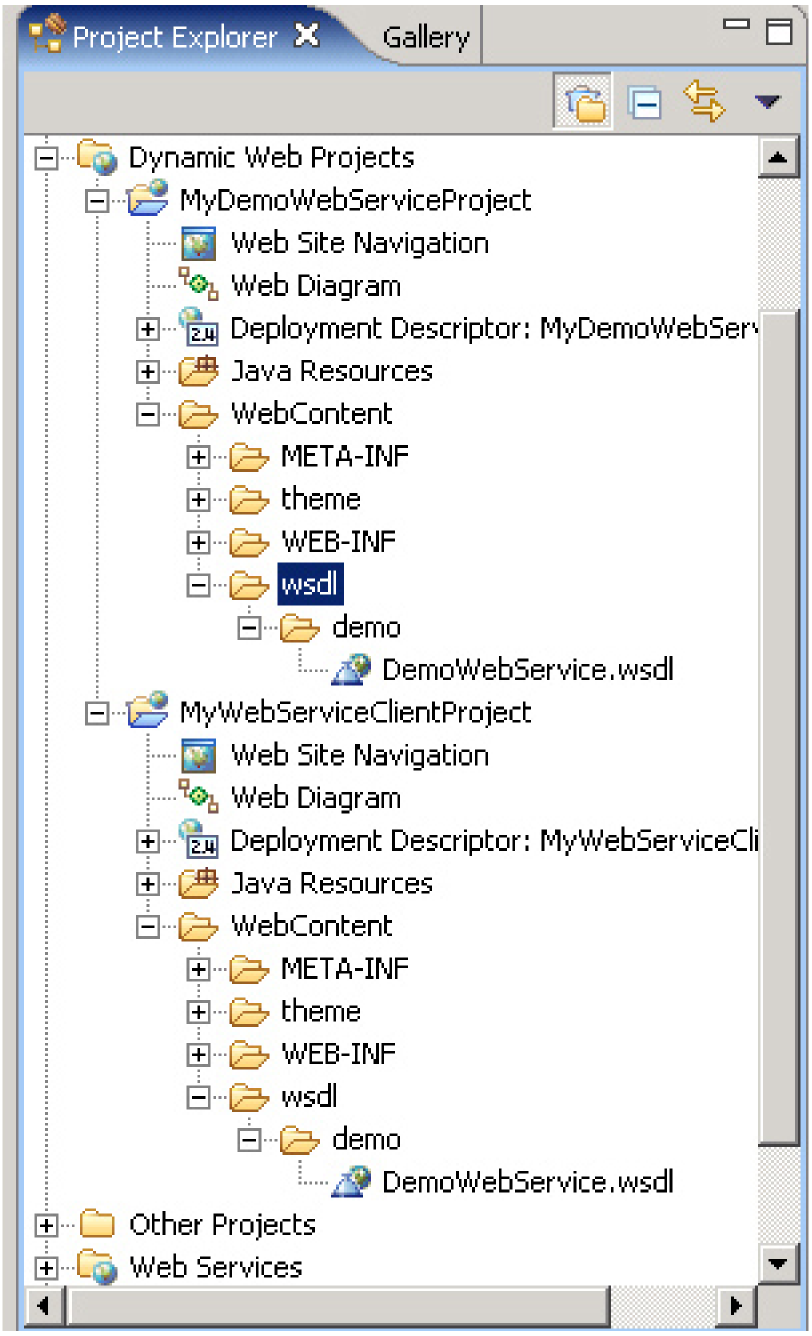
Copying the WSDL file from the host project

Copy the WSDL file from the MyDemoWebServiceProject to your client Web project:

1. In the Project Explorer view, expand **MyDemoWebServiceProject** > **WebContent**.
2. Copy and paste the **wsdl** folder from the MyDemoWebServiceProject / WebContent location to the MyWebServiceClientProject / WebContent

location.
Make sure the wsdl file is copied over to MyWebServiceClientProject under the WebContent folder:

Figure 10. Copying the wsdl file to MyWebServiceClientProject



In "real-life" work settings, if you are invoking an external Web service, request the WSDL file from the service provider.

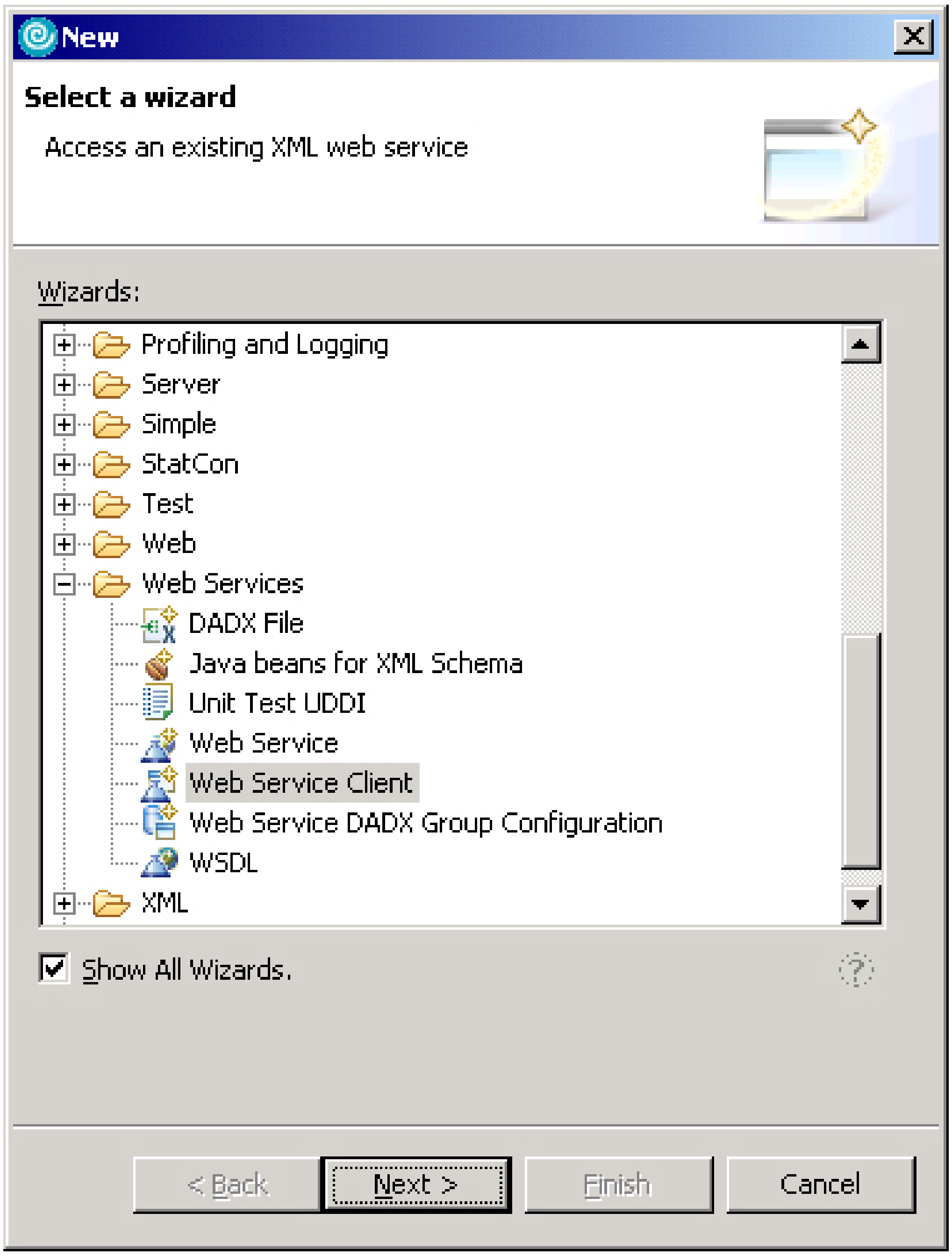
Generating a proxy

Generate a Web service client from the WSDL file that you copied previously:

1. Expand the wsdl folder under MyWebServiceClientProject. Right-click **DemoWebService.wsdl > New > Other**.

2. In the New Wizard window, select the Show All Wizards check box to see all the available wizards, including those for the capabilities not currently activated.

3. Expand **Web Services** and select **Web Service Client**. Click **Next**. Click **OK** if you are asked to enable the Web Services Development capability:
Figure 11. Creating a Web service client



New

Select a wizard

Access an existing XML web service

Wizards:

- Profiling and Logging
- Server
- Simple
- StatCon
- Test
- Web
- Web Services
  - DADX File
  - Java beans for XML Schema
  - Unit Test UDDI
  - Web Service
  - Web Service Client
  - Web Service DADX Group Configuration
  - WSDL
- XML

[✓] Show All Wizards.

4. In the resulting window, make sure **Java proxy** is selected. Click **Finish**. The wizard creates a Java proxy which is simply a Java class that can invoke the Web service.

5. Click **Yes** if you are asked to replace the web.xml file. If you get an exception, it is normal and safe to ignore it.

