

Create secure Java applications productively, Part 1

Use Rational Application Developer and Data Studio

Skill Level: Intermediate

[Tyler Anderson \(tyleranderson5@yahoo.com\)](mailto:tyleranderson5@yahoo.com)
Freelance writer and developer
Backstop Media

14 Apr 2008

This is the first in a two-part tutorial series creating secure Java-based Web applications using Rational® Application Developer, Data Studio and Rational AppScan. This first tutorial begins by showcasing how Data Studio with pureQuery can increase the efficiency of your database-driven Web development. You will be developing a Java™ Web application with Rational Application Developer, and then with Java Server Pages (JSP) you will deploy the application on WebSphere® Application Server.

Section 1. Before you start

This tutorial is recommended for Java and database application developers who want to take advantage of using Data Studio for its pureQuery capabilities and Rational Application Developer to facilitate the development of a Java-based Web application. Knowledge of JSP and Web development in general is also useful for this tutorial, but is not required.

About this series

This two-part series aims to broaden your Web application development skills through the use of Rational Application Developer, Data Studio and Rational

AppScan.

- Part 1 uses the IDE capabilities of Rational Application Developer and the pureQuery features of Data Studio to efficiently create a Java-based wealth management Web application.
- Part 2 takes advantage of the many Rational AppScan features available to harden/make secure the Java application by discovering vulnerabilities and fixing them so that you can be confident about deploying your Web application.

About this tutorial

This tutorial shows you how to use Data Studio to develop applications more efficiently by automatically creating reliable database connection classes. The efficiencies provided by Data Studio are shown by creating a Java-based Web application using Rational Application Developer and Data Studio that is deployed to WebSphere Application Server. You will also learn how to create and set up an application using Rational Application Developer and Data Studio, including:

- Creating a DB2 database connection
- Creating a pureQuery enabled dynamic Web application
- Generating pureQuery code from database tables
- Augmenting pureQuery code with custom queries
- Creating JSP that uses the pureQuery classes
- Deploy and test on WebSphere Application Server

You'll learn and implement the above concepts by creating a wealth management Web application that users will be able to log into and view their various holdings. The delayed value of stock and options holdings will be grabbed from Yahoo! Finance API. You'll also create a form to manually add new real estate holdings.

System requirements

For this tutorial you're only going to need three products:

- [Rational Application Developer](#)
Get a trial version of Rational Application Developer 7.0.0.6 from IBM (click **Download using the new IBM Installation Manager (recommended)** link). You'll use Rational Application Developer as a full

featured Eclipse based IDE for developing your Java Web application.

- [Data Studio](#)
This free plugin contains the pureQuery capabilities you'll use for efficient database driven Web development. This tutorial requires version 1.1.2 (the only version compatible with Rational Application Developer 7.0.0.6).
- [DB2 Express-C](#)
Data Studio requires a live database connection to generate pureQuery code for. This tutorial uses DB2 Express-C for the database.

Note that this tutorial should work with future releases of Rational Application Developer and Data Studio that are compatible (for example, Rational Application Developer 7.0.0.7 and Data Studio 1.1.3, if these versions were to exist). Please refer to IBM as to which of these future versions will be compatible with each other. In the mean time, Rational Application Developer 7.0.0.6 and Data Studio 1.1.2 are known to be compatible with each other.

Section 2. Install Rational Application Developer and Data Studio to the same package group

Before you can get started using Rational Application Developer and Data Studio together, you'll need to install them into the same package group. Follow along here to make sure you get the correct version of each product.

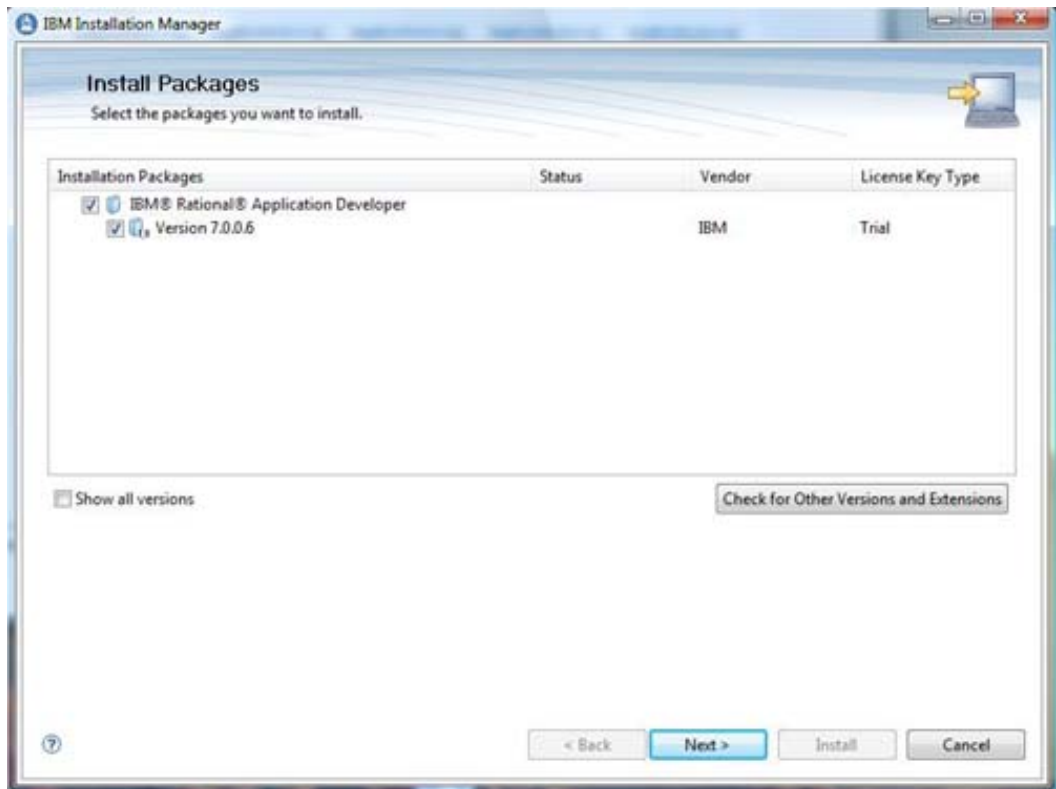
You will also have to install DB2 Express-C, although we won't cover that in this tutorial. For help installing DB2 Express-C and creating databases and database tables, see the [Resources](#) section. After you create a new database and a database table, manually insert new rows in the database by double clicking the icon of your newly created table in the main view of the control center.

Install Rational Application Developer

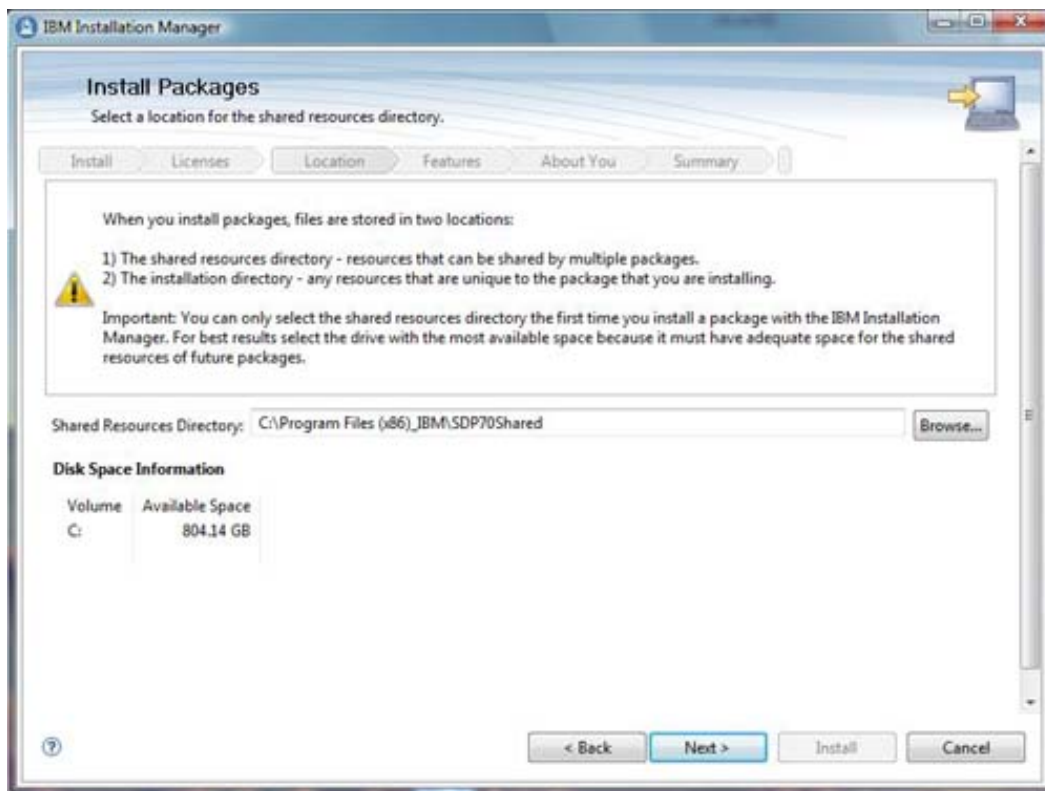
See the link in [System requirements](#) to use the IBM Installation Manager that will download and install only the files required by your installation options.

1. When you click the file you download above, you'll see the following screen:

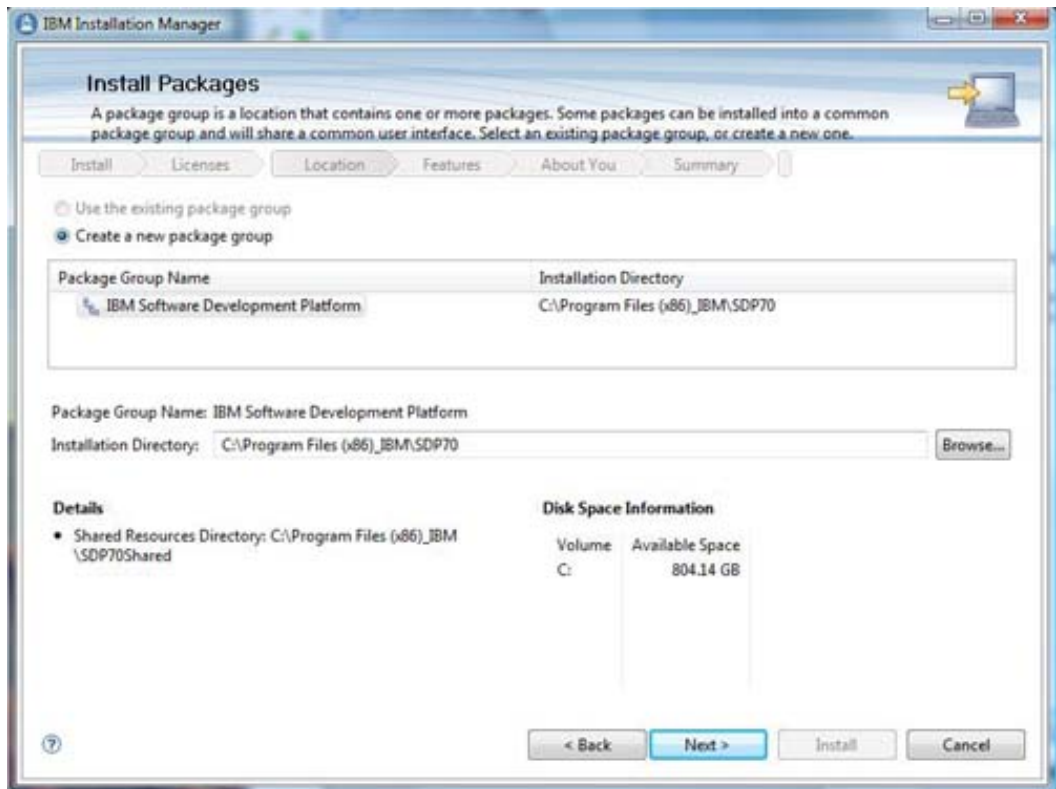
Figure 1. Installation packages



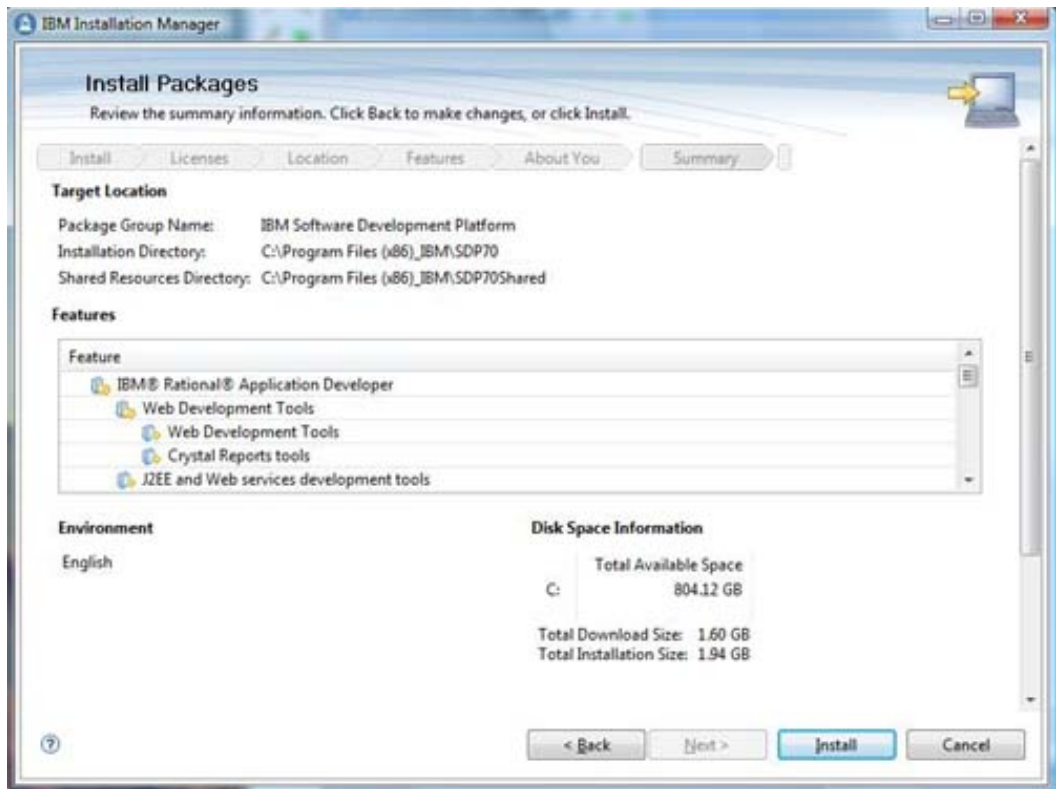
2. If 7.0.0.6 of IBM Rational Application Developer is not what appears, check **Show all versions** to force it to show. Check Version 7.0.0.6 and continue.
3. The "accept license terms" page opens. Select **I accept the terms in the license agreements**, and click **Next**.
4. The next screen asks you where to install the shared resources directory. **Figure 2. Where to install shared resources**



5. The default, shown above, should be fine. Click **Next**. Specify where to create a new package group. Select **Create a new package group**.
Figure 3. Creating a new package group

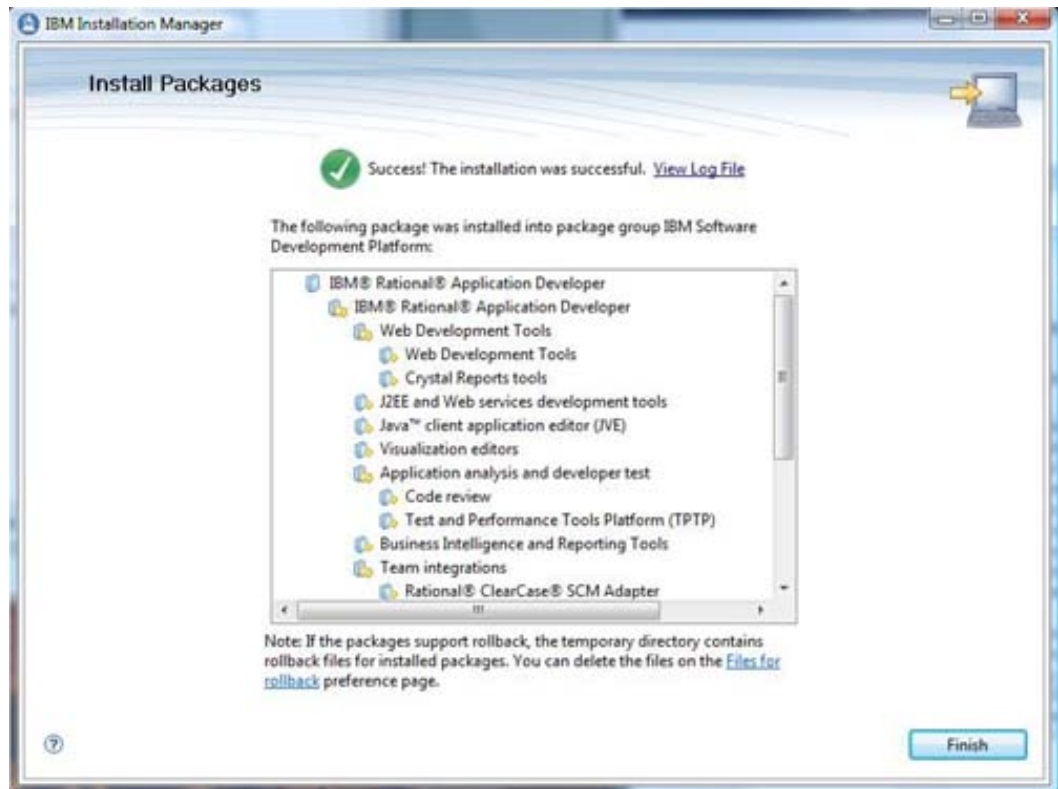


6. You'll install Data Studio into this package group later, so remember what it's named (IBM Software Development Platform is the name in Figure 4). The default directory will do, so click **Next**.
 7. The next screen asks you if you want to extend an existing version of Eclipse. You won't be doing that here, so make sure **Extend an existing Eclipse** is unchecked. Click **Next**.
 8. Select which Rational Application Developer features you'd like installed. Leave the defaults and click **Next**.
 9. Select which languages you'll be using and click **Next**.
 10. This brings you to a short three question survey. Choose an answer for all three, and click **Next**.
 11. Finally, you'll see an installation summary page.
- Figure 4. Rational Application Developer installation summary**



12. If everything looks good, click **Install**. This can take some time because everything needs to download (1.6GB in the case of Figure 4), and then unpackaged and installed. When installation completes you'll see a final window with a summary of what was installed, shown in Figure 5.

Figure 5. Rational Application Developer installation successful



Great! You now have an installed trial version of Rational Application Developer. Next, you'll install Data Studio! For more information on Rational Application Developer see the [Resources](#) section.

