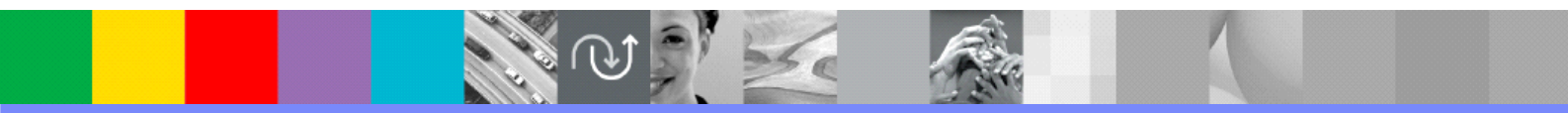
IBM Software Group

An IBM Proof of Technology

IBM Data Studio pureQuery For DBAs and Application Developers on z/OS (v2.2)

Lab Exercises



PoT.IM.09.1.077.00

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

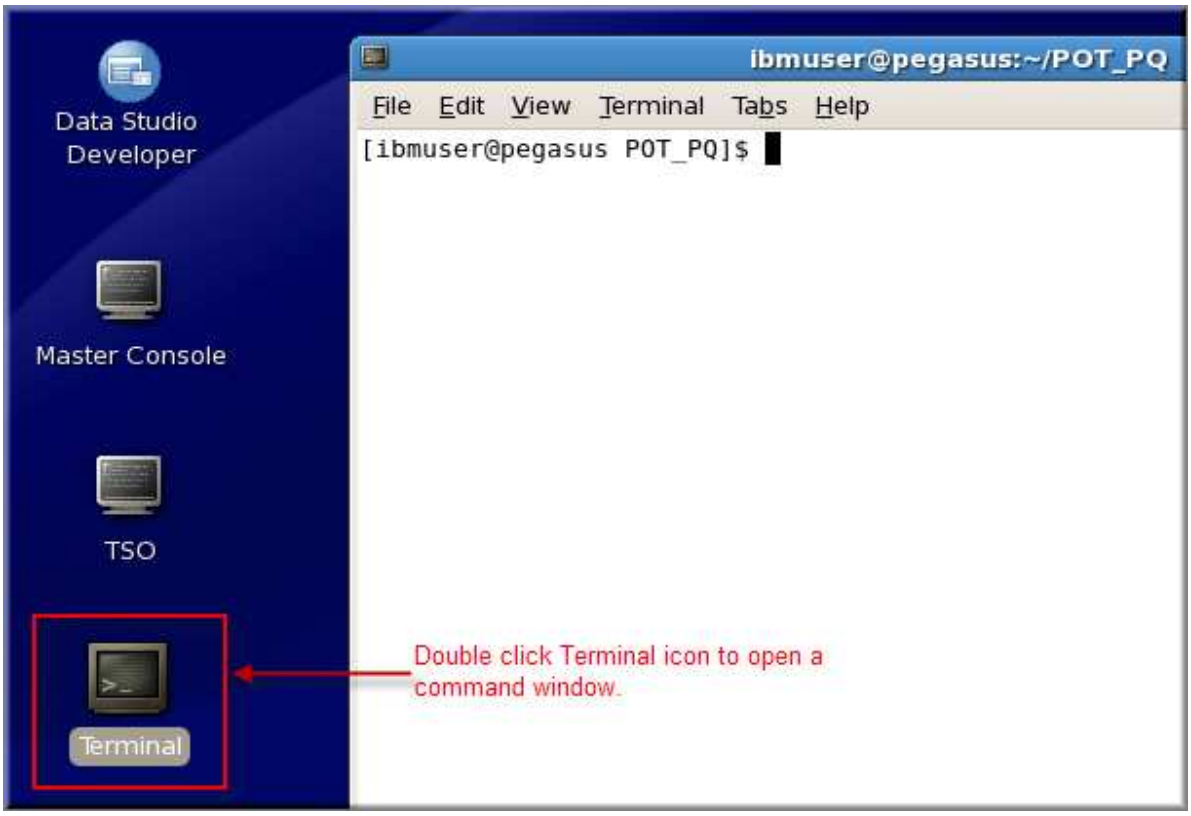
Lab 1- IPL z/OS and Introduction to Data Studio Developer

In this lab, you will first IPL (Initial Program Load) the IBM® z/OS® system, and then learn how to use IBM® Data Studio to open the Data and Java™ perspectives. You will see how to connect to a database and sample the contents of the tables in the database. Finally you will learn how to debug a stored procedure.

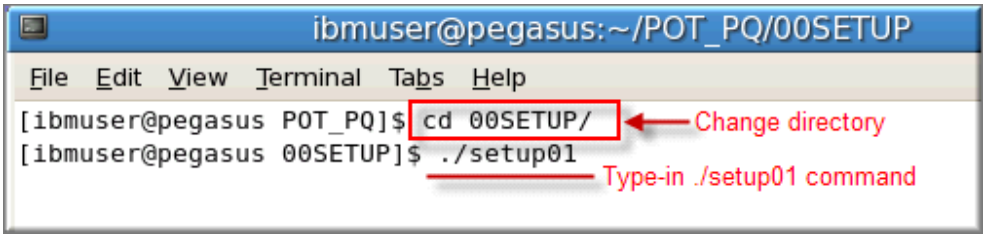
1.1 IPL z/OS System

__1. Make sure that you have the USB dongle connected to your laptop. If you do not see a USB dongle, please stop this lab and ask the instructor about it. You will not be able to continue with out it.

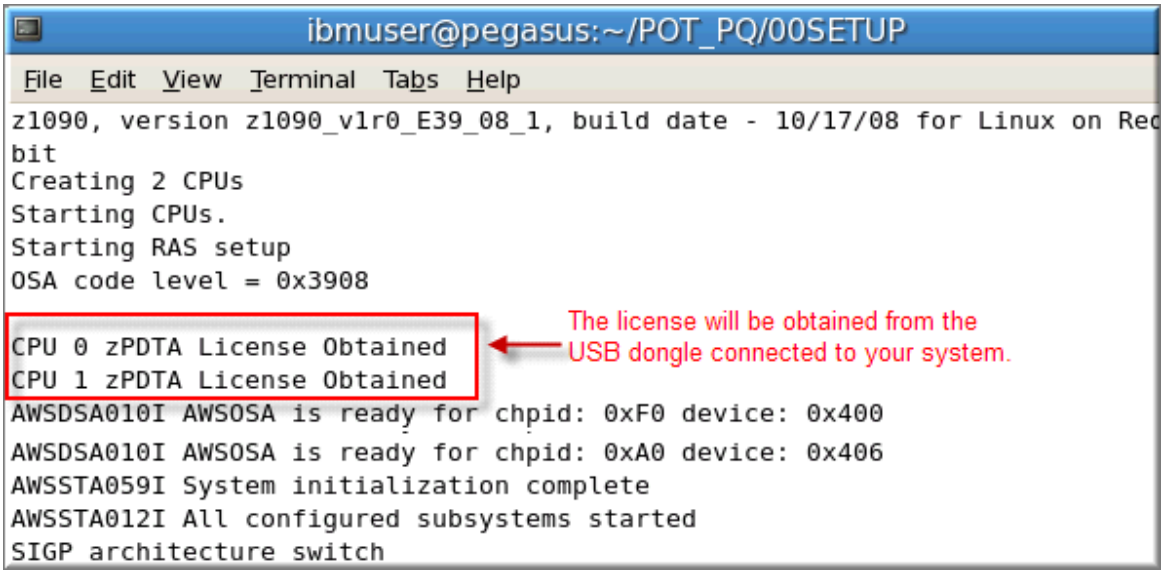
__2. Double click on the *Terminal* icon on the desktop to open up a command window.



You will see a command window as shown above and change directory to *00SETUP*. Type the command: *./setup01.*. This will create your database on the z/OS system in your virtual machine and IPL your own personal mainframe.

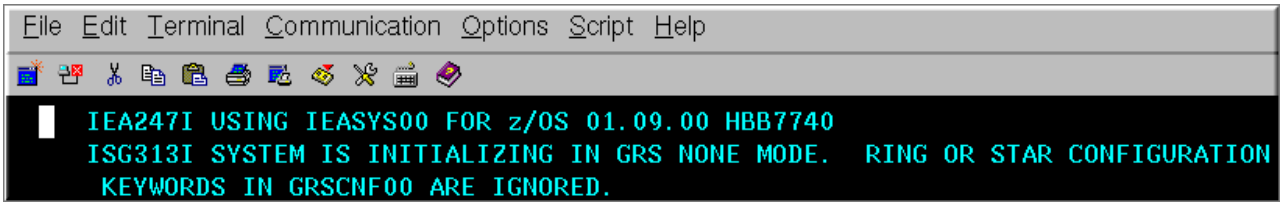


__3. After you type in the above command, the personnel z/OS system will start to come up. First you will see a license obtained message for 2 CPUs. This is your USB dongle working for you.



__4. Immediately after licenses are obtained, you will see a z/OS master console start up.

__5. When the z/OS starts to IPL, you will see messages showing in the master console.



__6. Occasionally the IPL of z/OS will wait for an operator response because of a time change since the last time z/OS was active (See IRL messages below) or the coupling facility will need to be reinitialized because of something like a time change in the BIOS of the ThinkPad (see the IXC messages below).

```
  IXC414I CANNOT JOIN SYSPLEX ADCDPL WHICH IS RUNNING IN MONOPLEX MODE:
   CONFIGURATION REQUIREMENT
  IXC404I SYSTEM(S) ACTIVE OR IPLING: ADCD
| IXC420D REPLY I TO INITIALIZE SYSPLEX ADCDPL, OR R TO REINITIALIZE XCF.
   REPLYING I WILL IMPACT OTHER ACTIVE SYSTEMS.
```

__7. Reply `00,I` on the master console and press right `CTRL` key.

```
00,I
```

```
00,I
IEE600I REPLY TO 00 IS;I
IXC413I MULTISYSTEM SYSPLEX CONFIGURATION PREVENTED BY SYSTEM COMPONENT
ISG150I GRS=NONE IS NOT SUPPORTED WHEN RUNNING IN A MULTISYSTEM SYSPLEX.
IXC418I SYSTEM ADCD IS NOW ACTIVE IN SYSPLEX ADCDPL
```

__8. Please wait for 4-6 minutes for IPL process to finish. When you see `Tivoli Enterprise Monitoring Server (TEMS) startup complete` message, it is an indication that the IPL process has finished.

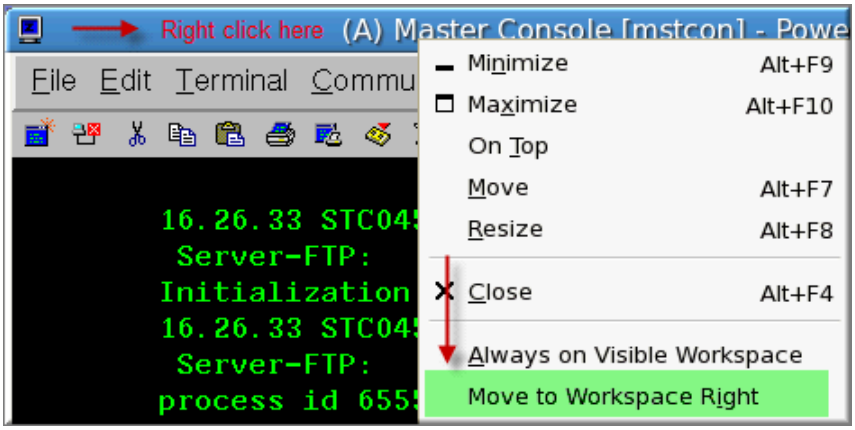
```
    21.29.04 STC01256  KDS9141I   The TEMS SHRERTE:CMS is connected to the
     hub TEMS ip.pipe:#192.168.100.160 5075 .
    21.29.07 STC01256  KO4SRV032  Tivoli Enterprise Monitoring Server (TEMS)
     startup complete.
IEE612I CN=L700     DEVNUM=0700 SYS=ADCD
```

__9. Please remember that the IBM 3270 `RESET` key is mapped to Windows® `PAUSE` key and the `ENTER` key is mapped to the right `CTRL` key.

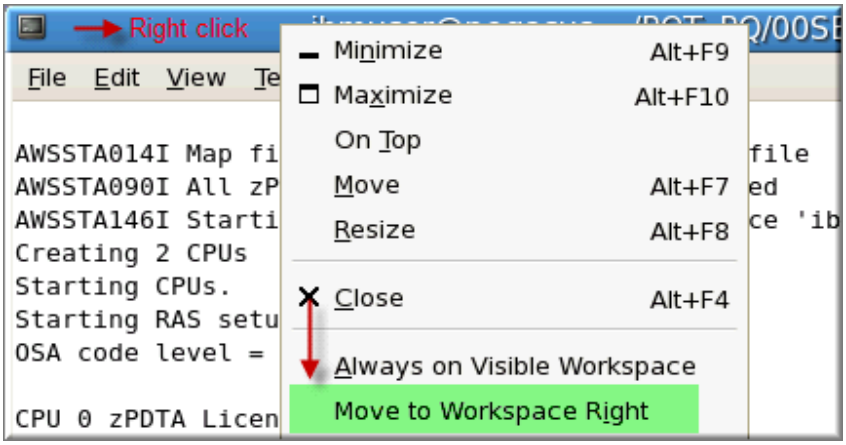
__10. Type `d t` command at z/OS master console to display current system data and time.

```
00- 16.42.57          d t
    16.42.57          IEE136I LOCAL: TIME=16.42.57 DATE=2009.079  UTC:
     TIME=21.42.57 DATE=2009.079
IEE612I CN=L700     DEVNUM=0700 SYS=ADCD
d t                 <-- Type d t here and hit right CTRL key
```

__11. Right click on the title bar of the master console window and click on `Move to Workspace Right`.



__12. Do the same thing for the terminal window where we started `setup01` command.

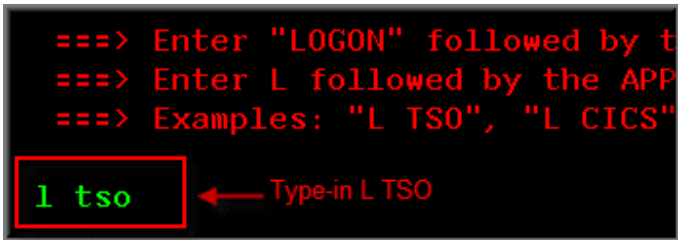


__13. Press <CTRL><ALT><Right Arrow> key on your keyboard to show Linux® Display #2.

__14. You should see both the windows that we moved from our desktop #1.

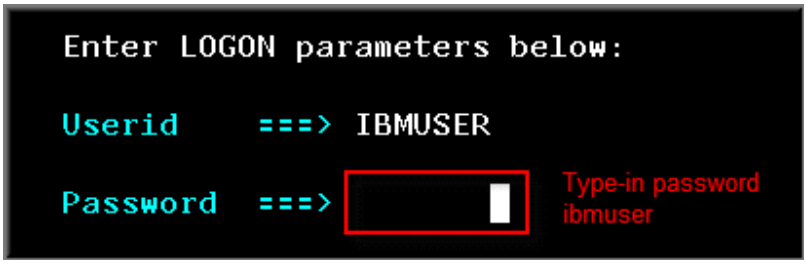
__15. Double click on the TSO icon on the desktop to launch the TSO session.

__16. Type-in `L TSO` and hit the right <CTRL> key.



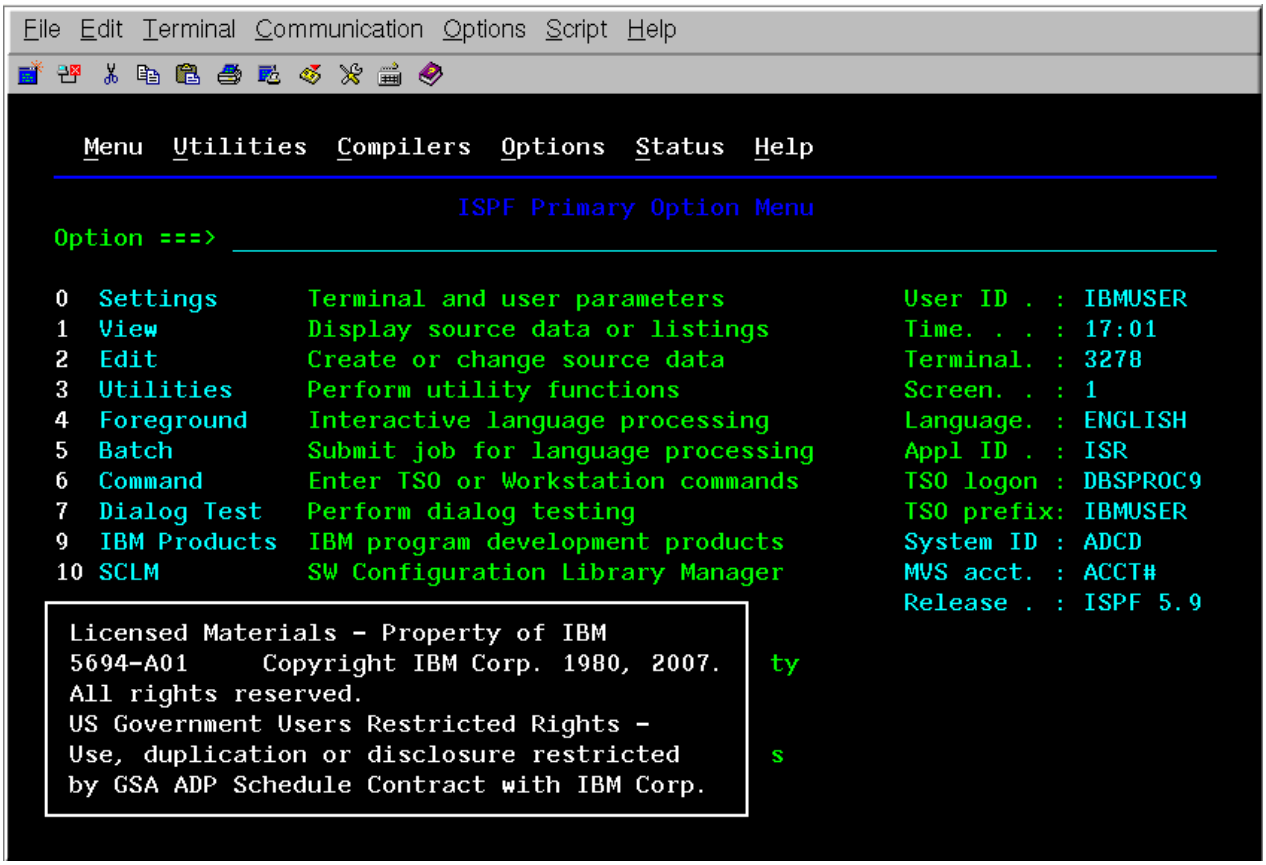
__17. Type in the user id as `ibmuser` and hit the right <CTRL> key.

__18. Type in the password `ibmuser` and hit the right <CTRL> key.



__19. The TSO login process will start and when you see ISPF, hit the right <CTRL> key.

__20. After this, you will see the main z/OS panel as shown below.



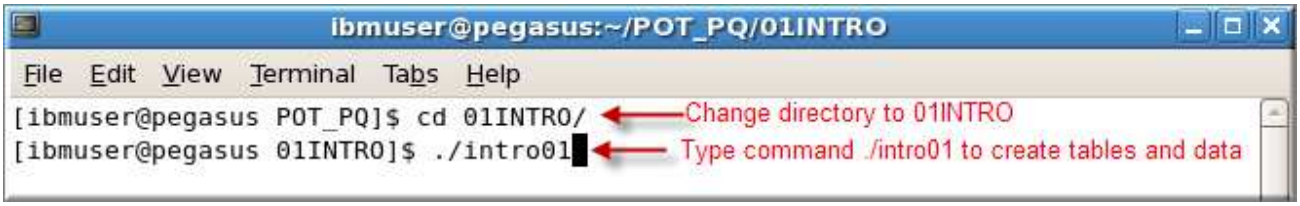
__21. This Linux desktop # 2 contains a z/OS master console and a TSO login. Please return to the Linux Desktop # 1 by pressing <CTRL><ALT><Left Arrow> keys and this will be our primary Linux workspace where we will be doing the remaining lab exercises.

Note: The reason we moved the z/OS related windows to the second Linux workspace is to protect them so that we do not close them accidently. You can switch back and forth between Linux workspaces by using <CTRL><ALT><Left Arrow> and <CTRL><ALT><RIGHT Arrow> keys.

1.2 Required Initial Setup

__22. Please make sure that you are in a Linux Workspace # 1. Double click on the *Terminal* icon on the desktop to open up a command window.

__23. You will see a command window. Change your directory to: `01INTRO`.

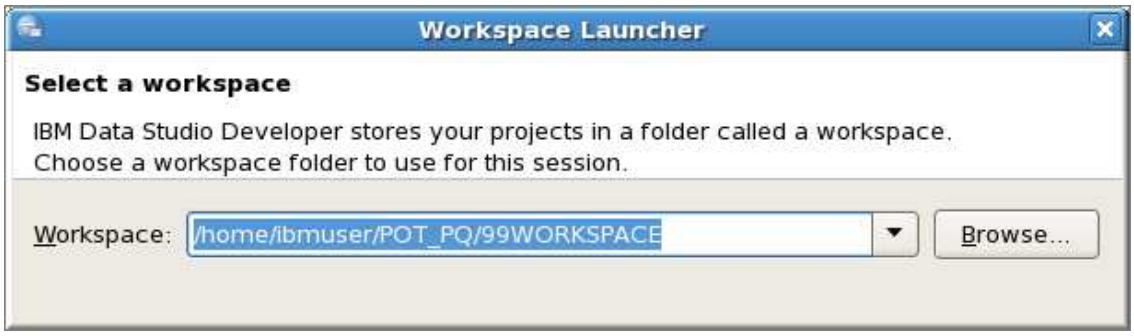


__24. Run `intro01` command to create tables and data in GOSALES and GOSALESCT schema and other objects for the lab exercises for this PoT. When the script finishes, continue with the next section. It may take up to **5 minutes** to create all the required objects and to insert data into tables.

1.3 Open Data Studio Developer

__25. Open the *IBM Data Studio Developer* by clicking on this icon on your desktop.

__26. Make sure that your workspace points `/home/ibmuser/POT_PQ/99WORKSPACE` directory. Click <OK> and wait for the *Data Studio Developer* to launch.



__27. You will see splash screen showing 2 products shell sharing with each other using a single package.

__28. If you get to a welcome screen, close it.

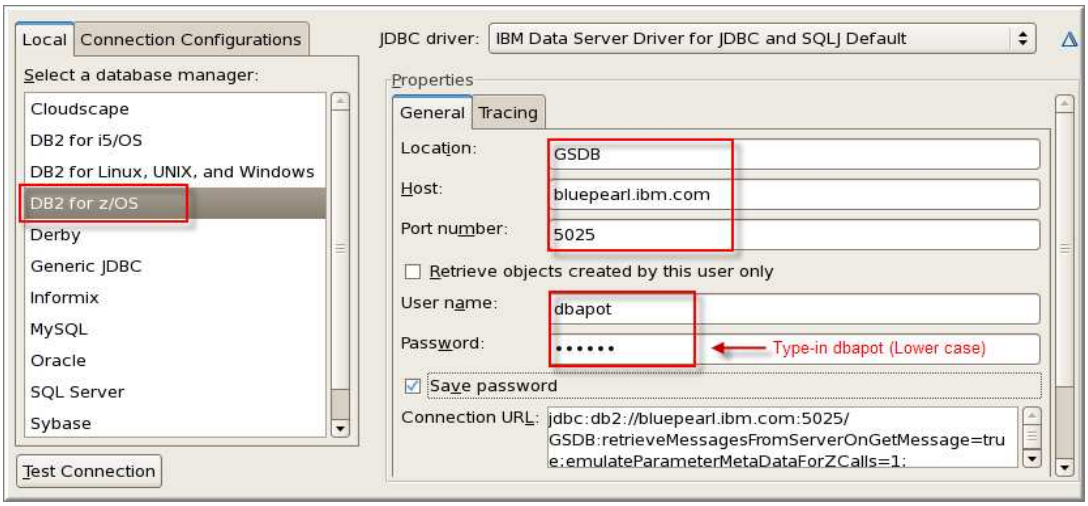
You will now be in the *Data Studio Developer* in a perspective called `Data`. You can see the perspectives on the top right corner of your screen.

1.4 Connect to a database

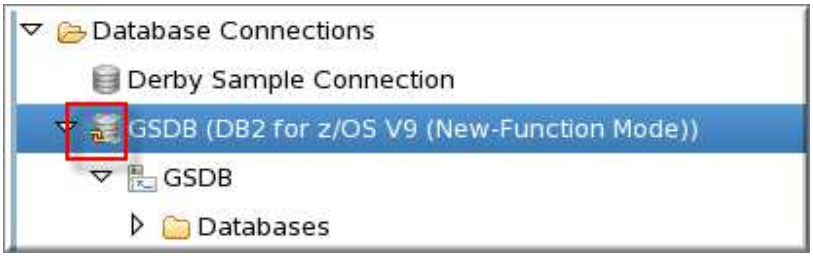
__29. Now we will connect to the `GSDB` location on the host `bluepearl.ibm.com`.

- In the *Data Source Explorer*, you will notice `GSDB2` connection. This is the connection to z/OS DB2® via local DB2 client. We will not use this, but rather we will create a new direct Type-4 JDBC connection to the DB2 on z/OS. Right click on *Database Connections.* Click on `New…`

- Select DB2 for z/OS and enter Location ⇨ *GSDB*, Host ⇨ `bluepearl.ibm.com`, Port Number ⇨ `5025`, User Name ⇨ `dbapot` and password ⇨ `dbapot`. Click on `Save password` and hit `Test Connection` button to test the connection. Click `<Finish>`.



- We now have a connection and the database icon changes to reflect that.



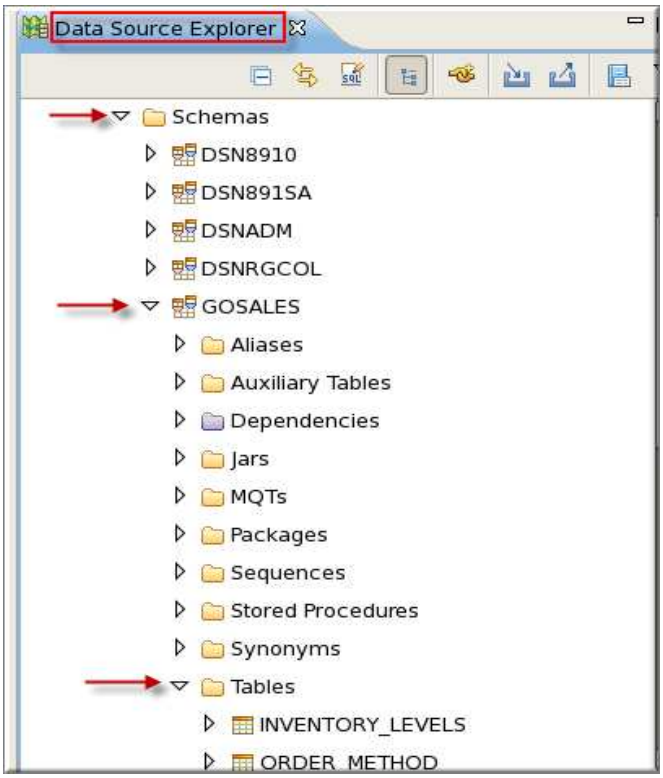
- We can now use *Data Studio Developer* to explore the *GSDB* database objects.

__30. Select GSDB database in *Data Source Explorer* and right click on it to select `Properties`.

__31. Click on the option `Connect every time the workbench is started` and hit `OK`.

__32. In the *Data Source Explorer*, expand `Connections` ⇨ `GSDB` ⇨ `GSDB`

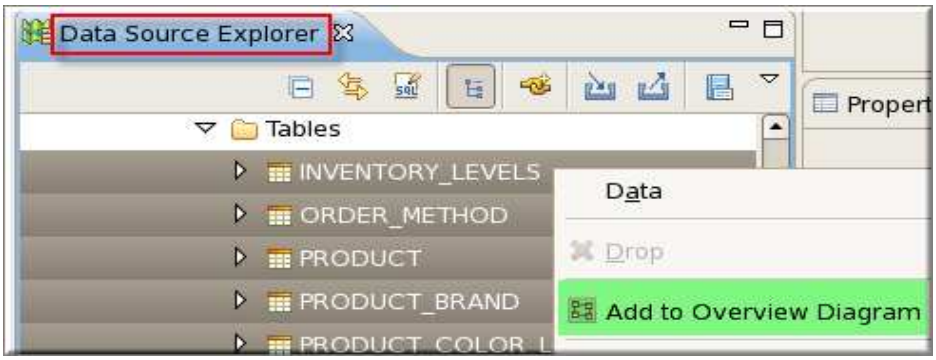
Again expand `Schemas` ⇨ `GOSALES` ⇨ `Tables`



See a visual relationship between tables

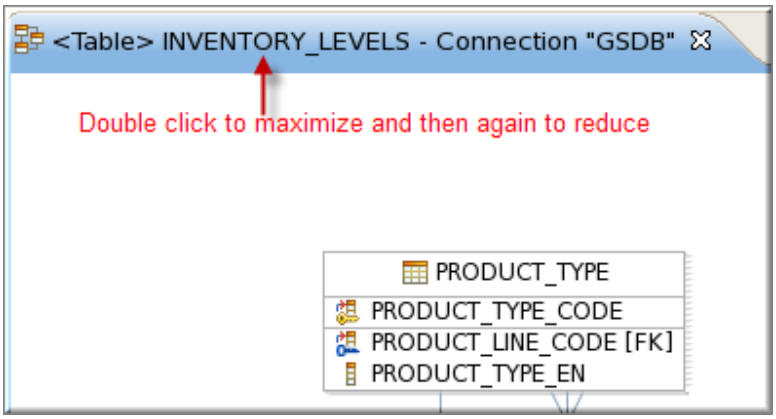
__33. Do the following to see a relationship between tables.