6. The proxy named DemoWebServiceProxy is generated in the Java Resources *JavaSource* folder. In the next step, create a JSP file to use this proxy class to call the Web service.

Creating a JSP file

1. In the Project Explorer view, right-click **MyWebServiceClientProject > New > JSP File**.

2. Enter `MyWebServiceClient` as the file name and click **Finish**.

3. Switch to the Source view of the JSP editor and enter the code shown in **bold** as follows:
   Listing 3. Code for MyWebServiceClient.jsp

```
<BODY>
<P>Place content here.</P>
<%
double quote =0;
try {
        demo.DemoWebServiceProxy proxy = new demo.DemoWebServiceProxy();
        quote = proxy.getQuote("IBM");

}catch (Exception e) {
e.printStackTrace();
}
 %>
<%="Quote: " + quote %>
</BODY>
```

4. Save the file by pressing Ctrl-S.

Running the JSP file in a browser

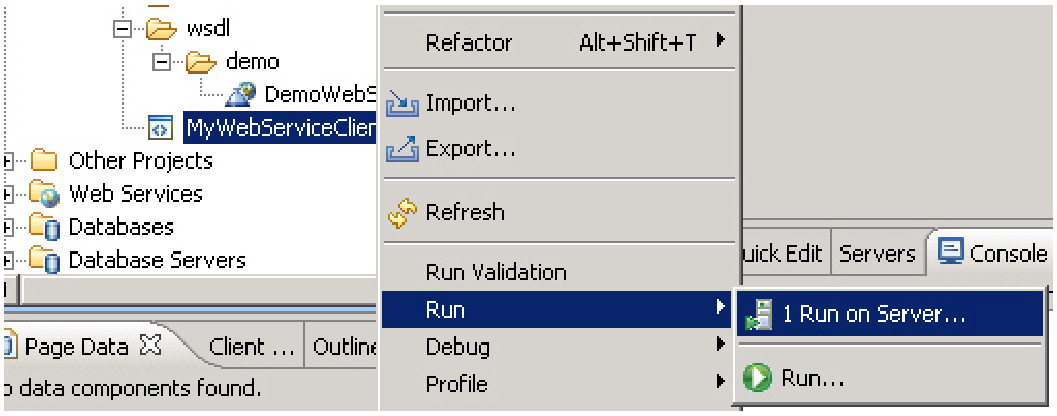
1. Start the server from the Server view and then run the JSP file using the URL: http://localhost:9080/MyWebServiceClientProject/MyWebServiceClient.jsp . Alternatively, right-click the JSP file and select **Run on Server**.

2. In the Project Explorer view, right-click the MyWebServiceClient.jsp file and select **Run > Run on Server**.

3. Select an existing server and click **Finish** to automatically add the project

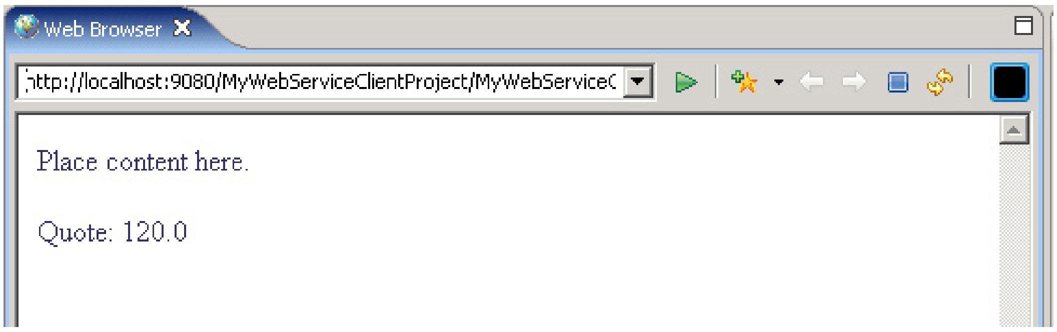
to the server and run the JSP file in the internal browser:

Figure 12. Running the MyCurrencyClient.jsp



4. The JSP file calls the proxy, which in turn invokes the Web service. In this exercise, both the Web service and the client run on the same WebSphere Application Server, but they can also run on different servers. Because the client only needs the WSDL file to invoke the Web service, it doesn't matter where it is hosted.

Figure 13. Running the MyWebServiceClient.jsp


Web Browser
http://localhost:9080/MyWebServiceClientProject/MyWebServiceC

Place content here.

Quote: 120.0

Section 6. Creating a Web application that accesses a database

In the final exercise, create a Web application that can access a Cloudscape database using a Relational Record List. First create a Web project, and then later add the database display capability to the Web application.

Would you like to see these steps demonstrated for you?

Show me

Creating a new Web project

Web application development can be done in the Web perspective. All Web elements, such as JSP pages and Java servlets, are stored in Web projects.

A J2EE application consists of several components: *Application Client*, *Web*, and *Enterprise JavaBeans (EJBs)* modules. Each *module* is a self-contained unit that can be deployed and run on a server individually or as a group in an enterprise application. You work with a Web module in this exercise.

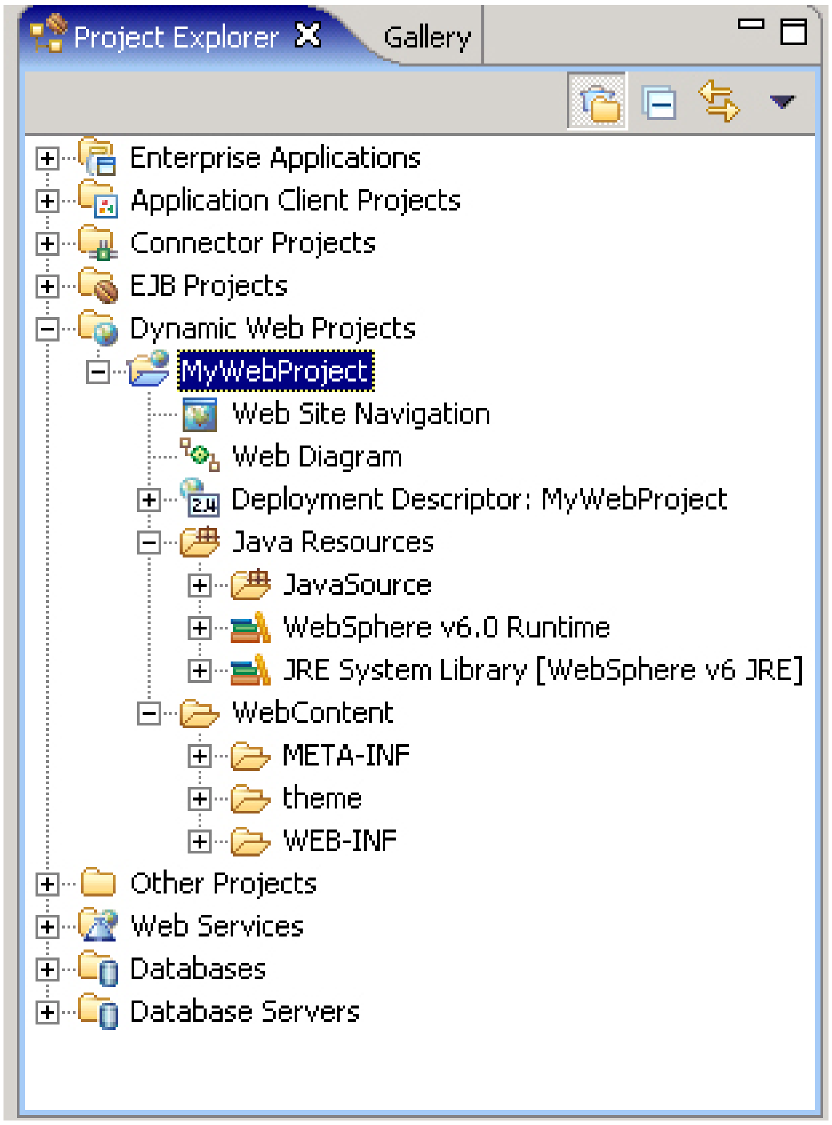
1. From the workbench, select **File > New > Project**.
2. Select **Web > Dynamic Web Project > Next**.
3. Enter `MyWebProject` as the project name. The Show Advanced button shows the advanced settings. Click it to display the default enterprise application project named MyWebProjectEAR. Click **Finish**.
4. When you are asked to switch to the Web Perspective, click **Yes**.

Creating a JSP file

The major contents of a Web project are JSP files and servlets. Other Web elements, such as HTML, style sheets, and images, are also stored in a Web project. Figure 14 shows the structure of a Web project, with its different Web elements in different locations.

After you create a Web project, you can see it in the Dynamic Web Projects folder. The Web project is set up in compliance with the J2EE standard to be exported as a WAR file for deployment. Java servlets and other supporting Java classes are stored in the JavaSource folder inside the Java Resources directory. JSP files, HTML pages, style sheets, Java script files, and images are stored in the WebContent directory. The Web Deployment Descriptor, web.xml, contains the deployment configurations of the Web application. It must be present in the Web module to be deployed on a J2EE server, such as WebSphere Application Server.