Install Data Studio to your existing package group

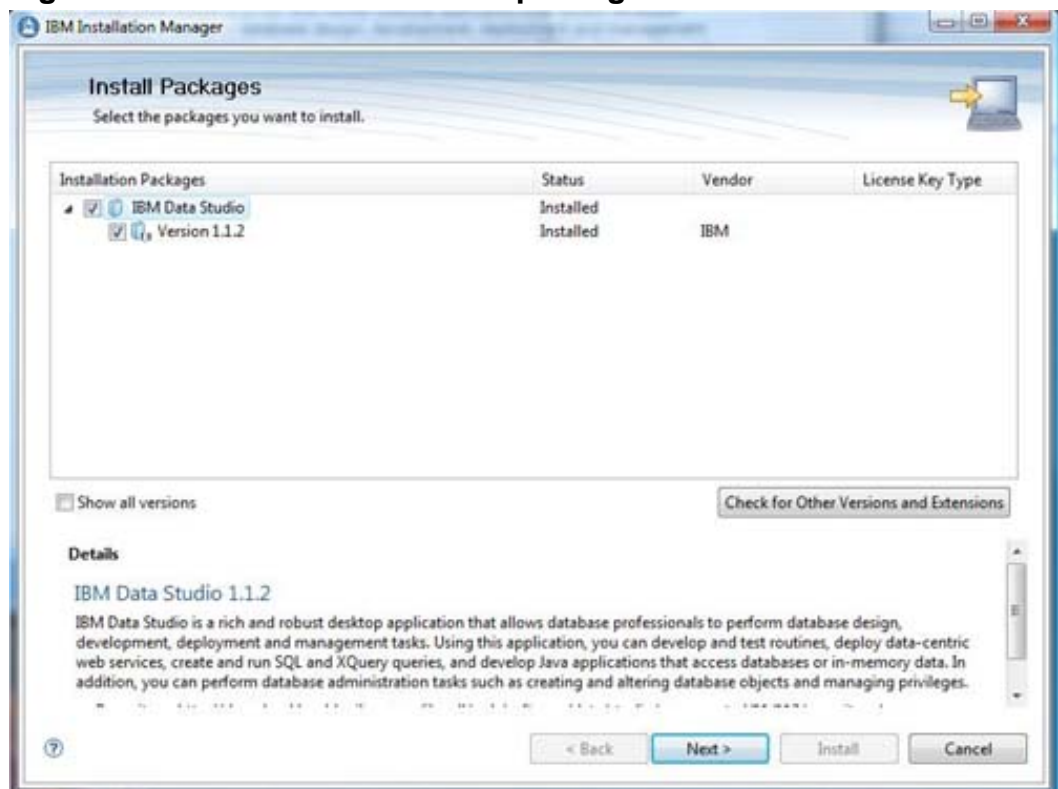
Now that you've downloaded Data Studio, there are a few steps to go through to install the package.

1. Double-click the setup.exe file. The setup page is displayed.
Figure 6. Data Studio setup page



2. Click **Install IBM Data Studio**. The Installation Packages page is displayed.

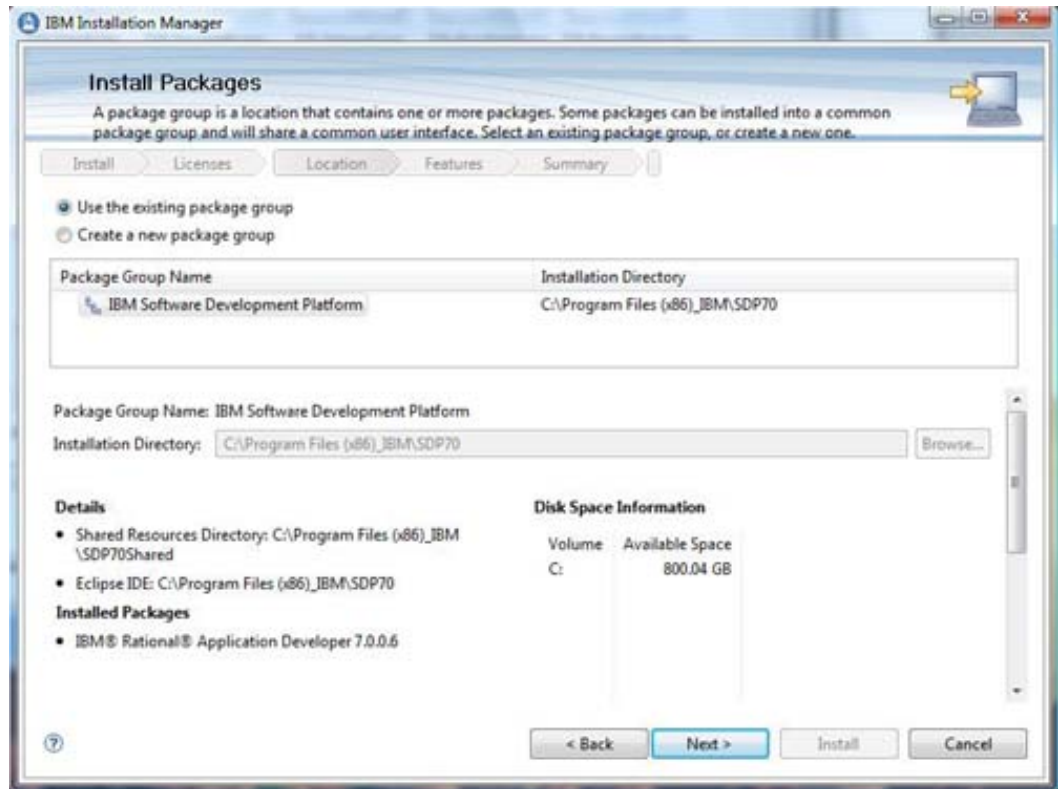
Figure 7. Data Studio installation packages



3. Make sure Version 1.1.2 is selected; if it's not listed, click **Check for Other Versions and Extensions**.

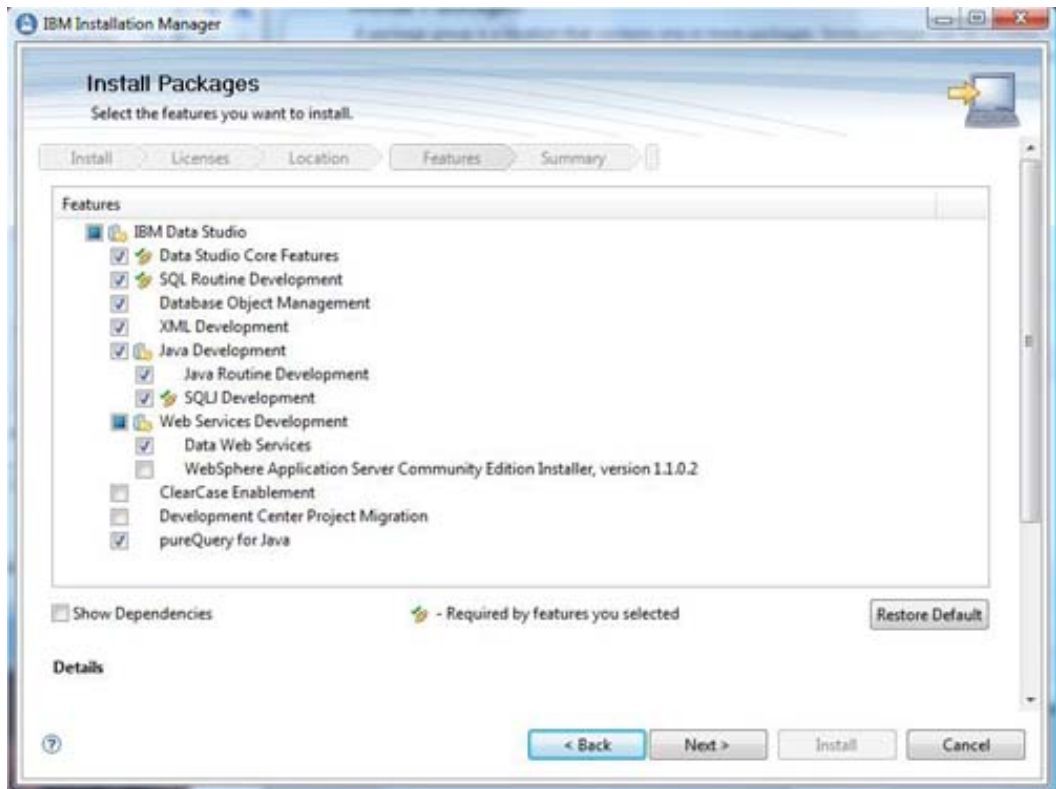
4. If Step 3 doesn't work, check **Show all versions**. Make sure version 1.1.2 is checked and click **Next**.
5. Select **Use the existing package group**, and make sure the same package group you installed Rational Application Developer into is selected, shown in Figure 8.

Figure 8. Using an existing package group



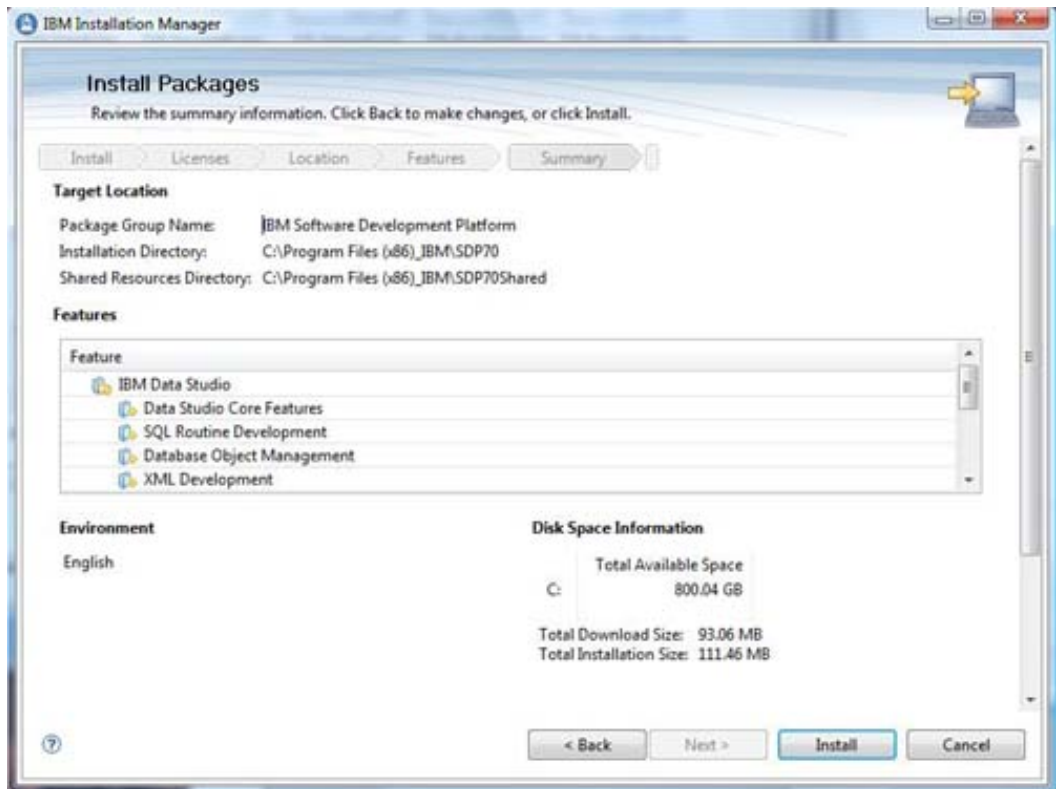
6. Click **Next**. You'll be asked which packages to include, as shown in Figure 9.

Figure 9. Features



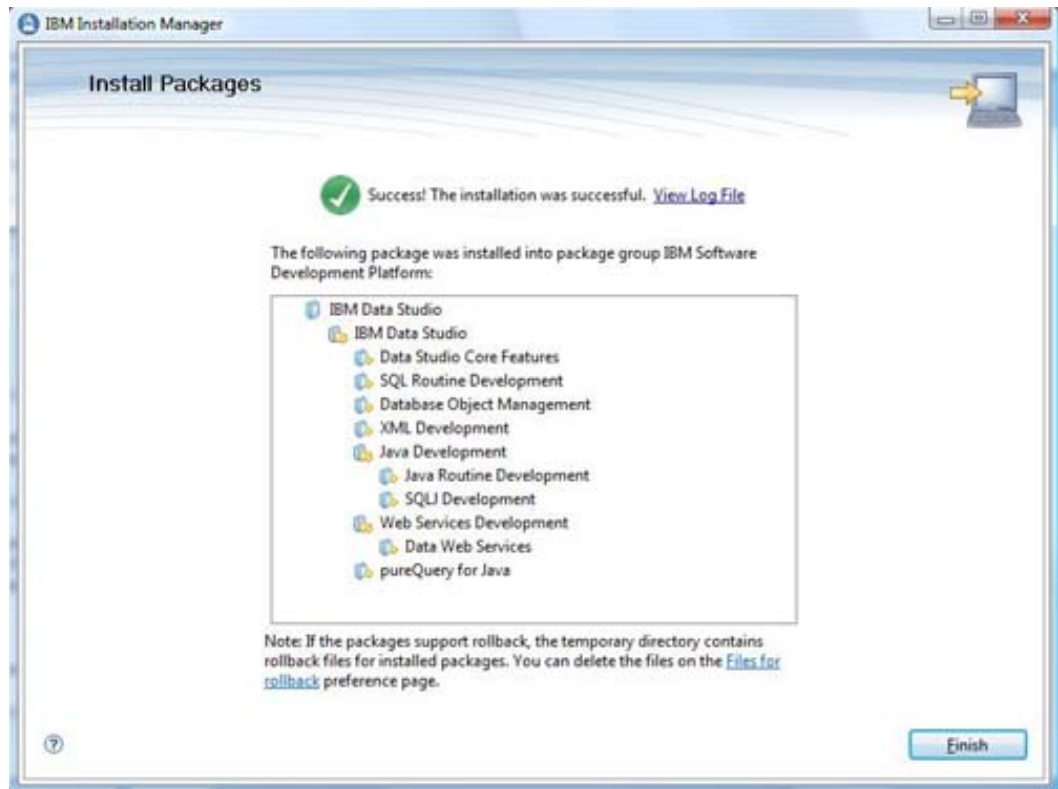
7. Leave the default and click **Next**. The installation summary page is displayed.

Figure 10. Data Studio installation summary page



8. Click **Next**. Installation should be much quicker than the Rational Application Developer installation. When installation is complete a final window shows you a summary of the installation.

Figure 11. Data Studio installation successful



There you have it -- a quick and simple install. Next you'll begin using your new Rational Application Developer and Data Studio duo! For more information on Data Studio, see the [Resources](#) section.

Section 3. Create a pureQuery enabled dynamic Web project

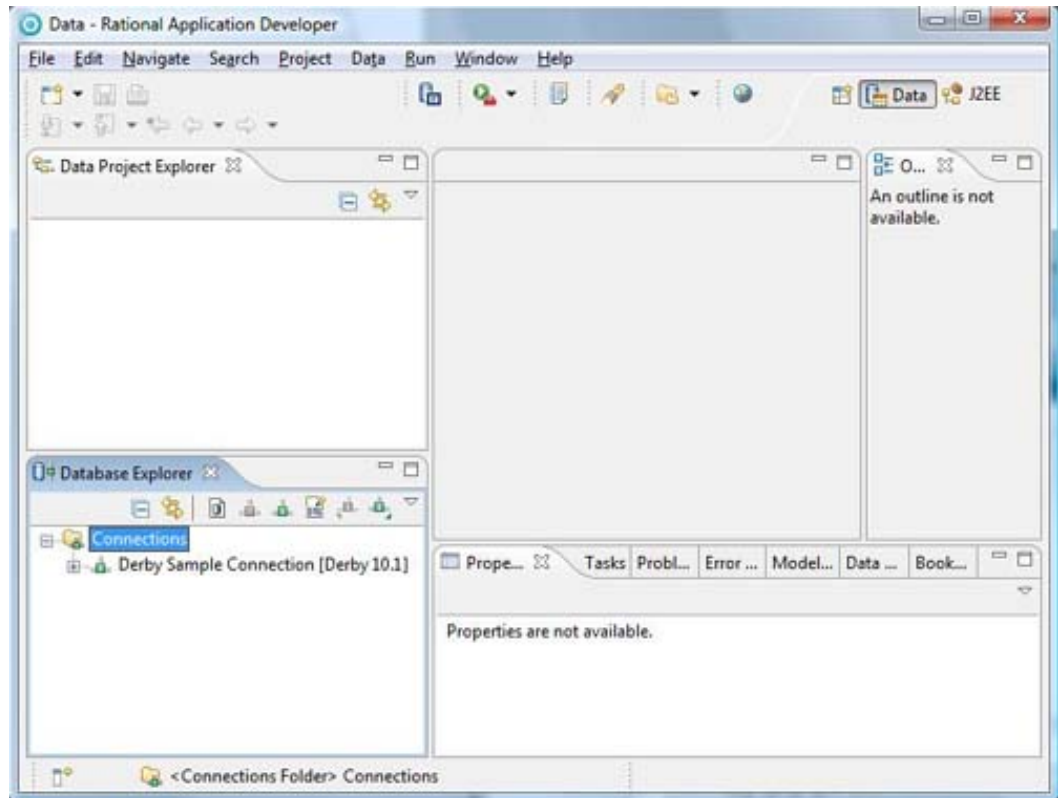
Now that you've got a working Rational Application Developer and Data Studio installation, you're ready to charge on! In this section you'll create a connection to your database, and create a new pureQuery enabled dynamic Web project.

Create a database connection

This tutorial uses DB2 as the database, though any database could be used, including the default Derby Sample Connection. The first thing to do is open Rational Application Developer and go to the data perspective.

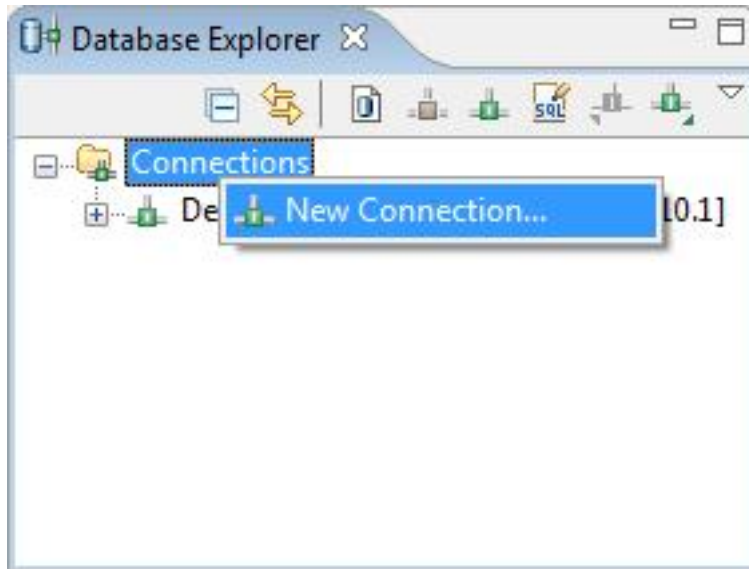
1. Click **Window > Show Perspective > Other** and select **Data** from the list. You should see the data perspective.