- Click on the *INVENTORY_LEVELS* table
- Hold shift key and click on the *PRODUCT_TYPE* table. By doing so, you will select all tables as shown below. Now right click (to show the context menu) and choose: `Add to overview diagram`.



- Check *Infer implicit relationships*, then *OK* in `<Next>` screen and you should see the overview diagram.

- Double click on the title of the screen to maximize the window to see the relationships diagram for the selected tables in full screen mode.



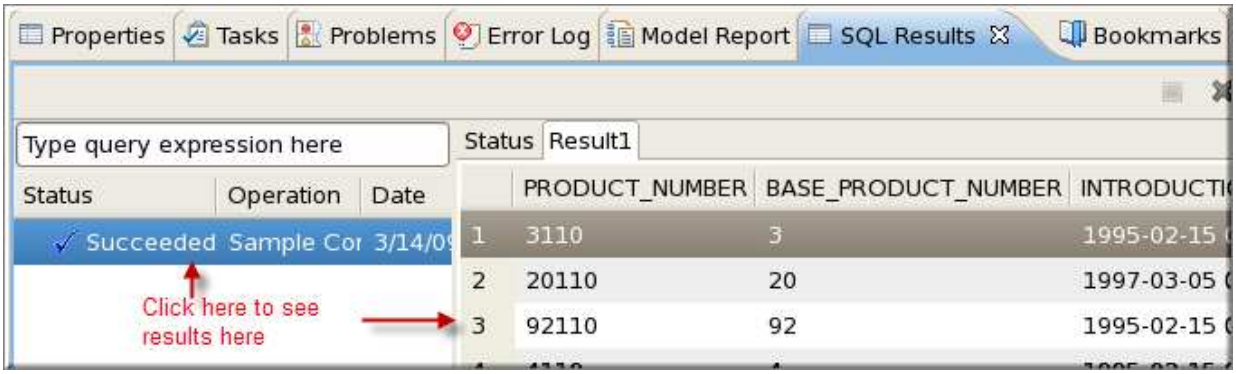
__34. After viewing this, double click on the title to bring the window in its original size.

__35. Now close this overview diagram window (click on the X in the tab). 

Sample some data from a table.

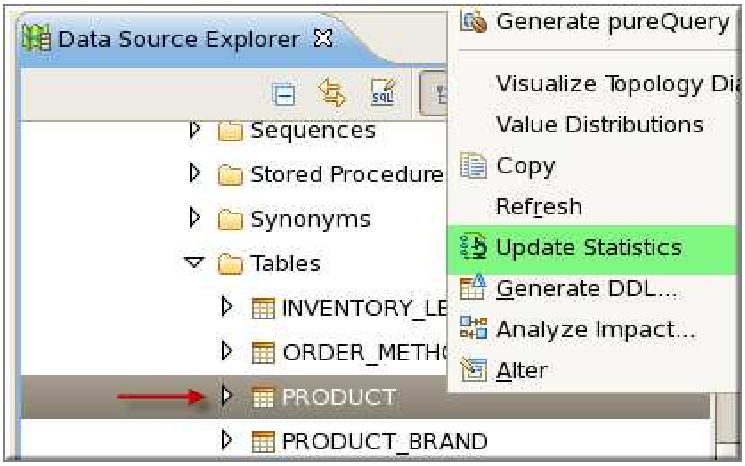
__36. In the *Data Source Explorer, Tables* folder, find the table *PRODUCT*. Right click on *PRODUCT*, then choose: `Data ⇨ Sample Contents`

__37. View the contents of the *PRODUCT* table in the *SQL Results* view. This view is in the bottom right corner of your *Data* perspective. Maximize it if you need to see more columns from the table by double clicking on the title. Double click again to minimize when finished.

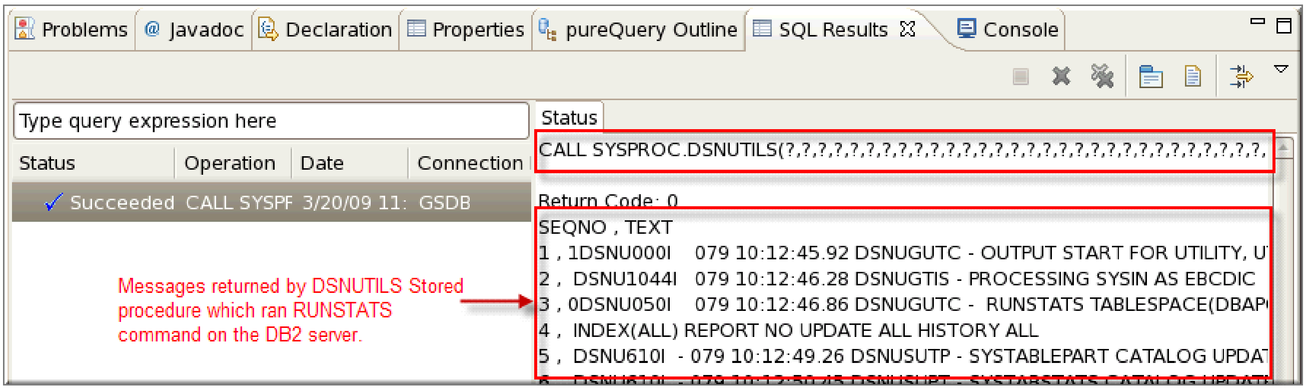


Update statistics on a table

In the *Data Source Explorer, Tables* folder, right click on table *PRODUCT*, then choose: `Update Statistics.`

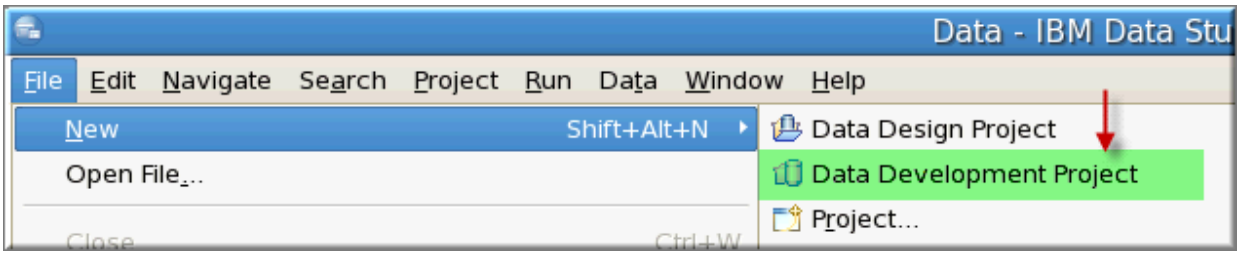


Review the *SQL Results* view to see how this was accomplished. Did you notice a call to `SYSPROC.DSNUTILS` procedure to do the statistics for `PRODUCT` table?

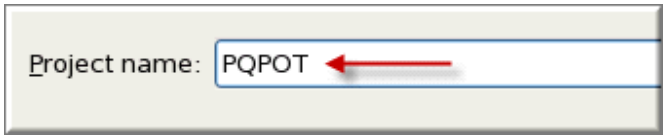


1.5 Create a Data Project

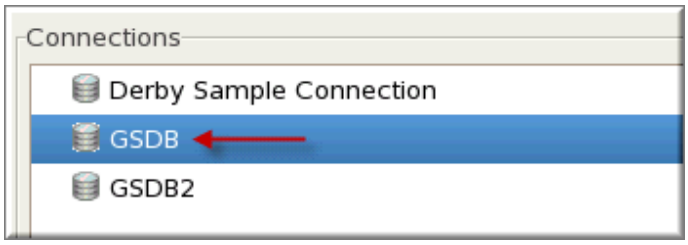
__38. Create a new data project. In the drop down menu at the top of the *Data Studio Developer*, select *File*, then: `New ⇨ Data Development Project`.



a. In <Next> screen, specify *PQPOT* as the project name and *GOSALES* as the schema name and click `<Next>`.

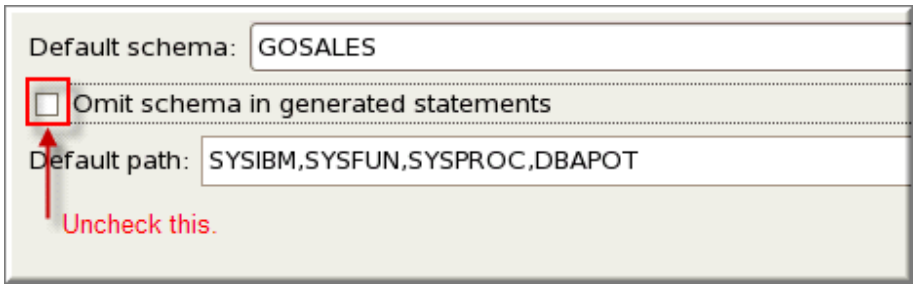


b. Choose the *GSDB* database. `<Next>`

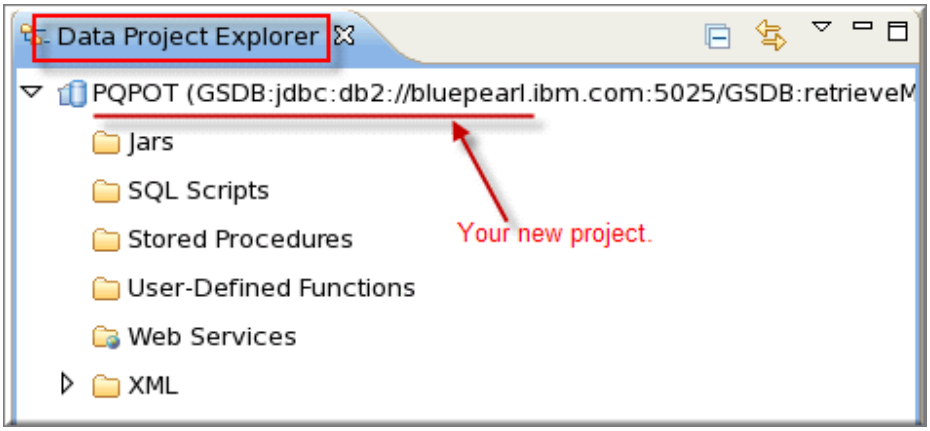


c. In the next screen, keep blank for the package and build owner. Click <Next>

d. Choose *GOSALES* as the default schema and uncheck `Omit schema in generated statements`. Click `<Finish>`

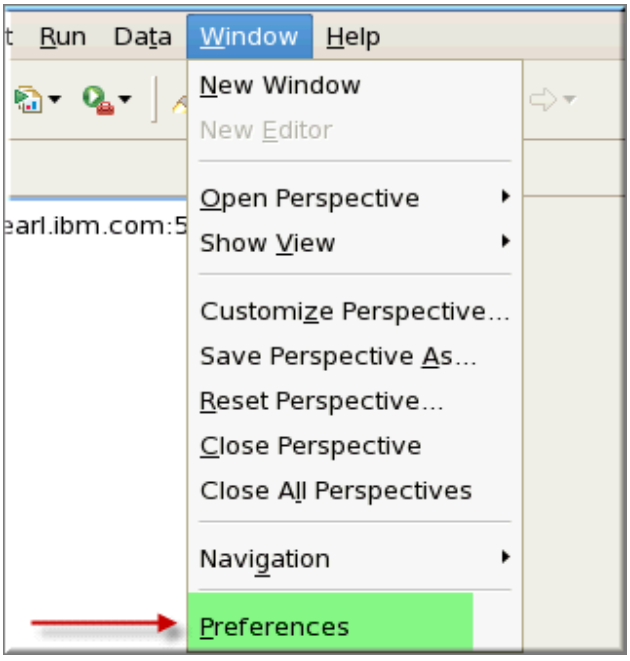


__39. In the top left quadrant of your *data* perspective, you will see your *Data Project Explorer*. Here is where your projects are managed.

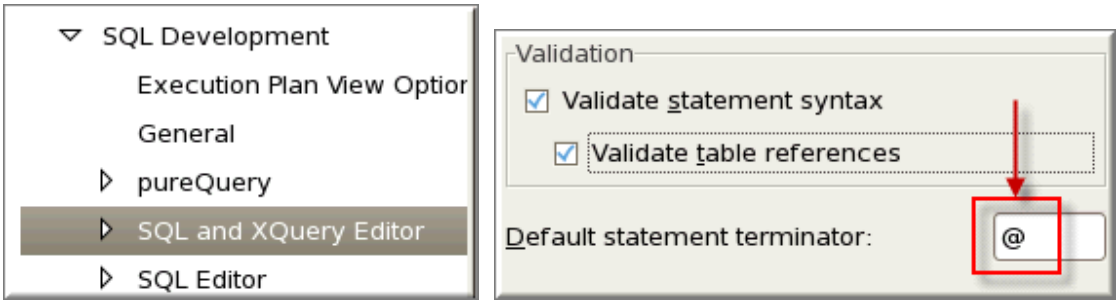


Customize your Editor Settings

__40. Go to the Data Studio Developer drop down menu bar and find *Window*, then choose: `Preferences`



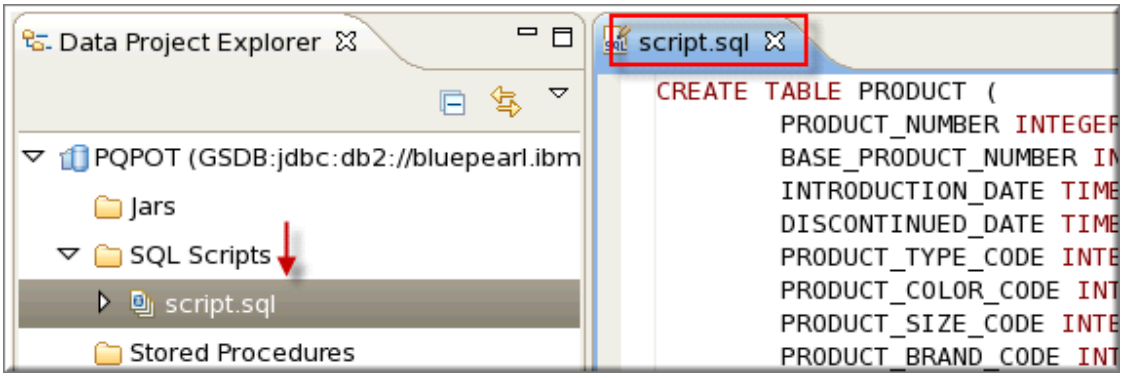
__41. Find the *Data Management* section, then find: `SQL Development ⇨ SQL and XQuery Editor`. Change the Default statement terminator to a `@`. Make sure all validations options are checked.



__42. Click: `<Apply> <OK>`

Generate DDL for a table

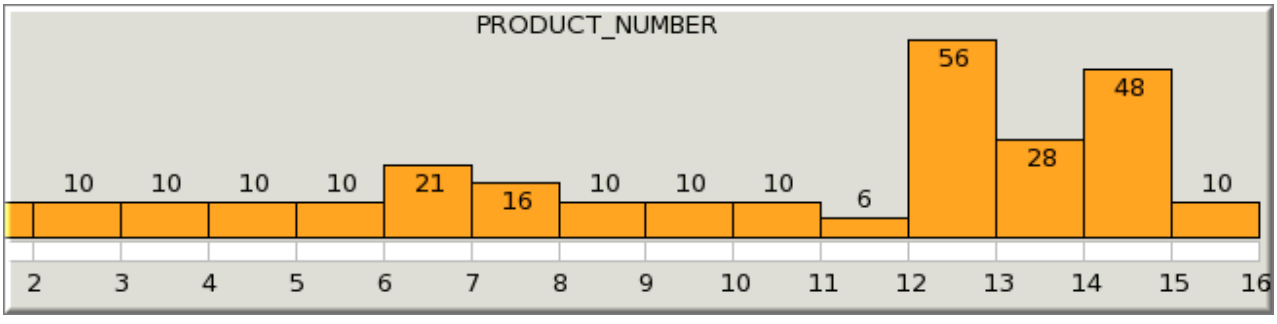
__43. In the *Data Source Explorer, Tables* folder, right click on table *PRODUCT*, then choose: `Generate DDL.` Click `<Next>` 3 times and click `<Finish>`. Double click `script.sql` in `PQPOT` project to open it. Review the DDL generated.



Look at value distributions

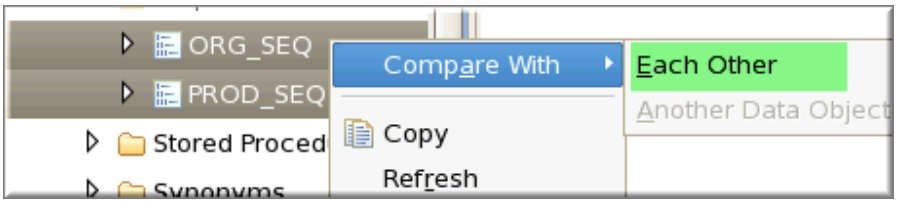
__44. In the *Data Source Explorer, Tables* folder, right click on table *PRODUCT*, then choose: `Value Distributions ⇨ Multivariate`

The output looks like this:

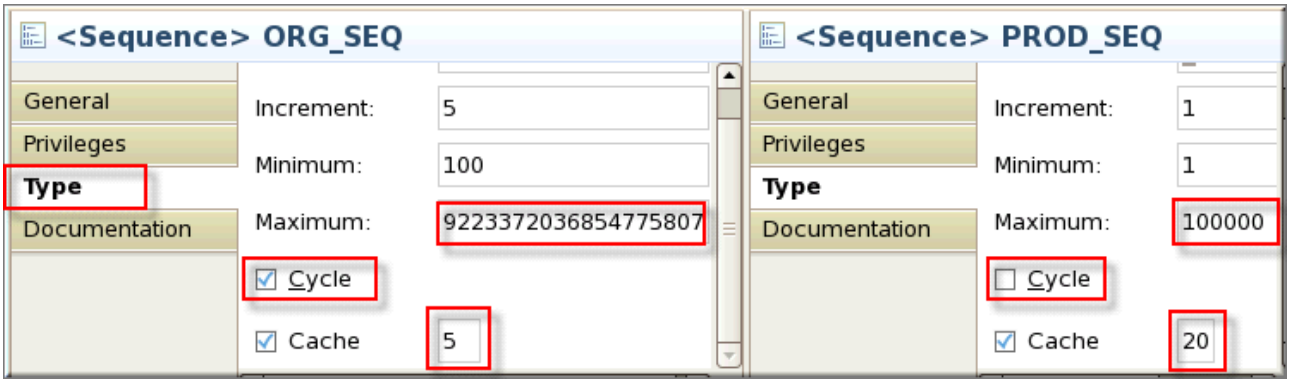


Compare objects

__45. In the *Data Source Explorer, Sequences* folder, right click on both: *ORG_SEQ* and *PROD_SEQ* Then choose: `Compare With ⇨ Each Other`



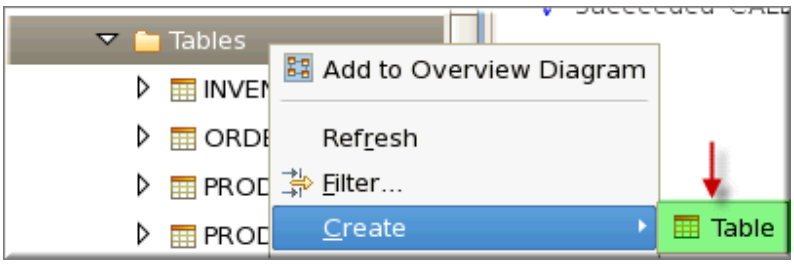
__46. Go to the *Type* screen and notice that the max values for the sequences are very different. Can you figure out why *ORG_SEQ* can be so large? (Hint: generate DDL for them both and see which data types they are defined to.)



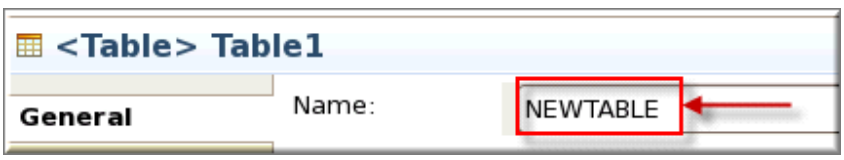
Creating a Table

__47. Close all the editors open by clicking `<CTRL><SHIFT><W>`

__48. Make sure you stay in the *GOSALES* schema in the *Data Source Explorer*, right click on the *Tables* folder itself then: `Create ⇨ Table`

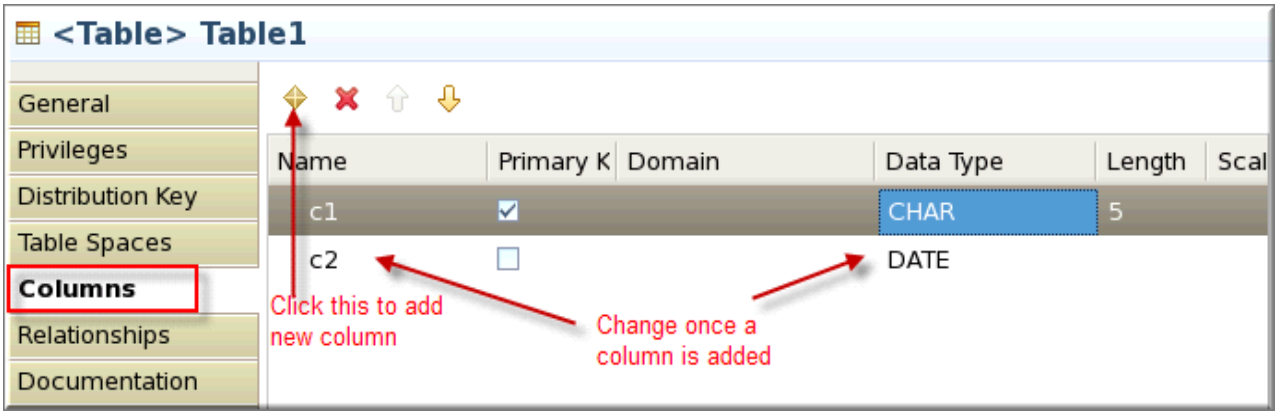


__49. Use the General screen of the *Data Object Editor* to create a table called: *NEWTABLE*

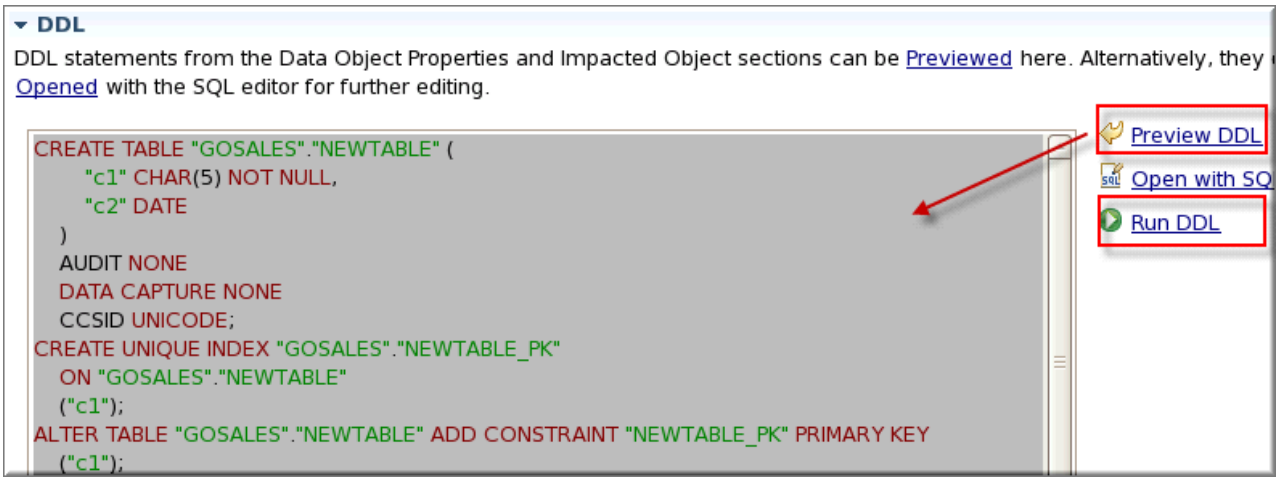


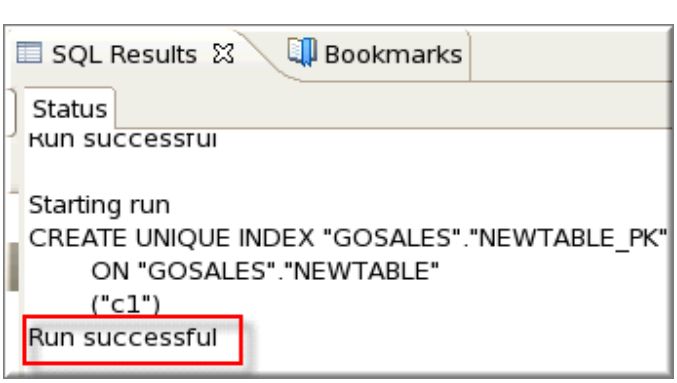
__50. Explore through each screen of the editor to get a feel for how it prompts you to create the table. Every option is available for you to take full advantage of the DB2 CREATE TABLE definition.

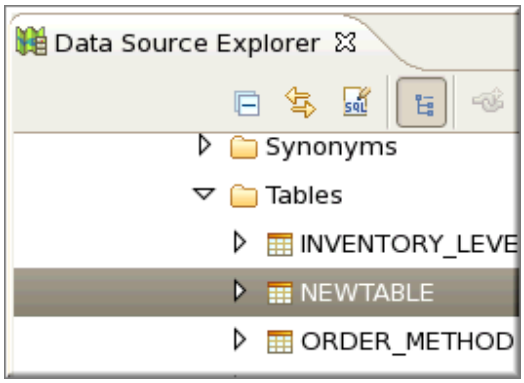
__51. On the columns screen, add a few new columns. It doesn't matter what you call them or what they are, just learn the interface. Below is an example of what you might do:



__52. `Preview` and then `Run DDL`. This same technique can be used to create any database object.







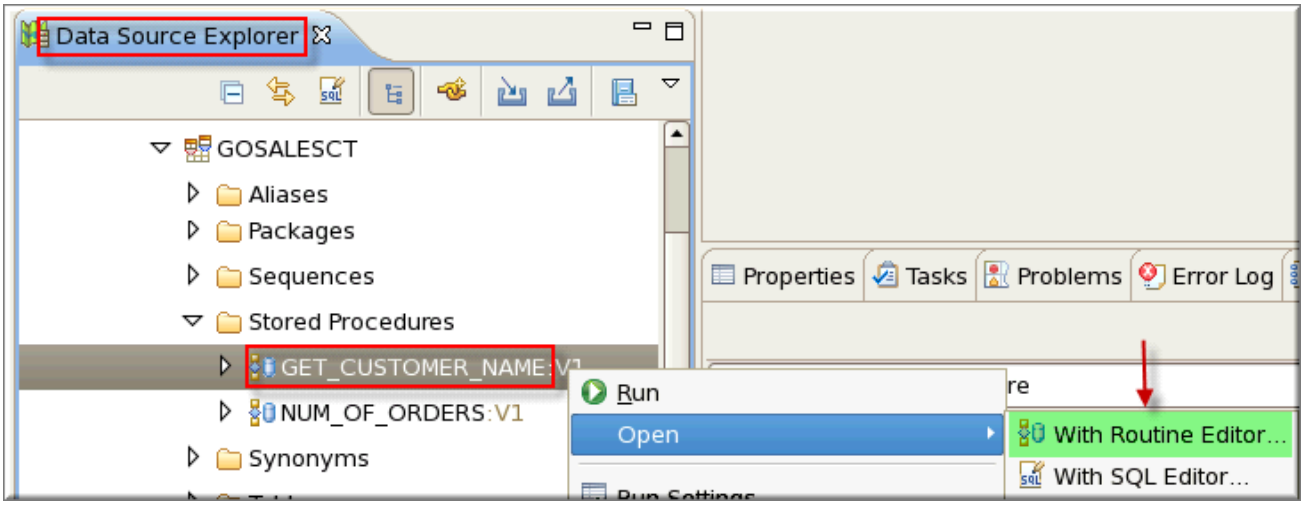
1.6 Debug a Stored Procedure

Bring stored procedure code into your project

__53. Close all the editors open by clicking <CTRL><SHIFT><W>

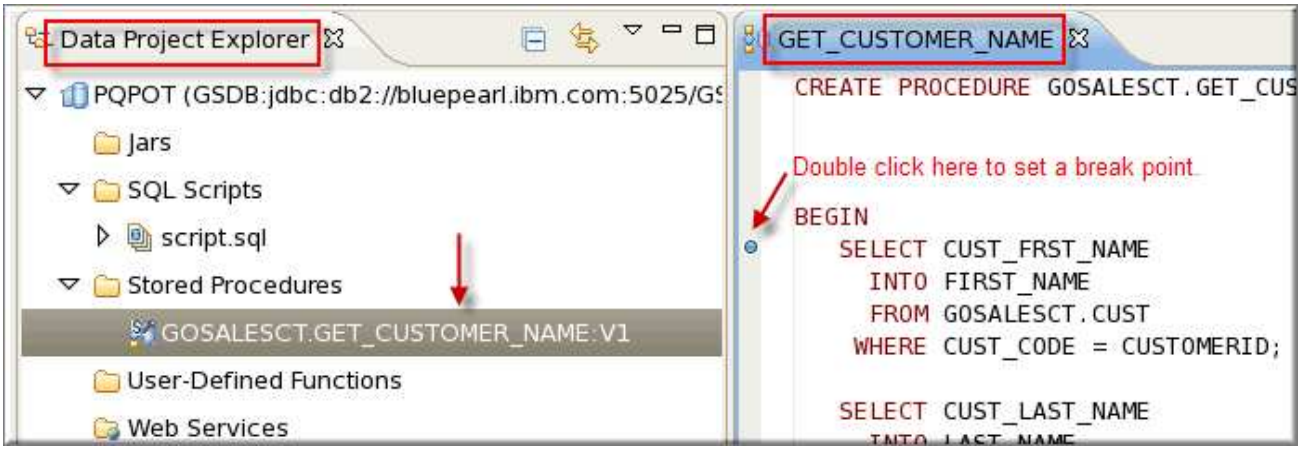
__54. In your *Data Source Explorer*, find the *GSDB* database schema called *GOSALESCT.*

__55. In this schema, find stored procedure *GET_CUSTOMER_NAME.* Right click on it then choose*: Open ⇨ With Routine Editor…*



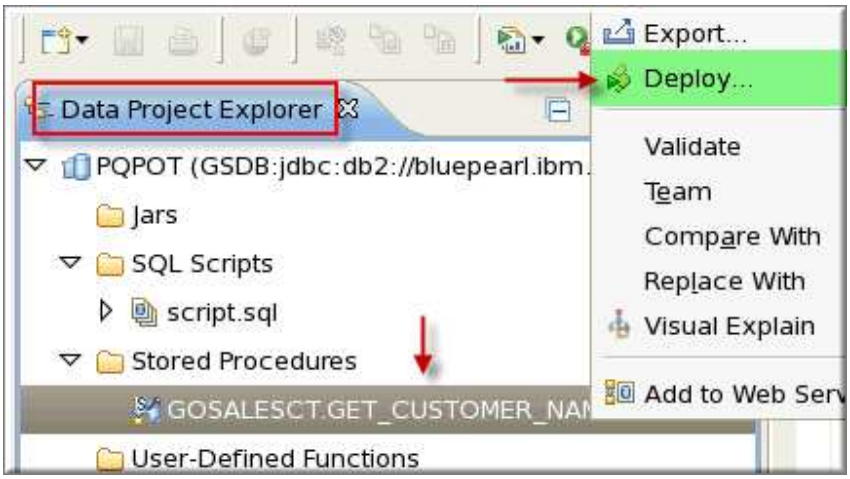
__56. Choose the project you just created (PQPOT) as the target to place this code and click <Finish>.