Figure 14. Structure of a Web project



1. Right-click on **MyWebProject > New > JSP File**.
2. Enter `MyFirstJSP` as the file name and click **Finish**. Notice that the JSP file is created under the folder WebContent.
3. Modify the JSP file from the editor using the *Design* or the *Source* view. Switch to the Source view.
4. Enter the **bold** code in the editor. This line is called a scriptlet, which is Java code enclosed by <% and %>. You can enter any Java code within these brackets. You can also use scriptlets to print something in a JSP file. For example, <%="hello"%> prints the text hello.

Listing 4. Code for MyWebServiceClient.jsp

```
<BODY>
<P>Place content here.</P>
<%
String a = "Hello";
a+= "World!!";
 %>
<%=new java.util.Date()  %>
```

```
<%=a %>
<BR>
</BODY>
```

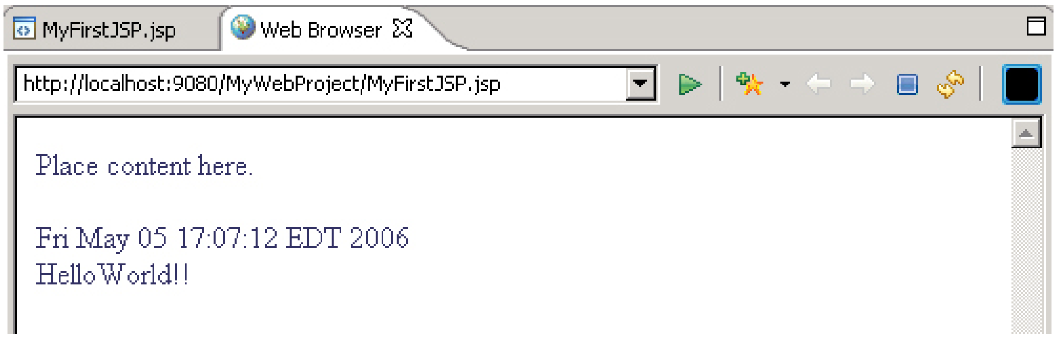
5. Press Ctrl-S to save.

Running the JSP file in a browser

With Application Developer you can run the JSP file in the embedded server. Start the server from the Server view and then run the JSP file using the URL http://localhost:9080/MyWebProject/MyFirstJSP.jsp. Alternatively, right-click the JSP file and select **Run on Server**.

1. In the Project Explorer view, right-click the MyFirstJSP.jsp file and select **Run > Run on Server**.

2. Select an existing server and click **Finish** to automatically add the project to the server and run the JSP file in the internal browser:
Figure 16. Running the MyFirstJSP



http://localhost:9080/MyWebProject/MyFirstJSP.jsp

Place content here.

Fri May 05 17:07:12 EDT 2006
HelloWorld!!

Creating a database connection

Application Developer has a number of relational database tools, or *views*, that can be accessed through the Data perspective. Some of the more important views in this perspective are the Data Definition, Database Explorer, and DB Output views.

In the Database Explorer view, connections can be made to a list of supported relational databases. Some of the supported databases are IBM Cloudscape, IBM DB2 Universal Database, Microsoft SQL Server, Sybase Enterprise Systems, and Oracle Database. Refer to the product help for a complete list of the supported databases.

In this step, create a database connection to a Cloudscape database that is shipped with Application Developer. Cloudscape database can use any file system folder as a database. You do not need to create the database before trying to establish a connection. If you select **Create the database if required**, the Cloudscape database is created before the connection is established.

Cloudscape accepts only one database connection at a time. For example, if a WebSphere Application Server is already connected to the Cloudscape database, an attempt to make a connection from the Database Explorer will fail.

Following are the steps to create a Cloudscape connection:

1. If the server is running, stop it now or you cannot establish a database connection. In the Servers view, right-click on **WebSphere Application Server v6.0** and select **Stop**.

2. Switch to the Data perspective. Select **Window > Open Perspective > Other**. Select the Show all check box and select **Data**. Click **OK**. Select **OK** if asked to enable the Core Database Development capability.

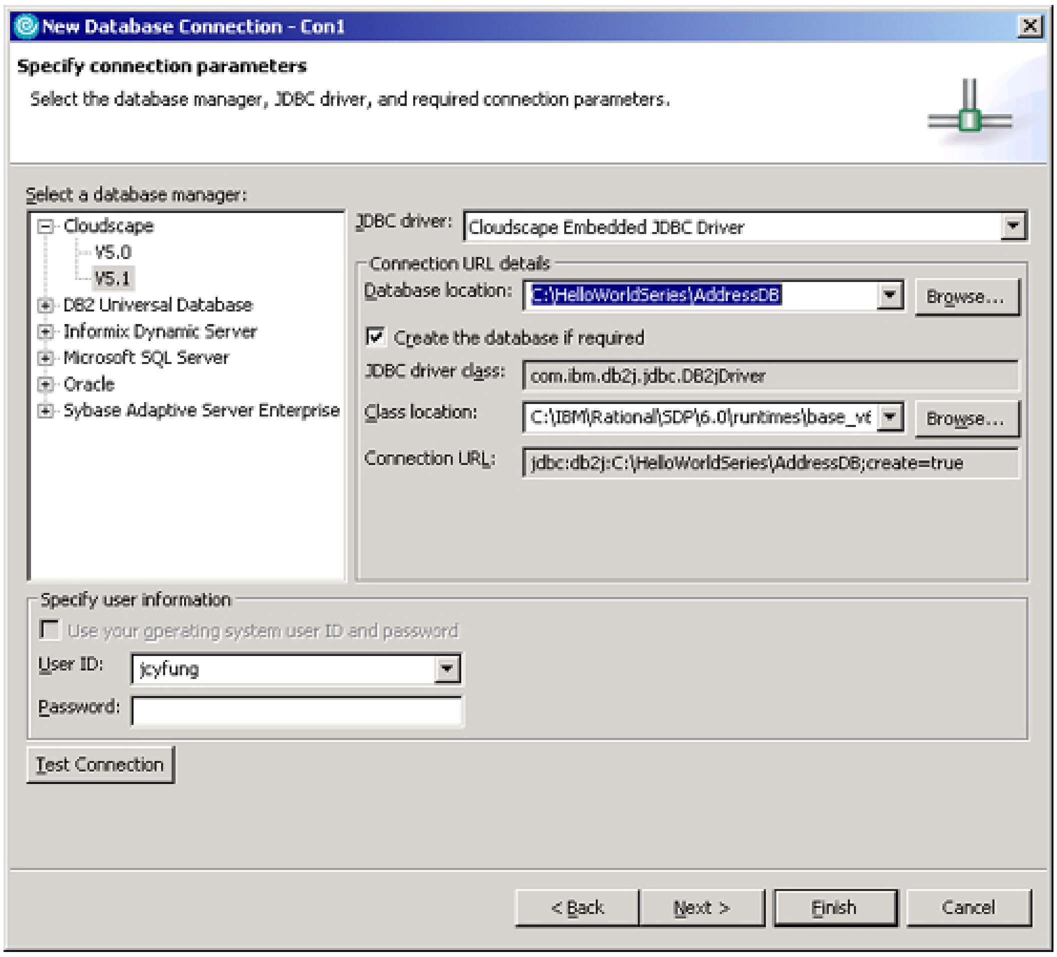
3. Right-click **Database Explorer**, located in the bottom left of the perspective and click **New Connection**.

4. Click **Choose a database manager and JDBC driver**. Enter `Con1` as the connection name. Click **Next**.

5. The database manager defaults to Cloudscape V5.1 as shown in Figure 17. Enter any file system directory in the **Database location**, for example, `C:\HelloWorldSeries\AddressDB`. You do not have to physically create the folder in the file system. Just type in a new folder name. Make sure the Create the database if required box is checked. If it is, the tool sets up the Cloudscape database at that location. Because Cloudscape is a test database, you do not need a user ID and password to connect to it. Enter any user ID in the user information section.

6. Click **Test Connection** to see if a connection can be established. You should see a window saying the connection is successful. Click **Finish** to create the connection. If it returns an error stating the user name is invalid, leave both user ID and password blank.

7. Click **No** when asked to copy database metadata to a project folder.
   Figure 17. Creating a Cloudscape database connection


New Database Connection - Con1

Specify connection parameters

Select the database manager, JDBC driver, and required connection parameters.