Figure 12. The data perspective



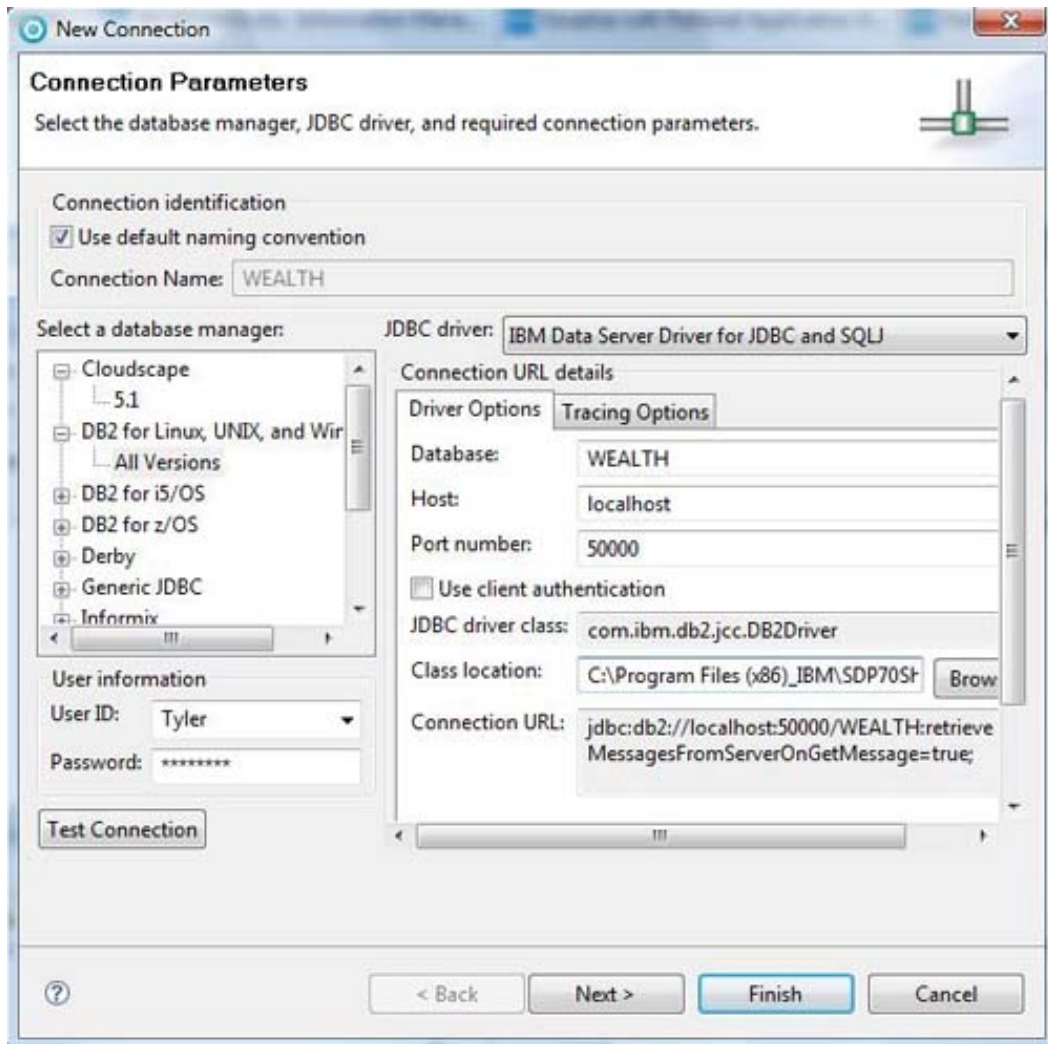
2. Notice the Data Explorer view in the bottom-left corner. To create a new database connection, right-click the Connections folder and click **New Connection**.

Figure 13. Creating a new connection



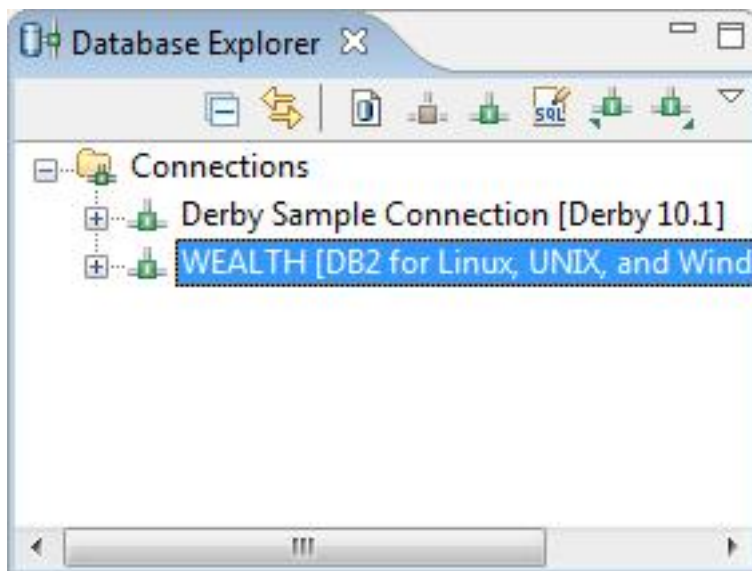
3. The New Connection page is displayed. Specify the database type (DB2) and database name (WEALTH).

Figure 14. Configuring a new database connection



4. Feel free to test the connection prior to finishing. Click **Finish** after everything looks good.
5. The new database connection appears in the Database Explorer.

Figure 15. New connection to the wealth database



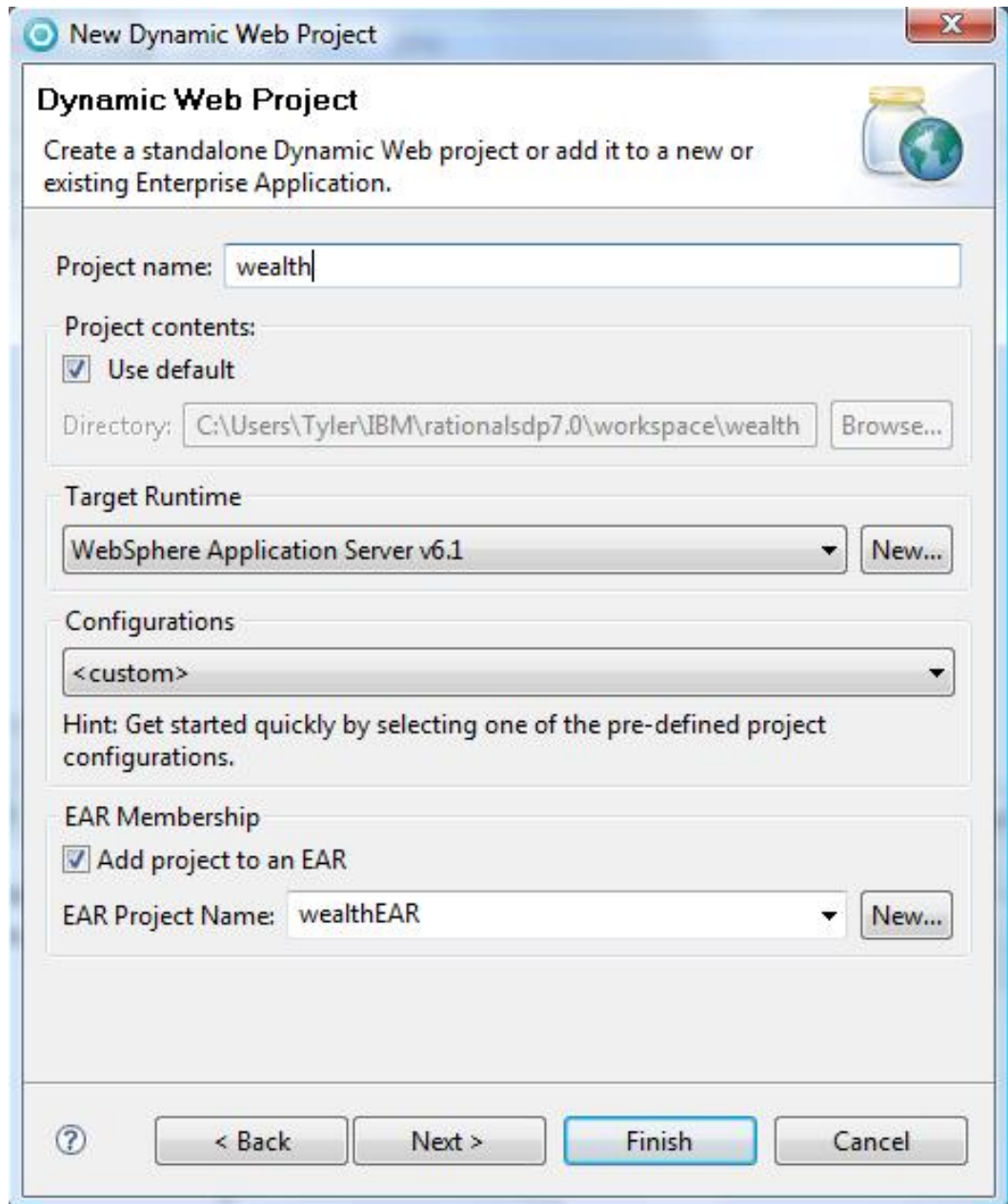
You're now set to create a new dynamic Web project.

Create a dynamic Web project

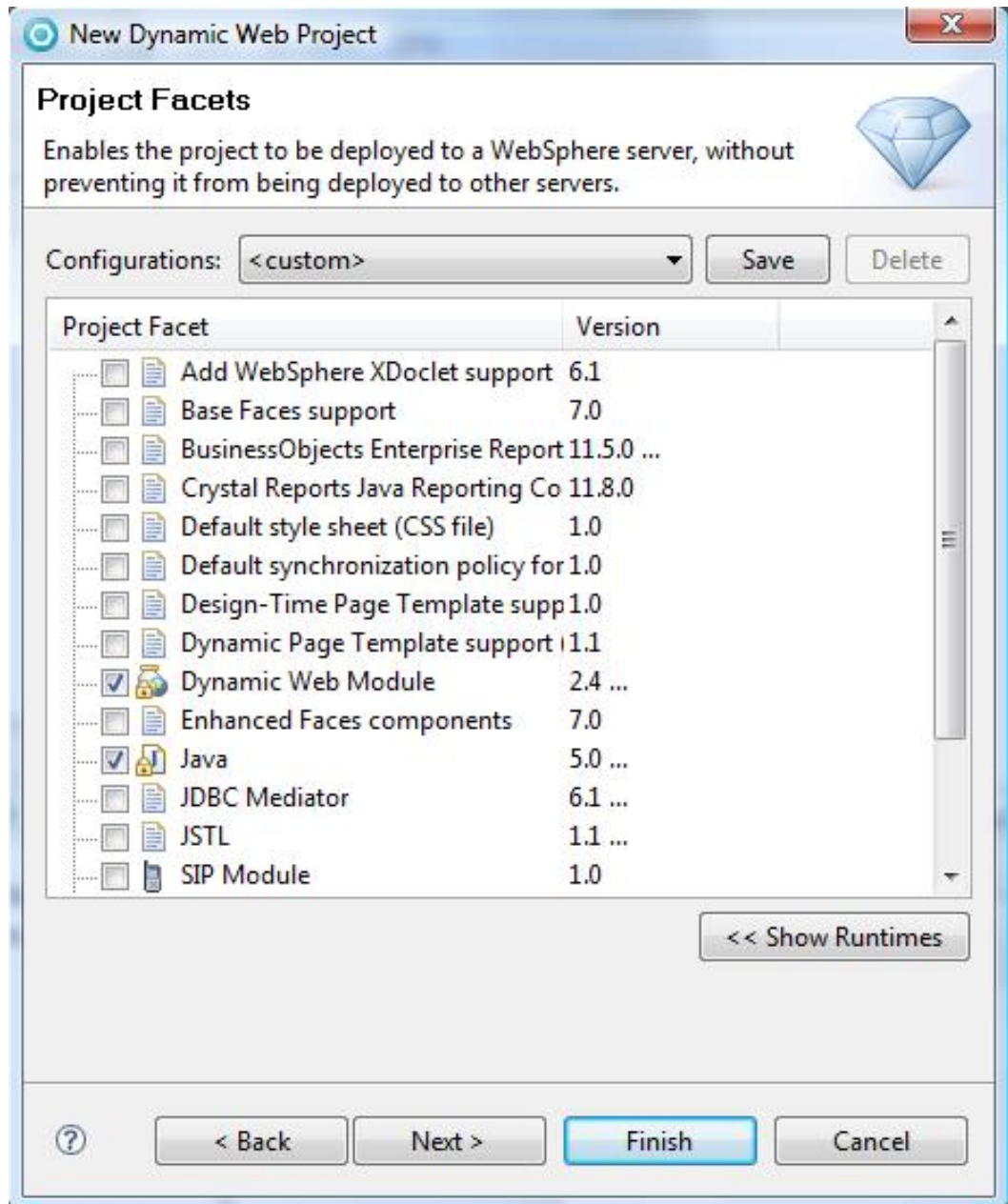
With the connection you're going to use in place, create a dynamic Web project.

1. Click **File > New > Project**, and select **Dynamic Web Project**. The New Dynamic Web Project window opens.

Figure 16. Creating a new dynamic Web project

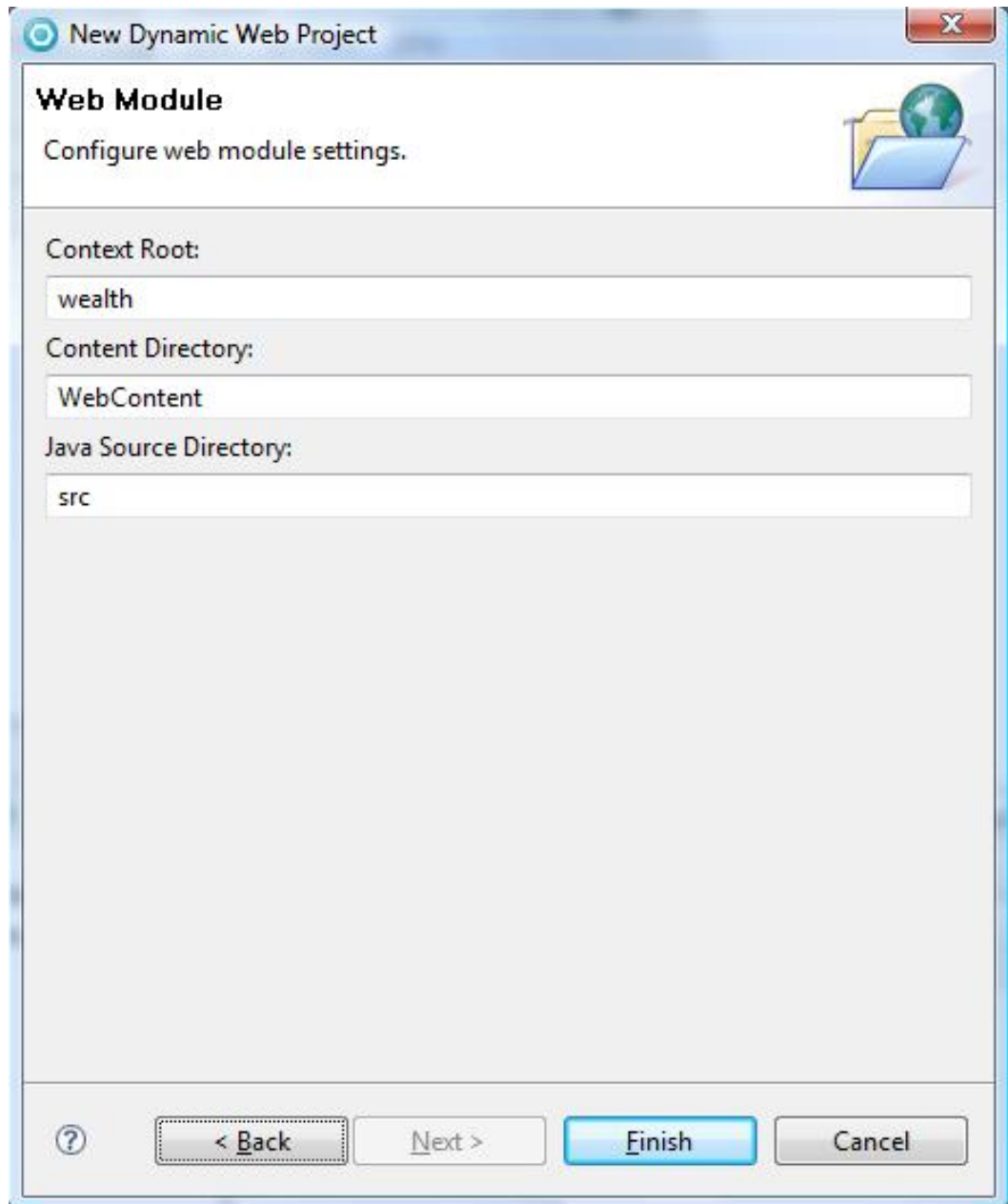


2. Name it `wealth`, choose **WebSphere Application Server V6.1** as your target runtime, a **custom** configuration, and leave **Add project to an EAR** checked. Click **Next**.
3. Configure the Project Facets included in your Web application.
Figure 17. Project Facets



Click **Next**.

4. Define the context root of your Web application, shown in Figure 18.
Figure 18. Web module configuration



5. Click **Finish** to create the dynamic Web project.

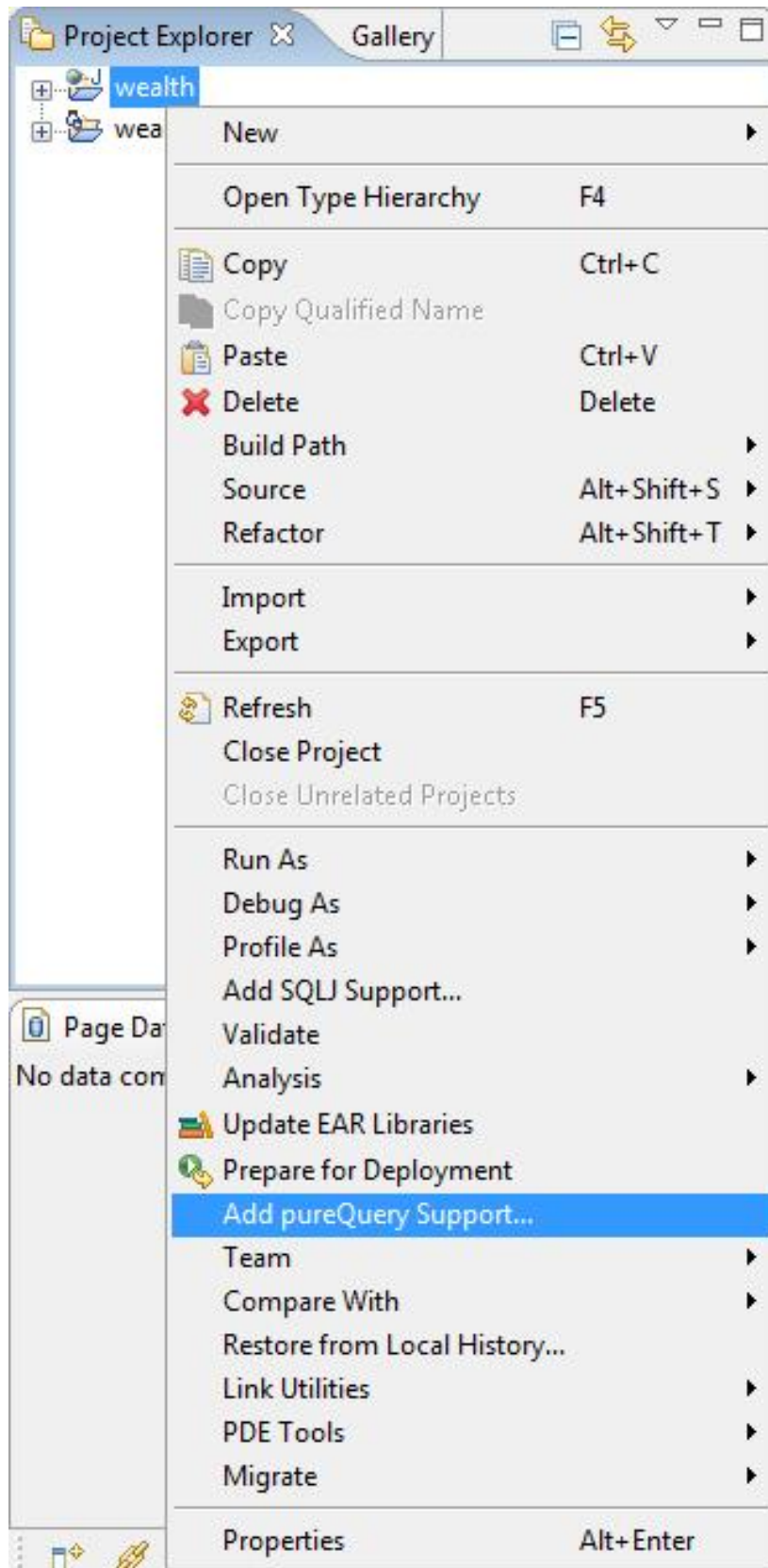
That does it for creating your new project! Next you'll add pureQuery support to it.