__57. Notice in the *Data Project Explorer*, the SQL PL for this stored procedure has been placed in your project *PQPOT*. Also, the SQL PL editor has this code loaded and is ready for you to start work with this code.



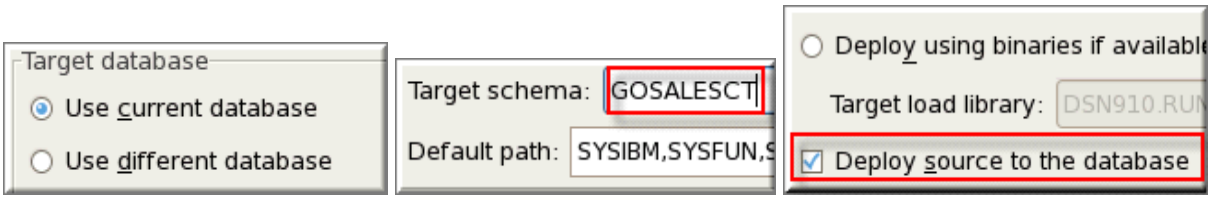
Debug your stored procedure

__58. In your *Data Project Explorer (*Please note: Not *Data Source Explorer)*, find the stored procedure *GET_CUSTOMER_NAME* again. Right click on it then choose: `Deploy`.

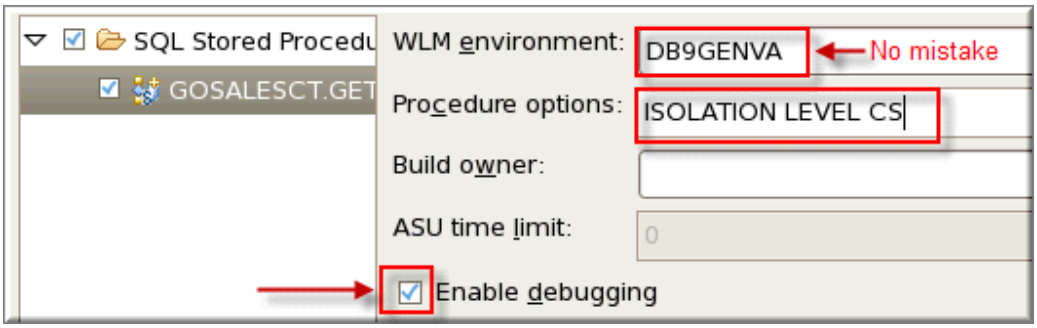


__59. In the *Deploy Routines* assistant, the *Deploy Options* screen, do the following:

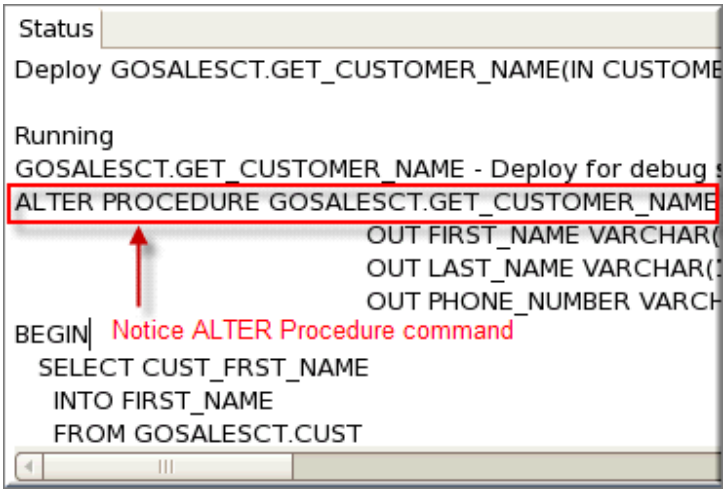
Select: `Use current database`.
Select: `Target Schema: GOSALESCT`.
Check: `Deploy source to the database`



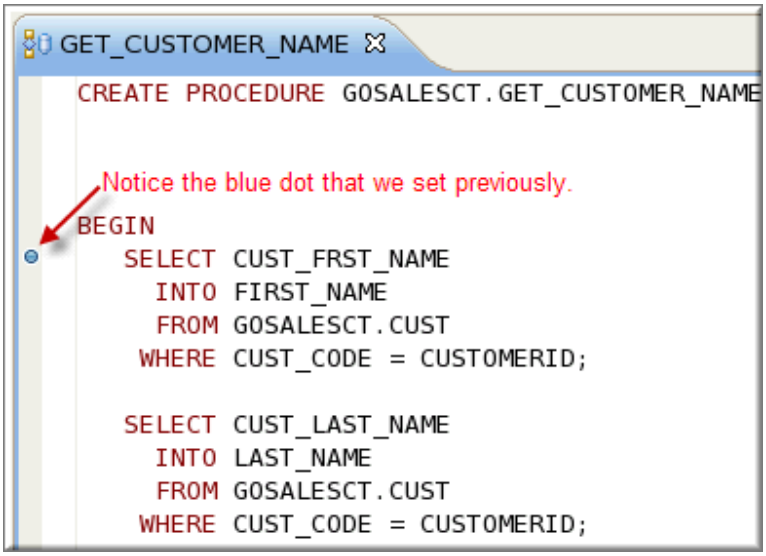
__60. In the next *Routine Options* screen, make sure you check: `Enable debugging.` Then `<Finish>.` You will now be able to debug this stored procedure from the *Data Project Explorer*.



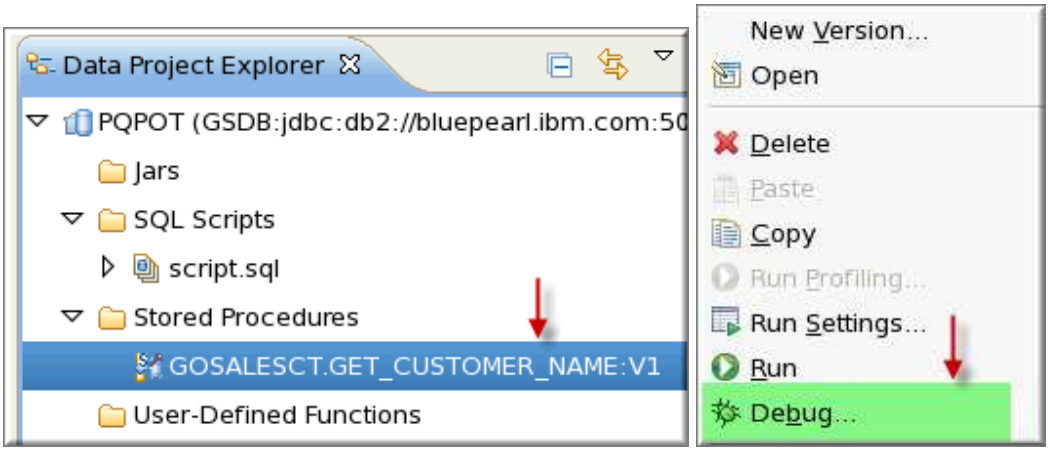
__61. In your SQL Results window, you should see the message about successful deployment of the stored procedure for debugging. Scroll your window to the right and you should see that the procedure was altered for debugging to use DB9GENVA WLM (Work Load Manager) application environment.



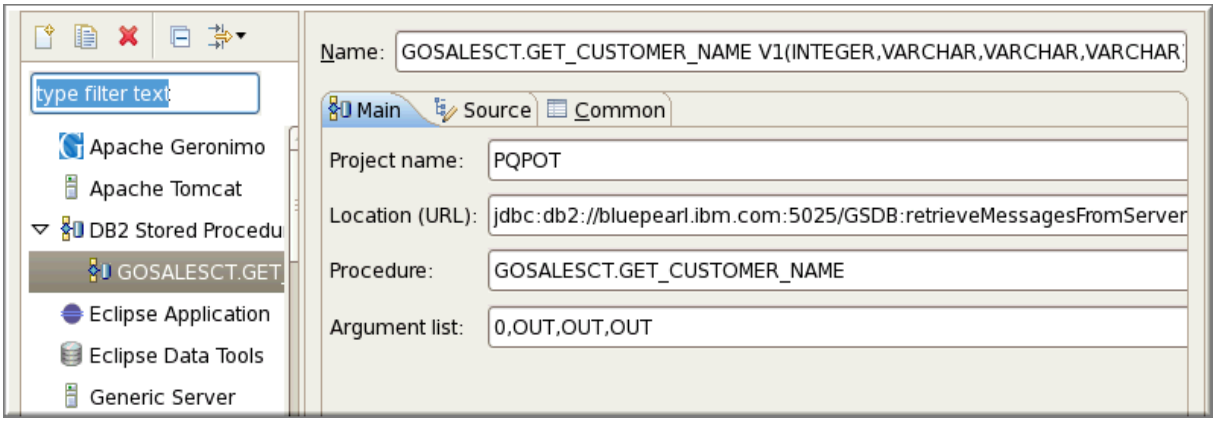
__62. Next, set any breakpoint you might need in the SQL editor itself. Do this by double clicking on the yellow boarder to the left of your code. A blue breakpoint dot will appear.



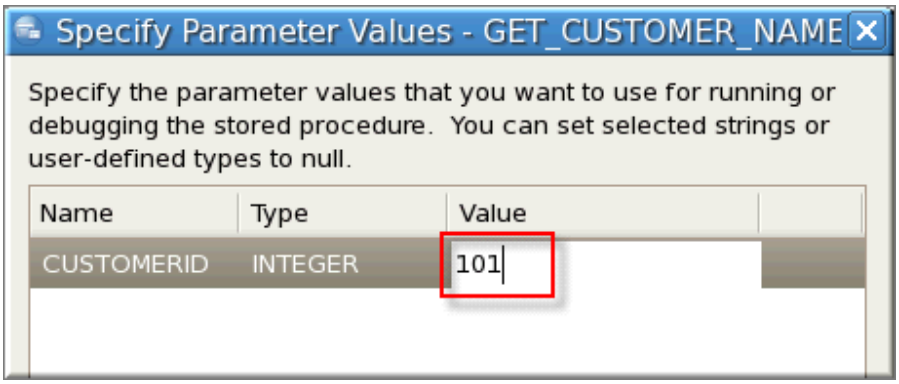
__63. Next, right click on *GOSALESCT.GET_CUSTOMER_NAME* stored procedure in *Data Project Explorer* view and choose: `debug`.



__64. In <Next> window, accept the default and click on Debug button. It may take a while to start processes on the server so please wait for the next window to appear.

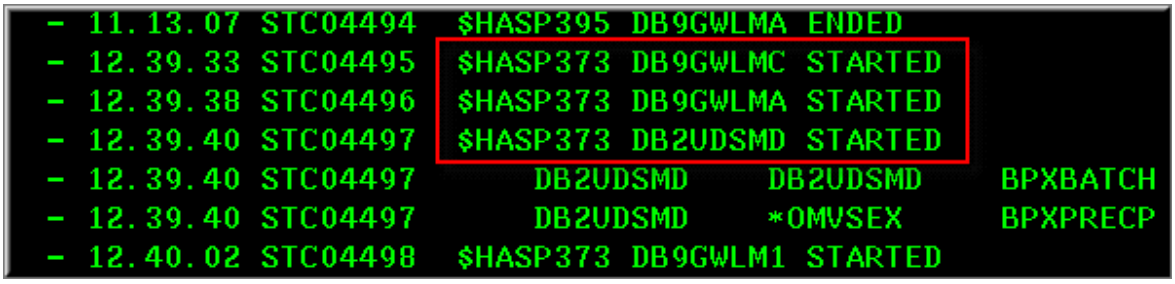


__65. Since this stored procedure expects an input parameter, you will see a window asking for a value for the *CUSTOMERID* parameter. Specify a value of *101* and ***do not*** click on <OK> yet. Please go to the next step



__66. Press <CTRL><ALT><Right Arrow> key to bring Linux Display #2.

__67. We have our z/OS master console on this display. Look at the master console and you will notice that DB2UDSMD has started. This is the server session manager configured on the z/OS for DB2.

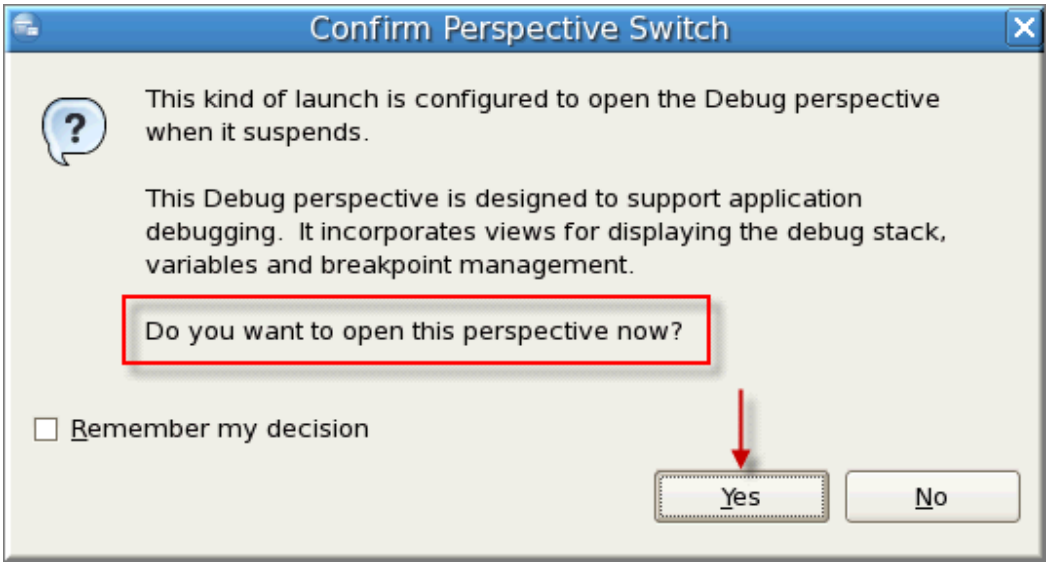


Note: You will need to follow steps given in IBM Redbooks® **DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond.** Refer to the chapter 28.2.2 for setting up unified debugger in your environment. In absence of the server session manager, you can also use local session manager from your workstation by launching dsdbgm.sh or dsdbgm.cmd from dsdev/bin directory in the data studio package.

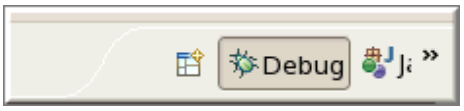
__68. Press <CTRL><ALT><Left Arrow> key to bring Linux Display #1.

__69. Now, you hit <OK> in your previous dialog box to do the debugging of the SP.

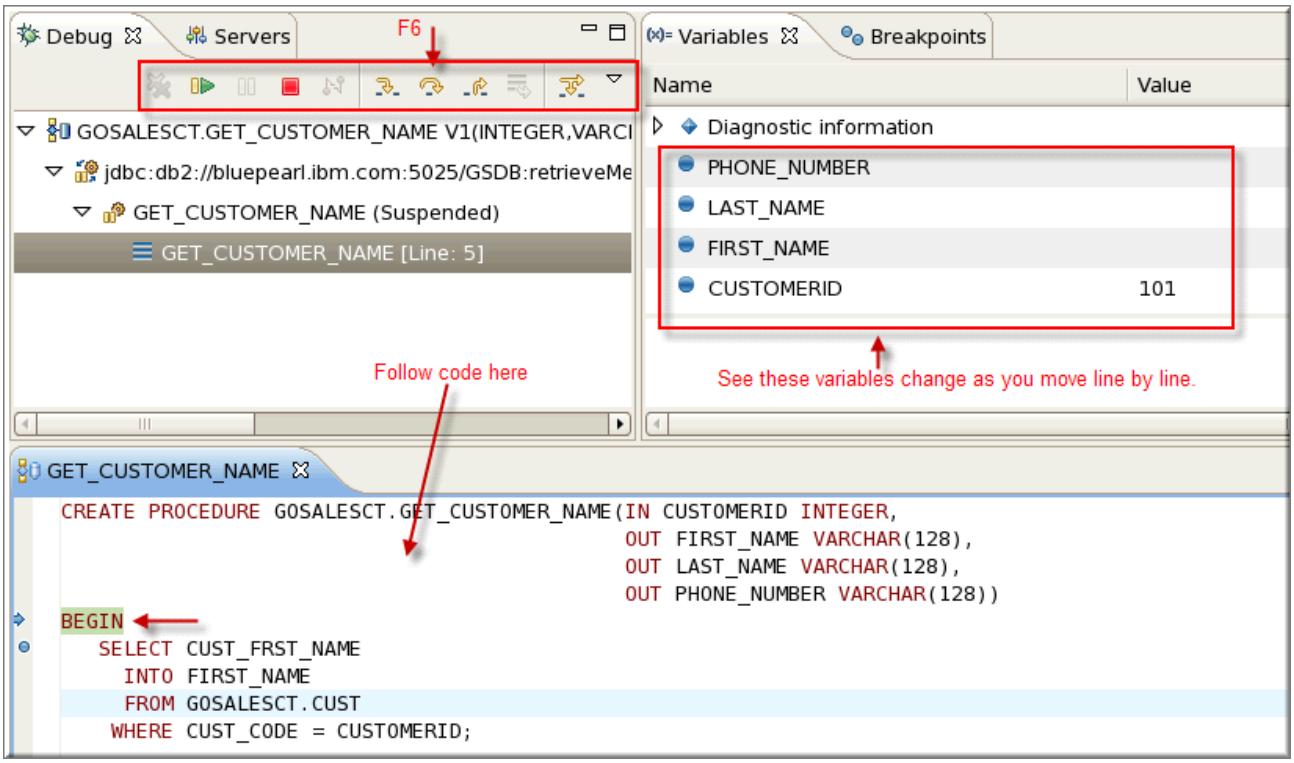
__70. *Data Studio Developer* is now doing something you haven't seen before. It is switching perspectives. It will now open the *Debug* perspective if you let it. Say *Yes* to open this perspective.



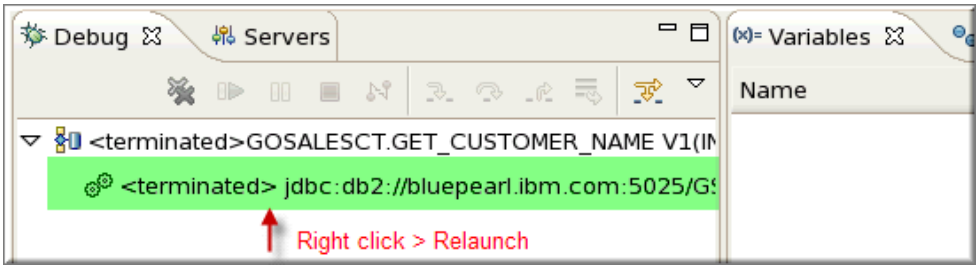
__71. Look in the upper right hand corner of the screen and notice you now have a *Debug* and a *Data* perspective opened. You can switch between them now if you want to. Stay in the debug perspective for the rest of this exercise.



__72. Go to *Debug* view in the *Debug Perspective* and either press *F6* to go from line to line or use the icon shown below to "*Step Into*" the code. Watch the variables change as you go through you code. Learn to use *F6* and *F7* for "*Step Return"* and "*Step Over"*. The code we are working with here is fairly simple, but try to imagine a very large many-lined stored procedure you can debug.



__73. If you run all the way through the code and the session is terminated, then just right click on the terminated session itself and choose: `relaunch`.



__74. Close the debug perspective when done.

**** End of pureQuery z/OS lab 1: Introduction to Data Studio Developer**

Lab 2 - Create pureQuery Project

Introduction:

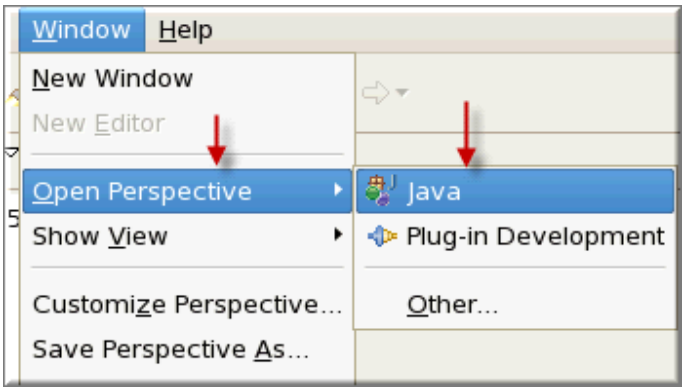
During this lab you will create a new Java project using IBM Data Studio Developer. You will enable a Java Project for pureQuery.

2.1 Creating a new Java Project

__1. Close all open files by clicking <CTRL><SHIFT><W>

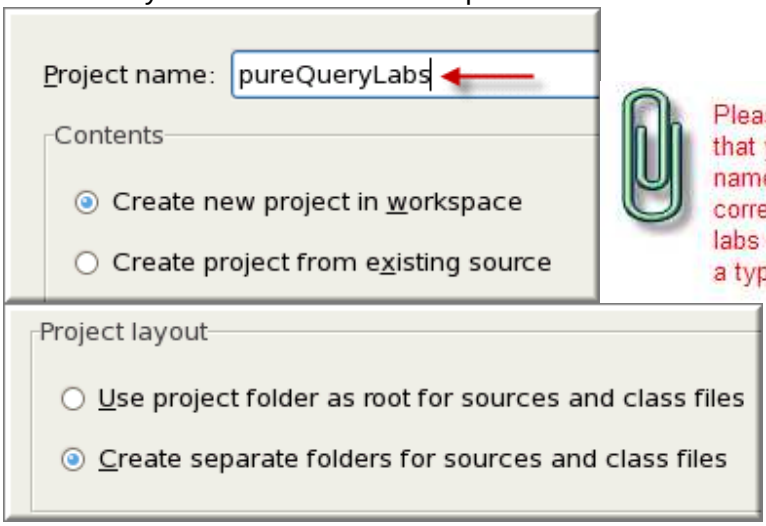
__2. Open *Java Perspective.* `Window ⇨ Open Perspective ⇨ Java`

You should now see:



__3. Now create a new Java project by going to:

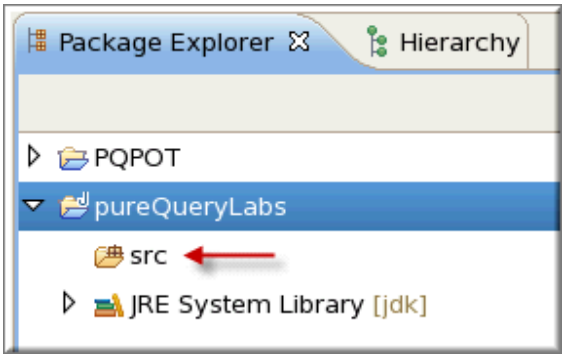
- `File ⇨ New ⇨ Java Project.`
- Name the project `pureQueryLabs`. Make sure options are chosen as shown below, so make sure you click on `Create separate source and output folders`:



Please make sure that you type the name of the project correctly. The later labs will fail if there is a typo in the name.

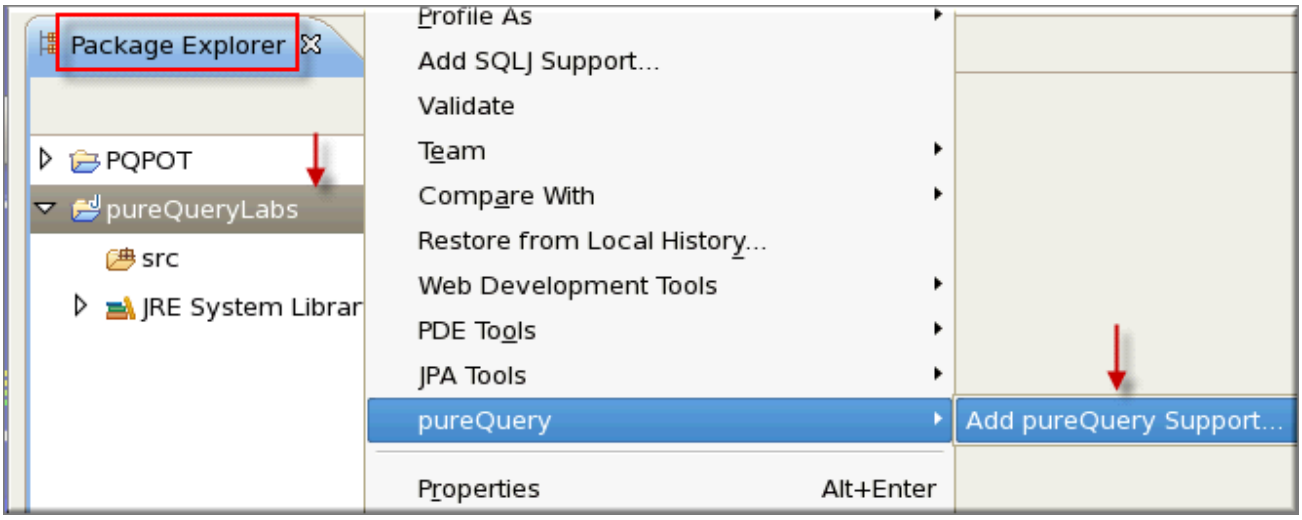
- Click <Next>.
- Click <Finish> to create the Java project.

- The newly created project `pureQueryLabs` along with the source folder, `src`, will now appear in the Data Studio in the `Package Explorer`:

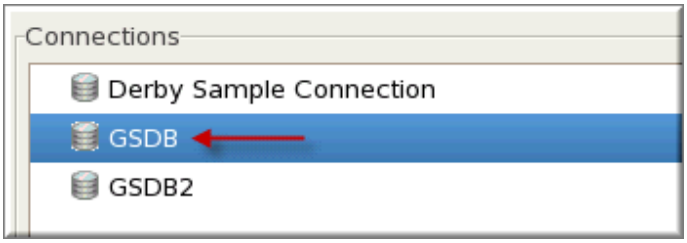


2.2 Enable Java project for pureQuery

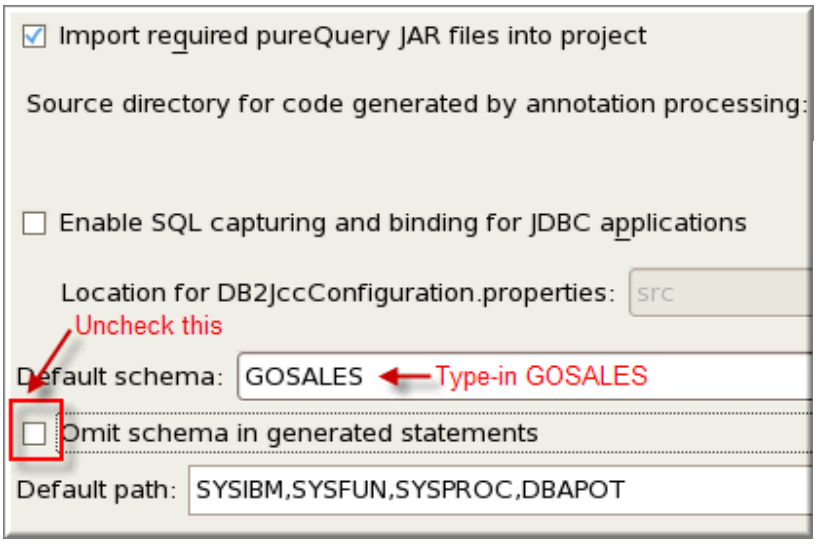
__4. Right click on pureQueryLabs Java project in Package Explorer view and click on `pureQuery` and click again on "`Add pureQuery Support`".