Select a database manager:
- Cloudscape
  - V5.0
  - V5.1
- DB2 Universal Database
- Informix Dynamic Server
- Microsoft SQL Server
- Oracle
- Sybase Adaptive Server Enterprise

JDBC driver: Cloudscape Embedded JDBC Driver

Connection URL details

Database location: C:\HelloWorldSeries\AddressDB

[✓] Create the database if required

JDBC driver class: com.ibm.db2j.jdbc.DB2jDriver

Class location: C:\IBM\Rational\SDP\6.0\runtimes\base_v6

Connection URL: jdbc:db2j:C:\HelloWorldSeries\AddressDB;create=true

Specify user information

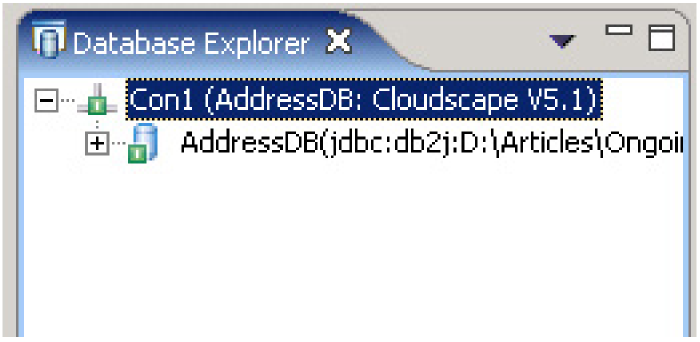
[ ] Use your operating system user ID and password

User ID: jcyfung

Password:

8. **Con1** displays in the Database Explorer view. Expand the connection to see AddressDB as shown in Figure 18:

Figure 18. Database Explorer Con1 connection

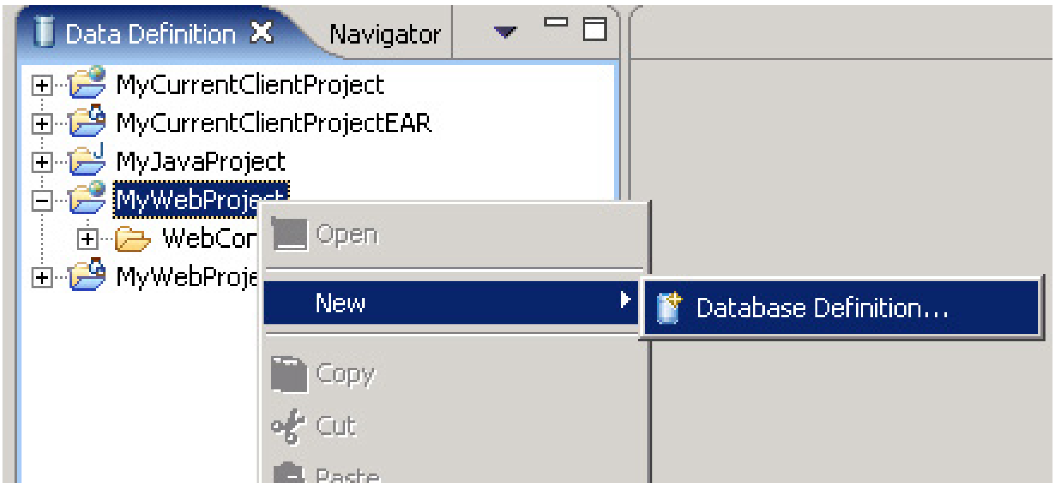

Database Explorer

Con1 (AddressDB: Cloudscape V5.1)

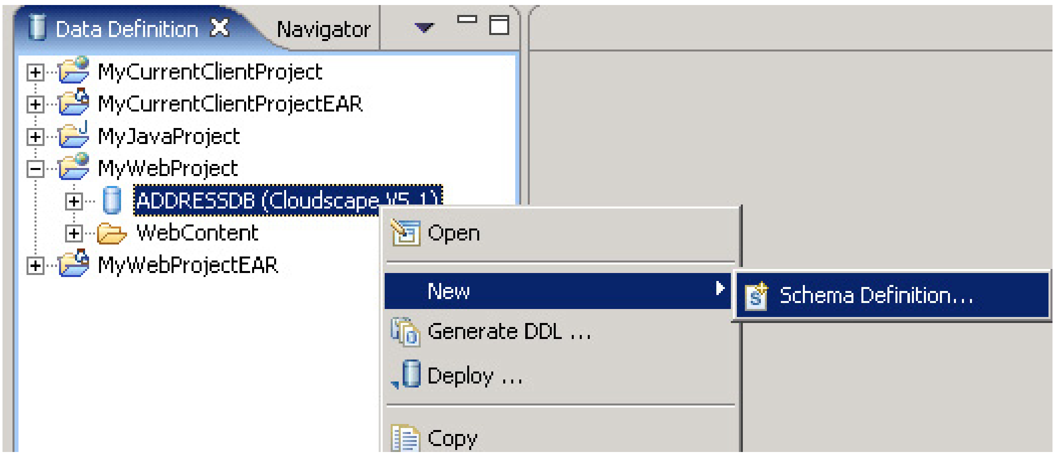
AddressDB(jdbc:db2j:D:\Articles\Ongoi

Creating a database definition

Right now, the database is empty. Use the Data Definition view to create a schema and table definition. The *database definition* is something you define in the workspace which doesn't exist in the database. Later in this hands-on, deploy the definitions to an actual database.

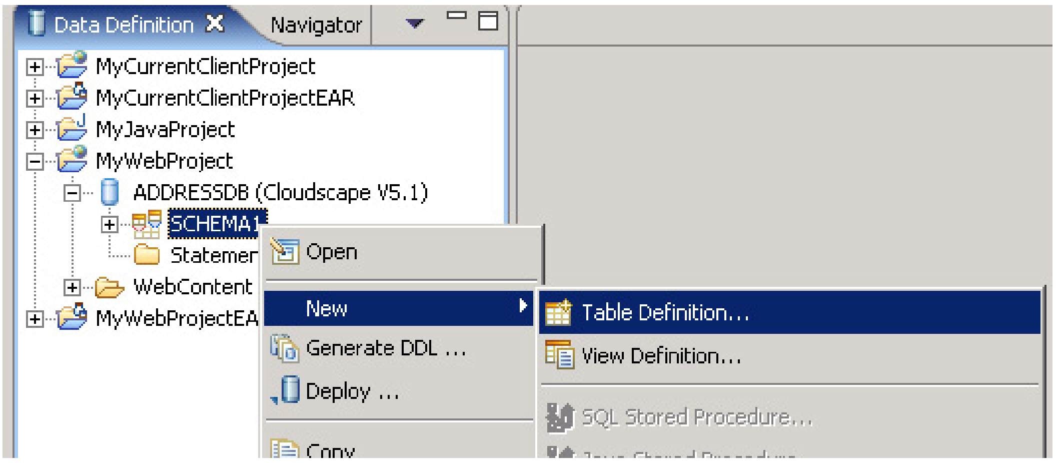
1. In the Data perspective's Data Definition view, right-click **MyWebProject** and click **New > Database Definition**:
   Figure 19. Creating a database definition
   

2. Enter `AddressDB` as the database name. Select **Cloudscape V5.1** as **Database vendor type**, and click **Finish**.

3. Expand **MyWebProject**. Right-click **ADDRESSDB** and click **New > Schema Definition**:
   Figure 20. Creating a schema definition
   

4. Accept all the defaults in the schema definition window. Click **Finish**.

5. Create a table definition named `AddressTable`. Right-click on the newly created **SCHEMA1 > New > Table Definition**:
   Figure 21. Creating a table definition



6. Enter `AddressTable` as the table name, and click **Next**.

7. Click **Add Another** to add a column. Enter `id` as the column name. Select **INTEGER** as the column type and click the Key column check box.

8. Click **Add Another**. Enter `name` as the column name. Select **VARCHAR** as the column type. Change the string length to **100**.

9. Click **Add Another**. Enter `address` as the column name. Select **VARCHAR** as the column type. Change the string length to **200**.

10. Click **Finish**.
Figure 22. Table columns for AddressTable

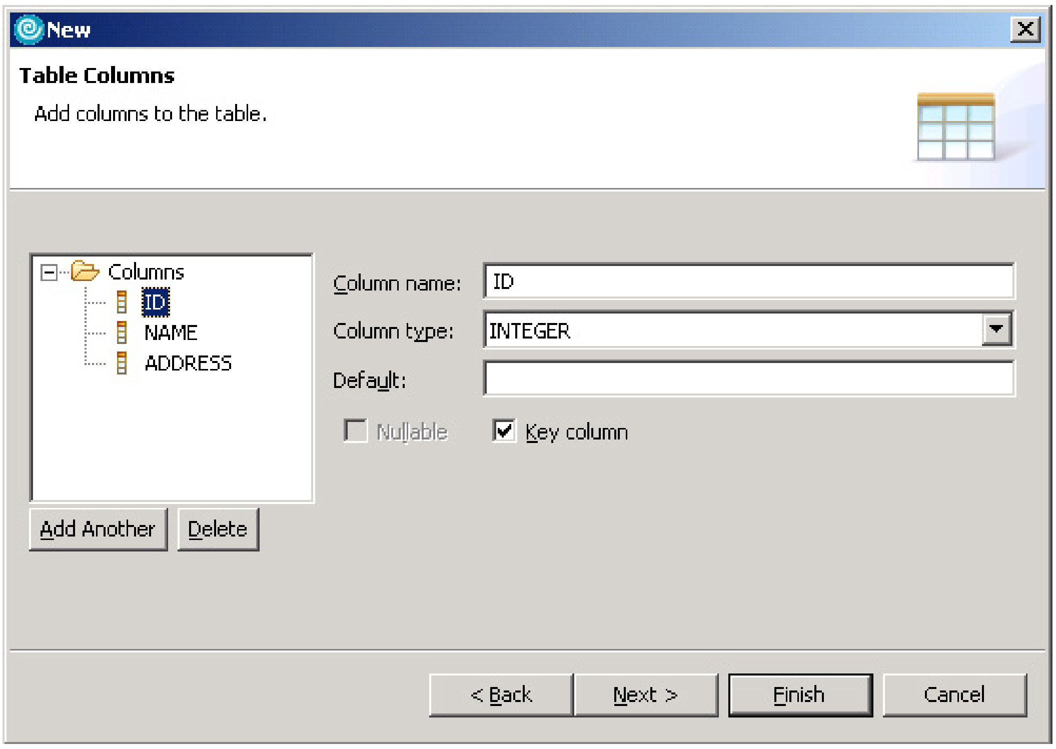

New

Table Columns

Add columns to the table.