Add pureQuery support

Adding pureQuery support allows you to use pureQuery features later in this tutorial (see [Resources](#) for some articles on pureQuery).

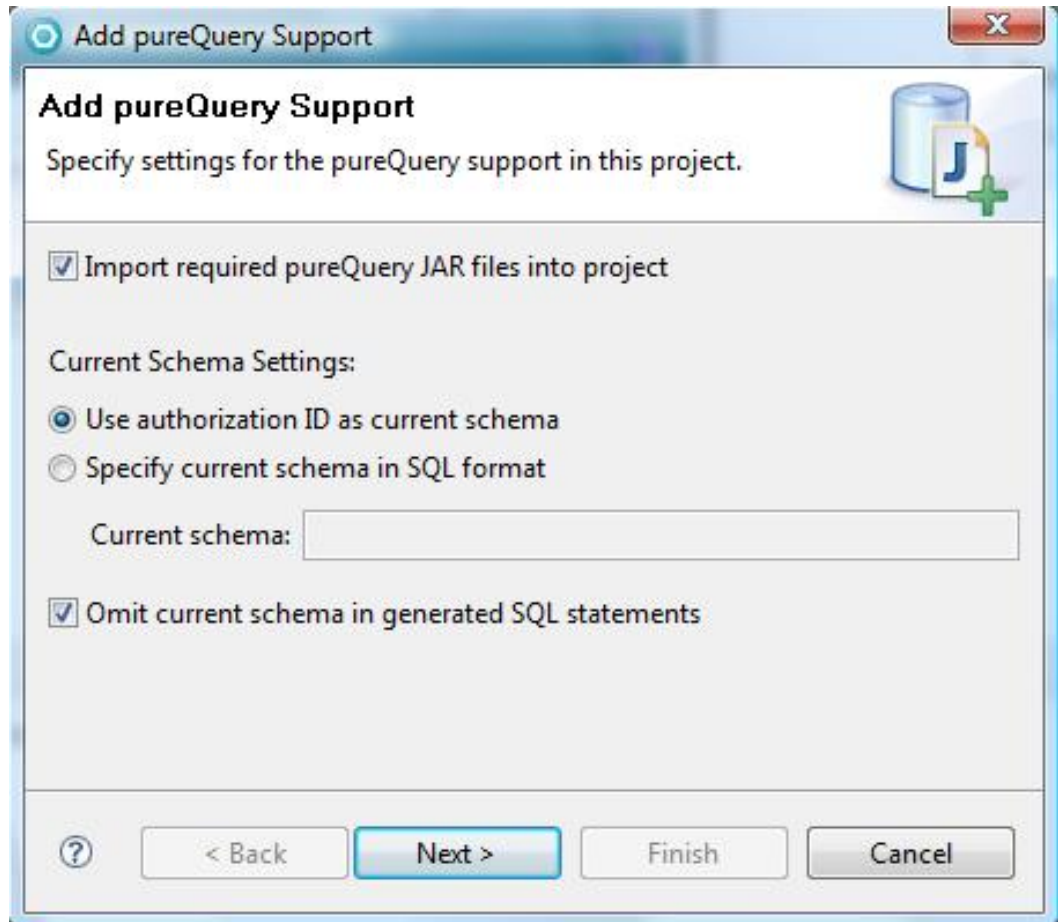
1. Right-click the wealth project in the Project Explorer view, and select **Add pureQuery Support**.

Figure 19. Add pureQuery support



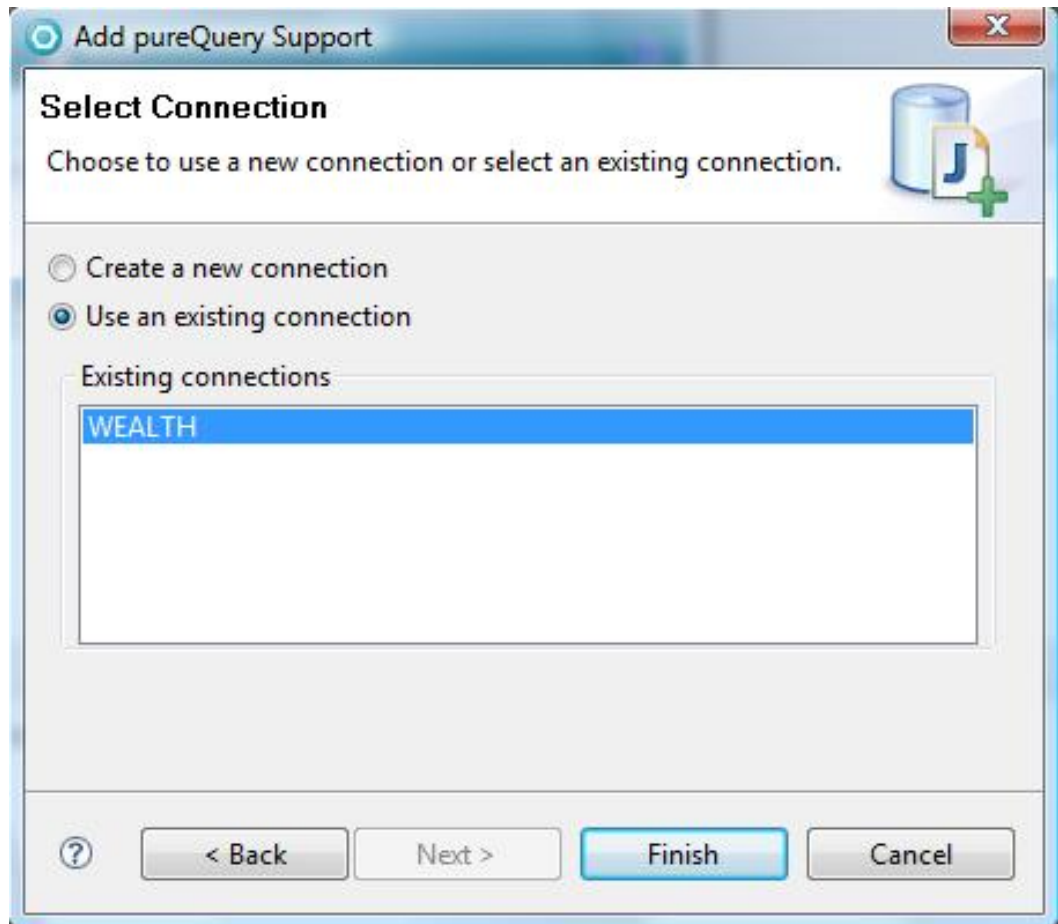
2. Select all the options shown in Figure 20 and click **Next**.

Figure 20. Configuring pureQuery support



3. Choose which database connection you want to include pureQuery support for (see Figure 21).

Figure 21. Selecting a database connection



4. Select **Use an existing connect** and choose the WEALTH connection.

Next you'll do some JAR shuffling to prepare your project for prime time.

Prepare your project for action

The pureQuery enabling of your project has added some jars to your project, however, you need to shuffle them in order for things to work smoothly.

To shuffle the jars:

1. Go to the directory of your wealth project on the file system (C:\Users\Tyler\IBM\rational\sd7.0\workspace\wealth on my machine).
2. Move pdq.jar to WebContent\WEB-INF\lib\.
3. Copy db2cc_license_cisuz.jar and db2jcc.jar from c:\Program Files

(x86)_IBM\SDP70Shared\plugins\com.ibm.datatools.db2_1.0.200.V200803071234\drive
(within the shared installation directory of the package you created during
the Rational Application Developer installation) to
WebContent\WEB-INF\lib

4. Go back to Rational Application Developer and right-click the WebContent\WEB-INF\lib directory in the Project Explorer window and click **Refresh**. These three class files should appear under the Libraries/Web App Libraries icon under the "Java Resources: src" icon.
5. Right-click db2jcc_license_cisuz.jar and db2jcc.jar and select **Build Path > Remove from Build Path**.

What you've essentially done is added the DB2 connection drivers and pureQuery runtime JARS into the EAR file that ultimately deploys to WebSphere Application Server. Without doing this your application on WebSphere Application Server won't know where to get the classes contained within these JARS and will fail to execute.

Excellent! Your dynamic Web project is all ready for you to begin development. First you'll begin by creating pureQuery code from the database tables.

Section 4. Generate pureQuery code from your database tables

With your project ready for action, you're going to generate pureQuery code from database tables. But before you can do that, you're going to learn the schema of the database and add a few sample rows.

See the [downloads](#) section for the sample tables and the rest of the source code to follow along.

The database schema

Whichever database you choose to use, you need to create tables and insert some test data for starters. Within the WEALTH database create three tables:

USERS

Contains the userid and password of authorized users

SECURITIES

Contains stock and options holdings for each user

REALESTATE

Contains the real estate holdings for each user

As for the schema of each, here's the schema for the USERS table:

- `userid varchar(50) NOT NULL PRIMARY KEY`
- `password varchar(50) NOT NULL`

Listing 1 shows some sample data for the USERS table.

Listing 1. Sample data for the USERS table

```
"tyler","tyler"  
"hilbilly","pass"
```

Here is the schema for The SECURITIES table:

- `autoid integer NOT NULL PRIMARY KEY AUTO_INCREMENT`
- `userid varchar(50) NOT NULL FOREIGN KEY on users.userid`
- `symbol varchar(20) NOT NULL`
- `quantity integer NOT NULL`
- `securitytype decimal (5,2) NOT NULL`

Listing 2 shows some sample data for the SECURITIES table.

Listing 2. Sample data for the SECURITIES table

```
"tyler","IBM",100,"stock"  
"tyler","IBMDT.X",5,"option"  
"hilbilly","XOM",200,"stock"  
"hilbilly","XOMDN.X",2,"option"
```

Note that the symbols must be valid, as they'll be used to query the Yahoo! Finance API and grab the delayed quotes for each.

And here is the schema for the REALESTATE table:

- `autoid integer NOT NULL PRIMARY KEY AUTO_INCREMENT`
- `userid varchar(50) NOT NULL FOREIGN KEY on users.userid`

- `propertyvalue decimal(8,2) NOT NULL`
- `address varchar(255) NOT NULL`
- `city varchar(100) NOT NULL`
- `zip varchar(10) NOT NULL`
- `state varchar(100) NOT NULL`

Listing 3 shows some sample data for the REALESTATE table.

Listing 3. Sample data for the REALESTATE table

```
"tyler",+199999.99,"100  
Roy st.", "Happy  
Town", "12345", "CO"  
"hilbilly",+049999.99,"100  
Depreciation  
ln.", "Sad  
Town", "12345", "AZ"
```

For now you'll create a form in the application that allows you to add more data manually via a form.

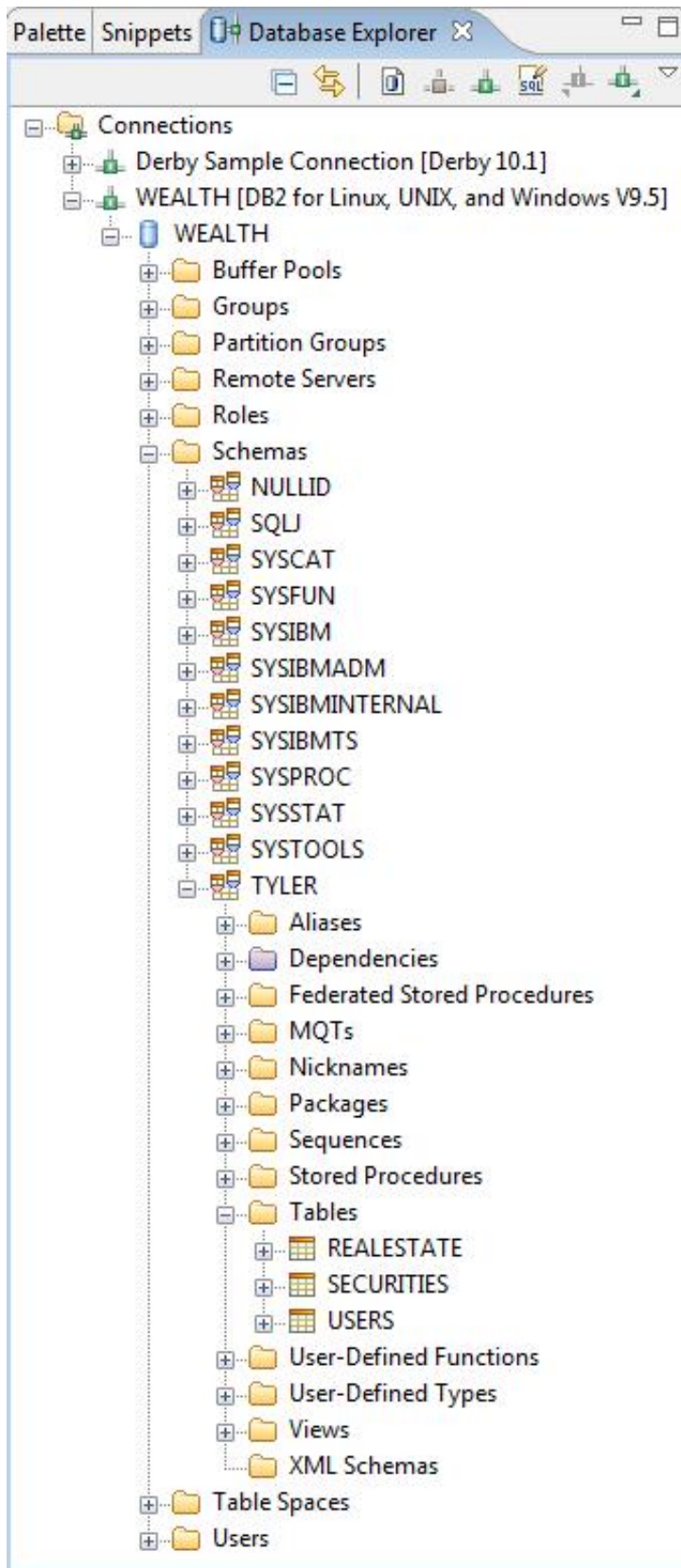
That's all you do here. Next, create the pureQuery classes!

Generate pureQuery code: USERS table

Now you're ready to create the pureQuery classes to help with your Web application development.

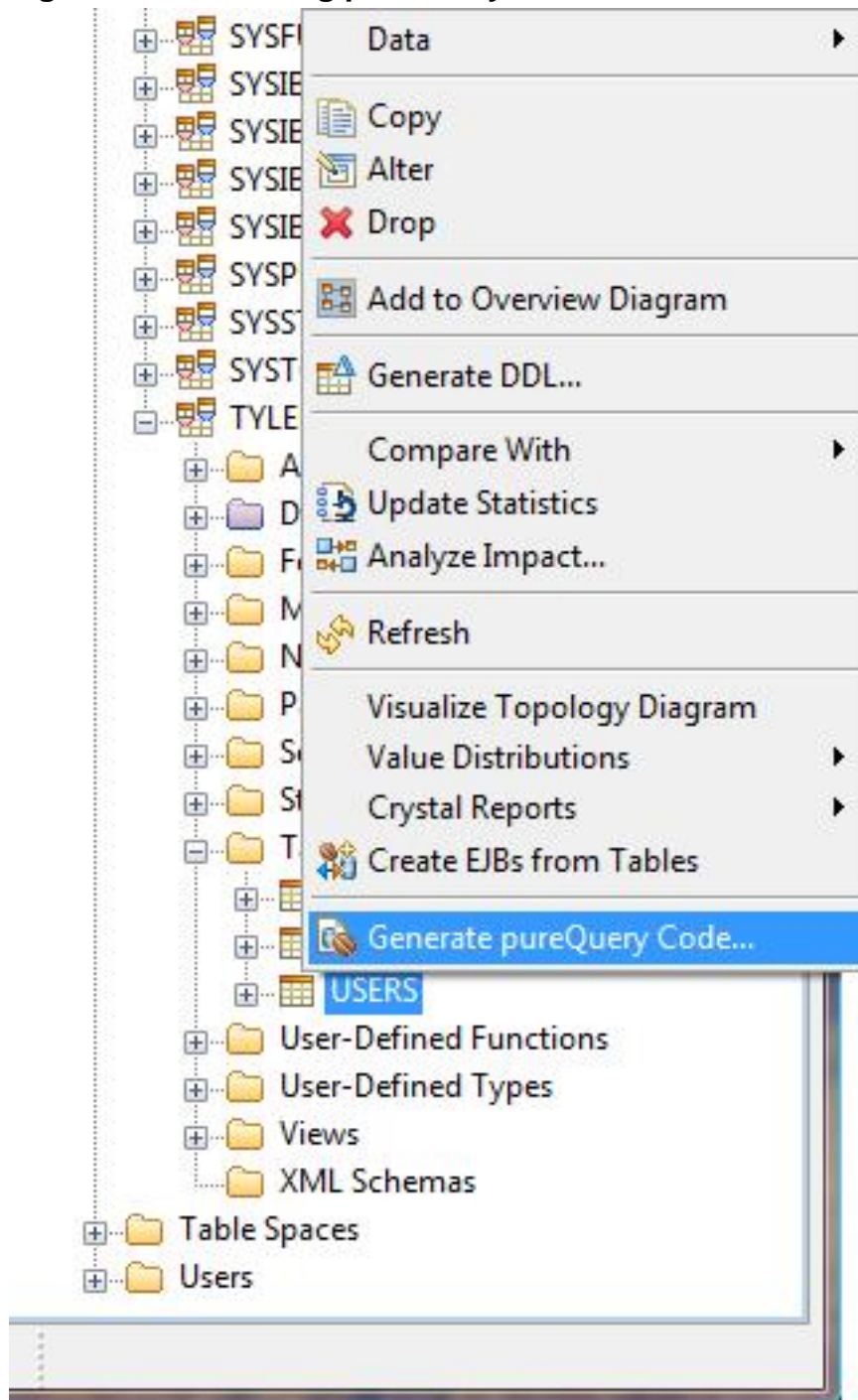
1. Scroll to the place in the Database Explorer window where the database tables are.

Figure 22. Viewing the database tables via the Database Explorer view



- Here you see the three tables. To generate pureQuery code, right-click the **USERS** table and select **Generate pureQuery Code**.