__5. Select GSDB database and click `<Next>`

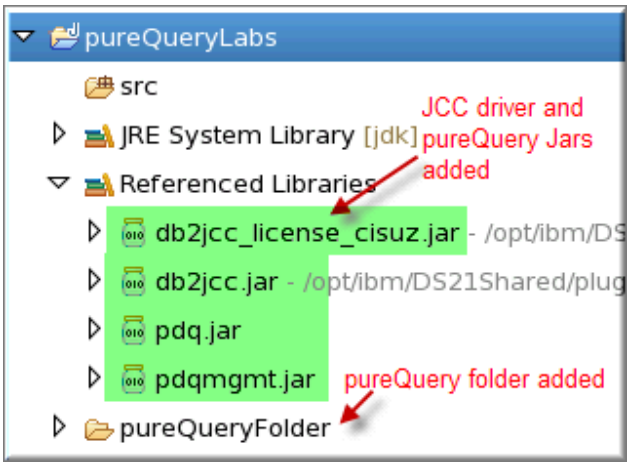


__6. Check default schema GOSALES and click <Finish>.



Please make sure that you type-in name of the schema GOSALES correctly. Later labs will fail if there is a typo in the name

__7. You should now see pdq.jar, pdqmgmt.jar, db2jcc.jar and db2jcc_license_cisuz.jar files added to the referenced libraries. A new pureQueryFolder directory is also added to the project. The Java project is now enabled with pureQuery support.

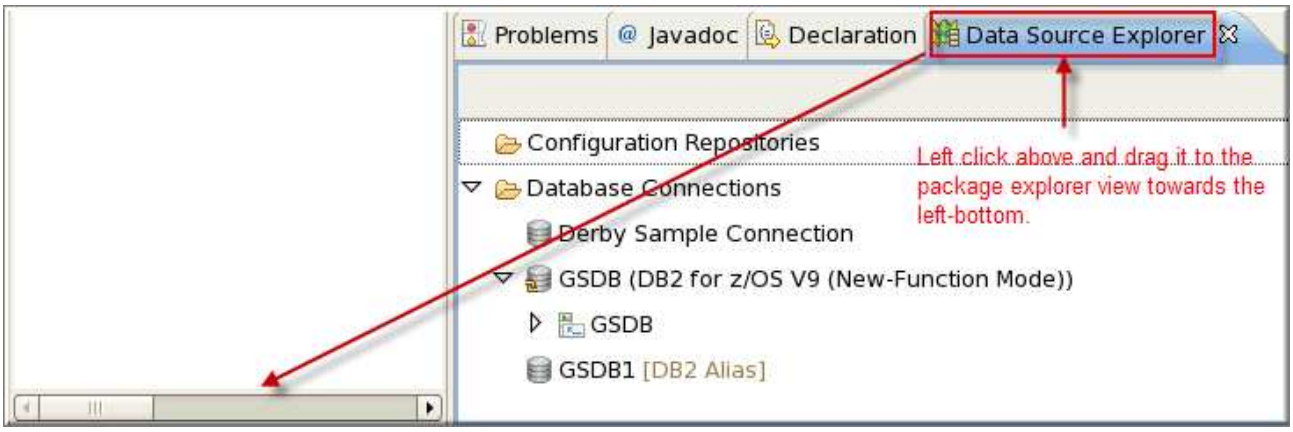


2.3 Enable Data Explorer View in Java project

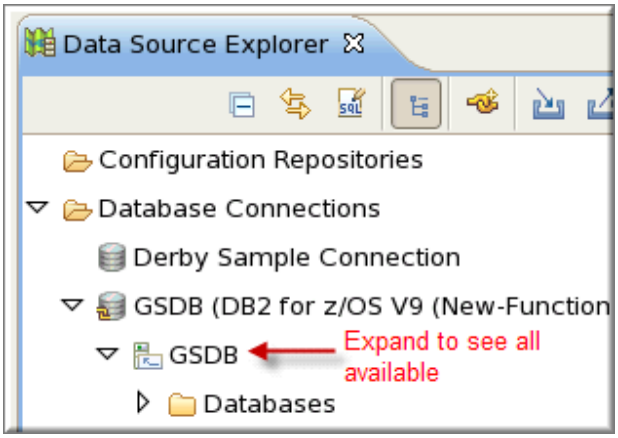
__8. Now we want to add the *Data Source Explorer* View to this perspective. To do this we will drag and drop this view to the package explorer.

__9. **Drag and drop**: Inside perspectives, we can drag and drop the various views to be placed where they are the most helpful to us. To drag and drop a view, left click on the tab of that view, hold down the mouse button and drag the view to where you want it to be within that perspective.

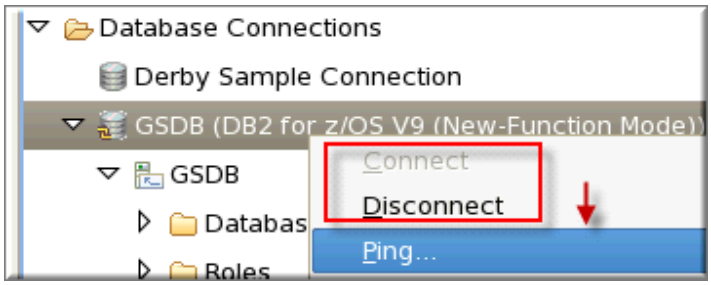
__10. Drag and drop the *Data Source Explorer* View right below the Package Explorer for a better view of the connections. The *Data Source Explorer* View will show all the available connections, if any, of your default database.



__11. Expand the database connections to see all that are available to you



__12. If you are not connected, connect to the GSDB database by right-clicking on it in the *Data Source Explorer* and choosing `Connect`. You can also ping the database without connecting.



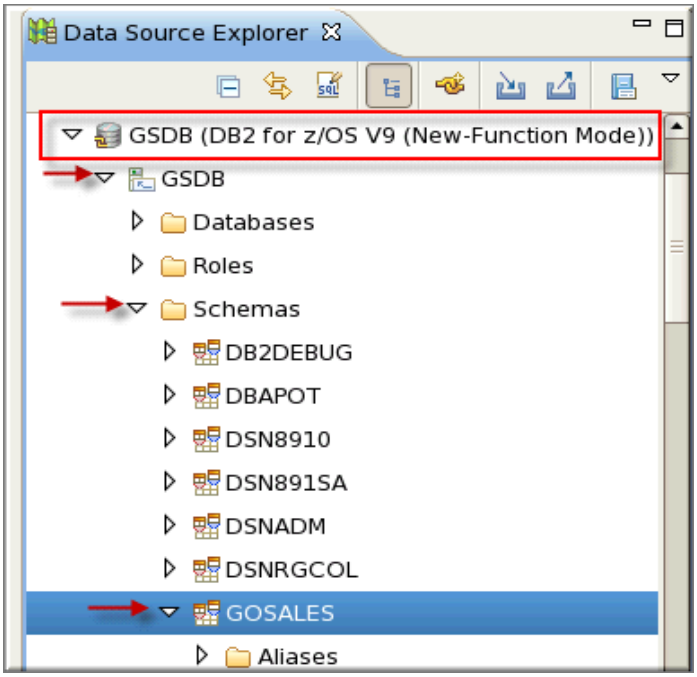
You are now ready to begin working with pureQuery code from the GSDB database.

**** End of lab 2: Create pureQuery Project**

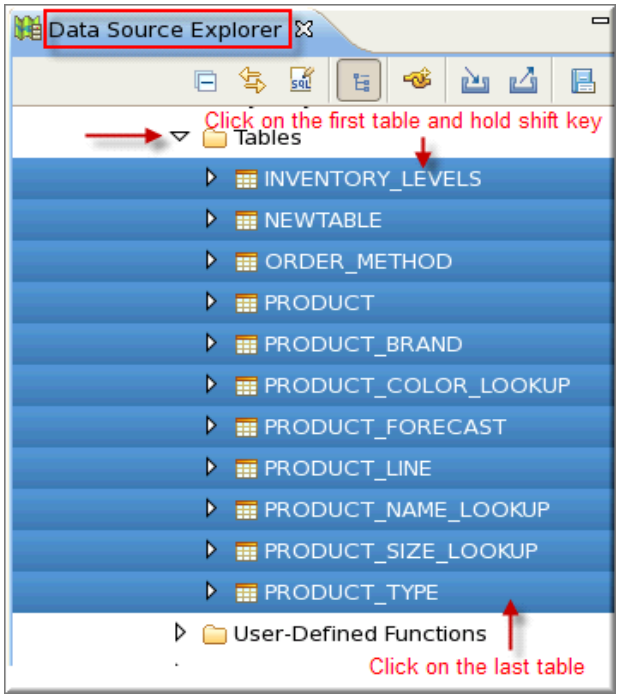
Lab 3 - Explore pureQuery Tools

3.1 Generate pureQuery code from database tables

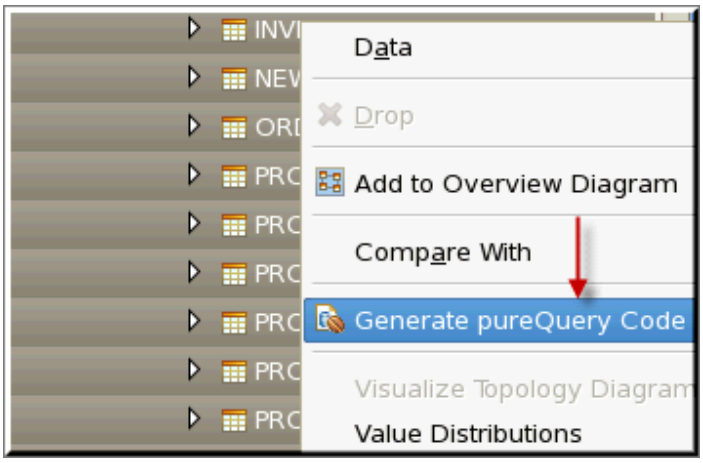
__1. Expand database GSDB, then expand Schemas, then GOSALES, and finally Tables.



__2. Select all tables. (Click on the first table, hold the shift key and click on the last table).

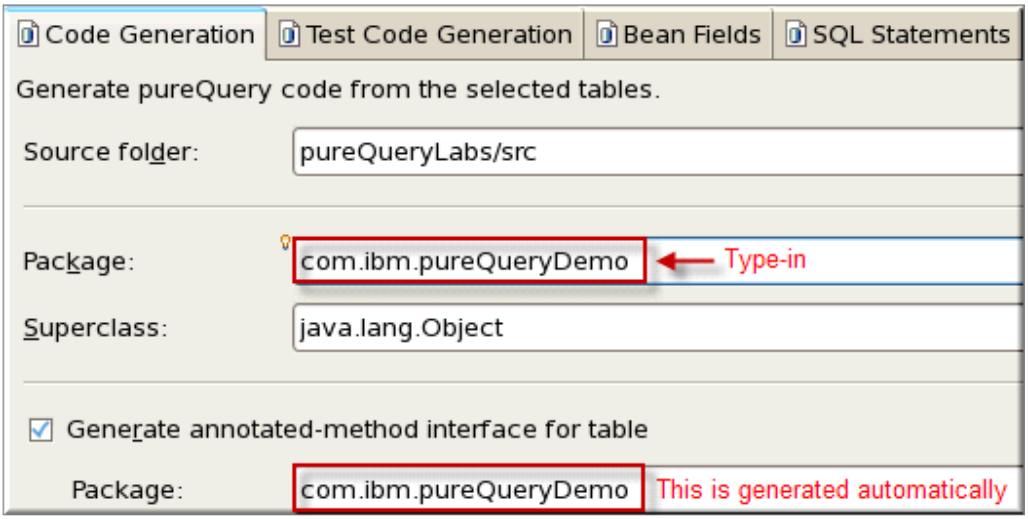


__3. Right click on any one of the selected tables and click on `Generate pureQuery Code`.

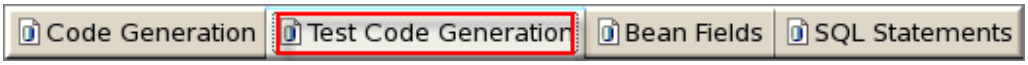


__4. The Generate pureQuery Code for a Table window will open.

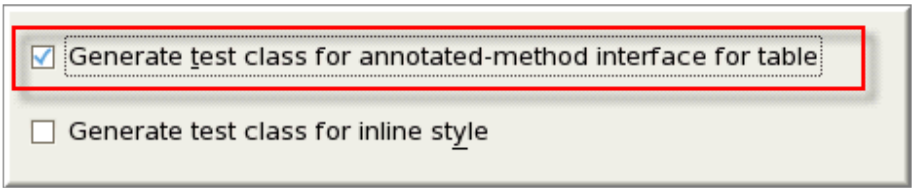
- Specify name of the package as `com.ibm.pureQueryDemo`.



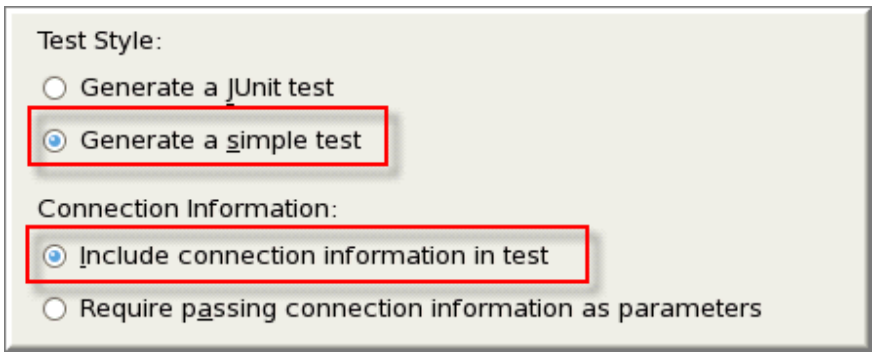
- Click on the `Test Code Generation` tab.



- Check option to `Generate test class for annotated-method interface for table`.

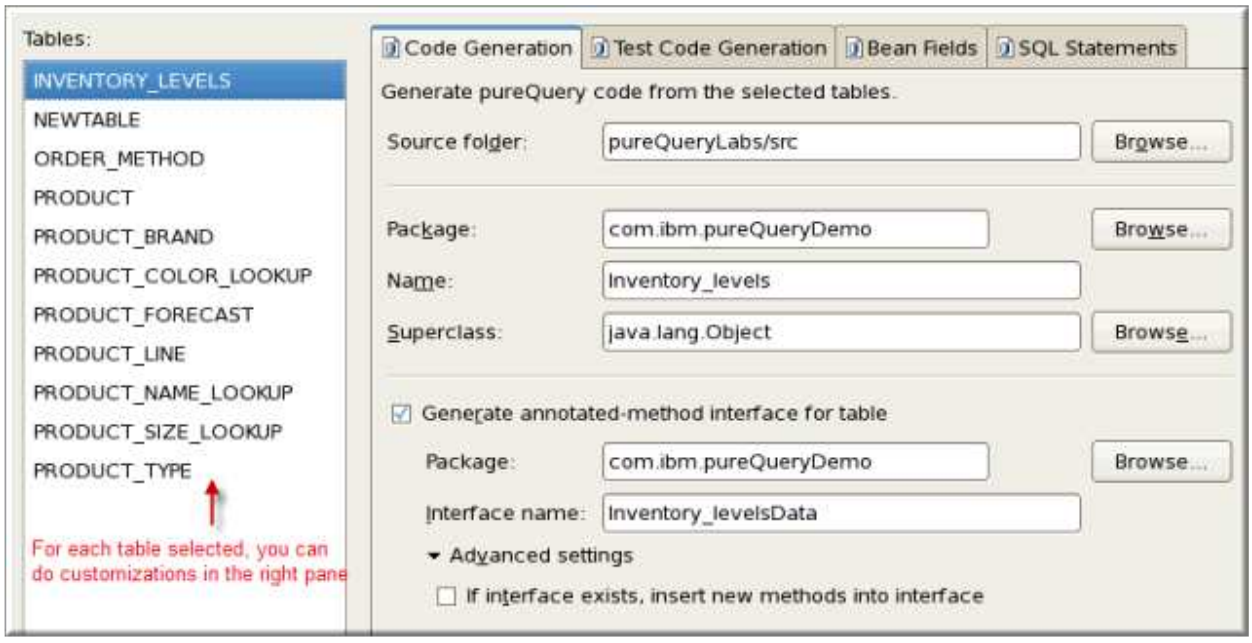


- Make sure that you have selected option for `Include connection information in test`.



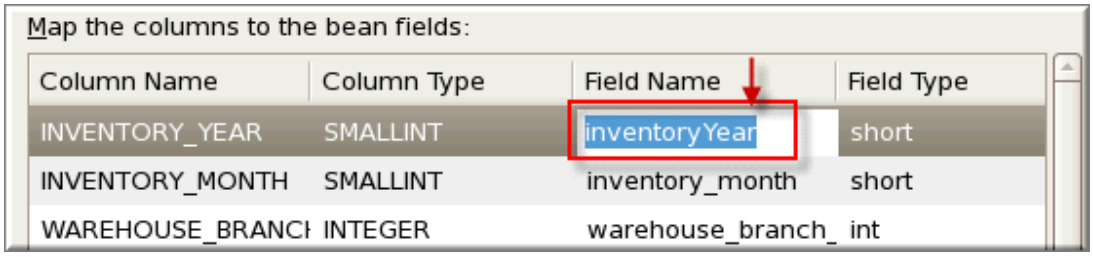
- Click `<Next>` button to go to the next screen.

- In this screen, you will see selected tables on the left hand side and same options shown above on the right pane. Here you can do customizations for each table for the Java source code that will be generated.



- Browse through each of the tab to see available options.

- Go to the *Bean Fields* tab and you will notice that you can map database column names to the java attributes as per your choice. We will map column `INVENTORY_YEAR` of the `INVENTORY_LEVEL` table to the Java bean attribute name `inventoryYear`.



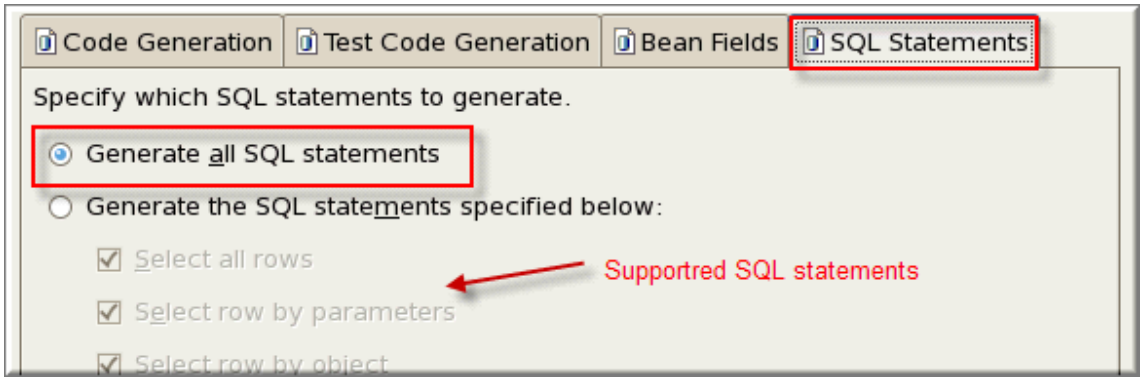
- The `inventoryYear` is mapped to the database column `INVENTORY_LEVEL .INVENTORY_YEAR` by adding the following annotations to the variable and to the setter and getter methods. (The following is an example of that mapping which will happen after you are done doing this step):

```
@Column(name="INVENOTORY_YEAR") protected short inventoryYear;
```

…and…

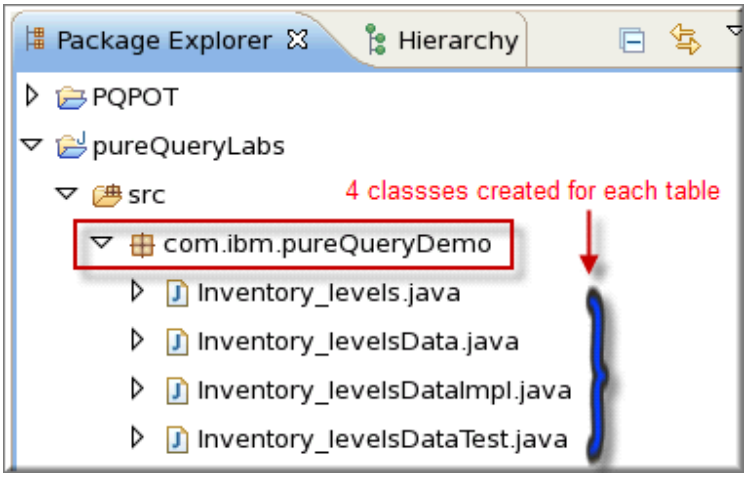
```
@Column(name="INVENOTORY_YEAR") public String getInventoryYear() {
return inventoryYear;
}
```

- Browse through last tab of `SQL Statements` and notice the type of statements supported for which code generation will happen.



- Now click `<Finish>`.

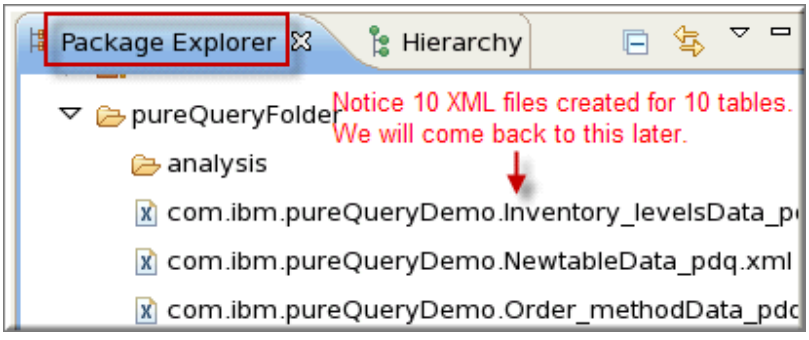
__5. Notice that a java package `com.ibm.pureQueryDemo` has been created with 40 classes for 10 tables selected in the previous step. There are 4 classes for each of the table.



__6. The following four classes have now been created for each table in the `pureQueryLabs src` folder under package `com/ibm/pureQueryDemo`.

- **Inventory_levels.java** — The Java file containing a one to one mapping from the data in the `INVENTORY_LEVELS` table to the Java object.
- **Inventory_levelsData.java** — An interface containing the abstraction of the data access layer for the querying of data or data manipulation.
- **Inventory_levelsDataImpl.java** — The implementation of the interface created above.
- **Inventory_levelsDataTest.java** — Sample class on showing pureQuery's functionality using the method-style.

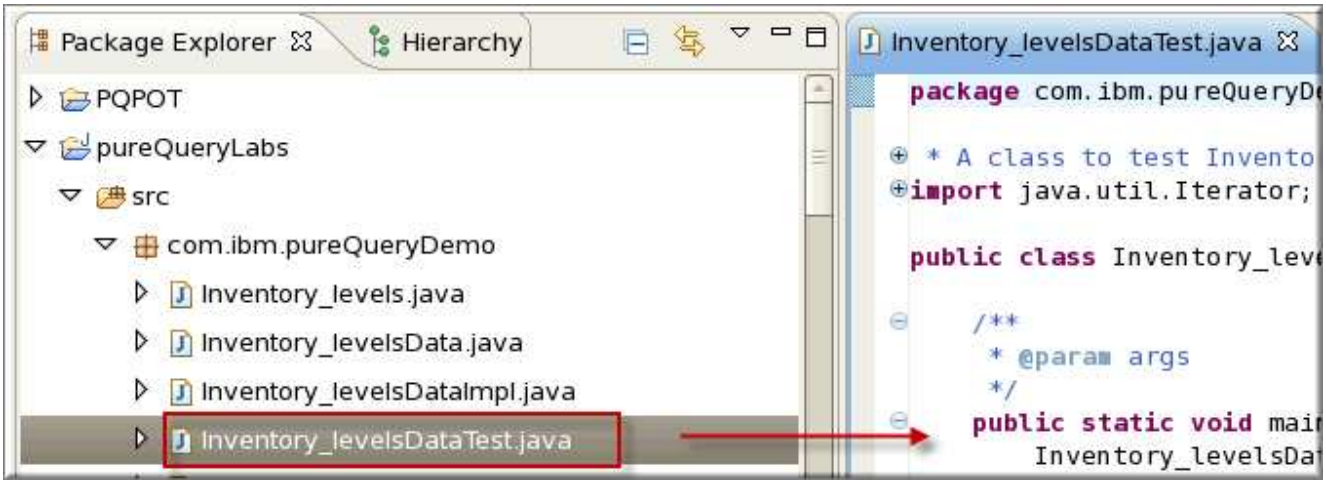
__7. Notice several XML files created for each table. We will come back to these later.



3.2 Quick overview and running the pureQuery Test Classes

The tool generated a test class for each table. These test classes are created to give the developer quick code samples of how to create a connection, create a bean instance or create a call method from the interface.

__8. Select the `Inventory_levelsDataTest.java` file:



__9. Description of the Class – a review:

- The *main(String[] args)* method expects to be passed one argument: the password to the database. If no arguments are passed, it will print to the console: "`All required arguments were not provided.`"

```
if (args.length < 1) {
    SampleUtil.println("All required arguments were not prov
    return;
}
```

- The class contains the code instantiating an object to call the methods defined in the `Inventory_levelsData` interface. This object has the `Inventory_levelsData` class, the connection string to the database and the username passed as arguments. The password will be passed as an argument when running the class.

```
data = SampleUtil
        .getData(
                Inventory_levelsData.class,
                "jdbc:db2://bluepearl.ibm.com:5025/GSDB:retrieveMessa
                "dbapot", args[0]);
((Data) data).setAutoCommit(false);
```

- Developers have control over the connection auto-commit mode so that the transactions may be committed individually, automatically or explicitly using `commit()`. The following line sets the connection auto-commit mode to false:
- Now the method, declared on the `Inventory_levelsData` interface to retrieve all Inventory Levels `getInventory_levelss()`, is called and its' return object is assigned to the `getInventory_levelss Iterator`. It then checks if any records were returned by

trying to retrieve the first element in the `Iterator`. If the `Iterator` is empty, it outputs "`result set is empty`," and does a rollback and stops executing the sample program. If the `Iterator` is not empty (there was at least one record returned) it assigns that record to the object `bean` of type `Inventory_level`.

```
Inventory_levels bean = null;
if (getInventory_levelss.hasNext()) {
    bean = getInventory_levelss.next();
    ((ResultIterator<Inventory_levels>) getInventory_levelss)
            .close();
} else {
    SampleUtil.println("Result set is empty.");
    ((Data) data).rollback();
    return;
}
```

- The following code deletes the bean that was retrieved in the previous example. An integer is returned with the number of records that were affected by the transaction.

```
Integer deleteInventory_levels = data.deleteInventory_levels(bean);
SampleUtil
        .println("Results for deleteInventory_levels(bean):  Deleted "
                + deleteInventory_levels.toString() + " rows");

data.createInventory_levels(bean);
```

- Finally, the `Inventory_level` deleted in the previous example is recreated, retrieved and its information is printed to the console.

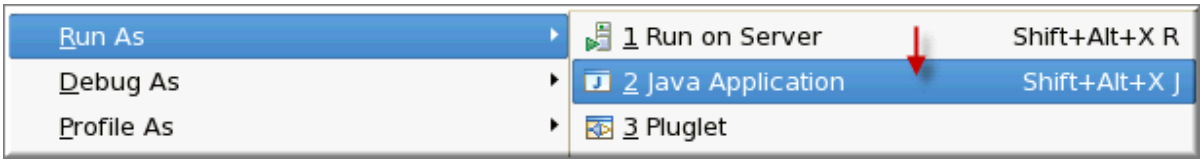
```
getInventory_levels = data.getInventory_levels(bean);
SampleUtil.println("Results for createInventory_levels(bean)");
SampleUtil.printClass(getInventory_levels);
```

- All the transactions are committed in the last statement.

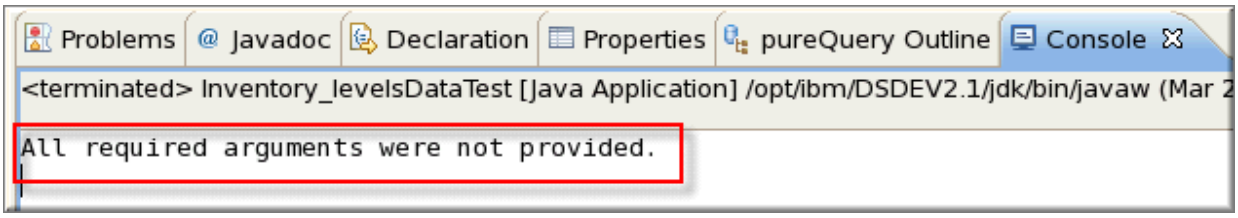
```
((Data) data).commit();
```

__10. Now that we understand what it is doing, we will run this test Class:

- Right-click anywhere on the `Inventory_levelsDataTest.java` class and select: `Run As ⇨ Java Application.`

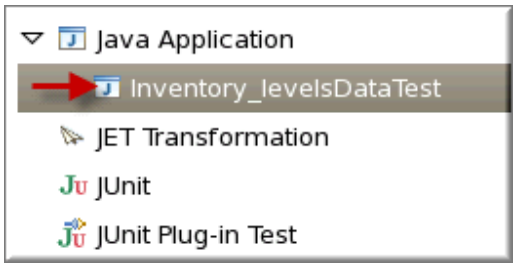


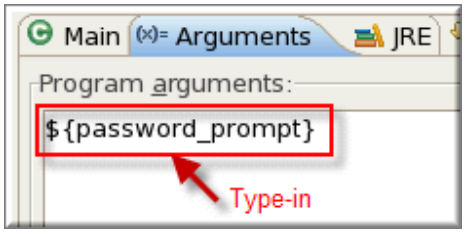
__11. You will notice "`All required arguments were not provided.`" in the console. It was expected since we did not specify the argument while running this test class.