Columns
- ID
- NAME
- ADDRESS

Column name: ID

Column type: INTEGER

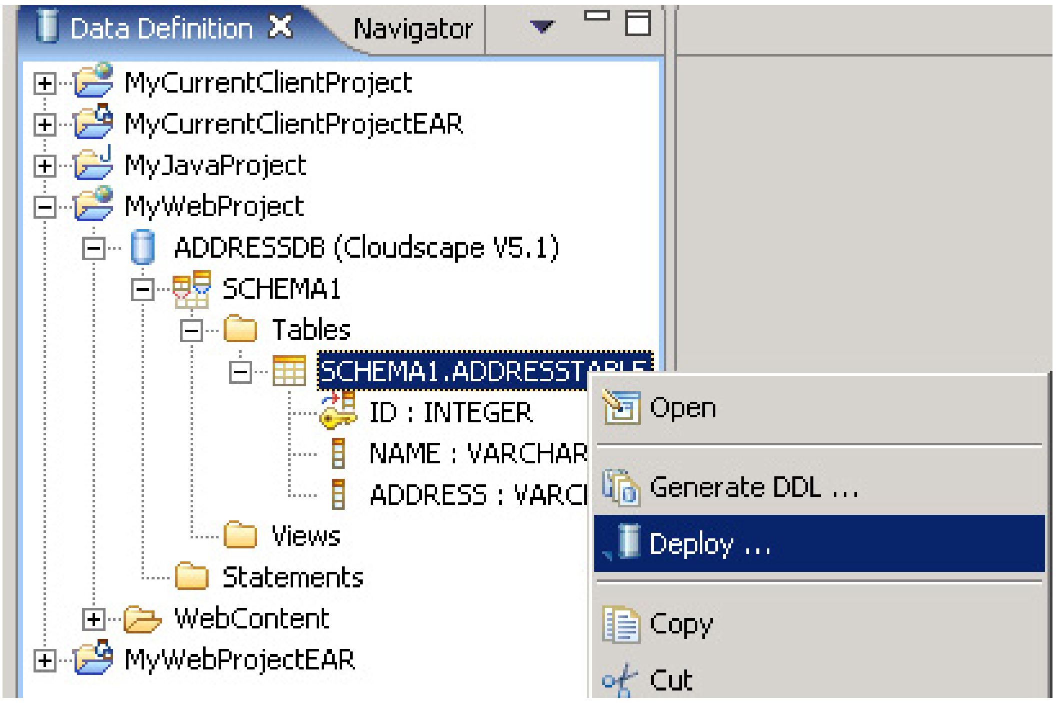
Default:

[ ] Nullable   [✓] Key column

11. In the Data Definition view, right-click the newly created

SCHEMA1.ADDRESSTABLE and click **Deploy**. Click **Next**, and then click **Next** again. This creates SCHEMA1 and AddressTable in the Cloudscape database:

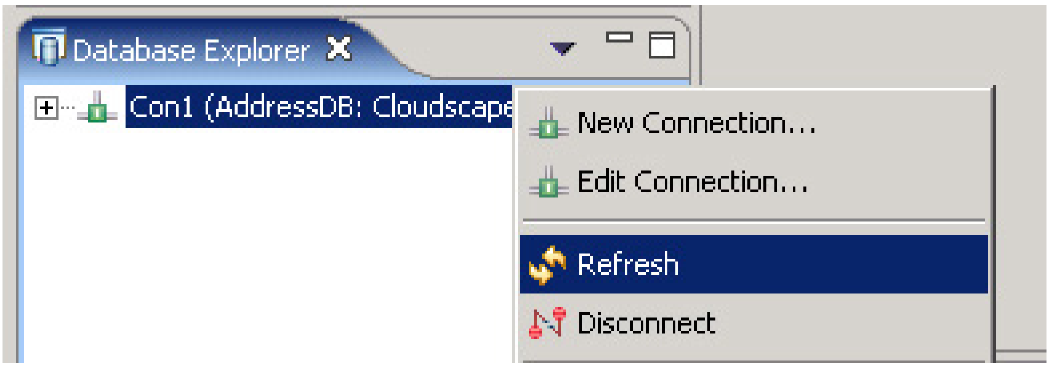
Figure 23. Deploy the schema definition



12. Click the Use existing connection check box. Select **Con1** as the database connection. Click **Finish**.

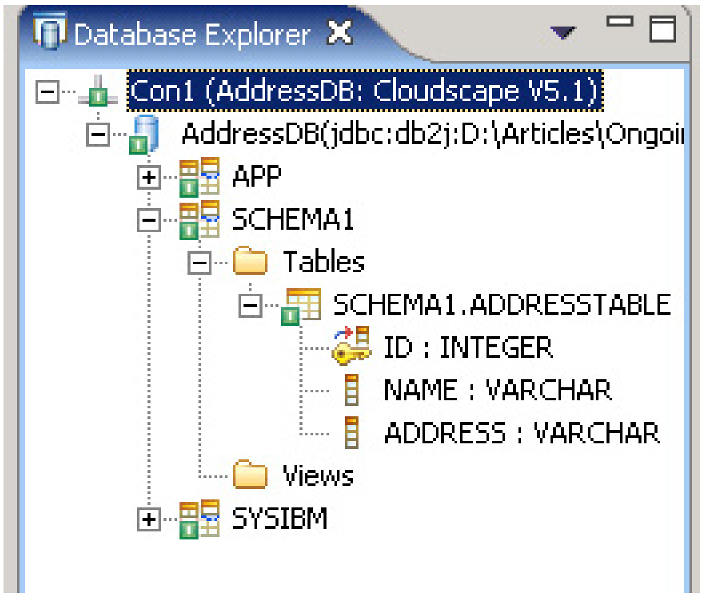
13. In the Database Explorer, right-click your connection **Con1** and click **Refresh**:

Figure 24. Refresh on the database connection



14. Expand your connection until you see the Tables folder. You see the schema and AddressTable you just created:

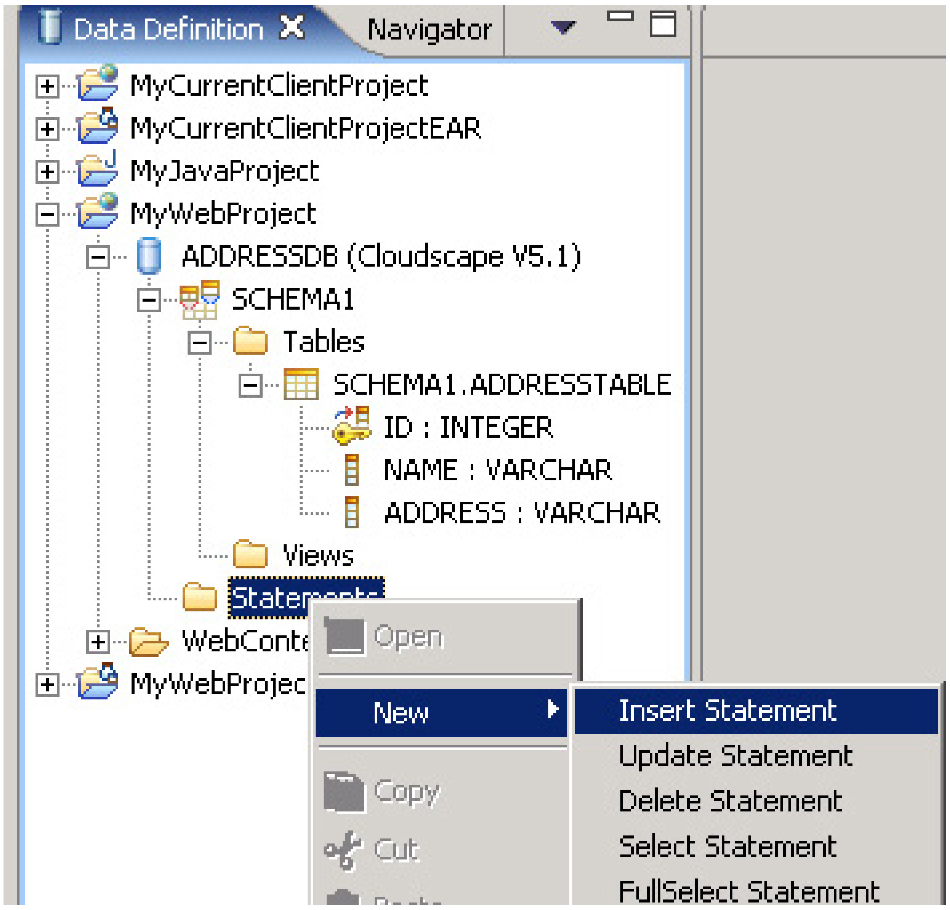
Figure 25. After the schema definition is deployed



Creating an insert statement

Although you have created the table in the database, it has no data. Create an Insert statement that can insert a record into the table.