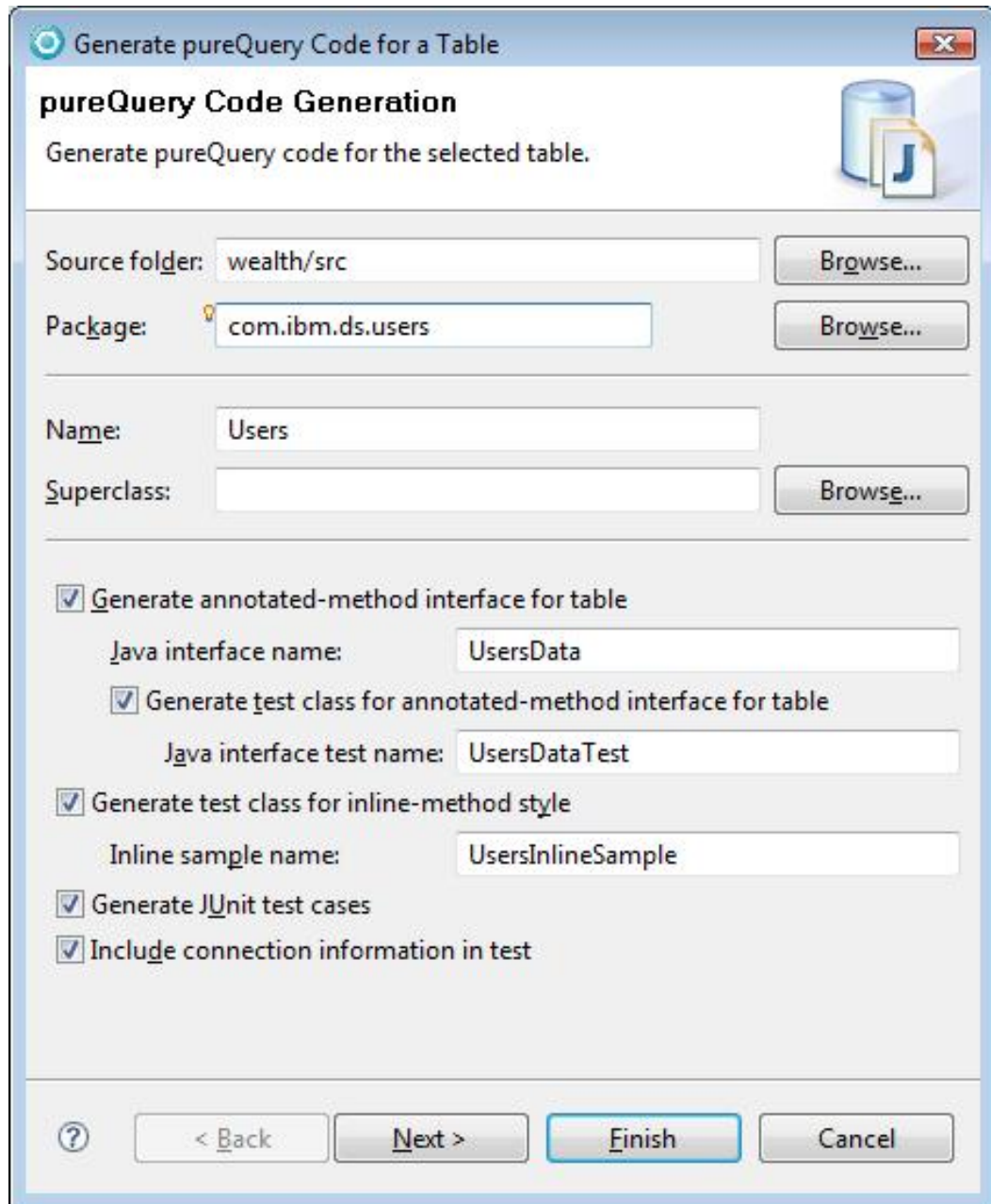
Figure 23. Generating pureQuery code



- To set the source folder, click **Browse** to navigate to the src folder of the

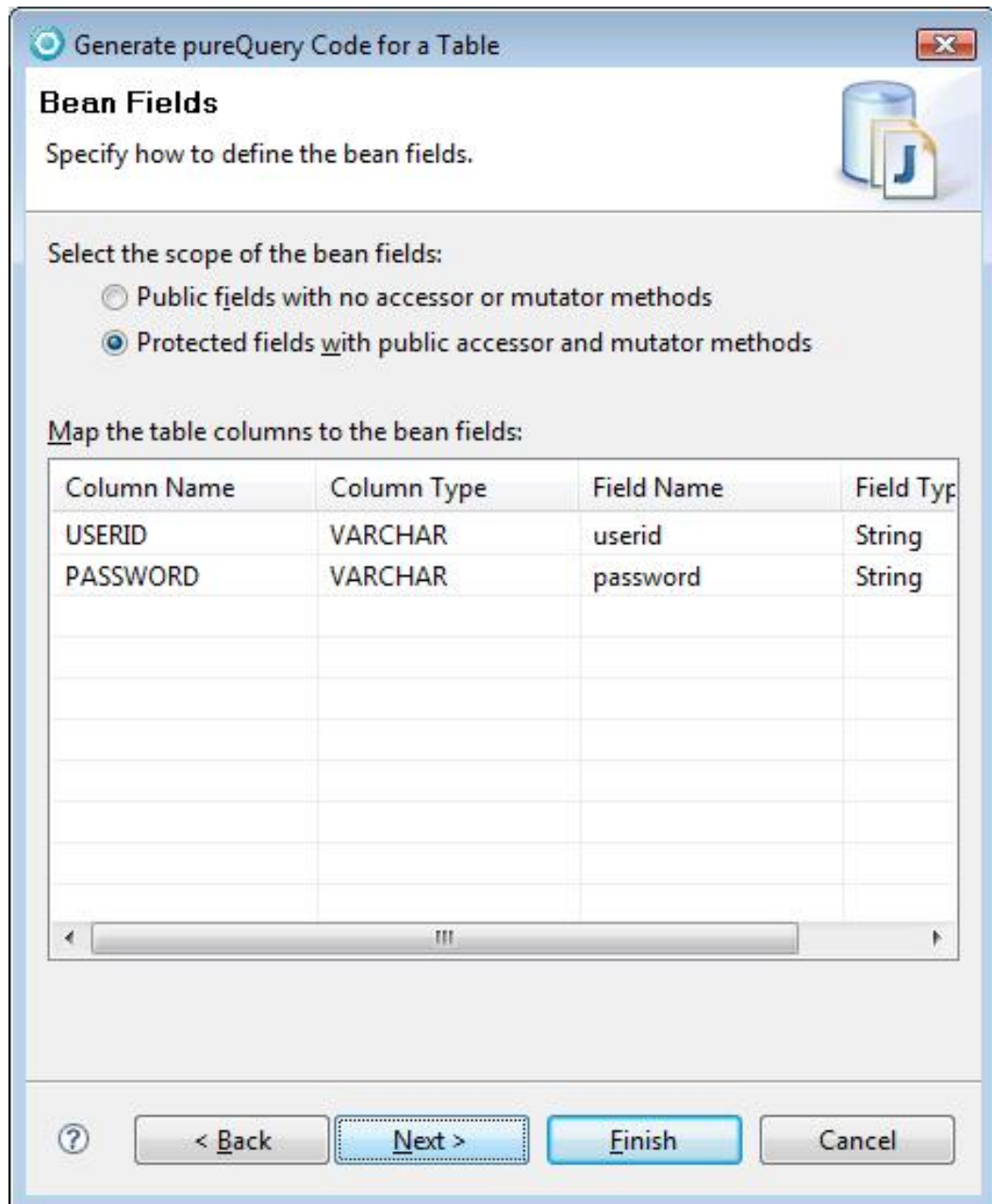
wealth project you created earlier. Name the package `com.ibm.ds.users`. Check all of the boxes for including test classes and connection information as shown in Figure 24.

Figure 24. Configuring the pureQuery code generation



4. Click **Next**. The Bean Fields window opens.

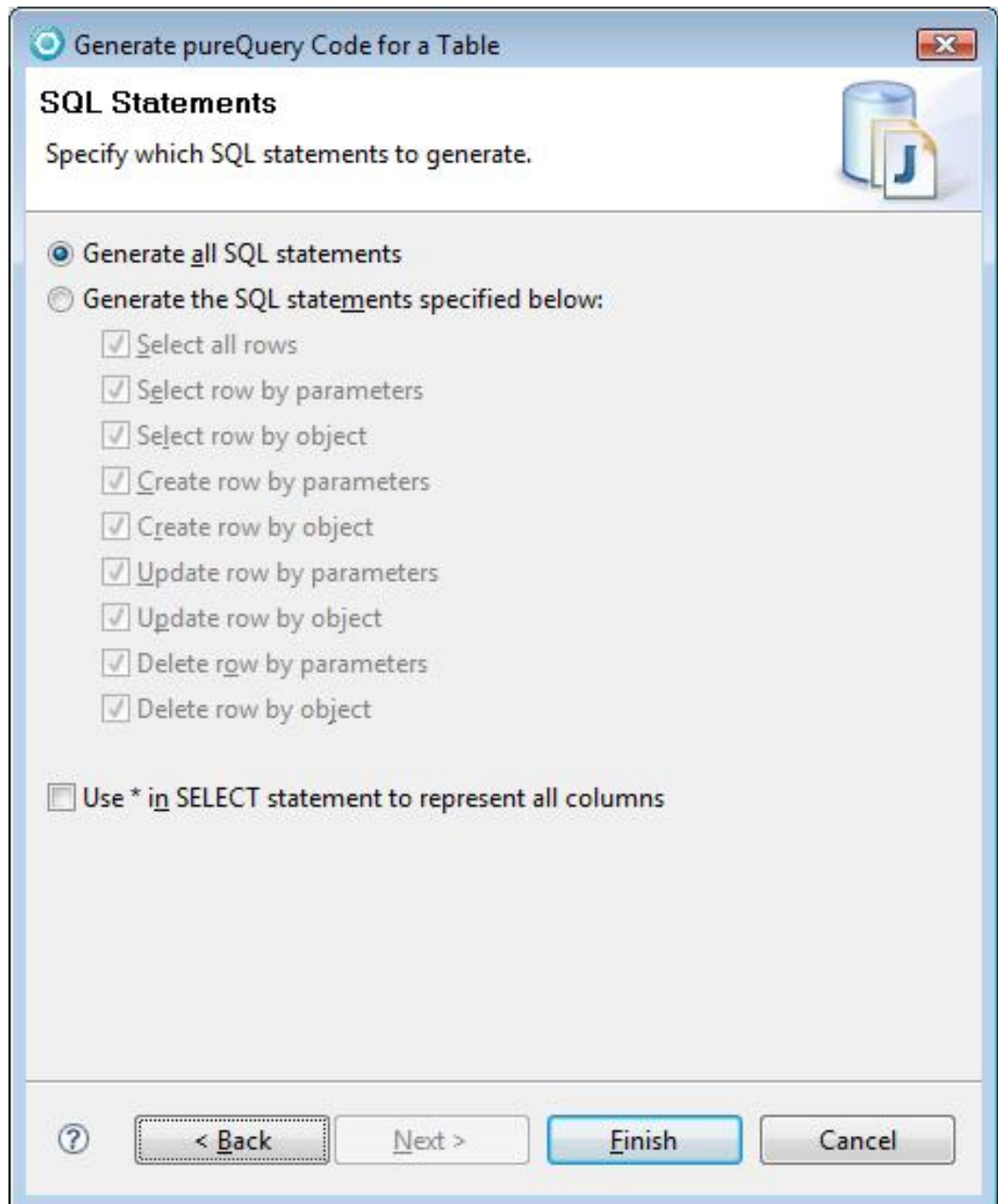
Figure 25. Bean fields



Notice that the same fields that are in the database table appear here. You can rename the Java name given to the variable by changing the values in the Field Name column.

5. Click **Next**. The final page is displayed, as shown in Figure 26.
6. Choose which preconfigured SQL statements to include in your bean by selecting **Generate all SQL statements** and click **Finish**. This should generate the new class files straight into your wealth project.

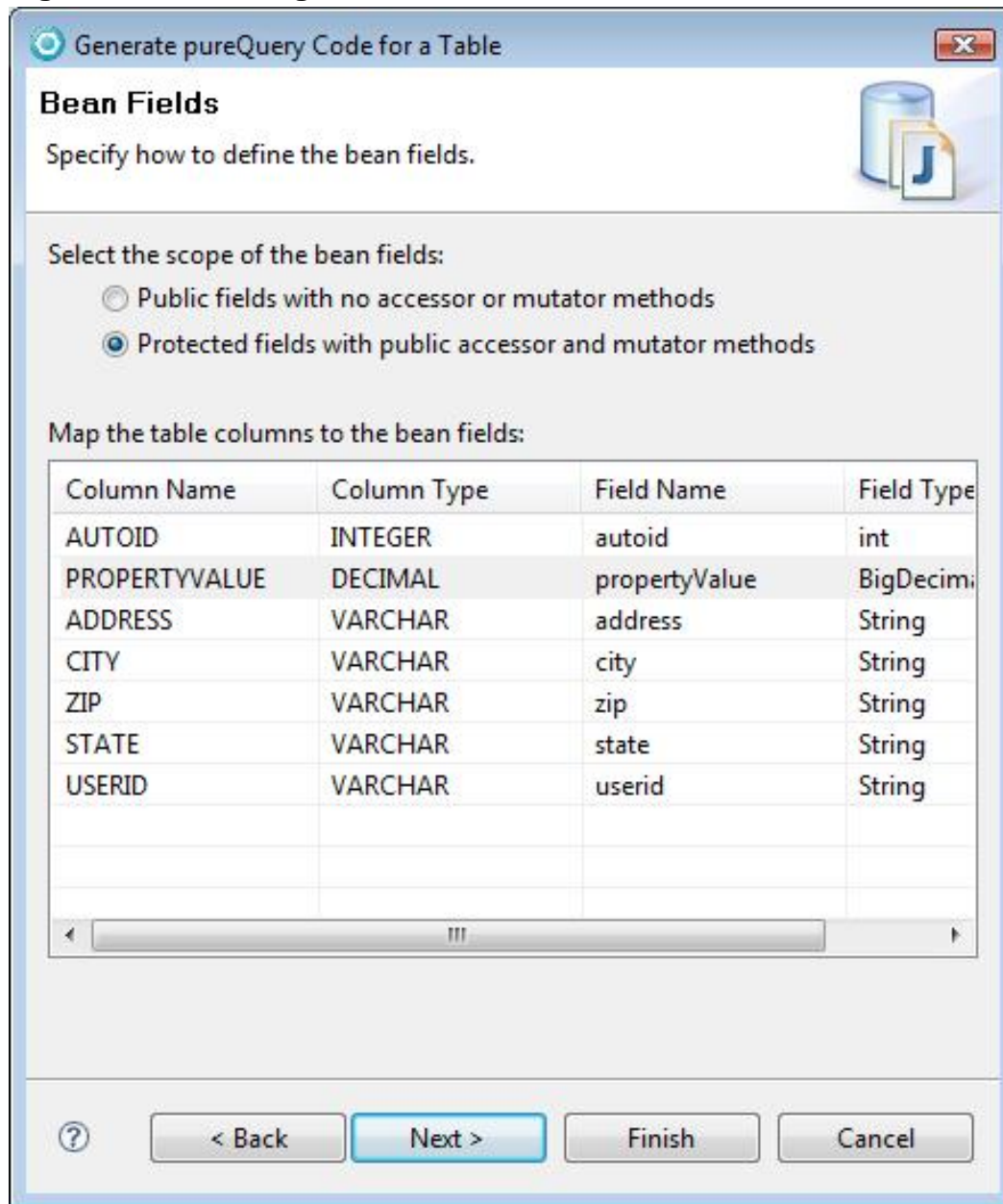
Figure 26. SQL statements



7. Go back to Step 1 and right-click the SECURITIES table. Follow steps 2-7 but be sure to name the package of the securities table `com.ibm.ds.securities`.
8. Go back to Step 1 and right-click the REALESTATE table. Follow Steps 2-7 but name the package of the REALESTATE table `com.ibm.ds.realestate`.
9. In the Bean Fields window, rename `propertyvalue` to

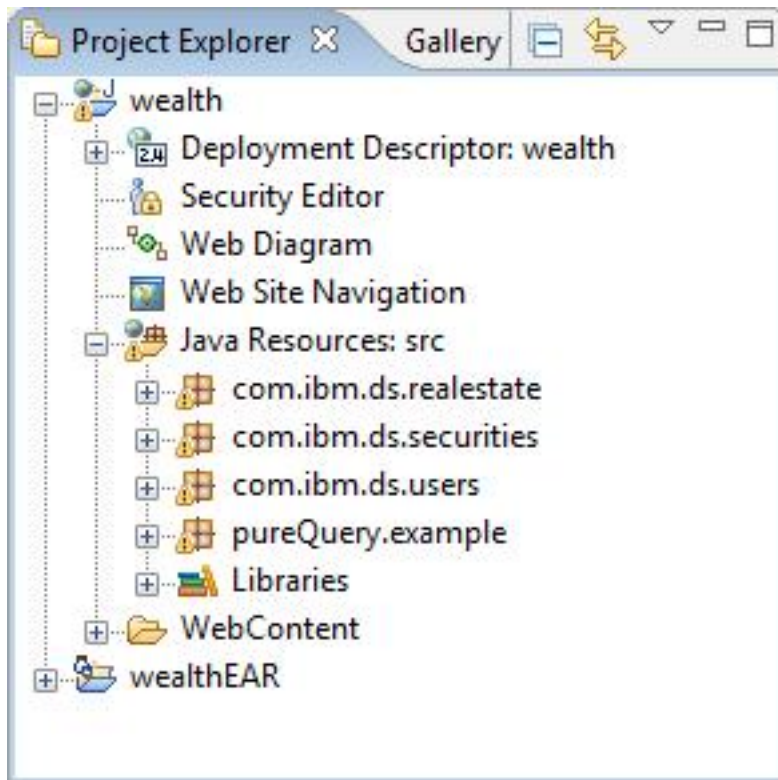
propertyValue, as shown in Figure 27. Note the renamed Field Name above. This feature gives you flexibility in the names given to the fields in each of your tables. The values in the Field Type column can also be modified, but the default works great in most cases.

Figure 27. Renaming a field in the Bean Fields window



10. With pureQuery code generated for all three tables, take a look at your project in the Project Explorer view, as shown in Figure 28. Note the four new packages under the Java Resources: src icon.

Figure 28. The Project Explorer view



Next you'll augment the generated classes with an additional SQL query.

Add a new query to the REALESTATE and SECURITIES objects

Since the REALESTATE and SECURITIES tables only have queries to grab the entire tables or to search only on the primary key, you need to create a new query to be able to, for example, grab all the REALESTATE records for a given userid.

1. Open the RealestateData.java interface class in the `com.ibm.ds.realestate` package, and add the following three lines from Listing 4.

Listing 4. New query to grab all REALESTATE records for a given userid

```
@Select(sql="select
AUTOID,
PROPERTYVALUE,
ADDRESS, CITY, "+
" ZIP,
\"STATE\", USERID
from REALESTATE
where USERID =
```

```
?")
Iterator<Realestate>
getRealestates(String
userid);
```

2. Once you save the file, the other files dependent on this interface are automatically updated! The `RealestateDataImpl` class is now primed for you to add this new SQL query, as you'll see later when you create the corresponding JSP Web page.
3. Now do the same for the `SecuritiesData.java` file in the `com.ibm.ds.securities` package (see Listing 5).

Listing 5. Query to grab SECURITIES records

```
@Select(sql="select
AUTOID, USERID,
SYMBOL, "+
QUANTITY, "+
"
SECURITYTYPE from
SECURITIES where
USERID = ?")
Iterator<Securities>
getSecuritiess(String
userid);
```

The `SecuritiesDataImpl` class is now ready for you to use this new query.