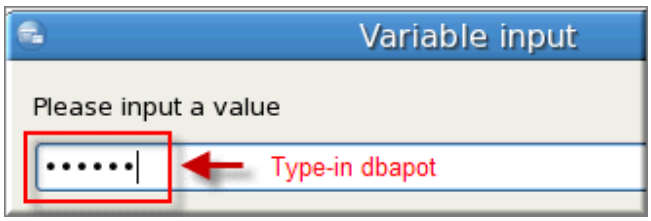
__12. Again right click anywhere on the `Inventory_levelsDataTest.java` class and select `Run As ⇨ Run Configurations.` (Please look at exhibit in item # 9.) This will open a *Run Configurations* dialog and select *Inventory_levelsDataTest* in the left hand side pane and click on Arguments tab on the right hand side pane to provide password as program arguments.

- Type in a variable name `${password_prompt}`

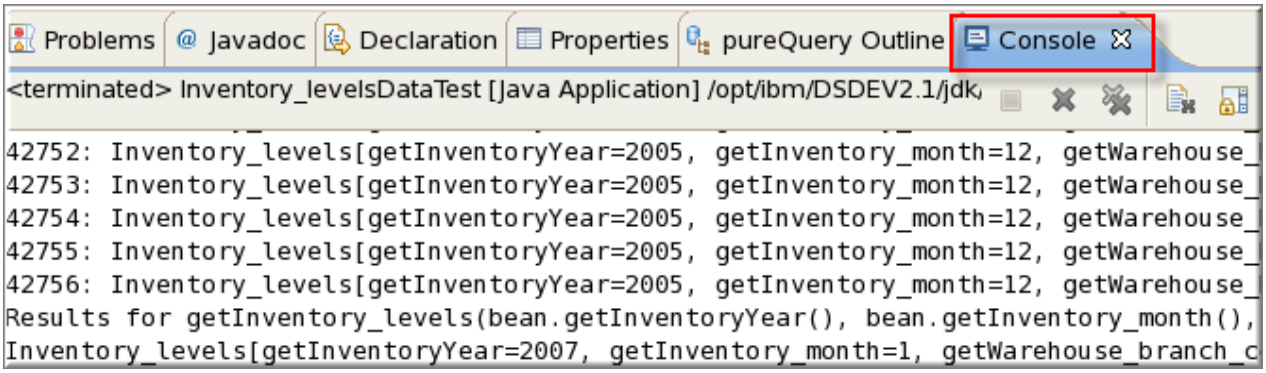




- Click <Run> and specify `dbapot` password.

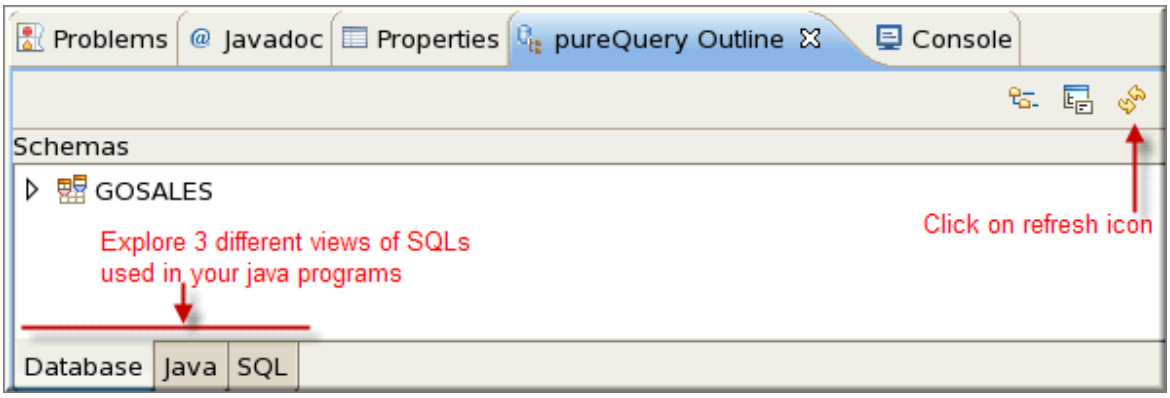


- You will see the results on the Console:

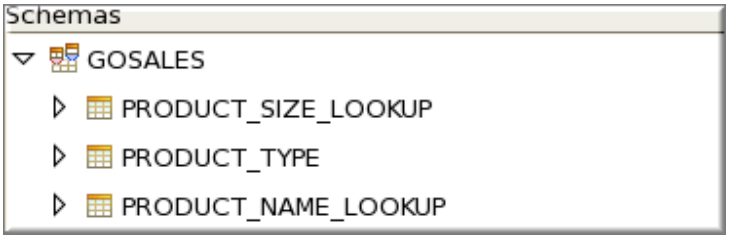


3.3 Explore pureQuery outline view

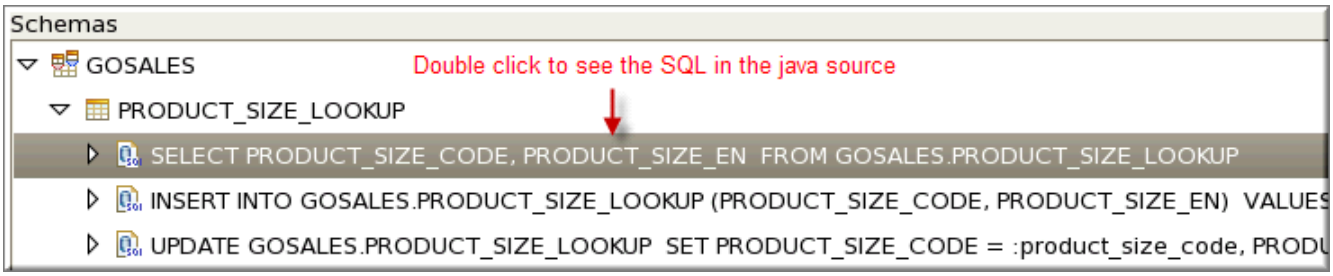
__13. You can view SQL statements used in a class (or projects in a workspace) with the help of pureQuery outline view. Click on *pureQuery Outline* view in the bottom pane.



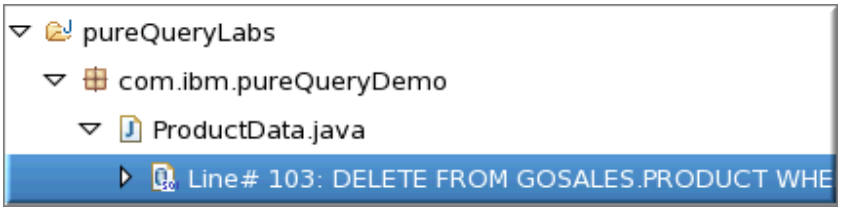
__14. After refreshing the outline view, you will see a view as shown below.



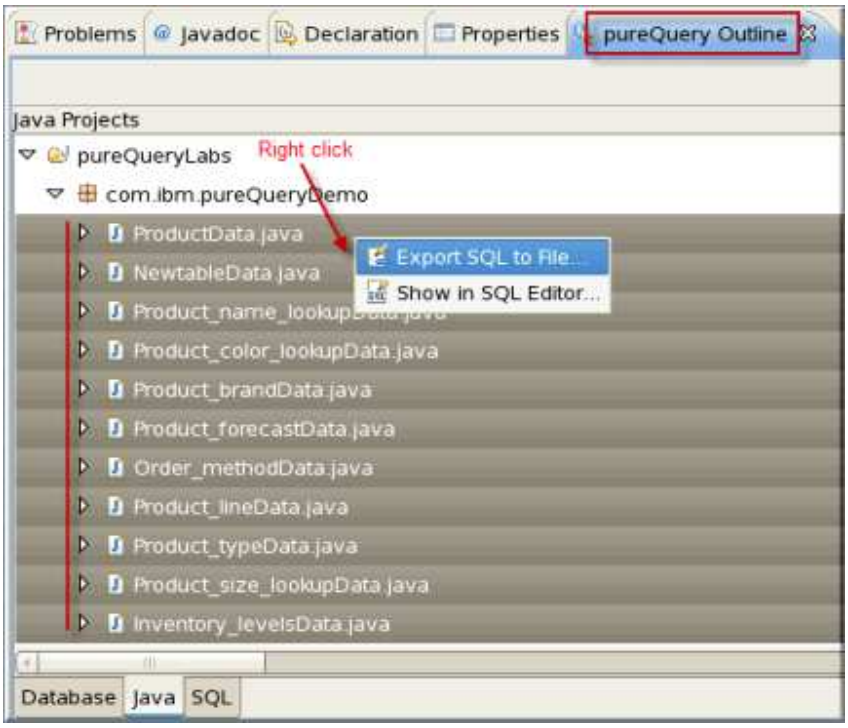
__15. Explore each view and you can easily see the relationships between Java classes, SQL statements and database using different views.

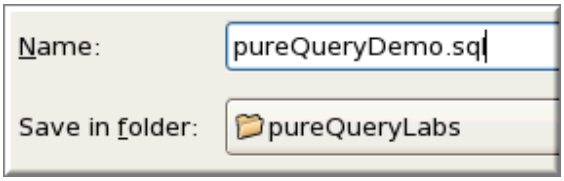


__16. Explore `Java` view to see SQL statements used in Java classes.



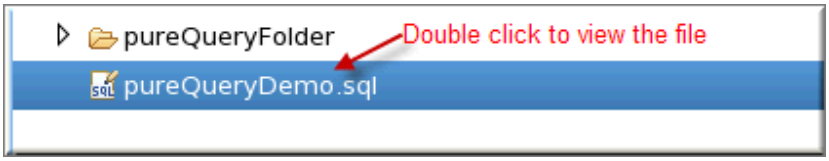
__17. Expand `pureQueryDemo` package and select all Java interface data access classes and right click. Select `Export SQL to File...` Save the file using any name you like and open it in an editor.



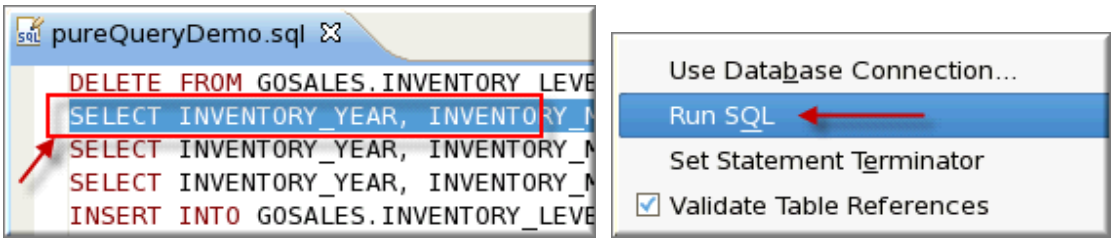


__18. Click on `pureQueryLabs` project and hit F5 to refresh it.

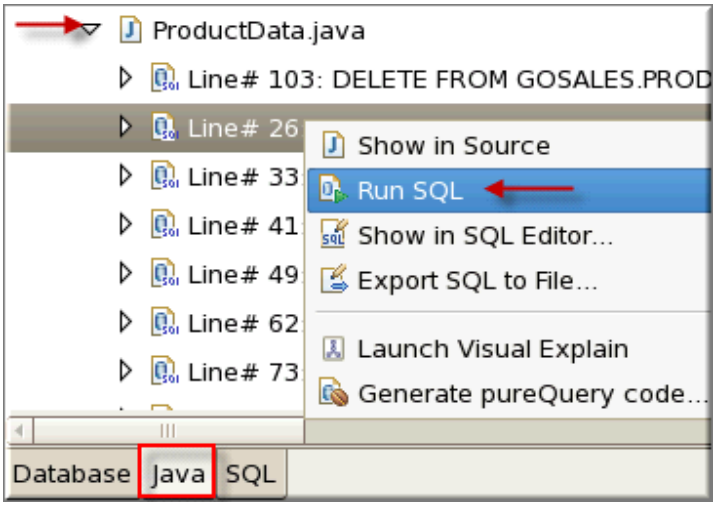
__19. Double click on `pureQueryDemo.sql` to open it in an editor. When prompted, specify `GSDB` database.



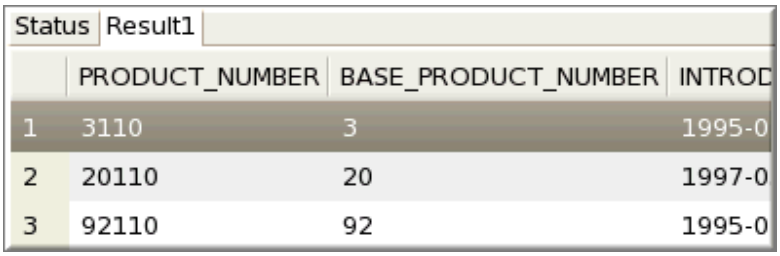
__20. You got all SQL statements used in `Inventory_levelsData.java` interface in a file. DBAs can use these SQLs to verify them for performance etc. Select a SQL statement and right click on it and click on `Run SQL` to run it. The results can be viewed in `SQL Results` window.



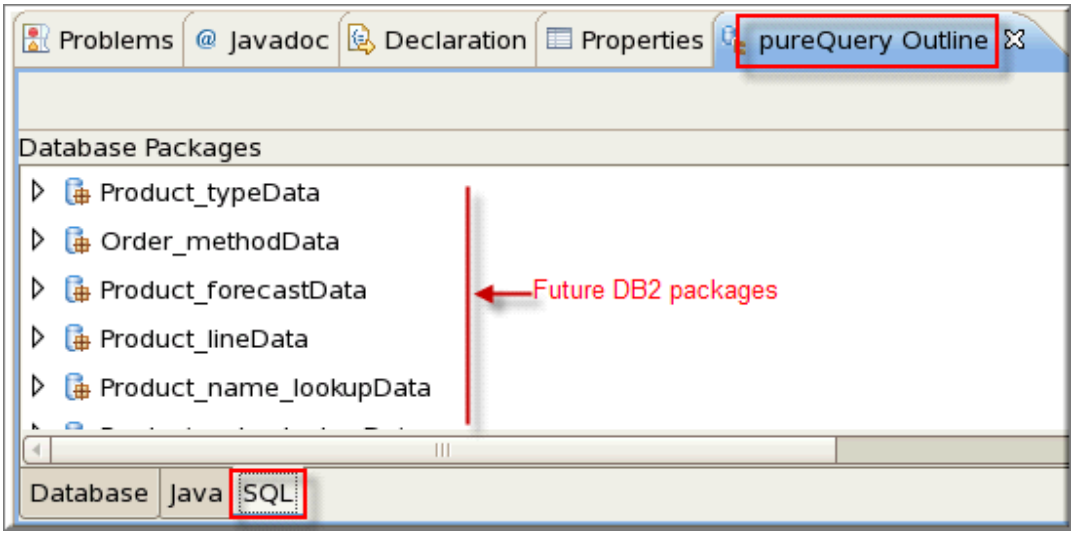
__21. In same `Java` view, expand `ProductData.java` and select `SELECT` statement and right click on it. Select `Run SQL`.



__22. The results can be viewed in `SQL Results` window.



__23. Explore SQL view to explore SQL statements in DB2 packages. Please note that these packages are not yet created.

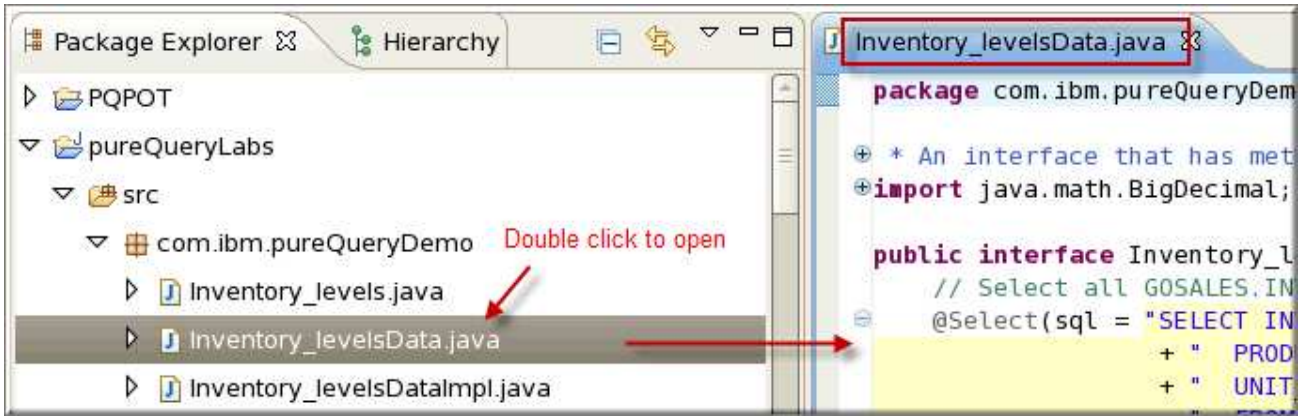


3.4 Explore pureQuery context assist capabilities

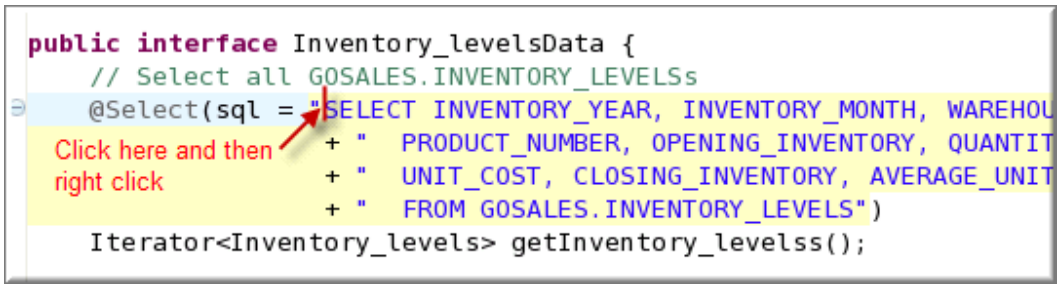
The pureQuery tools integrate the SQL editor inside the Java Editor providing developers a boost in productivity. Developers can now run SQL statements embedded in their Java programs as well as have SQL errors reported while typing the SQL statement inside the Java Editor.

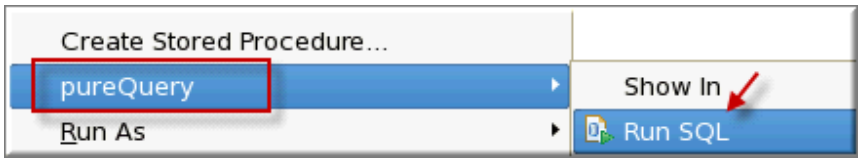
__24. Close all open files by clicking <CTRL><SHIFT><W>

__25. In the *Package Explorer*, double click on `Inventory_levelsData.java` to open the interface.

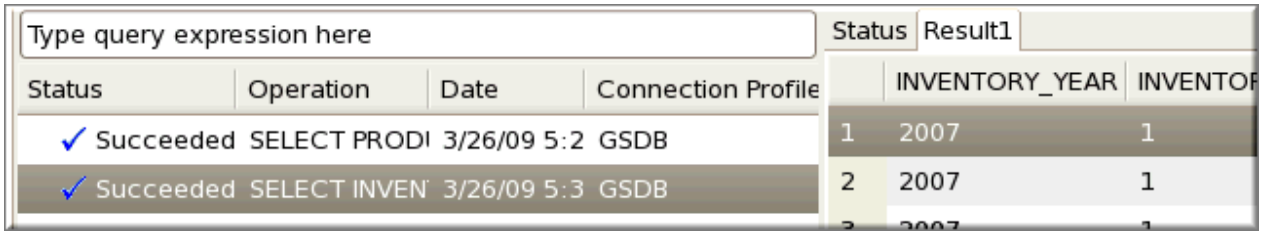


__26. Click at the beginning of the first SQL and then right click. Select `pureQuery` ⇨ `Run SQL`



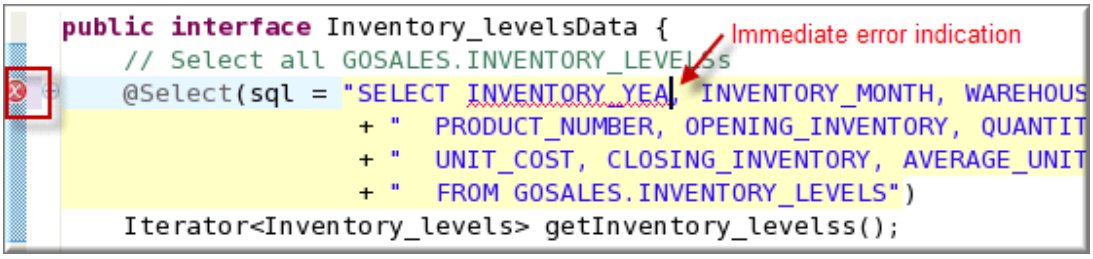


See the results of your query in the Data Output View in the bottom of the Data Studio:



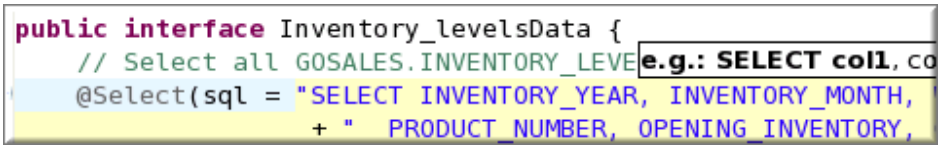
__27. While typing a SQL statement, errors will be underlined in red, just as in Java.

- Delete the letter “R” from `INVENTORY_YEAR` on the SQL statement. Notice that the editor underlines it in red displaying the message that it cannot find the column “`INVENTORY_YEA`” in the table `INVENTORY_LEVELS`:



__28. Using SQL Content Assist within the Java Editor:

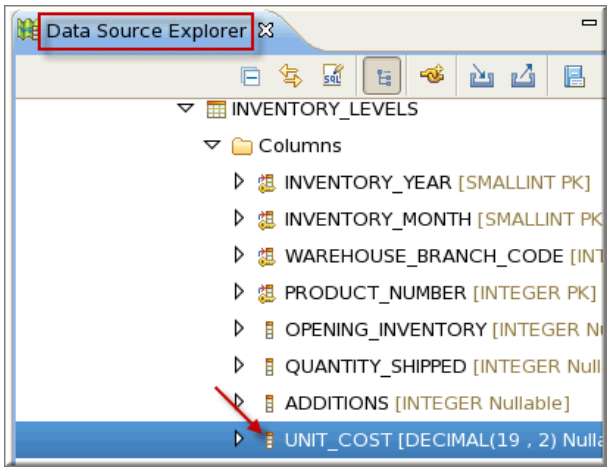
- After deleting the “R” from `INVENTORY_YEAR` in the previous example, put your cursor after “`INVENTORY_YEA`” and press the `<Ctrl>` key and the `<spacebar>` at the same time. This will change “`INVENTORY_YEA`” to `INVENTORY_YEAR`.



__29. If a developer wants to know the data type of a specific column or whether the column is *nullable* he/she can easily check with the help of the pureQuery tool.

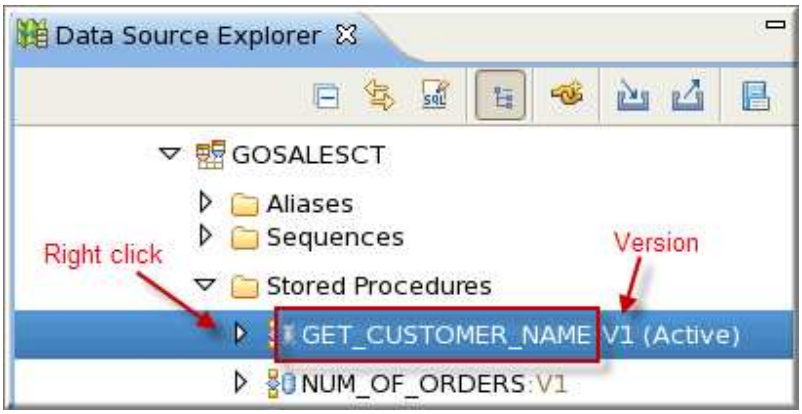
- Double-click on **`UNIT_COST`** so that it will be highlighted. Now right-click and go to *`pureQuery`* ⇨ *`Show in`* ⇨ *`Data Source Explorer`* or press `<SHIFT-F7>`.

__30. You will see the *Data Source Explorer* expanding the `INVENTORY_LEVELS` table and showing the information for the columns with the `UNIT_COST` highlighted. The developer now knows that the `UNIT_COST` column is of type `DECIMAL(19,2)`and is `Nullable`:



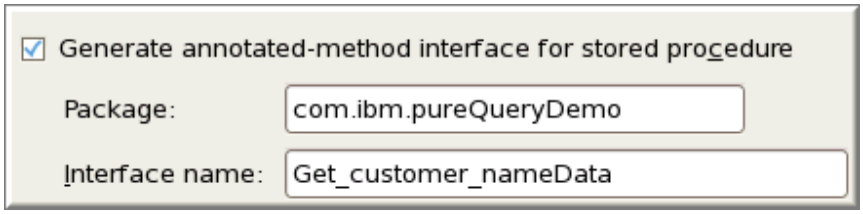
3.5 Generate pureQuery code for a SQL Procedure

__31. Expand the `Stored Procedure` folder under the `GOSALESCT` Schema (This is different schema than `GOSALES`) in the *Data Source Explorer*:

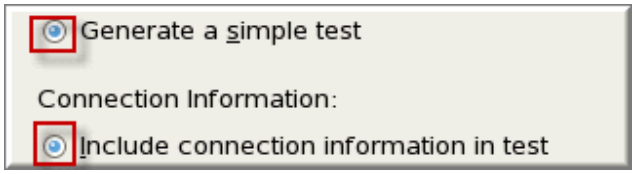


- Right-Click on the stored procedure *`GET_CUSTOMER_NAME`* and select *`Generate pureQuery Code…`*

- Fill the pop-up window as below if values are not already filled in.



- Click `<Next>`.
- Check on `Generate a simple test` and `Include connection information in test` and again click `<Next>`:



- The next screen allows you to modify mapping between parameters and bean attributes. Click on `<Next>` to go to the next screen.

Parameter Name	Parameter Type	Field Name	Field Type
CUSTOMERID	INTEGER	customerid	int
FIRST_NAME	VARCHAR	first_name	String
LAST_NAME	VARCHAR	last_name	String
PHONE_NUMBER	VARCHAR	phone_number	String

- In this screen, we are given a chance to discover result sets if stored procedure returns some result. Since our selected stored procedure does not return any result set, Click <Finish>.
- You will notice that 4 Java classes have been created for this stored procedure.

• `Get_customer_nameData.java`	Interface for data access
• `Get_customer_nameDataImpl.java`	Implementation layer generated
• `Get_customer_nameDataTest.java`	Test class for stored procedure
• `Get_customer_nameParam.java`	Parameters bean

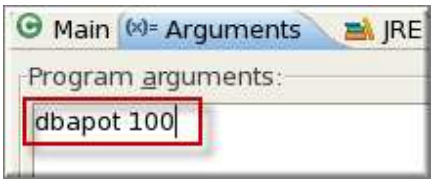
3.5.1 Calling a stored procedure

__32. Double click `get_customer_nameDataTest.java` in package explorer and this will open the Java test program in the editor view.

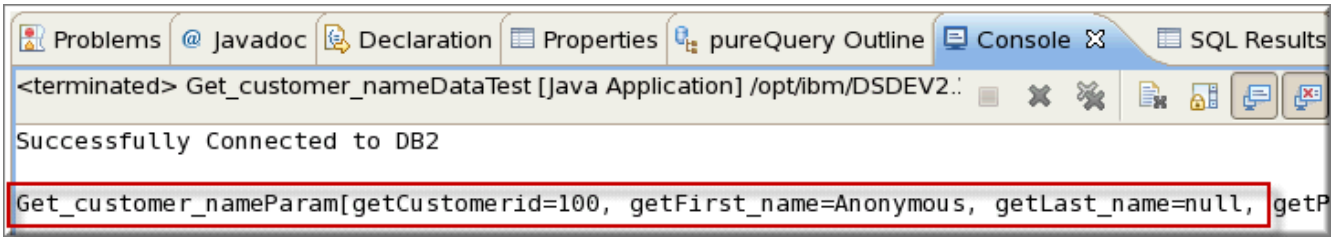
__33. Right click anywhere in Java source file and choose `Run ⇨ Java Application`. You will see console output stating that `All required arguments were not provided`. But doing so, you have created an instance of this application that can now be modified to specify input parameters.

__34. Right click on same Java source again and choose `Run As ⇨ Run Configurations..` which will open up `Run Configurations` window.

__35. Go to the *Arguments* tab and specify `dbapot` and `100`. The first argument is the password and second is the customer code for which the stored procedure will return a first name, last name and phone number. Click on <Run>.