1. In the Data Definition view, expand **MyWebProject > ADDRESSDB**. Right-click **Statements** and click **New > Insert Statement**:
   Figure 26. Creating an insert statement



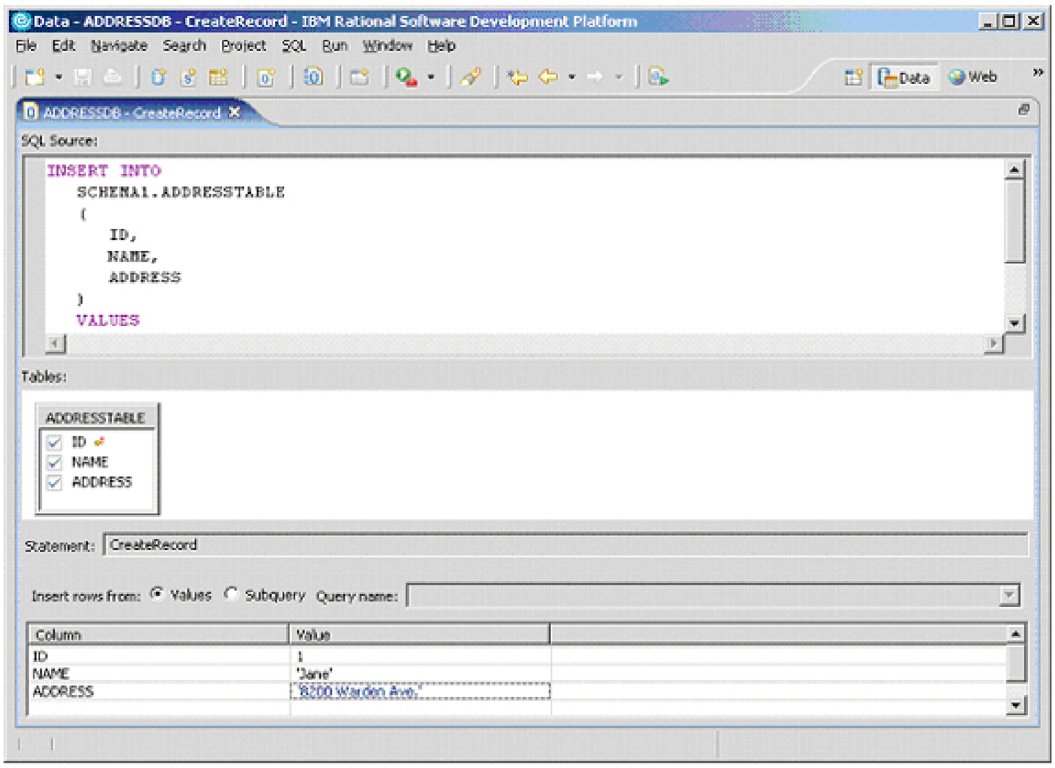
2. Enter `CreateRecord` as the name, and click **OK**.

3. Right-click in the middle panel labeled Tables and click **Add Table**. Select **SCHEMA1.ADDRESSTABLE** as the table name. Click **OK**.

4. Click all three check boxes in the AddressTable to select all the columns in AddressTable.

5. In the bottom panel of the editor, enter `1` for the ID column value. Enter `'Jane'` as the name value, `'8200 Warden Ave'` as the address value.

6. Save the file and close the editor.
Figure 27. Insert Statement editor


Data - ADDRESSDB - CreateRecord - IBM Rational Software Development Platform
File Edit Navigate Search Project SQL Run Window Help
Data
Web
ADDRESSDB - CreateRecord

SQL Source:

```
INSERT INTO
   SCHEMA1.ADDRESSTABLE
   (
      ID,
      NAME,
      ADDRESS
   )
   VALUES
```

Tables:

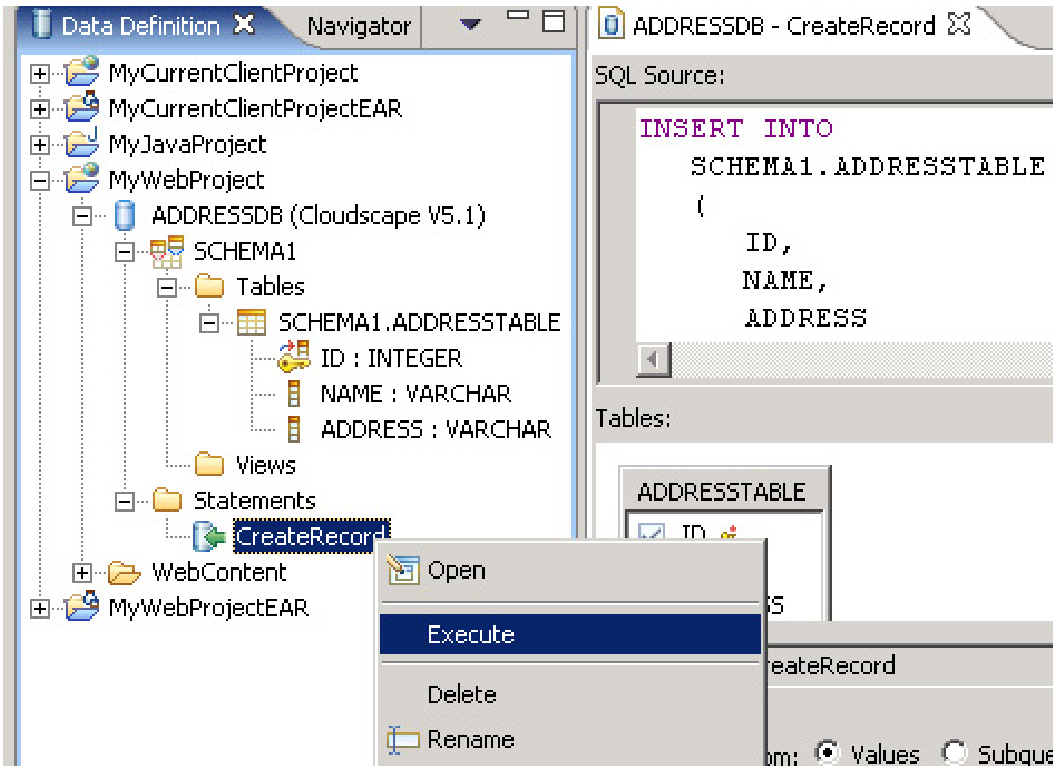
ADDRESSTABLE
- ✓ ID
- ✓ NAME
- ✓ ADDRESS

Statement: CreateRecord

Insert rows from: [✓ Values] Subquery   Query name:

Column	Value
ID	1
NAME	'Jane'
ADDRESS	'8200 Warden Ave.'

7. In the Data Definition view, right-click **CreateRecord** and click **Execute**. Select **Use an existing connection** and **Con1**. Click **Next** and **Finish**. If you get an error stating "Array index out of range: 0", use a new connection instead of reusing an existing connection when executing the statement.
   Figure 28. Execute CreateRecord



```
INSERT INTO
    SCHEMA1.ADDRESSTABLE
    (
        ID,
        NAME,
        ADDRESS
```

8. Go to the Database Explorer view. Right-click the connection **Con1** and click **Refresh**.

9. Expand **AddressDB** in the Database Explorer view until you see SCHEMA1.ADDRESSTABLE. Right-click **AddressTable** and click **Sample Contents**:

Figure 29. Sample contents

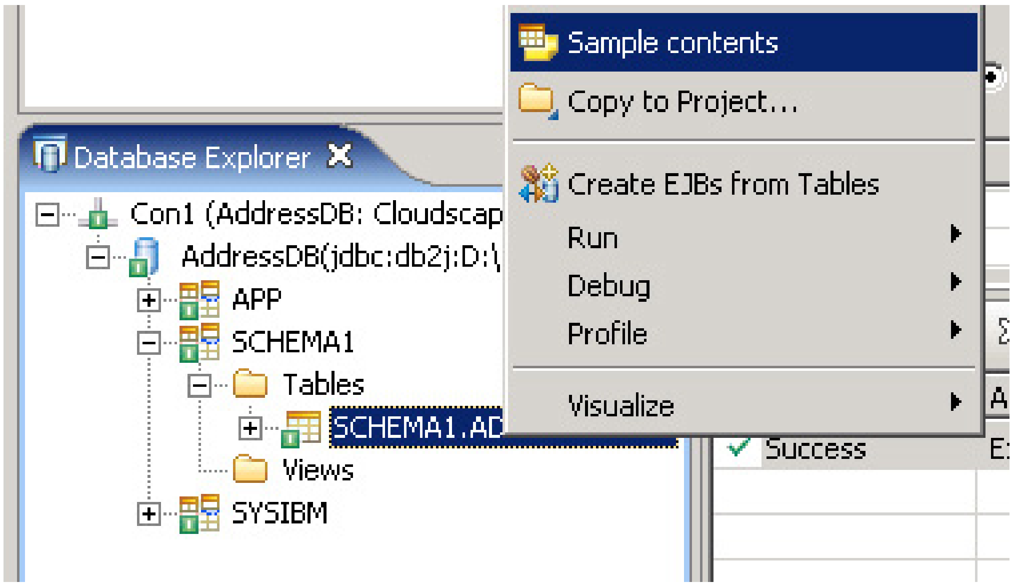
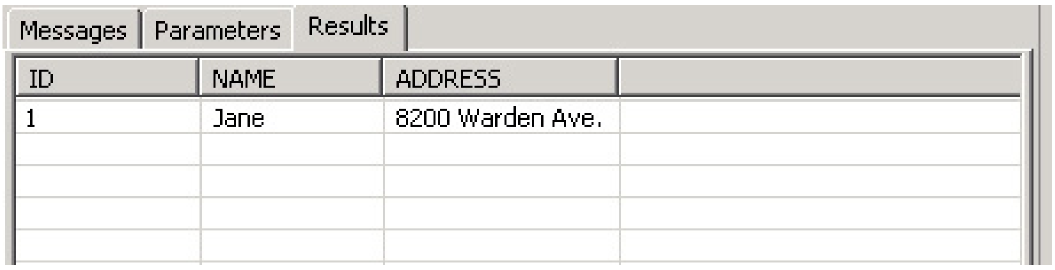


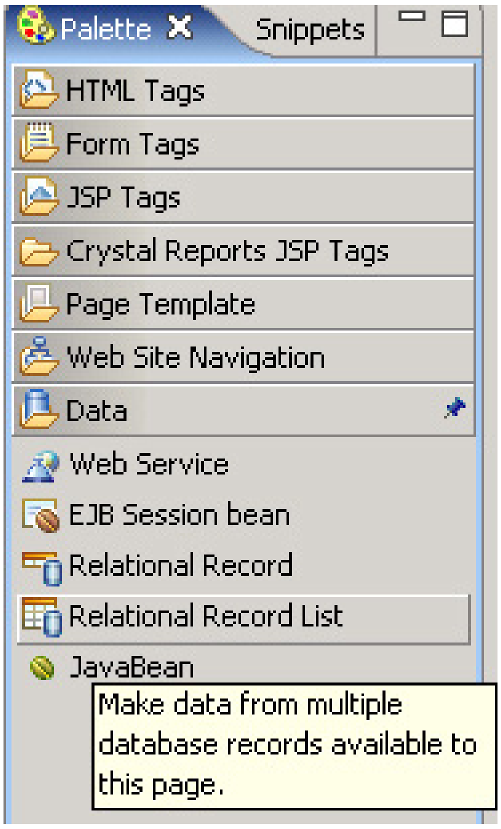
Figure 30. Sample contents results



ID	NAME	ADDRESS
1	Jane	8200 Warden Ave.

Modifying the JSP file to display database content

In this step, add a Relational Record List to the MyFirstJSP.jsp to display the table content. A *Relational Record List* lets you hook up to a database without doing any JDBC coding. Drag and drop the record list to the JSP file to make a read-only list.

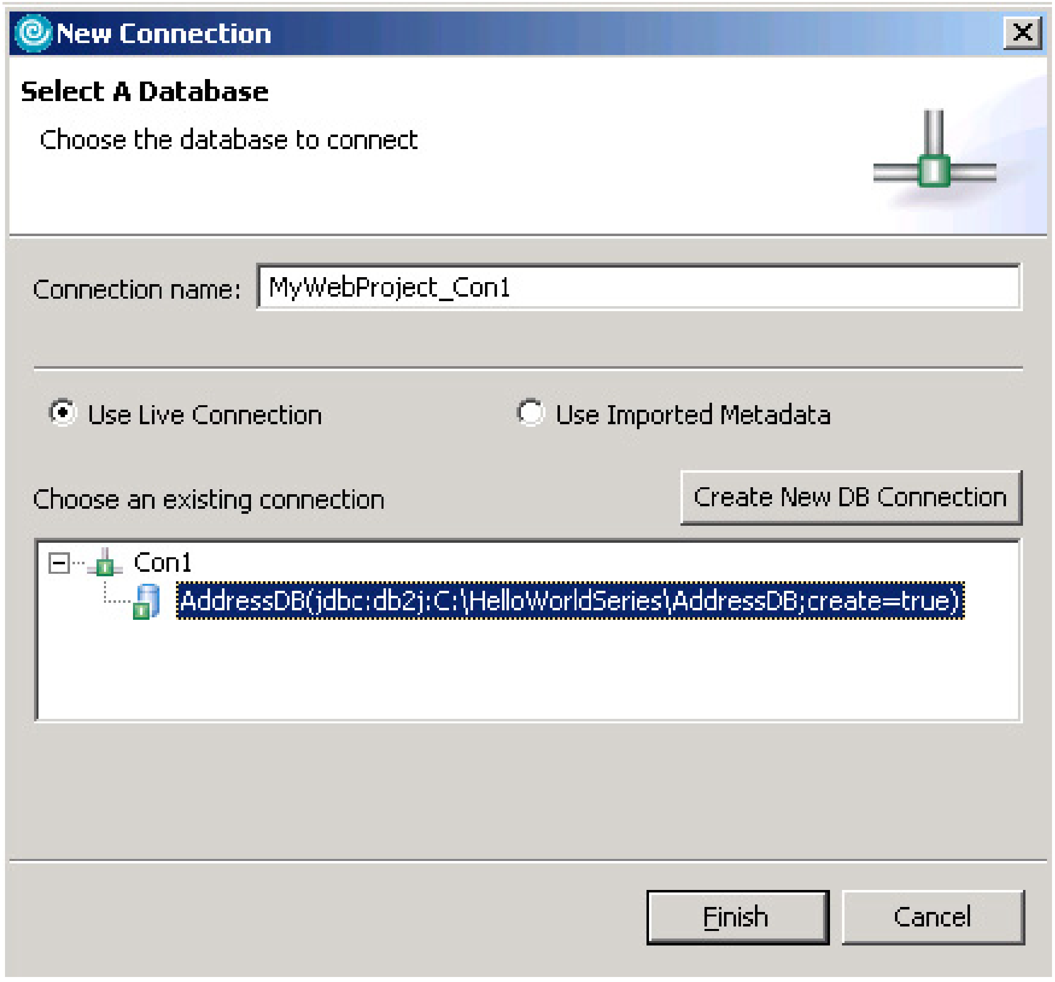
1. Switch to the Web perspective and open MyFirstJSP.jsp in the editor.
2. In the editor, switch to the Design view. From the Palette, click on **Relational Record List** and drop it to the bottom of the JSP editor's canvas. Use the Relational Record List to make multiple rows of data available to the page, for example all the rows from a table.
   Figure 31. Relational Record List
   

   Make data from multiple database records available to this page.
3. Enter `addressTableRecordList` as the Name. Click **Next**.

4. Click on **New** to create a new connection.

5. In the New Connection window, you should see the existing Con1 connection. Select **AddressDB** from Con1 as shown in Figure 32. Click **Finish**:

Figure 32. Create a new connection in the Relational Record List



New Connection

Select A Database

Choose the database to connect

Connection name: MyWebProject_Con1

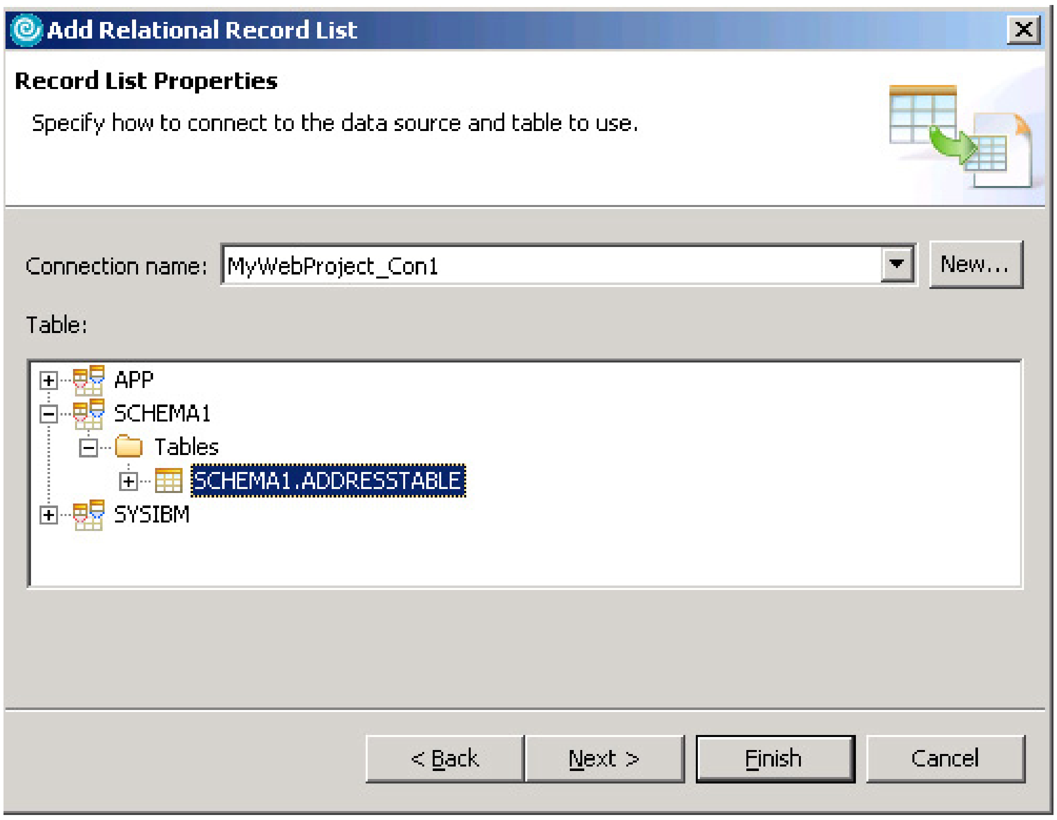
(● Use Live Connection) ○ Use Imported Metadata

Choose an existing connection

Con1
AddressDB(jdbc:db2j:C:\HelloWorldSeries\AddressDB;create=true)

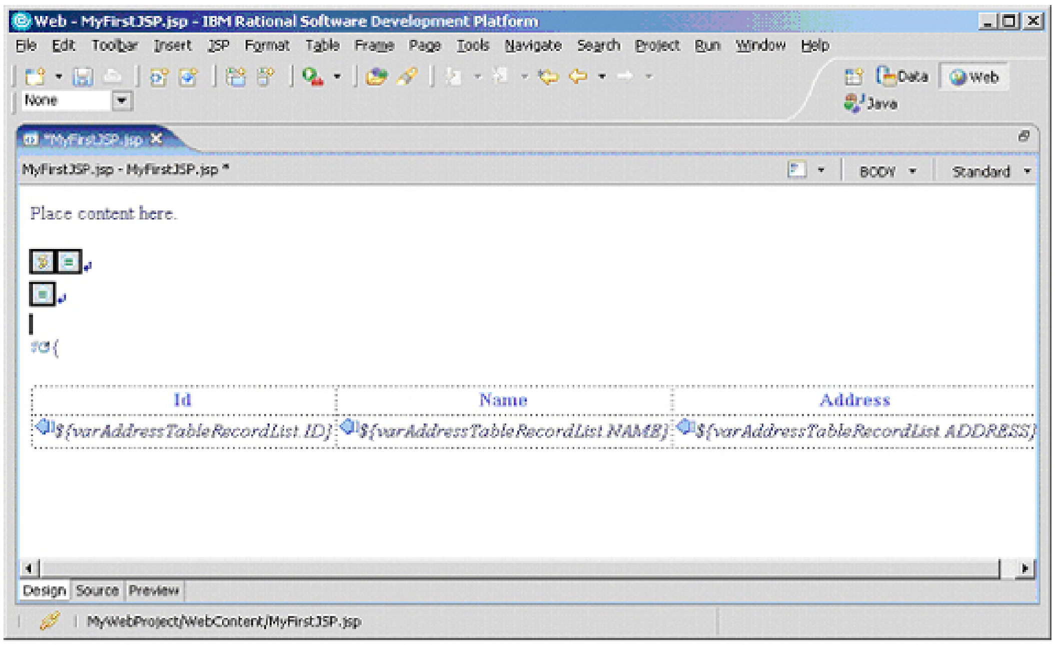
6. Select **SCHEMA1.ADDRESSTABLE** and click **Finish**. A table that displays the database rows is created on the JSP page.

Figure 33. Select a database row for the Relational Record List


Add Relational Record List
Record List Properties
Specify how to connect to the data source and table to use.
Connection name: MyWebProject_Con1
New...
Table:
APP
SCHEMA1
Tables
SCHEMA1.ADDRESSTABLE
SYSIBM
< Back
Next >
Finish
Cancel

7. Save the file. Figure 34 shows the final JSP page:

Figure 34. JSP editor with the Relational Record List


Web - MyFirstJSP.jsp - IBM Rational Software Development Platform
File Edit Toolbar Insert JSP Format Table Frame Page Tools Navigate Search Project Run Window Help
*MyFirstJSP.jsp
MyFirstJSP.jsp - MyFirstJSP.jsp *
BODY
Standard
Place content here.
Id
Name
Address
${varAddressTableRecordList.ID}
${varAddressTableRecordList.NAME}
${varAddressTableRecordList.ADDRESS}
Design Source Preview
MyWebProject/WebContent/MyFirstJSP.jsp

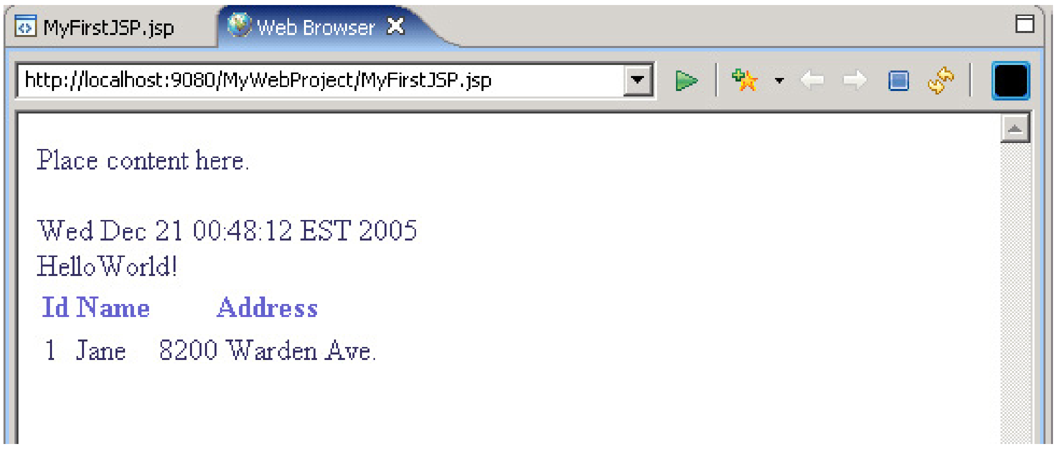
Running the JSP file in a browser

Start the server and run the JSP file.

1. In the Server view, right-click on **WebSphere Application Server v6.0** and select **Start**.

2. Right-click on **WebSphere Application Server v6.0 > Restart Project > MyWebProjectEAR**. Because you have modified the project, restart the EAR to pick up the changes. This prevents the occurrence of a NullPointerException error when you run the JSP file. If the error still persists, add or remove projects from the server. Right-click on the server **> Add or remove projects**, select **MyWebProjectEAR** and remove it from the right pane. Click **Finish**, then repeat the actions to add **MyWebProjectEAR** to the right pane.

3. Wait until the publishing is finished before running the JSP file. In the Project Explorer view, right-click **MyFirstJSP.jsp** and select **Run > Run on Server**. Click **Finish**.

4. The JSP displays the row from the AddressTable. Enter more rows to the AddressTable and run the JSP file again. You should see more rows in the JSP.
Figure 35. Running the JSP file


```
Place content here.

Wed Dec 21 00:48:12 EST 2005
Hello World!
```

Id	Name	Address
1	Jane	8200 Warden Ave.

Section 7. Summary

Wrap-up

This tutorial provided an introduction to Rational Application Developer. You created a Java application, a Web service application, and database applications.

Stay tuned for the next part of this series, which introduces you to WebSphere Integration Developer. To keep up with all the Hello World articles, check the Hello World overview page.

Resources

Learn

- Learn more about the Eclipse platform at Eclipse.
- IBM Service Oriented Architecture (SOA): Learn more about SOA from IBM.
- An Introduction of Rational Application Developer, A Guided Tour

Get products and technologies

- Download a free trial version of Rational Application Developer for WebSphere Software V6.0.
- Evaluate Rational Application Developer without downloading it. To evaluate Rational Application Developer without installing it on your system, try the developerWorks online trial version.
- Build your next development project with IBM trial software, available for download directly from developerWorks.
- Discover new SOA design resources that are available for download.

About the author

Jane Fung
Jane Fung works on the WebSphere Studio Application Developer Integration Edition tools team. Jane earned a Bachelor of Applied Science degree in Electrical Engineering at the University of Waterloo, Ontario, and is a Sun Java 2 Certified Programmer. You can reach Jane at jcyfung@ca.ibm.com .