Section 5. Create JSP to use the new pureQuery classes using Rational Application Developer

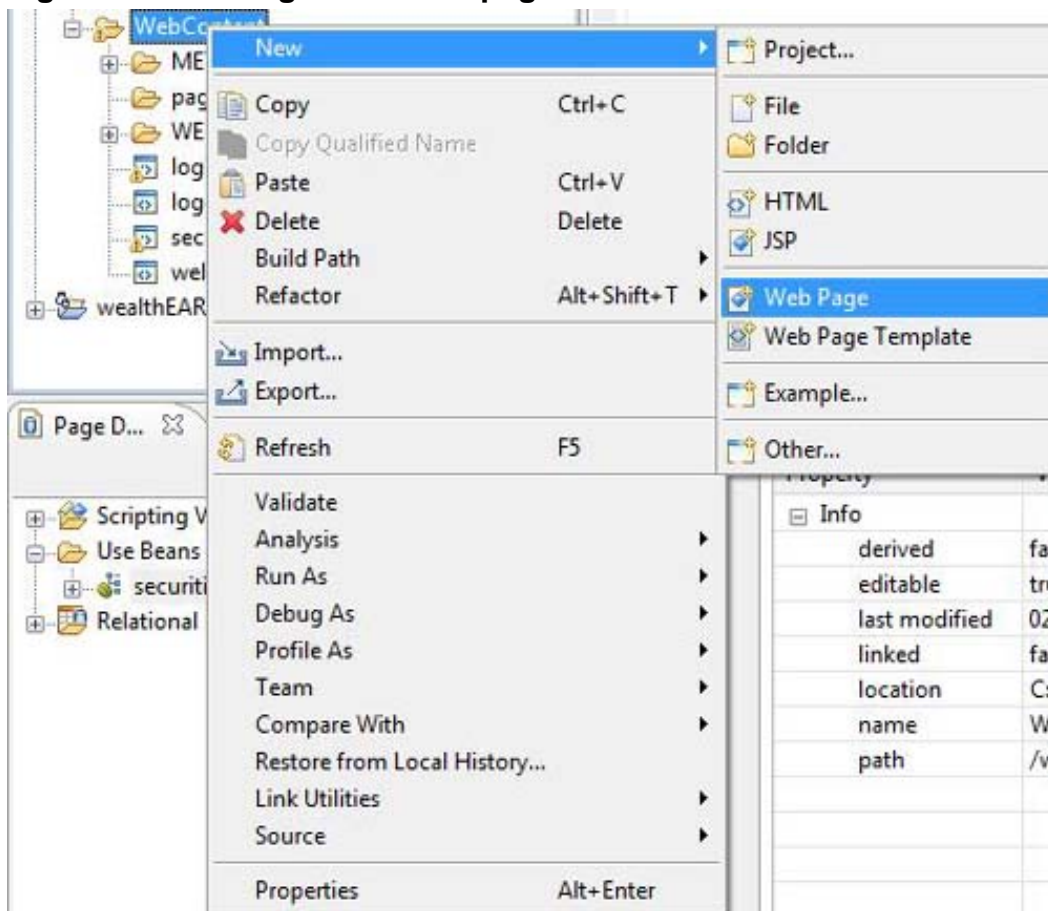
You now have all these classes to make development of your dynamic Web application more efficient in less time. In this section you'll create five new JSP files to comprise your Web application, followed by deployment and testing on WebSphere Application Server.

The login page

To create the login page to allow your users to login:

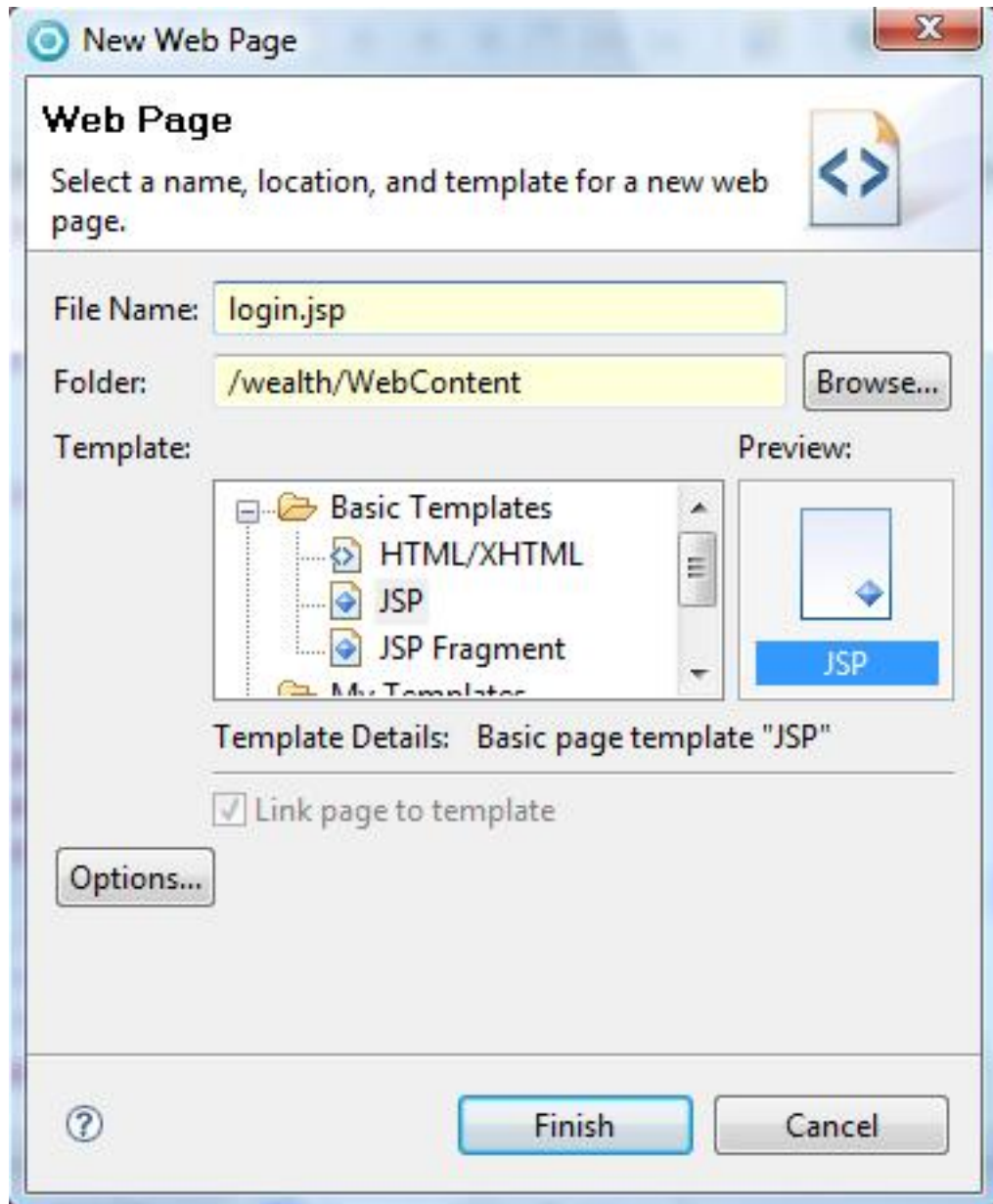
1. Left-click the `WebContent` folder and choose **New > Web Page**.

Figure 29. Creating a new Web page



2. The New Web Page window opens. Specify the details of your new Web page.

Figure 30. Configuring a new Web page



3. Name this file login.jsp.
4. Make sure the value of the folder field is /wealth/WebContent/.
5. Select **JSP** for Template.
6. Click **Finish** to create the file.
7. Go back and repeat Steps 1-6, but instead of creating login.jsp, create these four files: welcome.jsp, logout.jsp, securities.jsp, realestate.jsp.

You'll learn more about each of these JSP files and code them later in this section.

Great! Time to move on and begin coding some JSP!

Code the JSP files

Start with the login.jsp page and modify the template created for you, as shown in Listing 6.

Listing 6. Coding the login page

```
<!DOCTYPE HTML
PUBLIC
"-//W3C//DTD HTML
4.01
Transitional//EN">
<jsp:useBean
id="users"
class="com.ibm.ds.users.UsersDataImpl"
scope="page" />
<%@page
language="java"
contentType="text/html;
charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@page
import="com.ibm.ds.users.Users"
%>
<%@page
import="pureQuery.example.*"
%>
<%@page
import="com.ibm.pdq.runtime.*"
%>
<html>
<head>
<title>login</title>
<meta
http-equiv="Content-Type"
content="text/html;
charset=ISO-8859-1">
<meta
name="GENERATOR"
content="Rational
Application
Developer">
</head>
<body>
<h1>Login</h1>
<!-- database
connection will
be made using
URL:
jdbc:db2://localhost:50000/WEALTH
on users
table. Success
takes the user to
the welcome page.
-->
<%
```

```

        String userid
    =
    request.getParameter("userid");
        String
    password =
    request.getParameter("password");

        if((userid !=
    null &&
    !userid.equals(" "))
    &&
        (password
    != null &&
    !password.equals(" "))) {
            Data d =
    SampleUtil.getData("jdbc:db2://localhost:50000/WEALTH",
    "<enter_your_username>",
    "<enter_your_password>");
            users.setData(d);
            Users u =
    users.getUsers(userid);

            if(u !=
    null &&
    u.getPassword().equals(password)) {
                session.setAttribute("userid",
    userid);
                response.sendRedirect("welcome.jsp");
            }
            else {
                out.println("Error
    logging
    in.<br/><br/>");
            }
            else
            if(request.getParameter("login")
    != null)
                out.println("You
    must enter a
    userid and
    password.<br/><br/>");
            %>
            <form
    method="post">
                userid:
            <input
    name="userid"/><br/>
                Password:
            <input
    name="password"
    type="password"/><br/>
                <input
    type="submit"
    name="login"
    value="Login"/>
            </form>
            </body>
            </html>

```

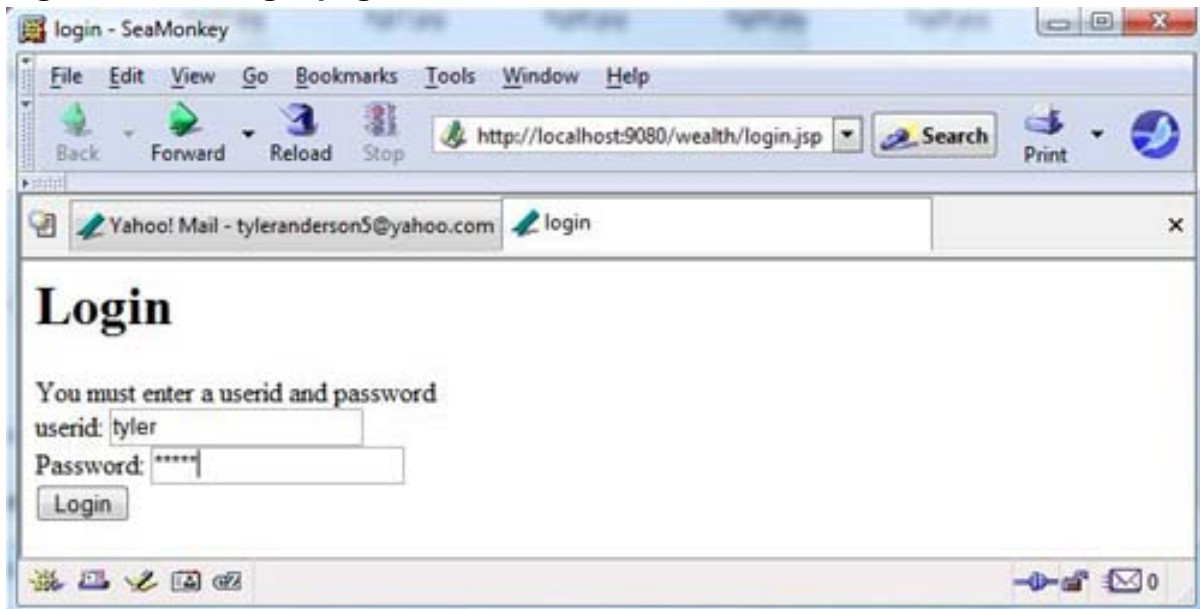
Notice the `<jsp:useBean tag />` on the second line. Here you've defined the `UserDataImpl` class for use by your JSP. Notice how it's used in the JSP code within the `<% %>` tags. Next of importance are the three import statements, followed by everything within the `<body>...</body>` tags.

First look at the form created at the bottom where you have input fields for the userid and password, with a Login button. When the form's submitted, you grab the fields and check their validity. Then you connect to the database using the pureQuery utility class that was created (`pureQuery.example.SampleUtil`). Then you set the data value (contains the connection to the database) in the `users` variable (bean declared at top of page). You can now execute queries on the database using the pureQuery classes.

The first query you execute using the pureQuery code is the one fetching the matching username from the database. If the password returned matches, you set the session variable, `userid`, to the `userid` passed in and redirect the user to the `welcome.jsp` page. If authentication failed, you output that it failed, and if the `userid` and `password` values entered are invalid you state so with output.

Take a look at the login page.

Figure 31. The login page



After logging in, you'll be sent to the `welcome.jsp` page, which you'll now define.

The welcome page

The welcome page is like a landing page for clients of your wealth management firm. Thus, it's a pretty simple file, so define it as shown in Listing 7.

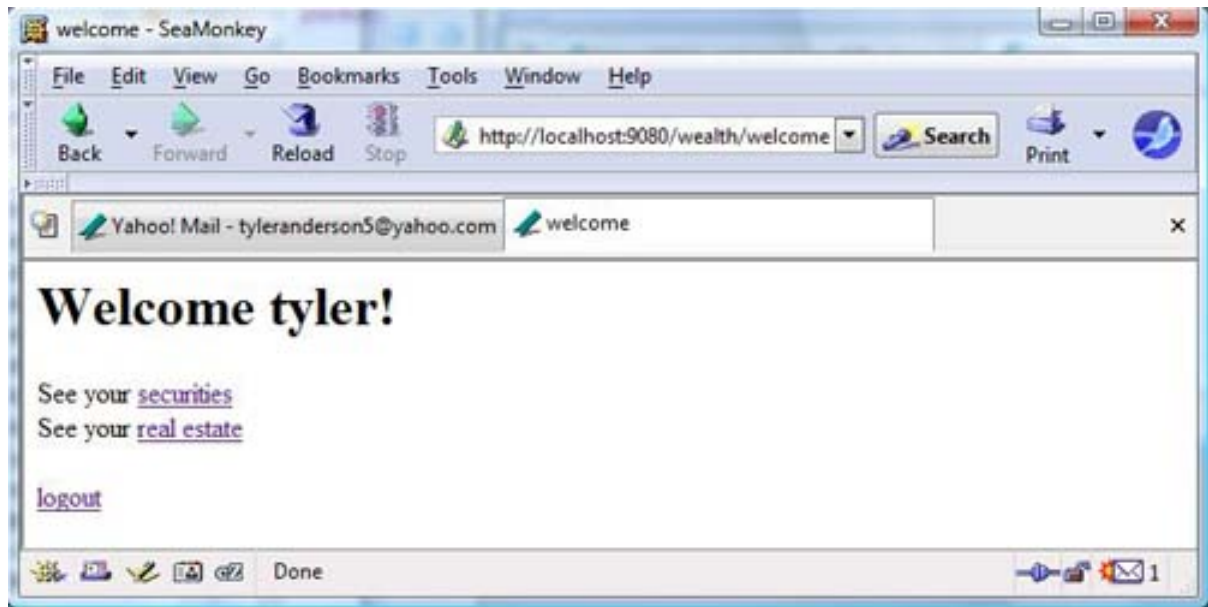
Listing 7. Defining the `welcome.jsp` Web page

```
<!DOCTYPE HTML  
PUBLIC
```

```
"-//W3C//DTD HTML
4.01
Transitional//EN">
<%@page
language="java"
contentType="text/html;
charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<html>
<head>
<title>welcome</title>
<meta
http-equiv="Content-Type"
content="text/html;
charset=ISO-8859-1">
<meta
name="GENERATOR"
content="Rational
Application
Developer">
</head>
<body>
<!-- If userid
session variable
is not null then
the user is
logged in. -->
<%
    String userid
=
(String)session.getAttribute("userid");
    if(userid ==
null){
response.sendRedirect("login.jsp");
    }
%>
<h1>Welcome <%
out.print(userid);
%>!</h1>
See your <a
href="securities.jsp">securities</a><br/>
See your <a
href="realestate.jsp">real
estate</a><br/><br/>
<a
href="logout.jsp">logout</a>
</body>
</html>
```

Make sure that the `userid` session variable exists, and if it doesn't, send the user back to the `login.jsp` Web page. Otherwise, code execution continues where the page welcomes the user, and displays three links: One to the securities page, another to the real estate page, and another to the logout page. Check it out in Figure 32.

Figure 32. The welcome page



Nice. Now move on and define the logout page.

The logout page

The logout page clears the userid session variable, blocking further access to the system. Define this page, as shown in Listing 8.

Listing 8. Defining the logout Web page

```
<!DOCTYPE HTML
PUBLIC
"-//W3C//DTD HTML
4.01
Transitional//EN">
<%@page
language="java"
contentType="text/html;
charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<html>
<head>
<title>logout</title>
<meta
http-equiv="Content-Type"
content="text/html;
charset=ISO-8859-1">
<meta
name="GENERATOR"
content="Rational
Application
Developer">
</head>
<body>
<!-- Logging out
by resetting the
userid session
variable to
```

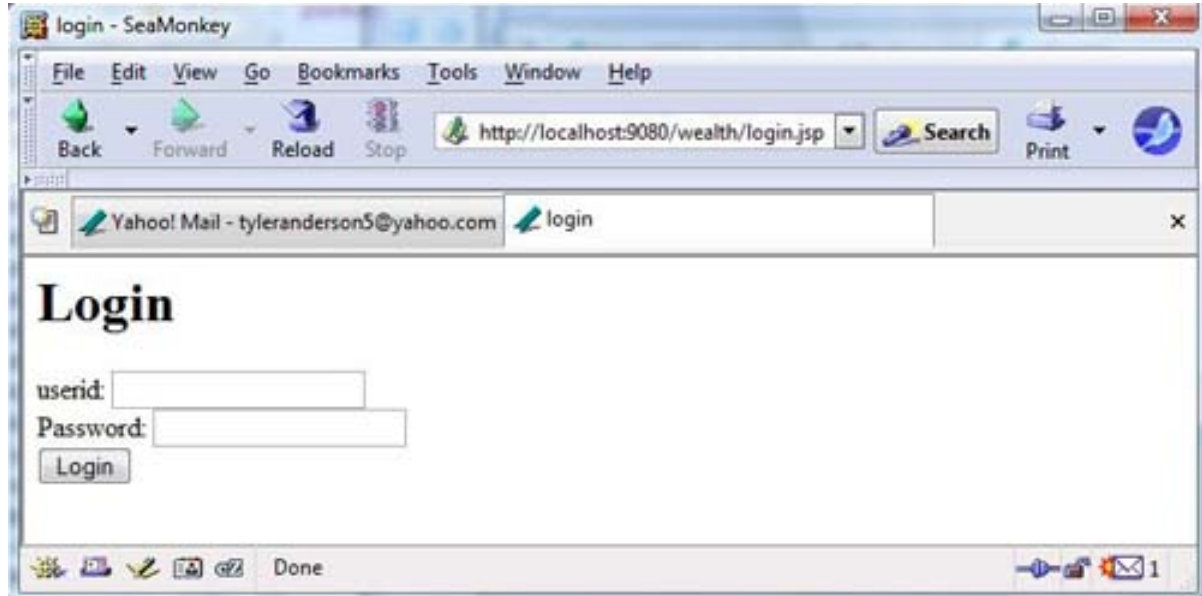
```

false. -->
<%
    String userid
=
(String)session.getAttribute("userid");
    if(userid !=
null){
session.setAttribute("userid",
null);
response.sendRedirect("login.jsp");
    }
%>
</body>
</html>

```

Nothing is displayed if the user isn't logged in, but if the user is logged in (userid is not null), the userid session variable is set to null, and the user is redirected to the login page. So try clicking the logout link now from the welcome page, you'll be brought back to the login page.

Figure 33. Back to the login page



Next you'll code up the realestate page so your visitors can view and enter their real estate holdings.

The realestate page

To create the realestate page where users can view their real estate holdings and enter new ones, define the realestate.jsp Web page, as shown in Listing 9.