__36. You will see the results in the Console.



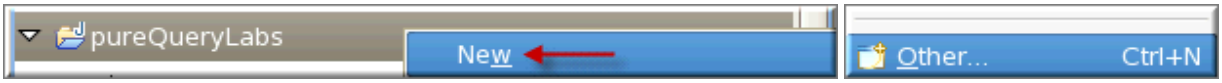
__37. Please review the Java source file `Get_customer_nameData.java` to see method `callGet_CUSTOMER_NAME` which was annotated with a CALL statement to the stored procedure. The implementation of this method was auto-generated and is shown in `Get_customer_nameDataImpl.java`.

```
public interface Get_customer_nameData {

    // Call GOSALESCT.GET_CUSTOMER_NAME
    @Call(sql = "Call GOSALESCT.GET_CUSTOMER_NAME( :customerid, :f
    StoredProcedureResult callGET_CUSTOMER_NAME(Get_customer_nameP

}
```

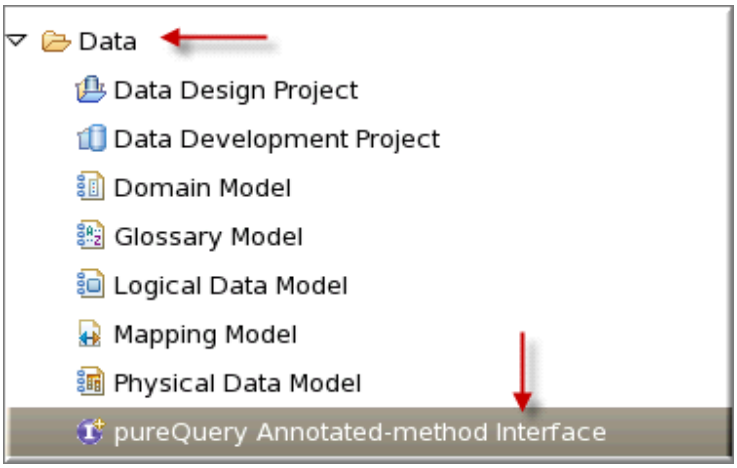
3.6 Generate pureQuery code from SQL Scripts

__38. You can generate pureQuery code from SQL defined in a file. You will use file `/home/ibmuser/POT_PQ/03TOOLS/CustomQueries.sql`.

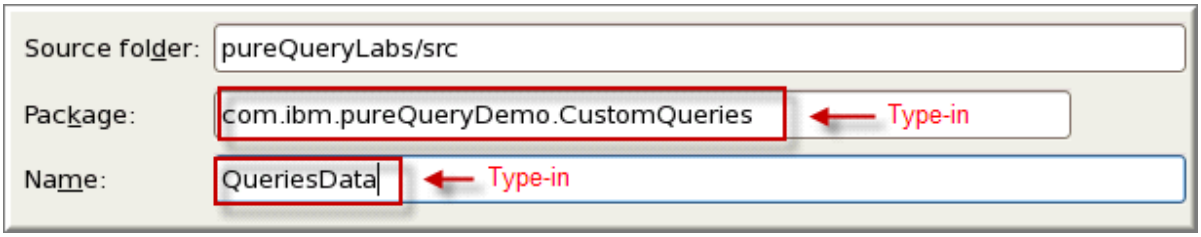
__39. Right click on `pureQueryLabs` project in the package explorer and click on `New` ⇨ `Other` …



__40. Expand the Data section and select the `pureQuery Annotated-method Interface` from next window click on `<Next>`.



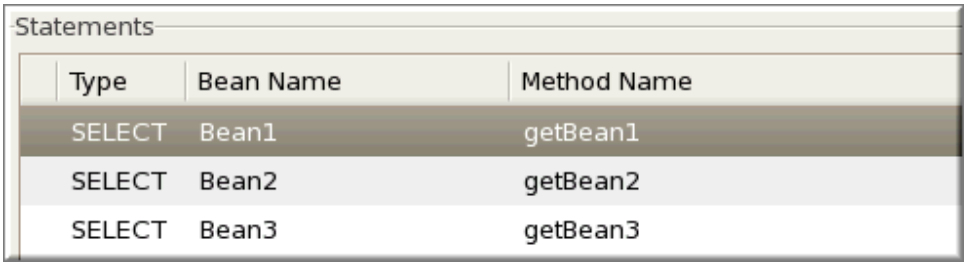
__41. Select `pureQuery Annotated-method Interface` from next window click on `<Next>`. Type in name of the package as `com.ibm.pureQueryDemo.CustomQueries` and name as `QueriesData` and click on `<Next>`.



__42. In SQL statements window, click on `Import` button.

__43. Double click on `File System` in the left pane. Double click on folders `home` ⇨ `ibmuser` ⇨ `POT_PQ` ⇨ `03TOOLS` and click on `CustomQueries.sql` and click <OK>.

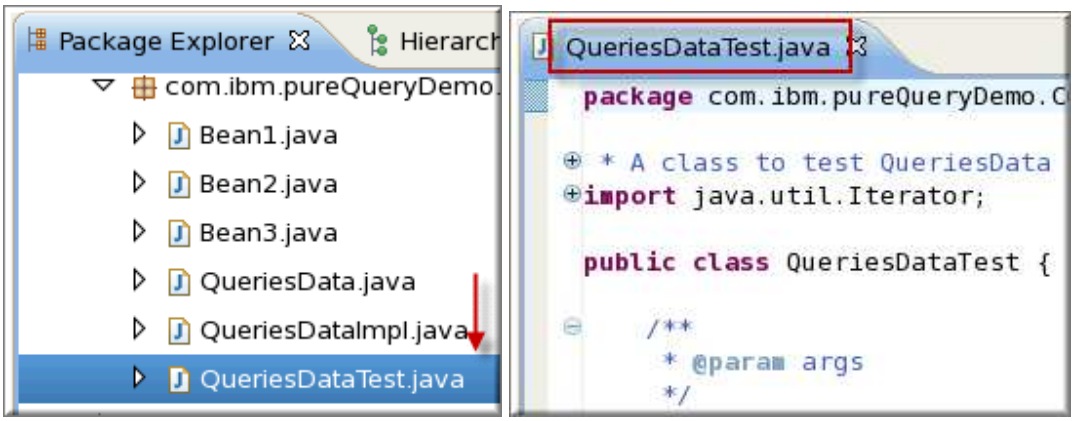
__44. You will see 3 `SELECT` statements imported in this window with default bean names as `Bean1`, `Bean2` and `Bean3`.



__45. Click on <Next> and then on <Finish> button to generate pureQuery code for these 3 SQL statements.

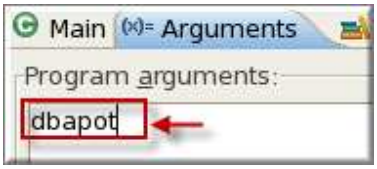
__46. Close all open files by clicking <CTRL><SHIFT><W>

__47. After generating pureQuery code, double click on `QueriesDataTest.java` in package explorer.

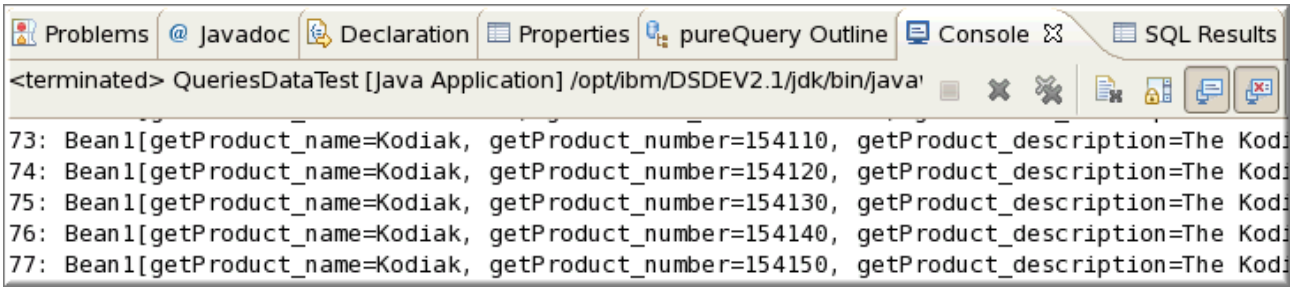


__48. Right click anywhere in the `QueriesDataTest.java` and choose `Run` ⇨ `Java Application`. You will see console output stating that `All required arguments were not provided`.

__49. Right click on same Java source again and choose `Run As ⇨ Run Configurations..` which will open up `Run Configurations` window. Type-in `dbapot` in the Arguments tab.



__50. Click on Run and you should see output from the `GetBean1` method.



**** End of Lab 3 – Explore pureQuery Tools**

Lab 4 - Explore pureQuery API

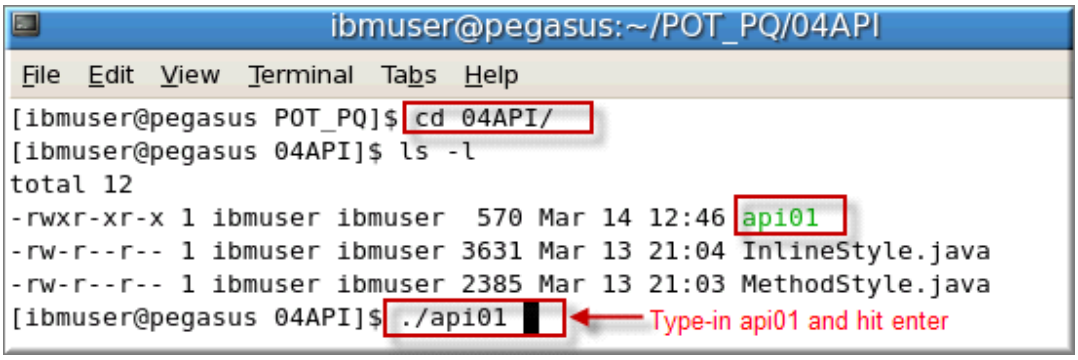
Prerequisites:

We need to copy the Java source files to the workspace.

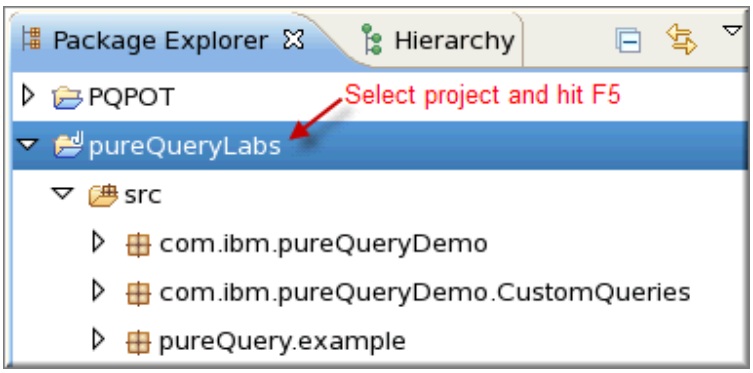
__1. Close all open files by clicking <CTRL><SHIFT><W>

__2. Double click on the *Terminal* icon on the desktop to open up a command window.

__3. You will see a command window. Change your directory to: `04API` and run `api01` command to copy `MethodStyle.java` and `InlineStyle.java` to the `pureQueryLabs` project.

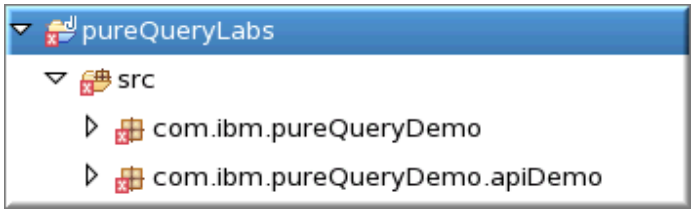


__4. Refresh the Java project so that the files copied in the previous step are reflected in *Package Explorer*.



__5. After you hit `F5`, you should see `apiDemo` package showing up in the *Package Explorer*. We created this package by running script `api01` in the previous step.

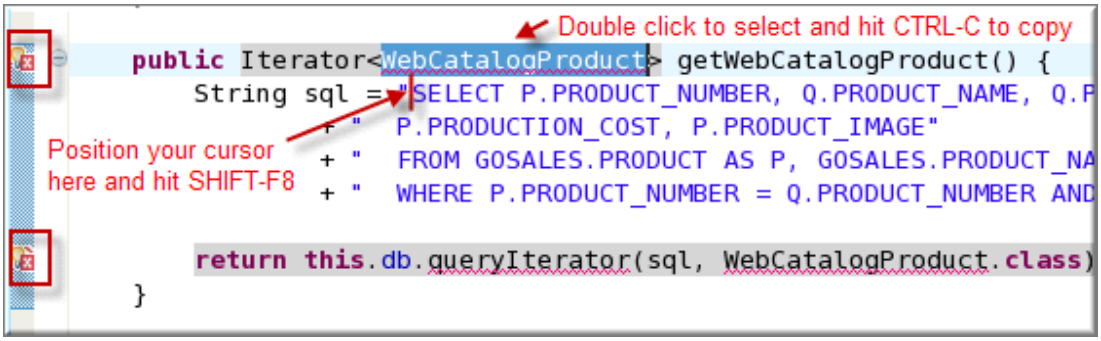
__6. You also notice a small cross icon on both the packages indicating errors. Do not worry about these errors and fixing them is part of this lab exercise.



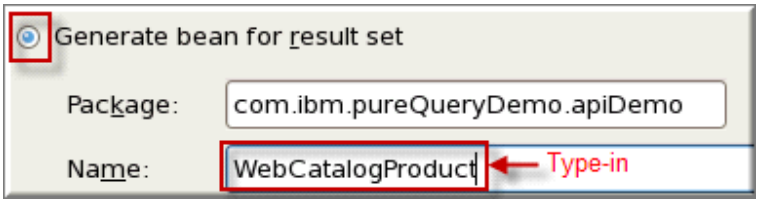
4.1 Practice Code Generation

__7. Expand `apiDemo` package and open `InlineStyle.java` by double clicking on it. We will fix some errors by creating beans from SQL statements.

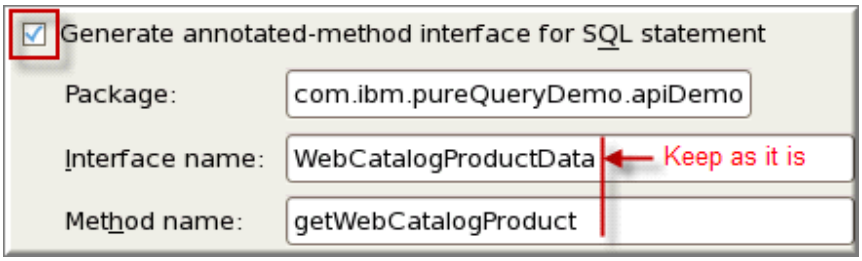
__8. Go to the 1st SQL statement or the first error marked in the file. We are referencing a missing `WebCatalogProduct` bean here. This bean maps to the SQL statement and we will generate it from the SQL. Double click on `WebCatalogProduct` to select it and hit `CTRL-C` to copy this in clipboard. Position your cursor at the beginning of the `SELECT` statement in the next line and hit `Shift-F8` to open pureQuery code generation dialog.



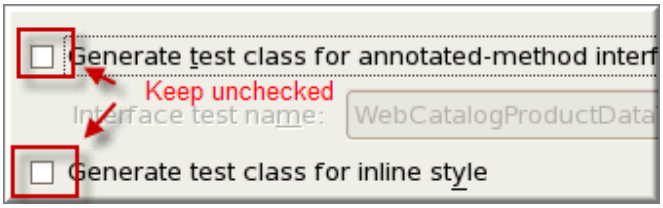
__9. Type-in `WebCatalogProduct` bean name as shown and make sure that `Generate bean for result set` is selected



__10. Make sure that `Generate annotated-method interface for SQL statement` is checked. Click `<Next>`.

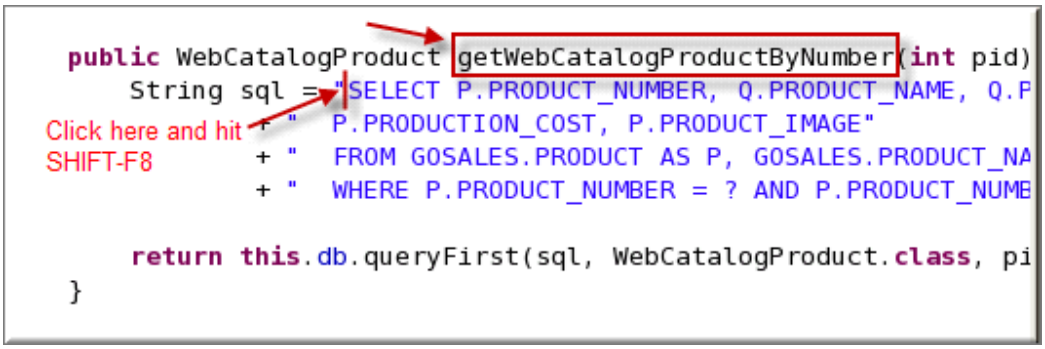


__11. Uncheck test program generation option and click `<Finish>` to generate bean for the SQL statement.



__12. The Java bean `WebCatalogProduct.java` is created and Data Studio will open it up for you. Review and close this and go back to the `InlineStyle.java` program.

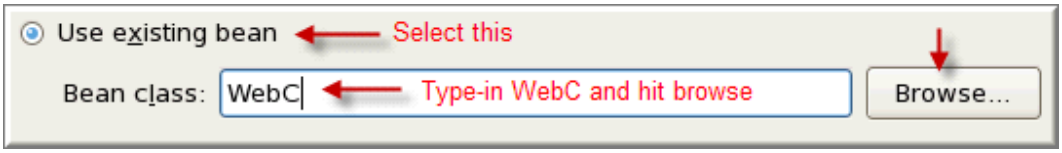
__13. Go to the 2nd SQL statement and position your cursor at the start of the `SELECT` statement and hit `SHIFT-F8`.



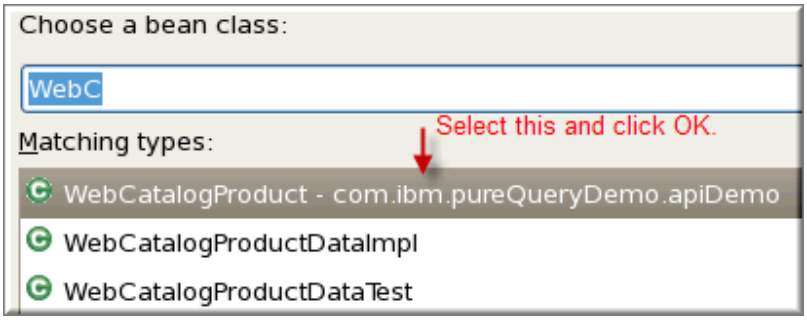
__14. In the `pureQuery Code Generation` screen, our choices will be different than the previous step.

- We will use the bean that we created in the previous step.
- We will also reuse the interface layer by appending new methods to it.

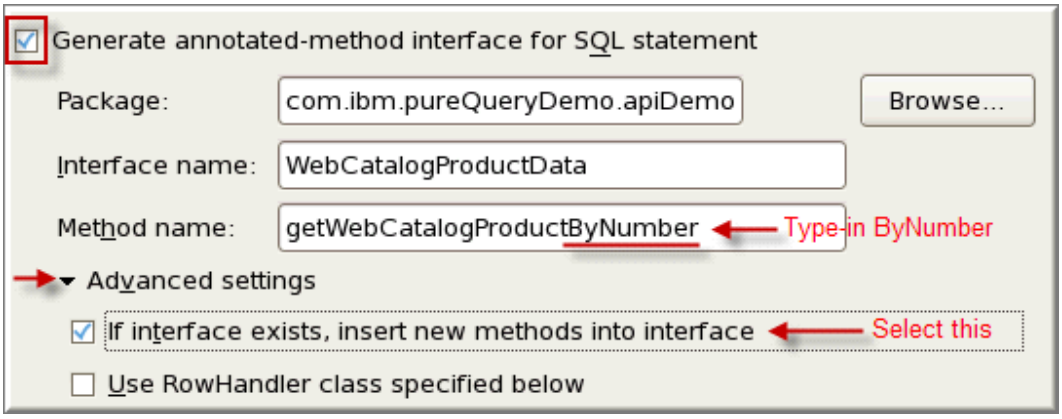
__15. Click on `Use existing bean` and type-in `WebC` and then click `Browse` button to select the bean class.



__16. Type-in `WebC` to reduce the number of beans and select `WebCatalogProduct` bean and hit `OK`.

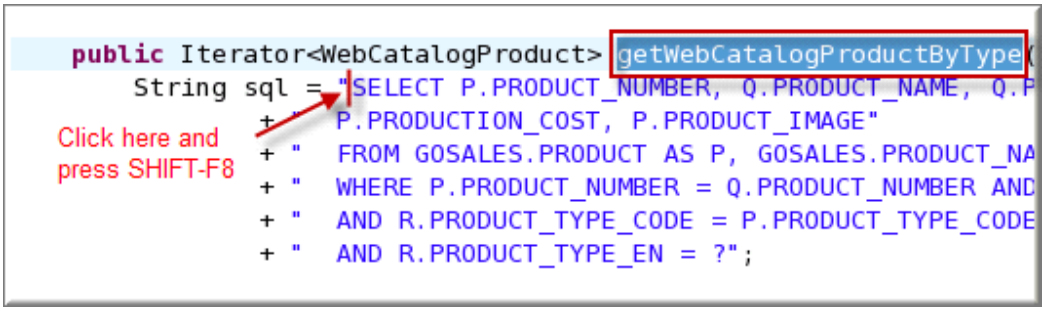


__17. When you come back to the same screen from the previous step, you should give the method name as `getWebCatalogProductByNumber`. You need to just add `ByNumber` at the end of already given name of `getWebCatalogProduct`. We already created the interface in the previous step and we are using the same one, so it is also necessary that you check `If interface exists, insert new methods into the interface`.

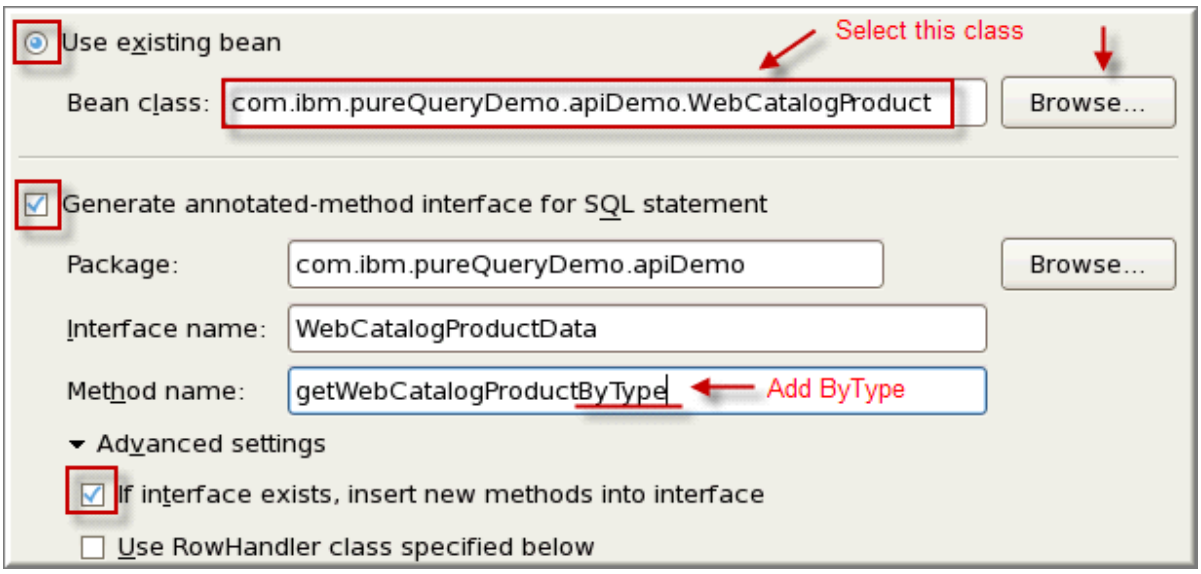


__18. Click `<Finish>` to generate additional method in `WebCatalogProductData.java` file. Open the file, review it and close it and go back to `InlineStyle.java` program.

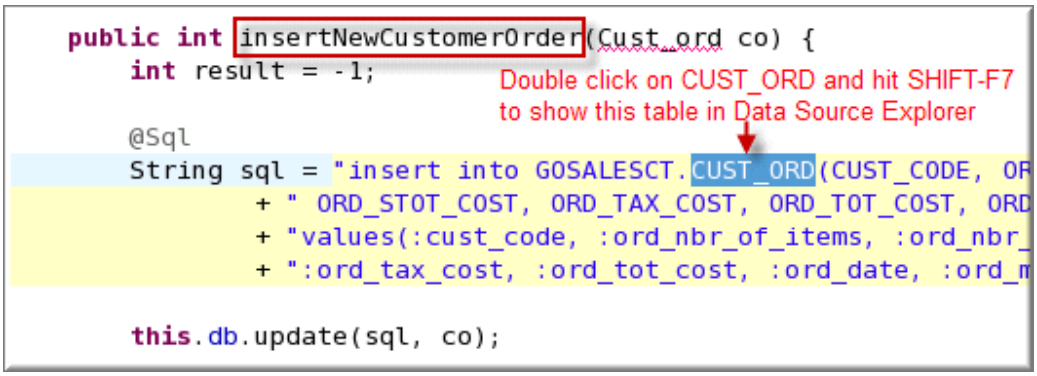
__19. Go to the 3rd SQL statement in `InlineStyle.java` and position your cursor at the start of the `SELECT` statement and press `SHIFT-F8` to open pureQuery Code Generation screen.



__20. Click on the Browse button to select existing `WebCatalogProduct` bean and append `ByType` to the method name and click `<Finish>` to append the pureQuery code to the existing `WebCatalogProductData` interface.

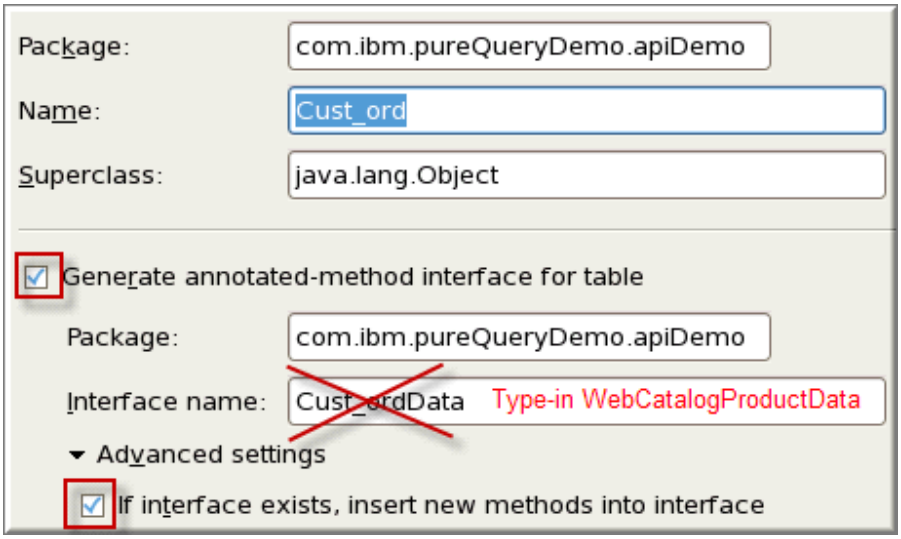


__21. Go to the 4th SQL statement in `InlineStyle.java` and position your cursor at the start of the `INSERT` statement and press `SHIFT-F7` to view this table in *Data Source Explorer*.

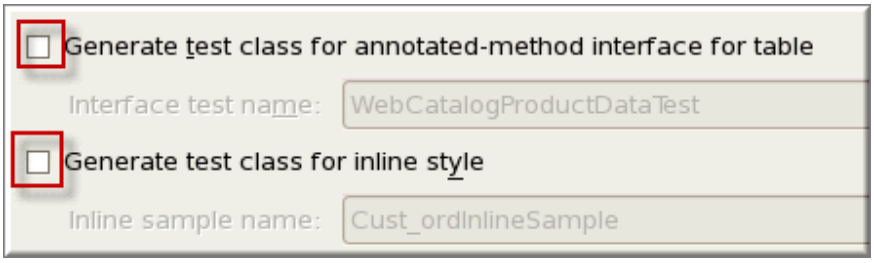


__22. Go to the Data Source Explorer and right click on table CUST_ORD and click on Generate pureQuery Code.

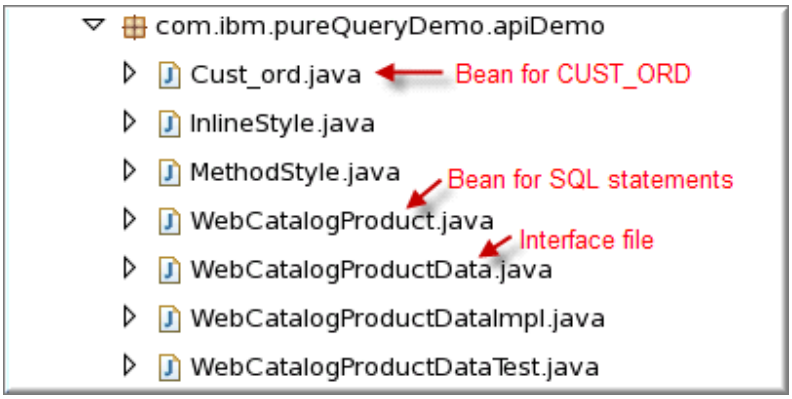
__23. In pureQuery Code Generation screen, check Generate annotated-method interface for table and If Interface exists, insert new methods into interface check boxes. Replace Interface name from the default value of Cust_OrdData to WebCatalogProductData. Click <Next> to go to the next screen.



__24. In the next screen, keep both the check boxes unchecked for creating test classes. Click <Finish> to append methods to fetch and create the customer order in an existing WebCatalogProductData data interface.



__25. After completing above steps, we should see error free `InlineStyle.java` and `MethodStyle.java` with additional files in the package.



Note: Review what we did in previous steps before going to the next step.

- Created `WebCatalogProduct` bean from 1st SQL statements.
- Created `WebCatalogProductData` interface containing annotation method API for the 1st SQL statement.
- Created additional two annotation methods for 2nd and 3rd SQL statement in `WebCatalogProductData` interface by using same bean created for the 1st SQL statement.
- Created a bean for `CUST_ORD` table and added two methods for getting and creating a customer order in the existing `WebCatalogProductData` interface.

__26. Close all open files by clicking `<CTRL><SHIFT><W>`

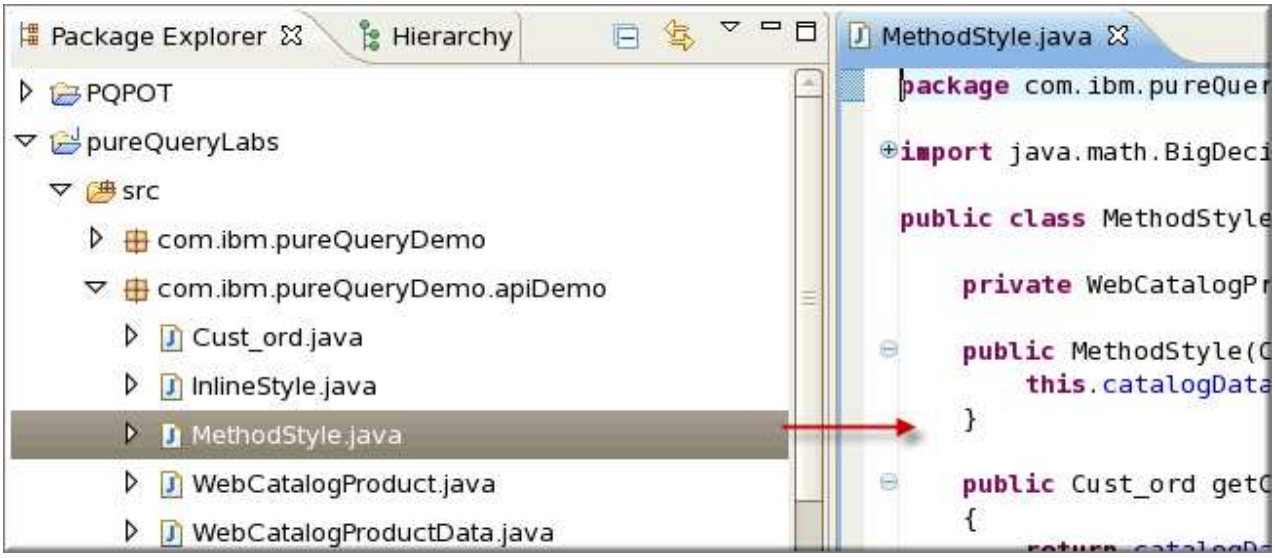
__27. Open `InlineStyle.java`, `MethodStyle.java` and `WebCatalogProductData.java` and review them. The `InlineStyle.java` contains inline SQL statements for which we created annotation methods in previous steps. Both the Java programs provide same output but by using inline and annotation APIs.

4.2 Using Method-style Program

Introduction:

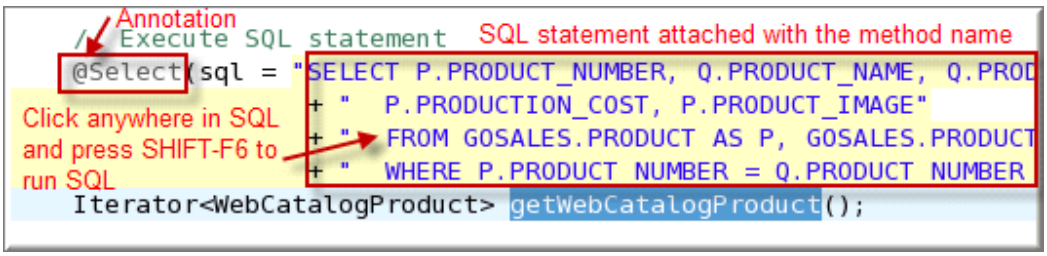
The pureQuery Annotated Method Style provides data accessor and update methods. These methods are declared in a user-created Java interface using annotations that express the specific query or update operations in standard SQL. Using Java annotated class definitions; a generator automatically creates the implementation of the specified methods. This style offers the advantage of separating the data access declarations and the associated SQL from the application's business logic. The application simply invokes the methods defined in the interface and uses familiar Java objects, beans and collections for providing parameters to the method and for receiving query results.

__28. Open the `MethodStyle.java` class by double-clicking the file and review the methods.

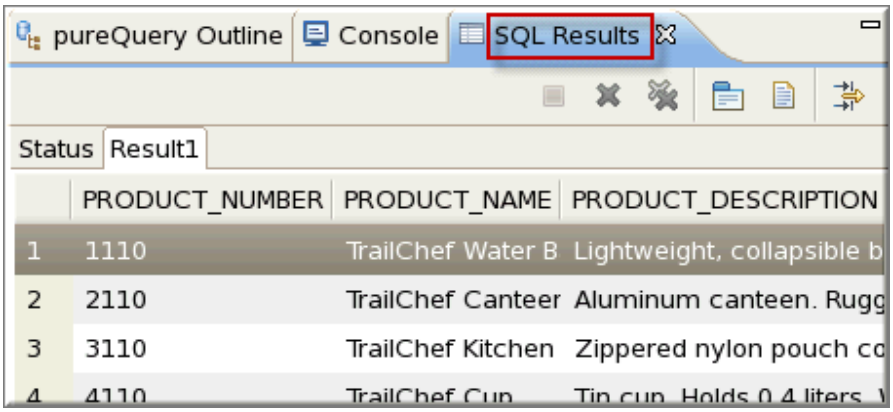


__29. Review method `getWebCatalogProduct`. Click on the method name inside the body of the method and hit F3 which will take you to the definition of the method in the Interface class.

__30. Review the method in the interface class. The method name `getWebCatalogProduct` is annotated with the SQL statements.



__31. Click anywhere inside the SQL statement and press `SHIFT-F6` to run the SQL statement. You will see the output from SQL statement in the `SQL Results` window in the lower bottom pane.



__32. The implementation of method `getWebCatalogProduct` is in `getWebCatalogProductDataImpl.java`. This implementation file gets generated whenever any change is made to the interface file by adding or removing the methods.

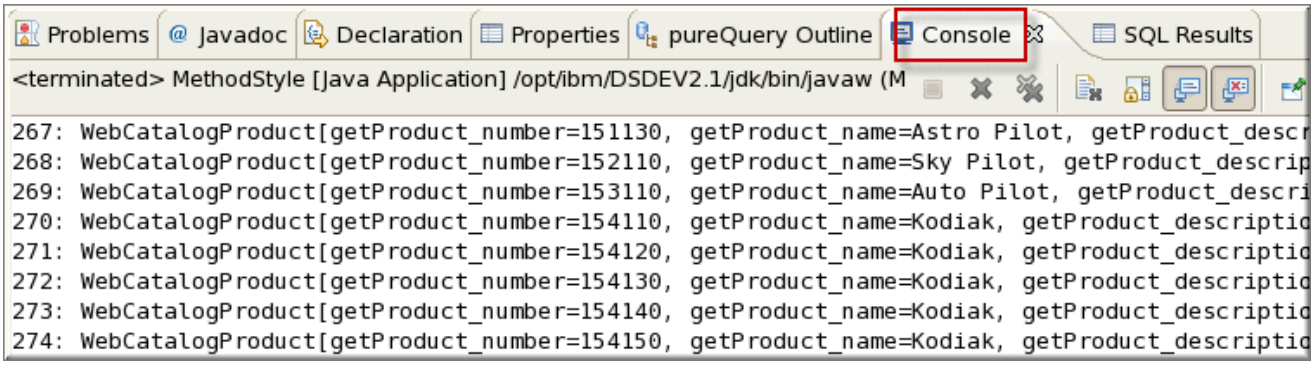
__33. Open `getWebCatalogProductDataImpl.java` and review the generated code.

__34. Go back to *MethodStyle.java* and review the `main` method. We will test each of the method through the main routine.

```
public static void main (String[] args)
{
    Connection conn = SampleUtil.getConnection("jdbc:db2://
    MethodStyle method = new MethodStyle(conn);
    int choice = 1;
    switch (choice) {
        case 1 :
            method.PrintCatalog();
            break;
        case 2 :
            method.PrintCatalog(1110);
            break;
        case 3 :
            method.PrintCatalog("Watches");
            break;
        case 4 :
```

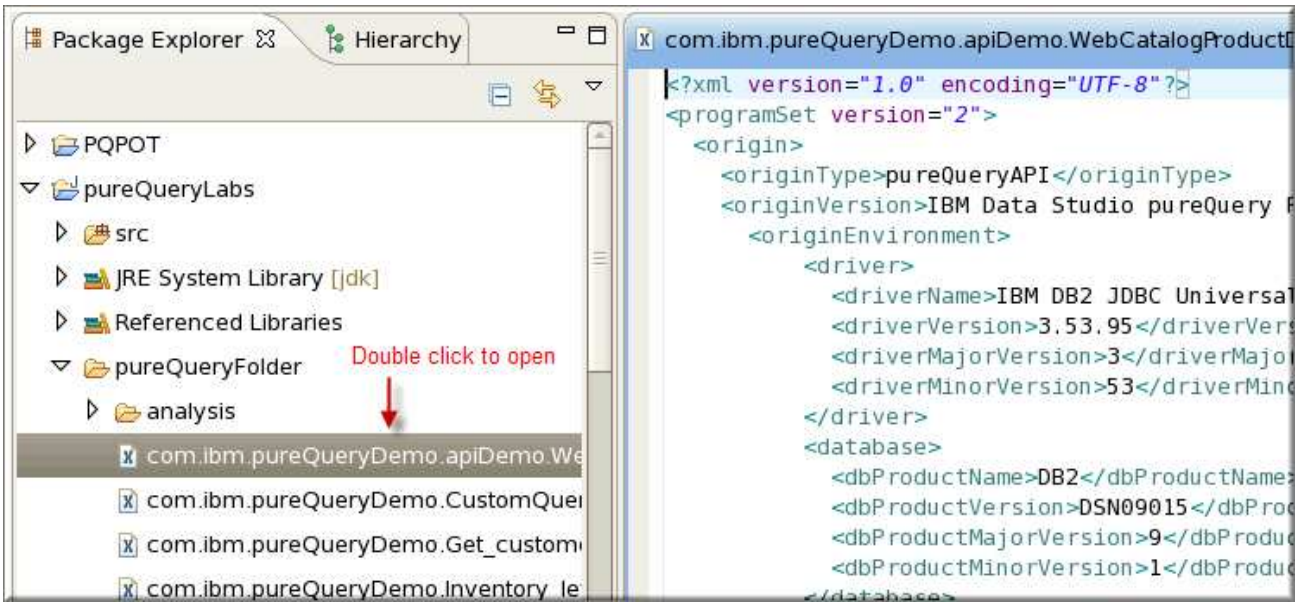
Change choice 1-4 to test each of the method.

__35. Right click anywhere in the program and click on `Run As ⇨ Java Application.` You will see the console output as shown:



__36. Close all open files by clicking `<CTRL><SHIFT><W>.`

__37. Go to the *Package Explorer* and expand `pureQueryFolder` folder and open `WebCatalogProductData_pdq.xml` file. This XML file contains all the SQL statements referenced in the `WebCatalogProductData` interface. The SQL in this XML is also called named query which is same as JPA standard. We will review this again when we go through pureQuery runtime.



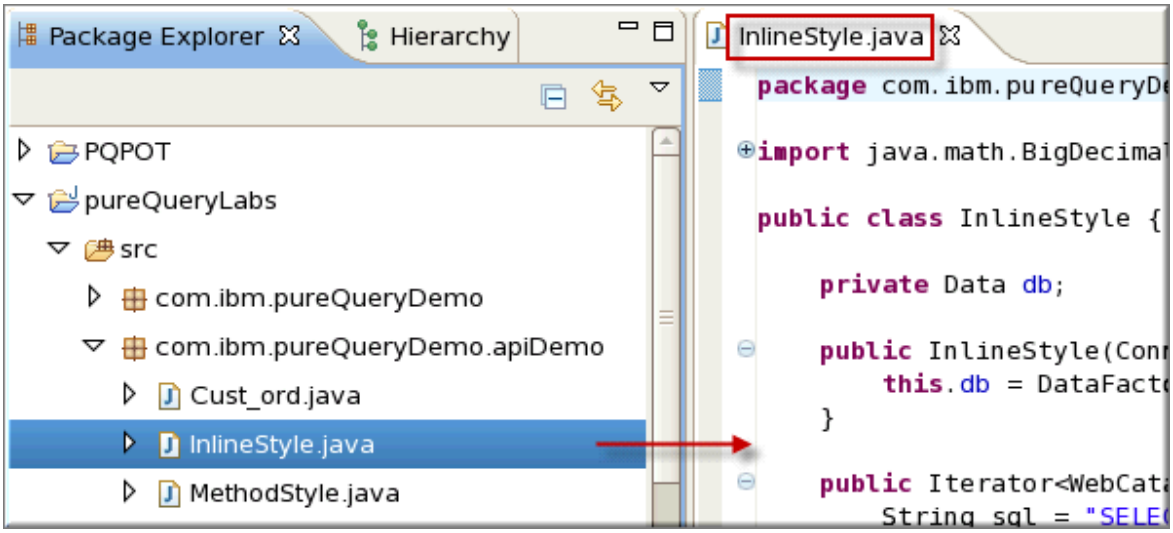
4.3 Using an Inline-style Program

Introduction:

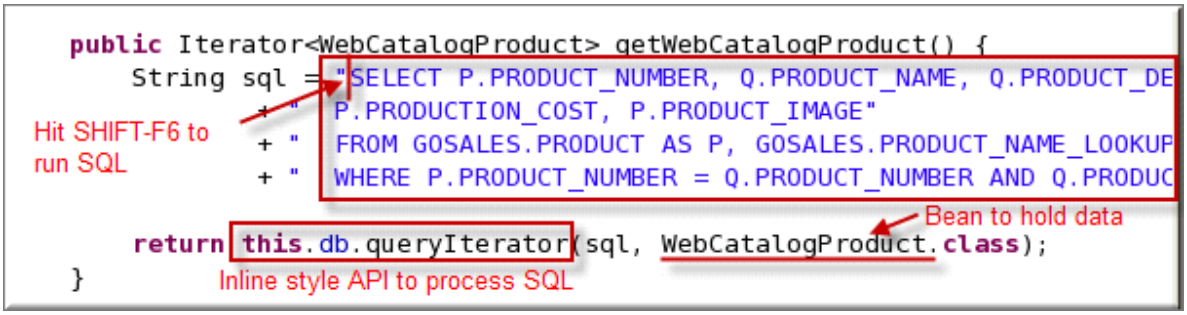
The pureQuery Inline-Style provides a complete set of Java methods for executing queries and update operations. These methods take an SQL statement and associated parameters as input and, where appropriate, return the results in numerous forms including a variety of Java collection types, as well as user-defined Bean types or as scalar and primitive values. With this style, the SQL query or update statement can be coded inline in the application and appears as a parameter on the method invocation. This programming style offers simplicity and tight integration between the SQL and the Java language.

__38. Close all open files by clicking <CTRL><SHIFT><W>.

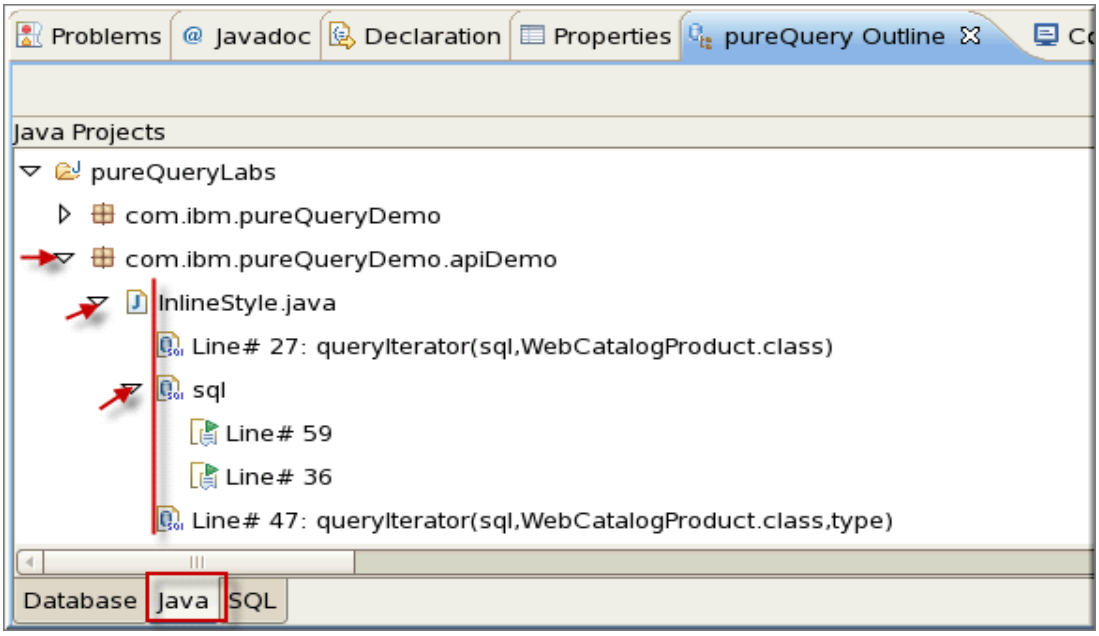
__39. Open the `InlineStyle.java` class by double-clicking the file and review the methods.



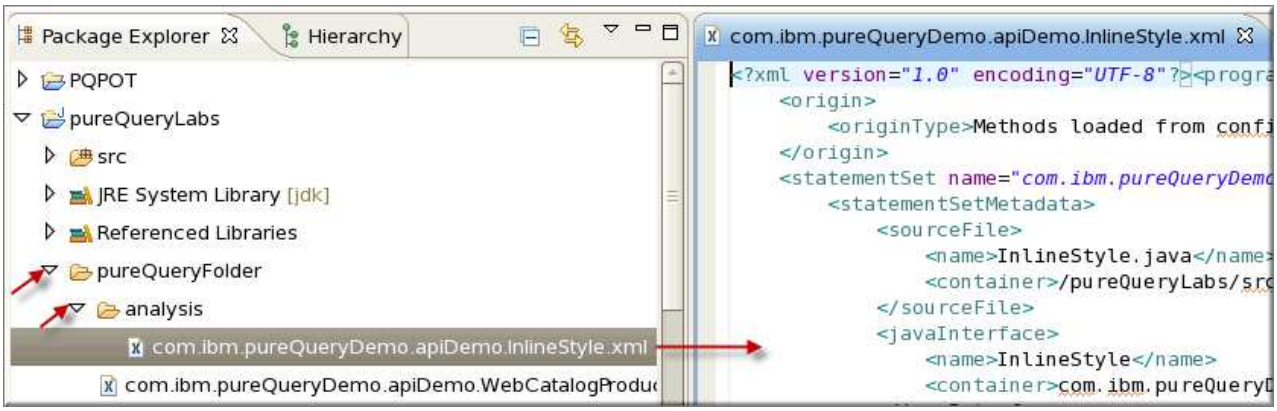
__40. Review all 4 SQL statements and associated in-line style pureQuery APIs to process the SQL statement.



__41. Select `Java` tab in `pureQuery Outline` view and expand `apiDemo` package. Expand `InlineStyle.java` and you can see line numbers at which SQL statements are used.



__42. Expand `pureQuery Folder` in `pureQueryLabs` project in the *Package Explorer*. Expand `InLineStyle.xml` under the analysis folder. The methods used in `InLineStyle.java` are saved in this XML file. We will come back to this later.



__43. Review `main` method and run the program. Right click anywhere in the program and click on `Run As ⇨ Java Application`. The console output for each of the method will be same as you did in the method style exercise.

__44. Change the value of choice parameter from 1 to 4 and run the program each time.

```
int choice = 1;
switch (choice) {
```

**** End of lab 4: Explore pureQuery API**

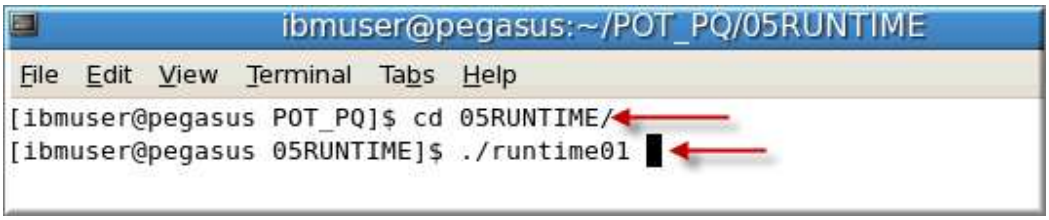
Lab 5 - Explore pureQuery Runtime

Note: By running the next command, you are setting the Data Studio *pureQueryLabs* project as if you have completed the *03 Tools lab* and the *04 API labs* correctly. If you are a DBA and have come to this lab by skipping *03 TOOLS* and *04 API* labs, wait for few seconds to allow workspace to compile and build Java source.

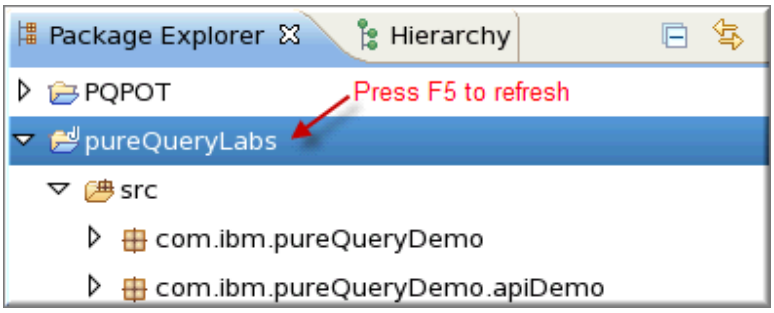
__1. Close all open files by clicking `<CTRL><SHIFT><W>.` If you have a command window open from the previous lab, close it.

__2. Double click on the *Terminal* icon on the desktop to open up a command window.

__3. Change your directory to: `05RUNTIME` and run `runtime01` command. (Running this command will refresh your project as if you completed labs 03 and 04.)



__4. Click on `pureQueryLabs` project in the Package explorer and hit F5 to refresh the project.

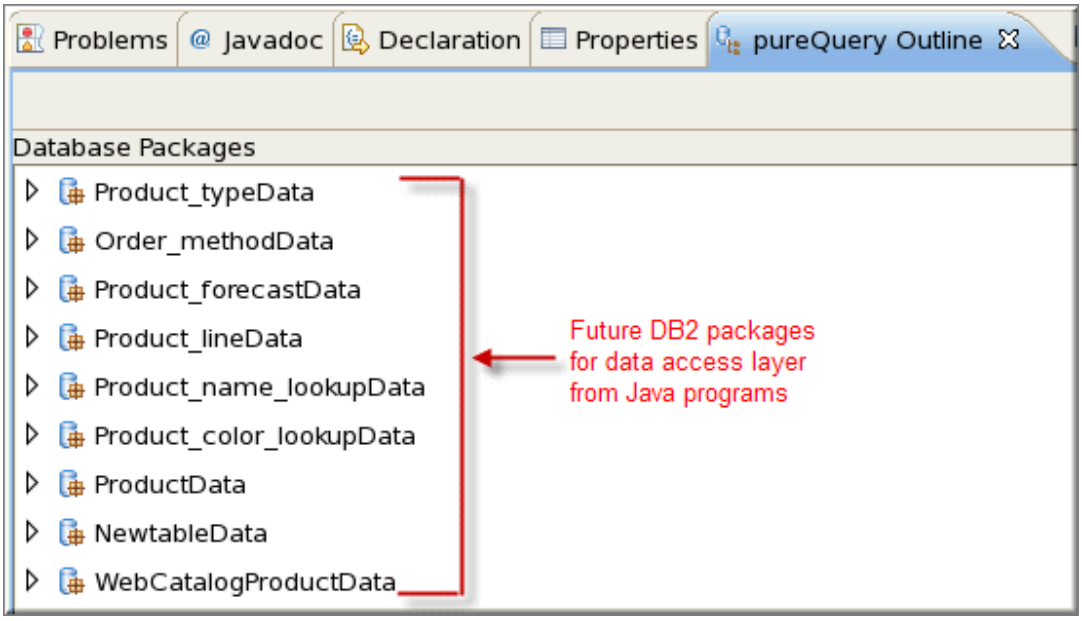


5.1 Explore pureQuery Outline View

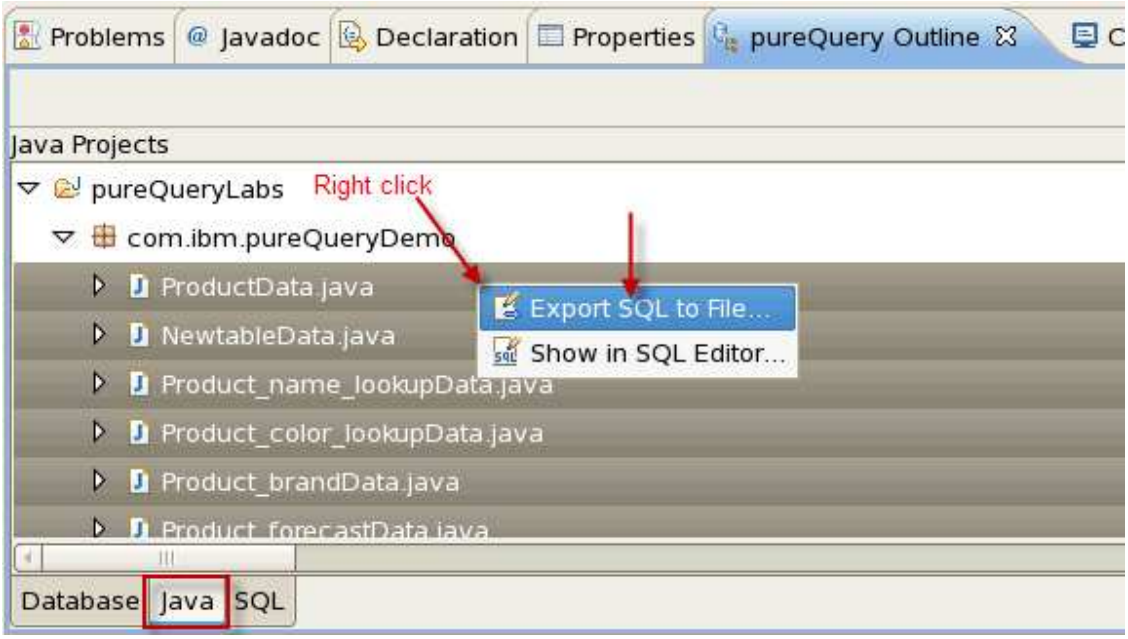
__5. Go to the *pureQuery Outline* view and click on refresh icon to rebuild it.

Note: If you do not see *pureQuery Outline* view, right click on *pureQueryLabs* project in *Package Explorer* and click on `pureQuery` ⇨ `Show pureQuery Outline`.

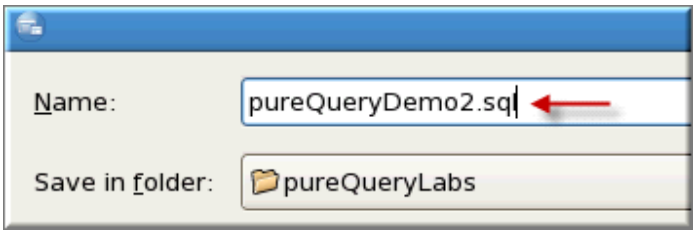
__6. In *pureQuery Outline* view, click on the SQL tab at the bottom. You will see a list of the future DB2 packages that are ready for bind or deploy.



__7. Click on `Java` tab at the bottom of the *pureQuery Outline* view and select all Java data access classes. Right click and export SQLs to a file to view them.

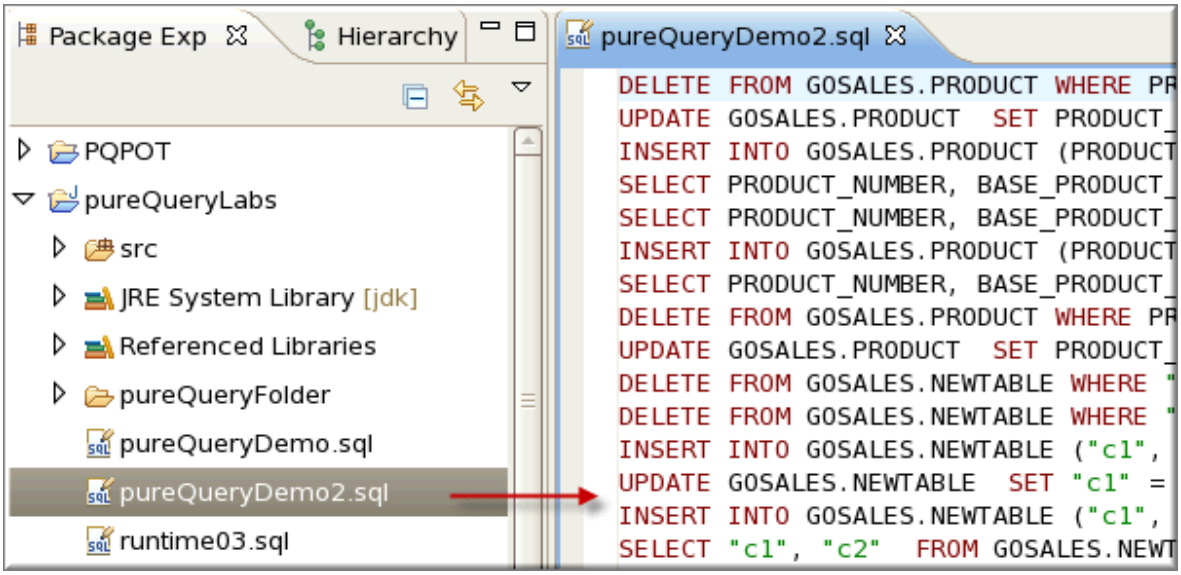


__8. Specify file name as `pureQueryDemo2.sql`.

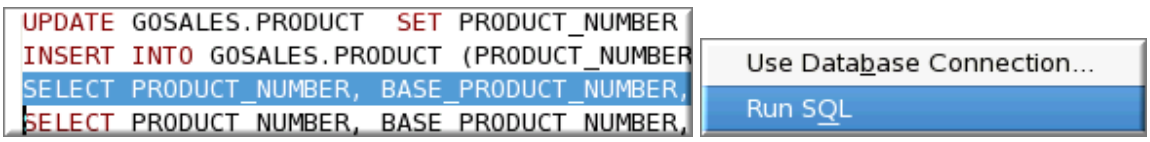


__9. Click on `pureQueryDemo` project to select it and hit F5 to refresh the project.

__10. Double click on `pureQueryDemo2.sql` to review SQL statements from data access classes. When prompted, select GSDB database.