Listing 9. Defining the realestate Web page

```
<!DOCTYPE HTML
```

```
PUBLIC
"-//W3C//DTD HTML
4.01
Transitional//EN">
<jsp:useBean
id="realestate"
class="com.ibm.ds.realestate.RealestateDataImpl"
scope="page" />
<%@page
language="java"
contentType="text/html;
charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@page
import="java.util.Iterator"
%>
<%@page
import="com.ibm.ds.realestate.Realestate"
%>
<%@page
import="pureQuery.example.*"
%>
<%@page
import="com.ibm.pdq.runtime.*"
%>
<%@page
import="java.math.BigDecimal"
%>
<html>
<head>
<title>realestate</title>
<meta
http-equiv="Content-Type"
content="text/html;
charset=ISO-8859-1">
<meta
name="GENERATOR"
content="Rational
Application
Developer">
</head>
<body>
<!-- Here we
login to the
database again to
fetch all real
estate
holdings in the
realestate
table, same
database
connection URL as
before. -->
<h1>Your
realestate
holdings</h1>
<table
border="1"><tr><th>Address</th>
<th>City</th><th>Zip</th>
<th>State</th><th>Value</th></tr>
<%
String userid
=
(String)session.getAttribute("userid");
if(userid ==
null){
response.sendRedirect("login.jsp");
}
Data d =
```

```

SampleUtil.getData("jdbc:db2://localhost:50000/WEALTH",
"<enter_your_username>",
"<enter_your_password>");
realestate.setData(d);
    Iterator i =
realestate.getRealestates(userid);

    Realestate
curr = null;

    double total
= 0;
while(i.hasNext()){
    curr =
    (Realestate)
i.next();
out.println("<tr><td>"+curr.getAddress()+"</td>
<td>"+curr.getCity()+
"</td><td>"+curr.getZip()+"</td>
<td>"+curr.getState()+
"</td><td>$"+curr.getPropertyValue()+"</td>
</tr>");
    total +=
curr.getPropertyValue().doubleValue();
}
    if(total > 0)
        total =
Math.round(total*100.0)/100.0;
%>
</table>
<h3>Total value:
$<%
out.print(total);
%></h3>
<h4>Enter new
real estate:</h4>
<%
    String
address =
request.getParameter("address");
    String city =
request.getParameter("city");
    String zip =
request.getParameter("zip");
    String state
=
request.getParameter("state");
    String value
=
request.getParameter("value");
if(request.getParameter("create")
!= null){
if(!address.equals(""))
&&
!city.equals("")
&&
!zip.equals("")
&&
!state.equals("")
&&
!value.equals("")){
realestate.createRealestate(new
BigDecimal(value),
address, city,
zip, state,
userid);
response.sendRedirect("realestate.jsp");
}
    else

```

```
out.println("You
must fill in all
fields.<br/><br/>");
}
%>
<form
method="post">
  Address:
  <input
name="address"/><br/>
  City: <input
name="city"/><br/>
  Zip: <input
name="zip"/><br/>
  State: <input
name="state"/><br/>
  Value:
  $<input
name="value"/><br/>
  <input
type="submit"
name="create"
value="Create"/>
</form>
<a
href="welcome.jsp">back</a>
</body>
</html>
```

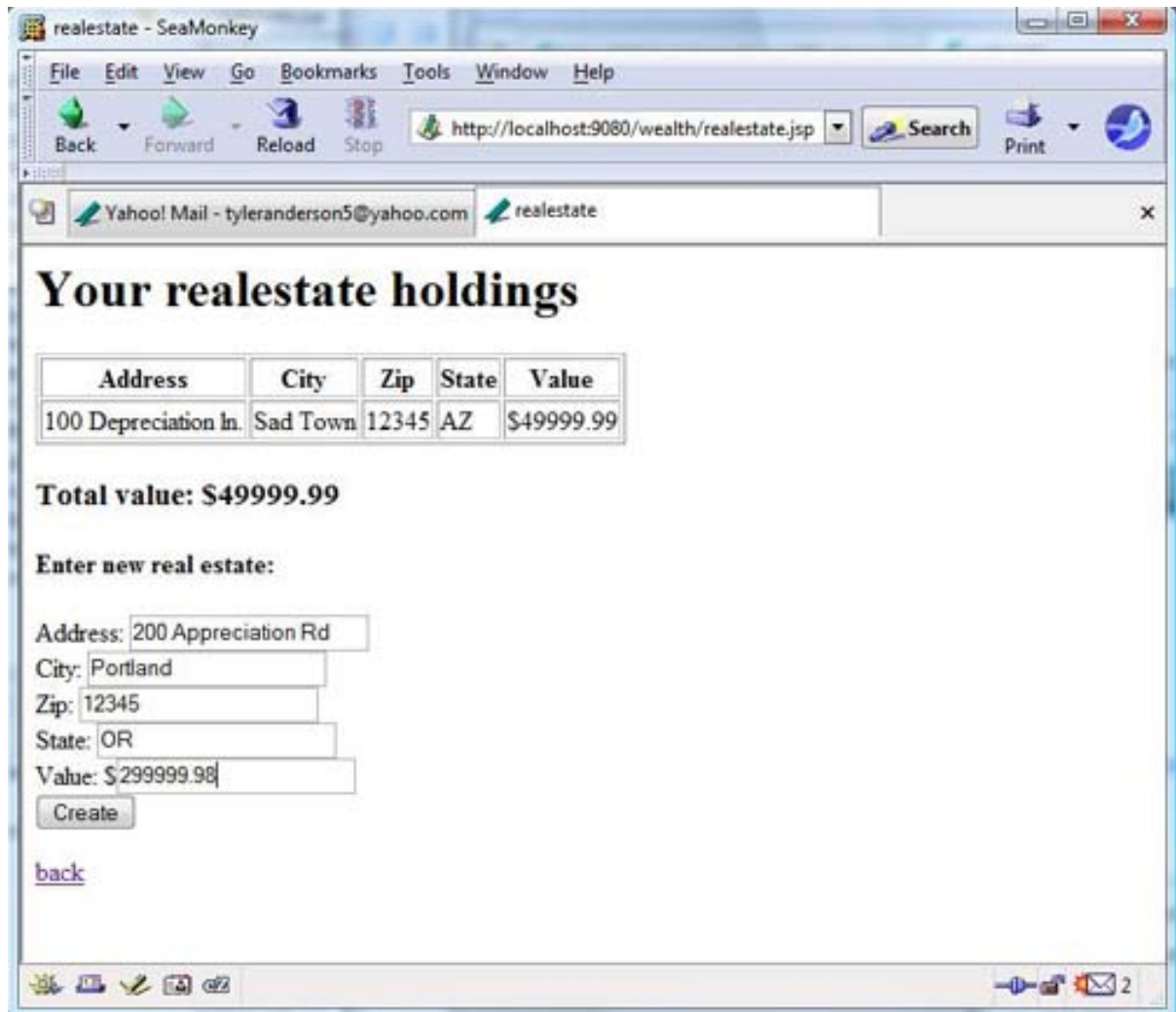
This one is more like the login.jsp page where you've declared the realestate bean at the top of the page, along with several import statements of classes you'll use throughout this JSP Web page.

Starting at the beginning of the <body> tag, the start of a table is setup to display all the real estate holdings. Here you reconnect to the database, and note that here is where you use the new pureQuery statements you created in the realestateData.java class by querying the database on userid only to retrieve a list of matching records. These are then displayed to the browser within the table, followed by a total value of the user's real estate holdings.

Lastly, a form is setup to take input for new real estate holdings. Entering valid data into this form calls the realestate.createRealestate() function of the realestate bean. After the new realestate record is successfully added, the user is redirected back to the same page. If invalid data is entered, it's stated with an output statement.

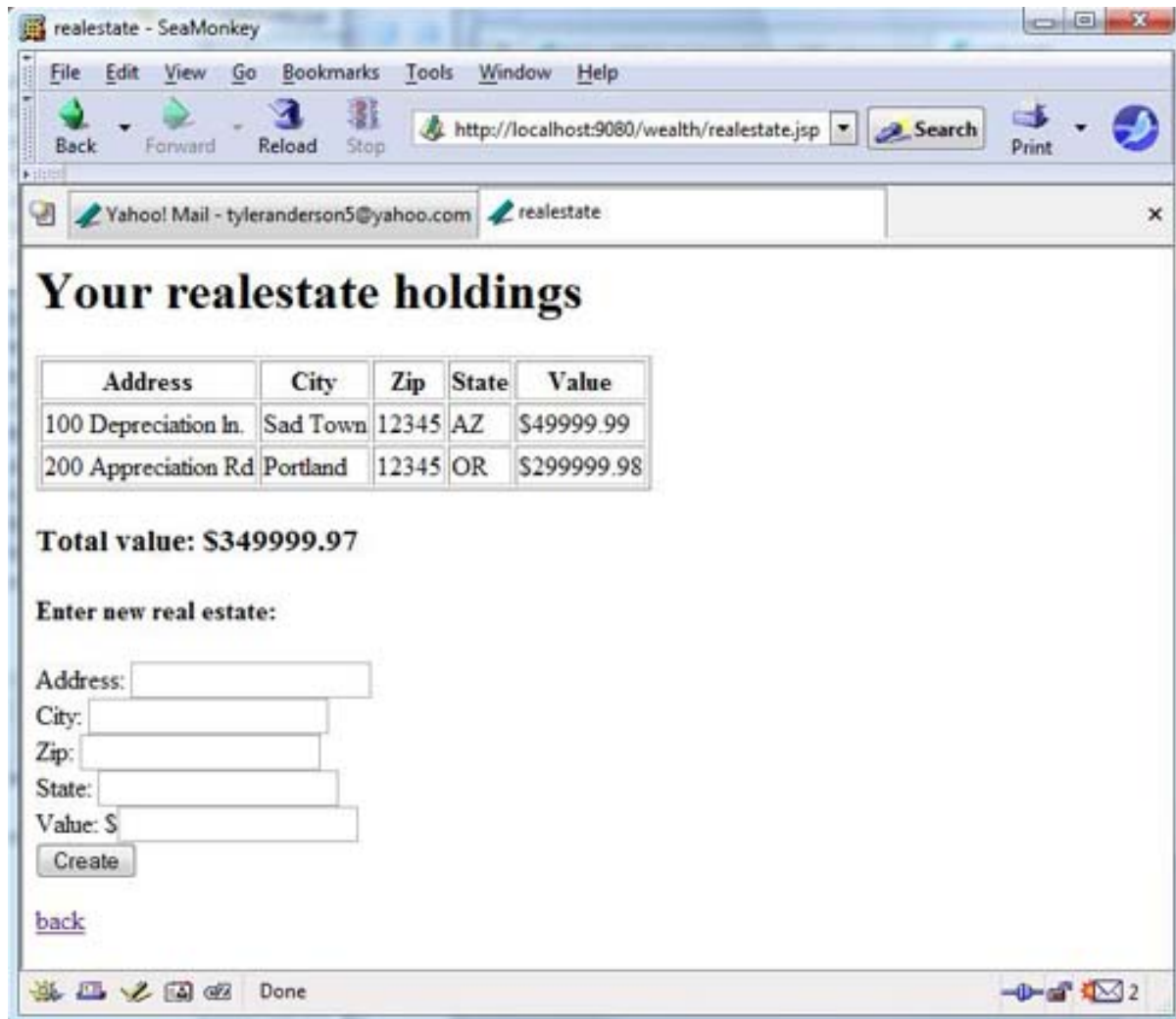
Login to the system using hilibilly, and then see the realestate Web page, as shown in Figure 34.

Figure 34. The realestate Web page



Click **Create** to execute the code and add the new realestate record into the system.

Figure 35. Entering a new realestate record



The record was successfully added, and the total updated.

Next you'll move onto the securities page where you'll get to query the Yahoo! Finance API.

The securities page

You're now on the last page, where users will be able to view the delayed values of their securities holdings. Define this page, as shown in Listing 10.

Listing 10. Defining the securities Web page

```
<!DOCTYPE HTML
PUBLIC
"-//W3C//DTD HTML
4.01
Transitional//EN">
```

```

<jsp:useBean
id="securities"
class="com.ibm.ds.securities.SecuritiesDataImpl"
scope="page" />
<%@page
language="java"
contentType="text/html;
charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@page
import="java.util.Iterator"
%>
<%@page
import="com.ibm.ds.securities.Securities"
%>
<%@page
import="pureQuery.example.*"
%>
<%@page
import="com.ibm.pdq.runtime.*"
%>
<%@page
import="java.net.*"
%>
<%@page
import="java.io.*"
%>
<html>

<head>
<title>securities</title>
<meta
http-equiv="Content-Type"
content="text/html;
charset=ISO-8859-1">
<meta
name="GENERATOR"
content="Rational
Application
Developer">
</head>
<body>
<!-- Here we
login to the
database to fetch
all securities
      in the
securities table,
      same
database URL as
before. -->
<h1>Your stock
and options
holdings</h1>
<table
border="1"><tr><th>Sybmol</th><th>
Description</th><th>Quantity*</th>
<th>Type</th><th>Current
price**</th>
<th>Total
value</th>
<th>Quote
date</th></tr>
<%
      String userid
=
      (String)session.getAttribute("userid");
      if(userid ==
null){

```

```

response.sendRedirect("login.jsp");
    }
    Data d =
SampleUtil.getData("jdbc:db2://localhost:50000/WEALTH",
"<enter_your_username>",
"<enter_your_password>");
securities.setData(d);
    Iterator i =
securities.getSecuritiess(userid);

    Securities
curr = null;
    double total
= 0;
while(i.hasNext()){
    curr =
(Securities)
i.next();

    String
str =
"http://finance.yahoo.com/d/quotes.csv?s="+
curr.getSymbol()+"&f=nd111";
    URL url =
new URL(str);
    URLConnection
conn =
url.openConnection();
    DataInputStream
in =
        new
DataInputStream (
conn.getInputStream
( ) );
    BufferedReader br
= new
BufferedReader(new
InputStreamReader(in));
    String
line = null;
    if(br.ready())
        line
= br.readLine();

    String
company =
line.substring(line.indexOf('')+1,
line.indexOf('"',
line.indexOf('')+1));

    String
date =
line.substring(line.indexOf("\","\")+3,
line.indexOf("\",",
line.indexOf("\","\")+4));

    String
price =
line.substring(line.indexOf("\",",
line.indexOf("\","\")+4)+2);

    int
multQuantity =
curr.getQuantity();
    if(curr.getSecuritytype().equals("option"))
multQuantity =
curr.getQuantity()*100;

    float

```

```

totalValue =
Float.parseFloat(price)*multQuantity;
totalValue =
(float)(Math.round(totalValue*100.0)/100.0);

out.println("<tr><td>"+curr.getSymbol()+"</td>
<td>"+company+
"</td><td>"+curr.getQuantity()+"</td><td>"+
curr.getSecuritytype()+"</td><td>$"+price+
"</td><td>$"+totalValue+"</td><td>"+date+"
</td></tr>");

        total +=
totalValue;
    }
    if(total > 0)
        total =
Math.round(total*100.0)/100.0;

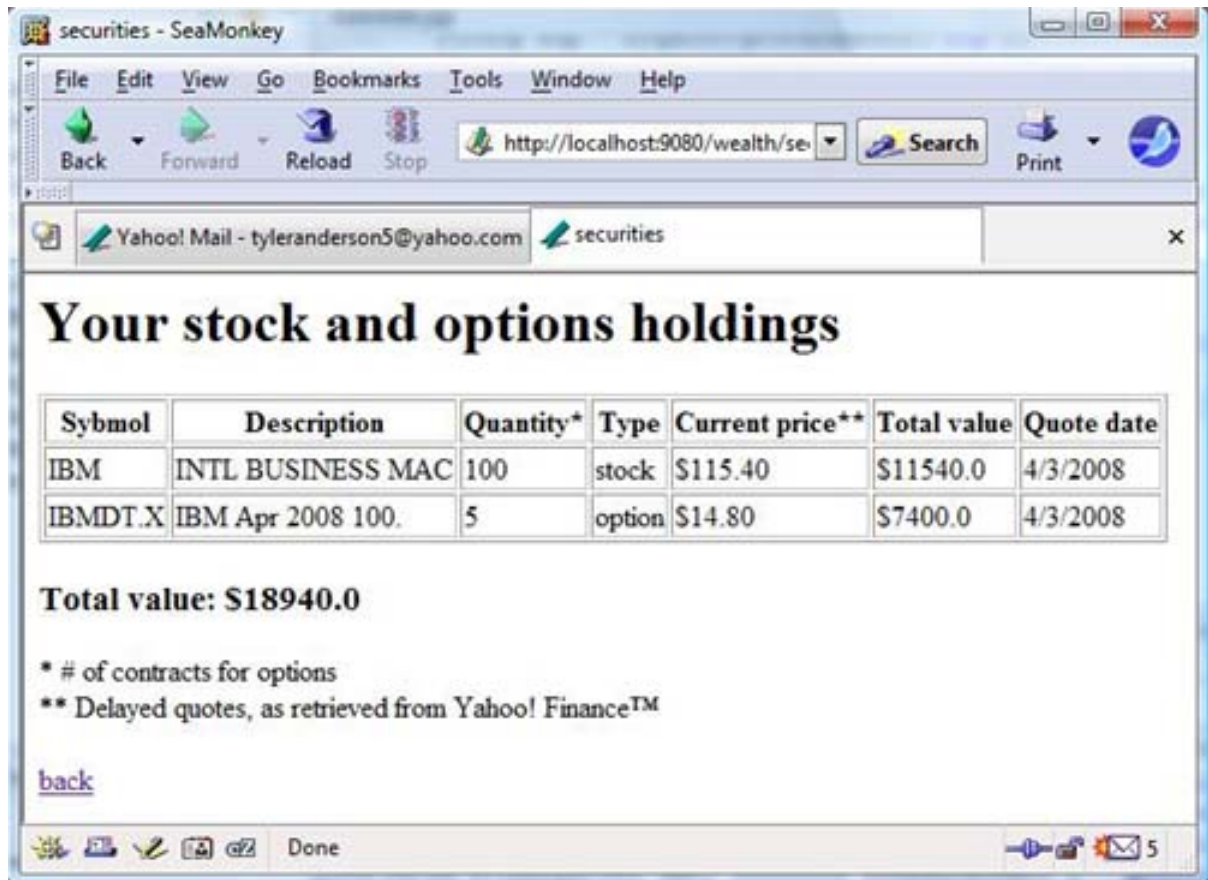
%>
</table>
<h3>Total value:
$<%
out.print(total);
%></h3>
* # of contracts
for
options<br/>**
Delayed quotes,
as retrieved from
Yahoo!
Finance&trade;
<br/><br/><a
href="welcome.jsp">back</a>
</body>
</html>

```

Ignoring the similarities to the realestate Web page, here you query the database for all this user's securities, and setup a table to display them. Then for every security, you query the Yahoo! Finance API by setting up the URL, connecting to the Web page, and grabbing results into the line variable with the call to `line = br.readLine()`; . The next three lines parse the results into three variables. Note that if the `securityType` is equal to "option", the quantity for computing the value of the security is multiplied by 100 (as there are 100 securities per options contract).

Get a first look of the securities page in action in Figure 36 (logged in with user tyler).

Figure 36. The securities Web page



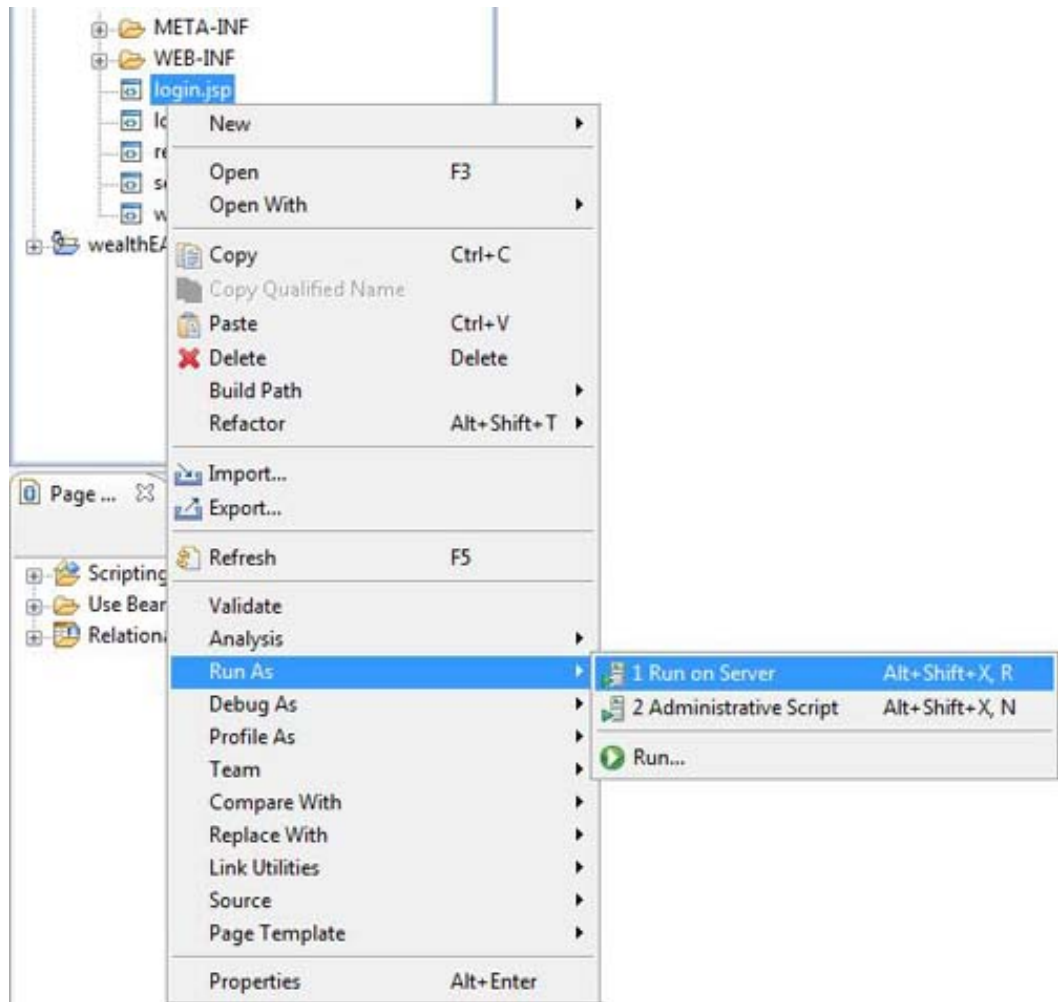
Like a pro, you've completed the code! Now time to suit up and deploy on WebSphere Application Server!

Deploy and test on WebSphere Application Server

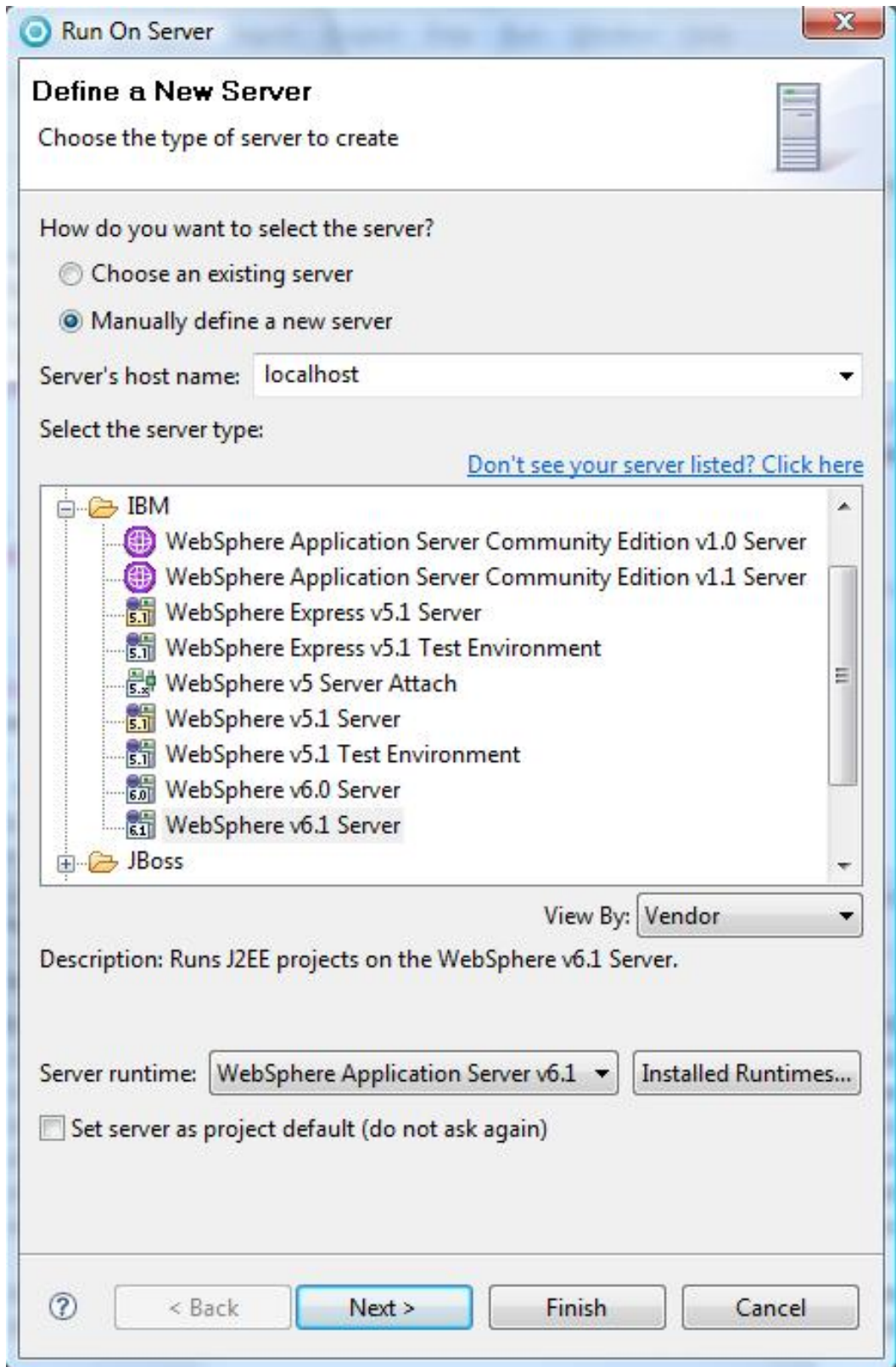
To run your app on WebSphere Application Server, there are a few last steps to prepare your app.

1. Right-click login.jsp within the Project Explorer window and select **Run As > Run on Server**.

Figure 37. Running on WebSphere Application Server

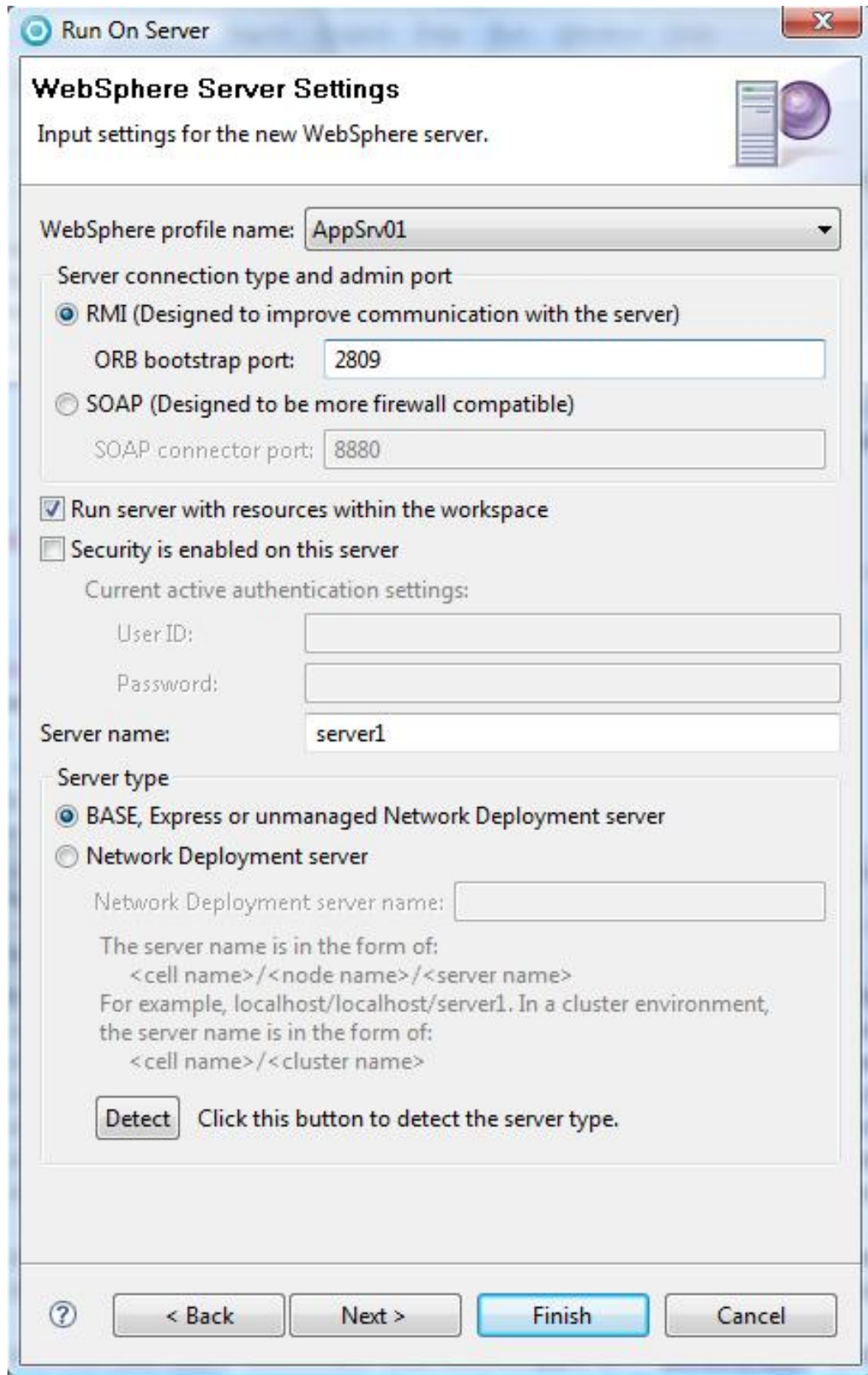


2. The Run on Server window opens.
Figure 38. Defining a new server



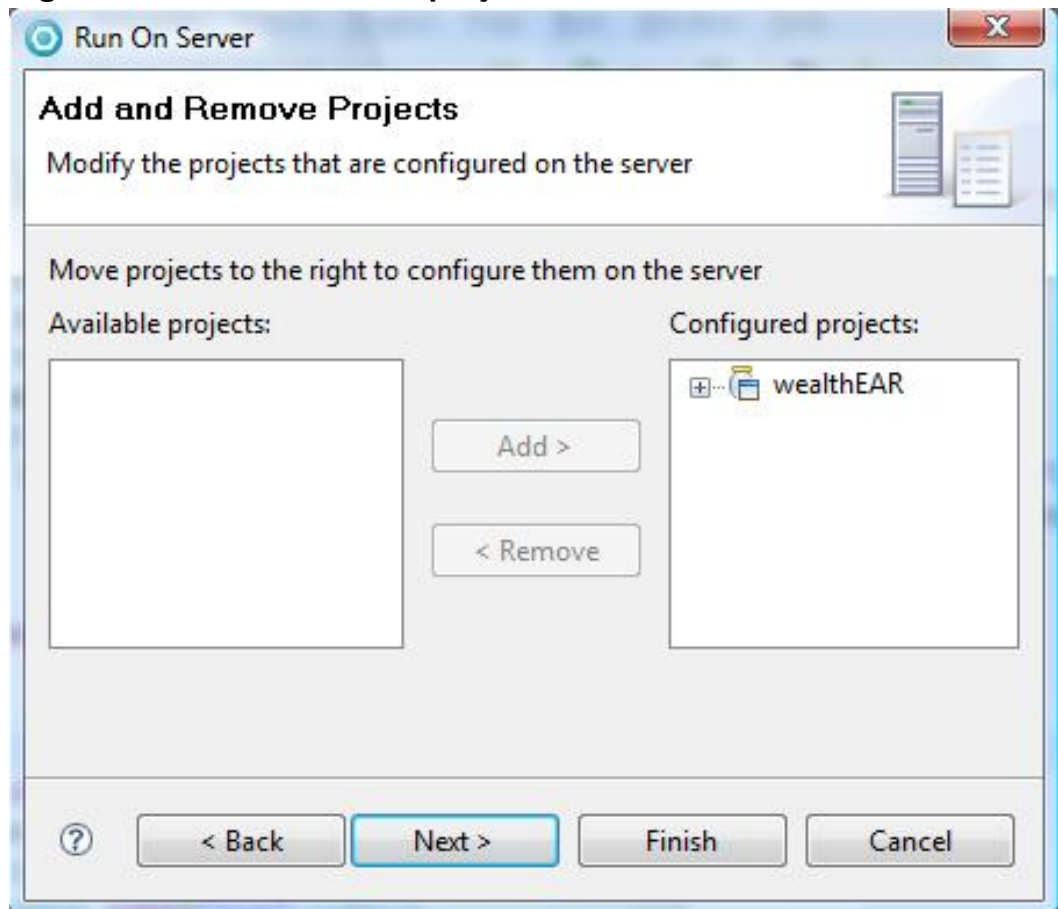
3. Select **Manually define a new server** and choose **WebSphere v6.1 Server** under the IBM folder. If everything looks good, click **Next**.
4. The WebSphere Server Settings window opens.

Figure 39. Configuring WebSphere server settings

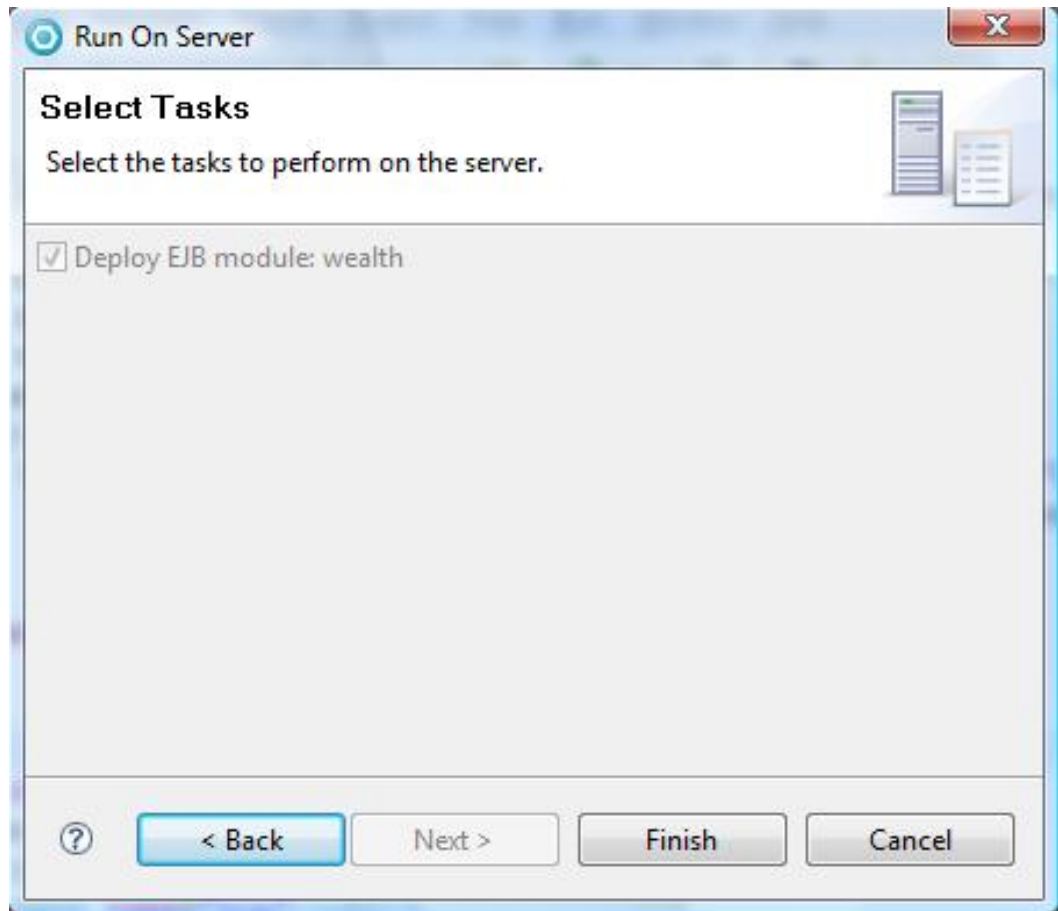


5. Keep the defaults and click **Next**.
6. The Add and Remove Projects window opens. Make sure the wealthEAR project is listed under Configured projects.

Figure 40. Add and remove projects



7. Click **Next**.
- Figure 41. Select tasks**



8. That's the end, so click **Finish**.

The server should now start. Open a browser and view the login page at the following URL: <http://localhost:9080/wealth/login.jsp>.

You now have had a true front-to-back Rational Application Developer with Data Studio experience.

Section 6. Summary

You've successfully completed this tutorial and have mastered the use of pureQuery support to your JSP Web pages. You've installed Rational Application Developer together with Data Studio, successfully created a pureQuery enabled Dynamic Web Project, generated pureQuery code from your database tables, and wrote five JSP Web pages to demonstrate their functionality.

Make sure to come back for Part 2 where you'll learn how to check your Java-based Web application for vulnerabilities and fix them with the help of AppScan!

Downloads

Description	Name	Size	Download method
Part 1 source code	r-appscan1-code.zip	5.8MB	HTTP

[Information about download methods](#)

Resources

Learn

- If you want a refresher on JavaServer Pages, be sure to read this [tutorial](#).
- To learn about the features in IBM WebSphere 6.1, start with [this](#) article.
- Start [here](#) to learn about what's new in IBM Rational Application Developer 7.0.
- Get an overview of Rational Application Developer V7 in the tutorial, [Hello World: Learn how to create Java, Web service, and database applications with Rational Application Developer V7](#) (Jane Fung, developerWorks, January 2007).
- Follow step-by-step installation instructions in the tutorial [Use an XML database in PHP and Java applications](#) (Tyler Anderson, developerWorks, March 2008).
- To learn how to use the DB2 Control Center to create a new database and database tables, read [XForms and Ruby on Rails at the Doctor's Office, Part 1: Setting up IBM DB2 9 pureXML](#) (Michael Galpin, developerWorks, December 2007).
- To learn about pureQuery, which gives database application developers an easy, GUI-based means to significantly increase productivity in both the design and implementation phases, read [Understanding pureQuery, part 1: pureQuery: IBM's new paradigm for writing Java database applications](#) (Azadeh Ahadian, developerWorks, December 2007) and [Understanding pureQuery, part 2: Assist class modelers with data modeling](#) (Azadeh Ahadian, developerWorks, January 2008).
- See the unofficial [Yahoo! Finance API](#).
- Visit the [developerWorks resource page for DB2 for Linux, UNIX, and Windows](#) to read articles and tutorials and connect to other resources to expand your DB2 skills.
- [developerWorks technical events and webcasts](#): Stay current with developerWorks technical events and webcasts.
- [Podcasts](#): Tune in and catch up with IBM technical experts.

Get products and technologies

- Download [Rational Application Developer version 7](#).
- Download [Data Studio v1.1.2](#).

Discuss

- [Participate in the discussion forum for this content](#).

- Participate in the [Rational products discussion forum](#).
- Participate in the [Data Studio discussion forum](#).

About the author

Tyler Anderson



Tyler Anderson has graduated with a degree in computer science in 2004 and a Master of Science degree in computer engineering in December, 2005, both from Brigham Young University. Tyler is currently a freelance writer and developer for Backstop Media.