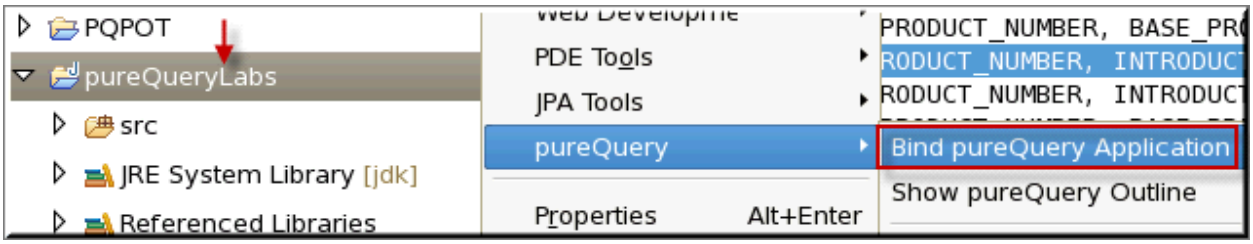


__11. Select any SELECT statement and right click on it. Click on Run SQL and view the results in *SQL Results* windows.

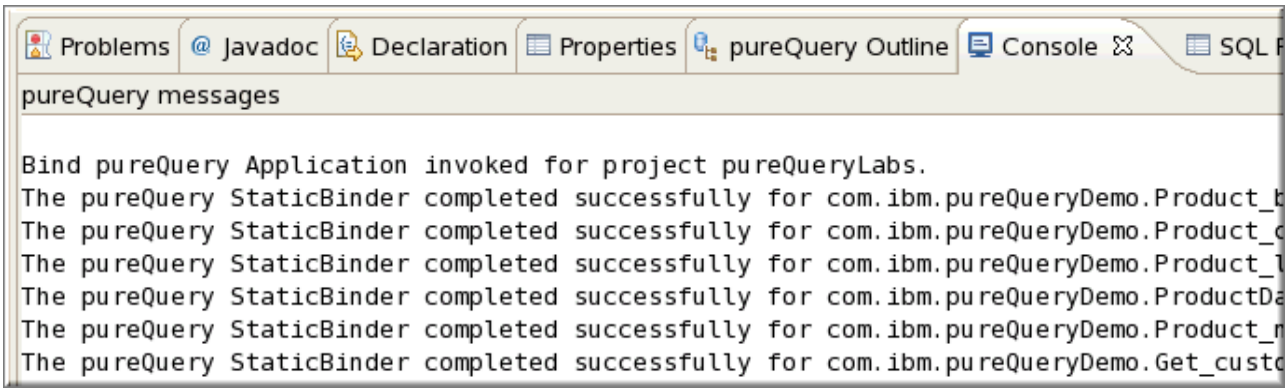


5.2 Bind packages for a pureQuery project

__12. From the *Package Explorer* view right-click on the `pureQueryLabs` Java project and select `pureQuery` ⇨ `Bind pureQuery Application`. Select `GSDB` database when prompted.

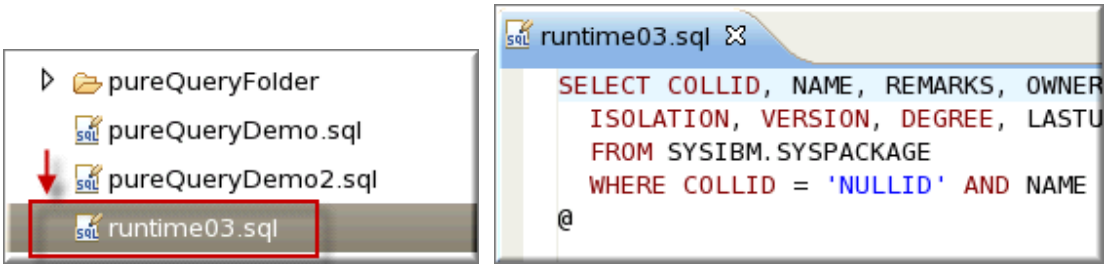


__13. Please wait for `BIND` process to complete. Look at the output in the `Console` view and you will notice that the `BIND` should complete successfully.



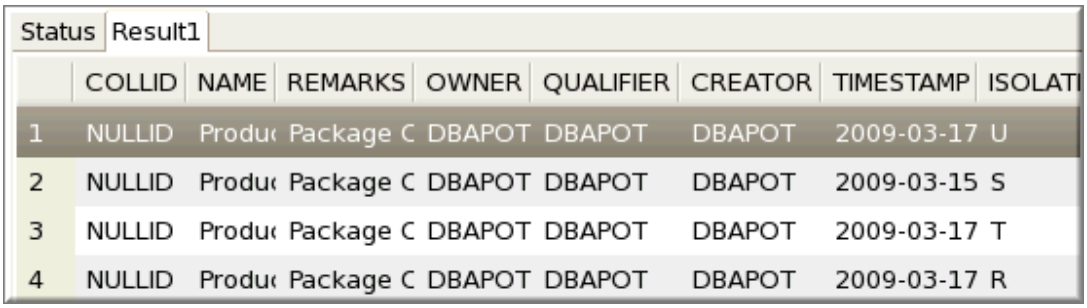
__14. View the info of the packages through SQL:

- Double-click the `Runtime03.sql` file under the `pureQueryLabs` project and select `GSDB` database when prompted.



- Go to main menu and click on `Script` ⇨ `Run SQL`.

- You should now see the following results:

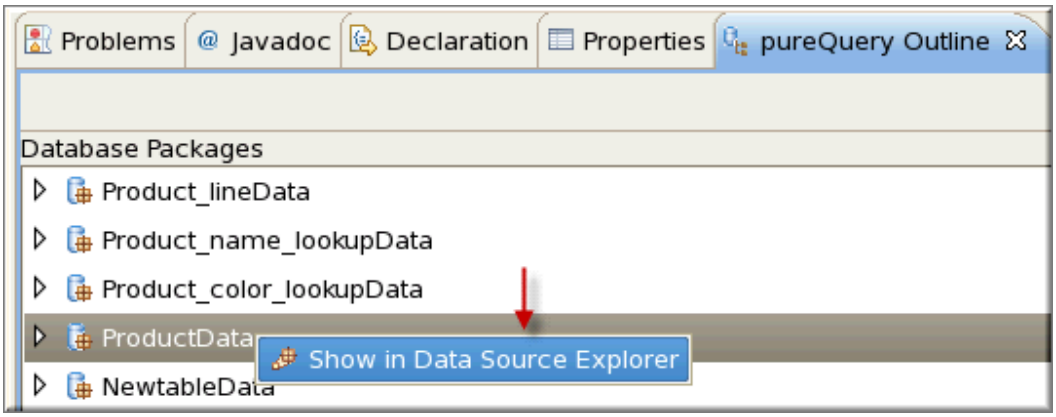


	COLLID	NAME	REMARKS	OWNER	QUALIFIER	CREATOR	TIMESTAMP	ISOLATI
1	NULLID	Produ	Package C	DBAPOT	DBAPOT	DBAPOT	2009-03-17	U
2	NULLID	Produ	Package C	DBAPOT	DBAPOT	DBAPOT	2009-03-15	S
3	NULLID	Produ	Package C	DBAPOT	DBAPOT	DBAPOT	2009-03-17	T
4	NULLID	Produ	Package C	DBAPOT	DBAPOT	DBAPOT	2009-03-17	R

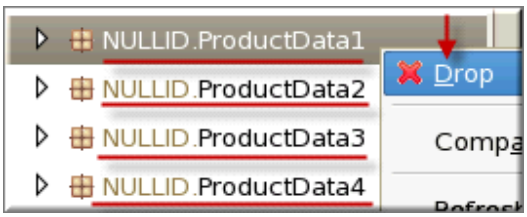
__15. Did you notice 4 packages created for the `ProductData` interface? This happened since we did not specify the `ISOLATION LEVEL.` Expand `pureQueryFolder` and double click `Default.bindProps` file to open it and add following line. Hit `CTRL-S` to save the file.

```
defaultOptions= -isolationLevel CS
```

__16. We will need to drop these 4 packages before we bind `ProductData` interface. Go to the *pureQuery Outline* view and select `SQL` view to view packages. Right click on `ProductData` package and click on `Show in Data Source Explorer`.



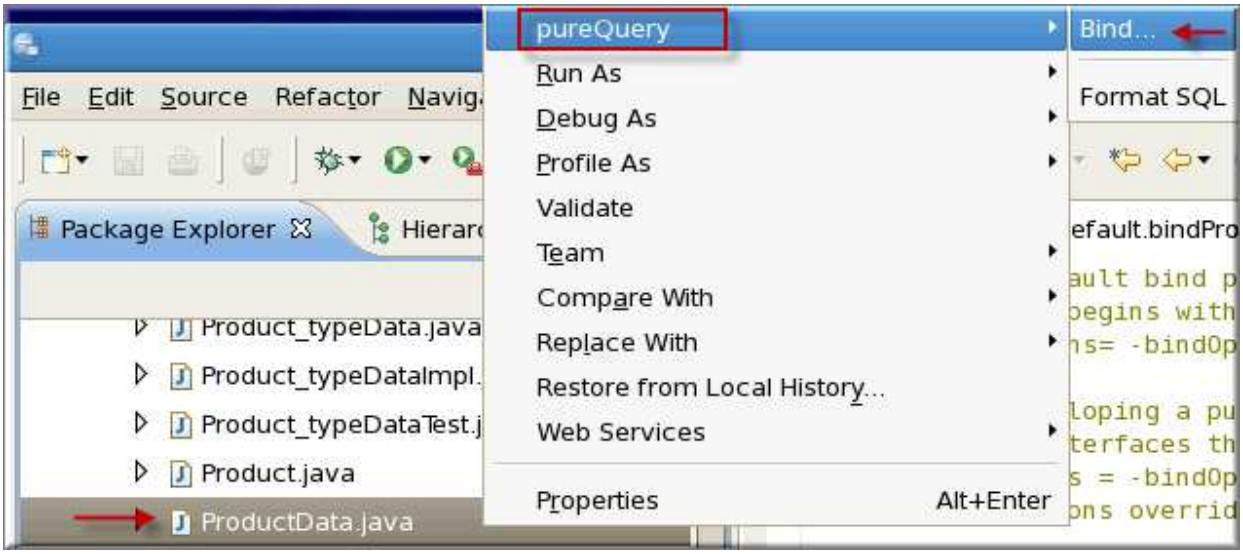
__17. All 4 packages will be highlighted in *Data Source Explorer*. Drop `ProductData1` package by choosing `Drop` option when you right click on it.



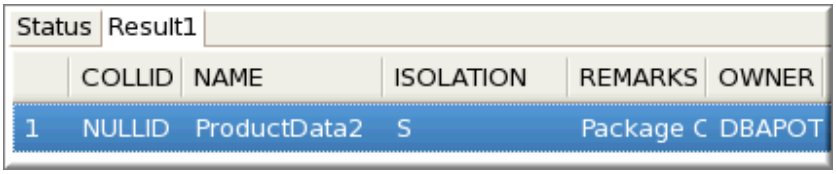
__18. Repeat previous 2 steps to drop `ProductData2` through `ProductData4` packages.

__19. Expand `com.ibm.pureQueryDemo` package and locate `ProductData` interface and select it.

__20. Right click on `ProductData.java` and click on `pureQuery` ⇨ `Bind` to bind the interface. When prompted, select `GSDB` database. Check console for the `BIND` completion message.



__21. After successful bind, go back to the `Runtime03.sql`. Right click on your first SQL and run it. You should only see one package with `ISOLATION LEVEL CS`.



5.3 Turn Dynamic SQL into Static SQL

After binding the packages for data access classes, the SQL in the Java application continues to run in dynamic mode unless we turn on the switch also known as `executionMode`.

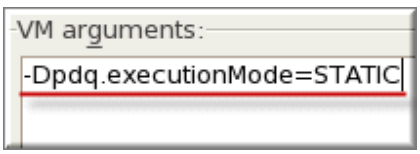
__22. There are many ways to turn `executionMode` to `STATIC` or `DYNAMIC`. For example:

Scope	Method	Description and how to set
Global	JVM™	Set the value as a JVM system property and is applicable to all of the pureQuery XML files in the application that you start with the `Java` command.
Global	Property file	Use `pdq.properties` and it is applicable to all connections for an application
Connection Specific	URL or DS property	`jdbc:db2://localhost:50000/GSDB:pdqProperties =executionMode(STATIC)`
Class Specific	Property file	Modify the application to create a Properties object, set the property there, and pass it to the factory that creates the Interface implementation instance.

__23. We will use the `JVM` option to set this property.

- Expand `apiDemo` package and double click `MethodStyle.java` to open it in an editor.

- Right click anywhere in the `MethodStyle.java` program and choose `Run As ⇨ Java Application`.
- Again right click and choose `Run As ⇨ Run Configurations…` and go to the *Arguments* tab and specify `-Dpdq.executionMode=STATIC` in *VM arguments* window and click on <Run> button.



- Notice the output on the Console is the same as if you were running dynamic SQL.

- But how did you know if it ran using DB2 package or not? Go to the *pureQuery Outline* view and click on SQL tab at the bottom. Go to the `WebCatalogProductdata` package and right click on it. Click on `Show in Data Source Explorer`.

- In the *Data Source Explorer* view, right click on `WebCatalogProductdata` package and click on `Drop`. Click `OK` to drop the package.

- After dropping the package, run `MethodStyle.java` program again.

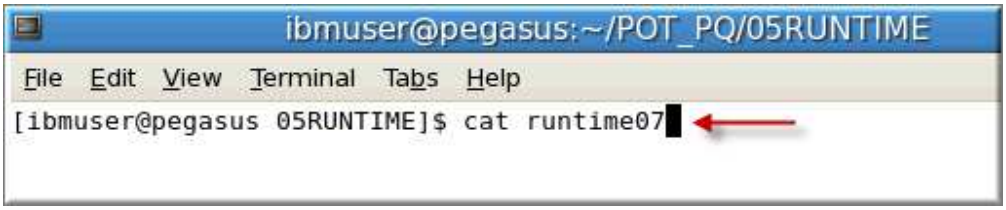
- You will see SQL -805 error indicating that the `WebCatalogProductdata` package was not found.

```
NOT FOUND IN PLAN DISTSERV. REASON 04. SQLCODE=-805, SQLSTATE=51002,
```

5.4 Bind Packages through Command Line

As a DBA, you might need to run *Static Binder* through command line if there is no option for a GUI tool like Data Studio to be deployed in a production environment.

__24. Go to the command window and see script `runtime07`.



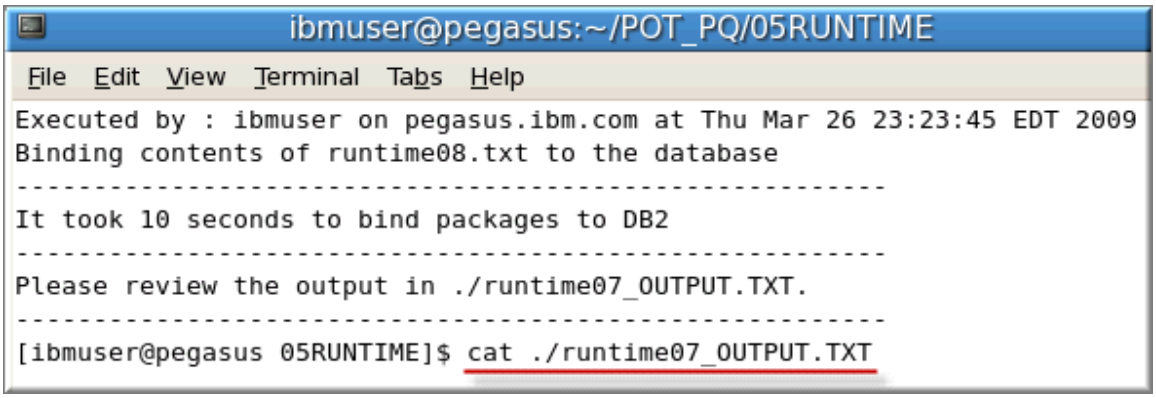
__25. Review the contents of this file and notice how a Static Binder is invoked.

```
CLPATH=$JCC_HOME/db2jcc.jar:$JCC_HOME/db2jcc_license_cisuz.jar
CLPATH=$CLPATH:$PQHOME/pdq.jar
CLPATH=$CLPATH:$PQHOME/pdqmgmt.jar   <-- License file
CLPATH=$CLPATH:$PQHOME/bin

URL="-url jdbc:db2://bluepearl.ibm.com:5025/GSDB"
USERINFO="-user dbapot -password dbapot"
BINDER=com.ibm.pdq.tools.StaticBinder   <-- Static Binder

echo Binding contents of runtime08.txt to the database

$JAVA -cp $CLPATH $BINDER $URL $USERINFO -optionsFile runtime08.txt >>
```

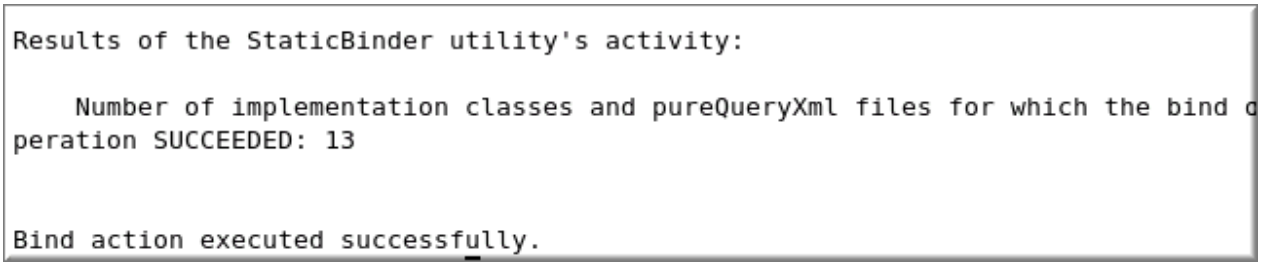
__26. Review the contents of the `RunTime08.txt` option file where data interface classes are listed.

```
[ibmuser@pegasus 05RUNTIME]$ cat runtime08.txt
defaultOptions= -bindoptions "QUALIFIER GOSALES" -isolationLevel CS
com.ibm.pureQueryDemo.apiDemo.WebCatalogProductData
com.ibm.pureQueryDemo.CustomQueries.QueriesData
```

__27. Now run script `./runtime07`.



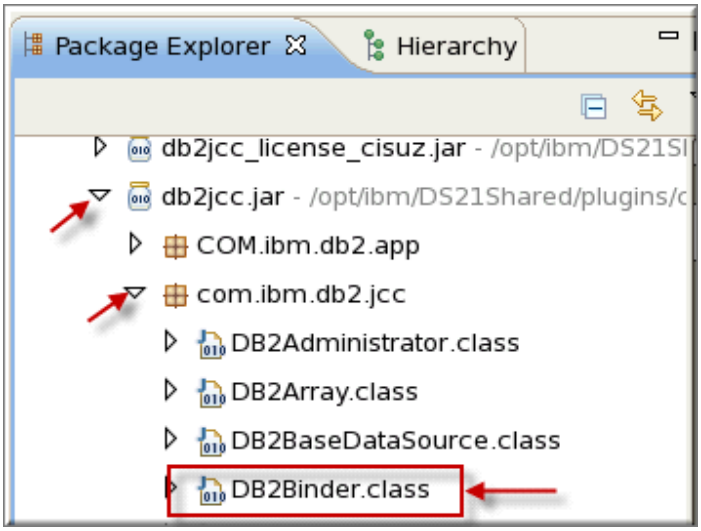
__28. After it has completed the work, review output file `Runtime06_OUTPUT.TXT` for the results.



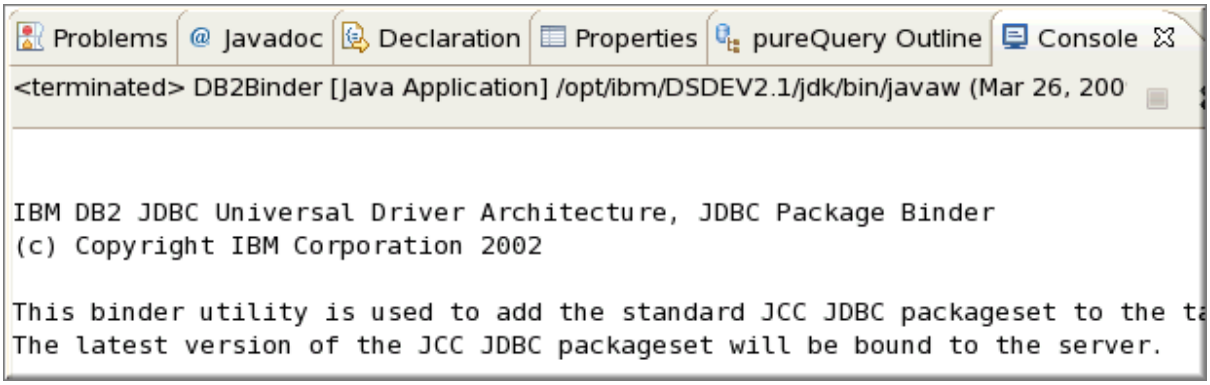
5.5 DB2Binder command to REBIND a package

__29. Under the `pureQueryLabs` project expand the `db2jcc.jar` file.

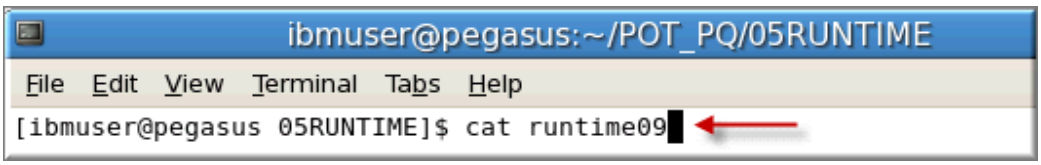
__30. Expand the `com.ibm.db2.jcc` package and notice the `DB2Binder` class.



__31. Right-click the `DB2Binder` class and select `Run As ⇨ Java Application` and you will see the help message.



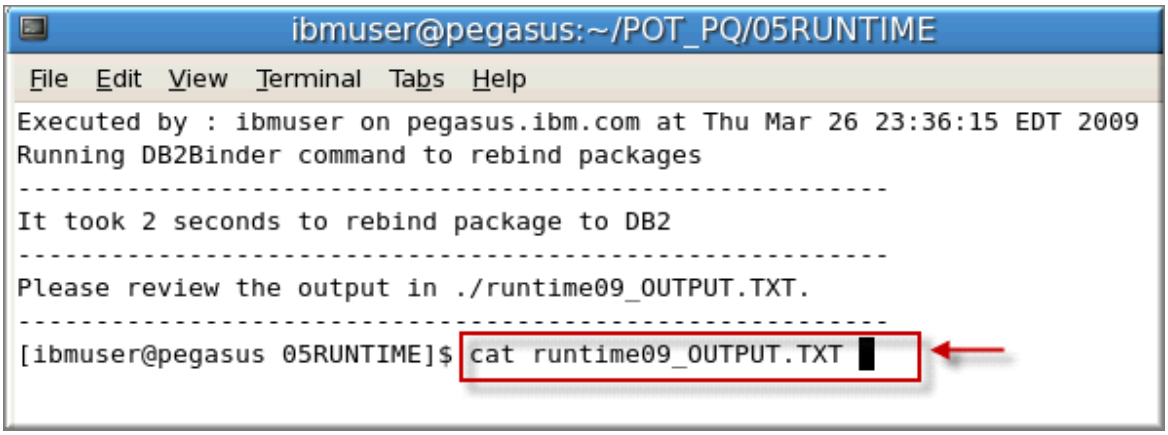
__32. In command window, review script `runtime09`. This is an example script that can be used and customized to `REBIND` DB2 packages.

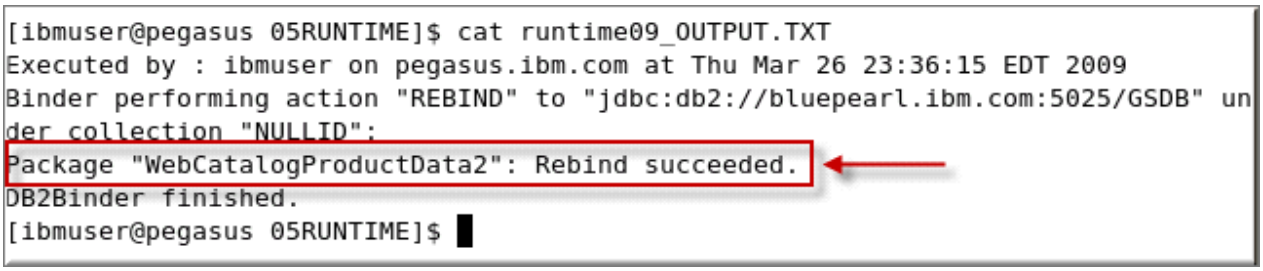



```
CLPATH=$JCC_HOME/db2jcc.jar:$JCC_HOME/db2jcc_license_cisuz.jar
CLPATH=$CLPATH:$PQHOME/pdq.jar
CLPATH=$CLPATH:$PQHOME/pdqmgmt.jar

URL="-url jdbc:db2://bluepearl.ibm.com:5025/GSDB"
USERINFO="-user dbapot -password dbapot"
BINDER=com.ibm.db2.jcc.DB2Binder

echo Running DB2Binder command to rebind packages

$JAVA -cp $CLPATH $BINDER $URL $USERINFO -collection NULLID -action REBIND -gene
ric -package WebCatalogProductData2 >> $OUTFILE
```

__33. Run script `runtime09`.

__34. Review the `runtime09_OUTPUT.TXT` to notice that the interface has been rebound to DB2

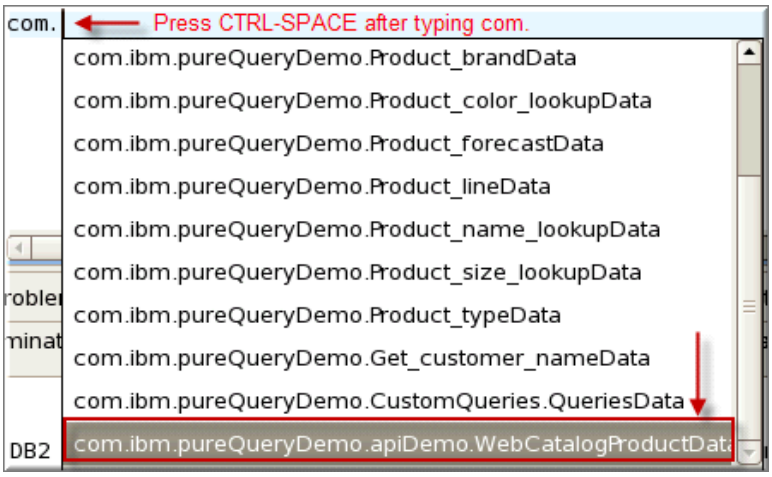




5.6 Customize `BIND` options for DB2 packages

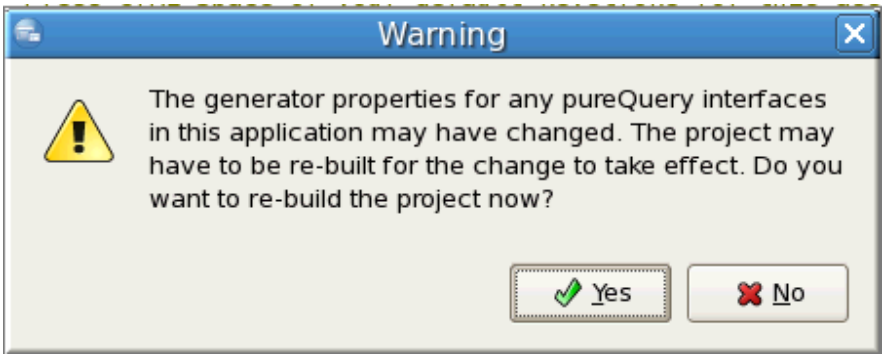
From Data Studio Developer, you can set a number of *pureQuery* properties to be associated with the project. Several of these are input to the *Interface implementation Generator*, which creates a working version of the interface whenever that interface file is saved. Some of those options are saved in the compiled implementation and become input to the `BIND` process. The following step demonstrates how to modify those properties.

__35. Double-click on `Default.genProps` in `pureQueryFolder` of `pureQueryLabs` project. Add following line to force collection schema to be `PDQCOL` instead of the default value of `NULLID`. You can use context sensitive help while selecting the interface name.



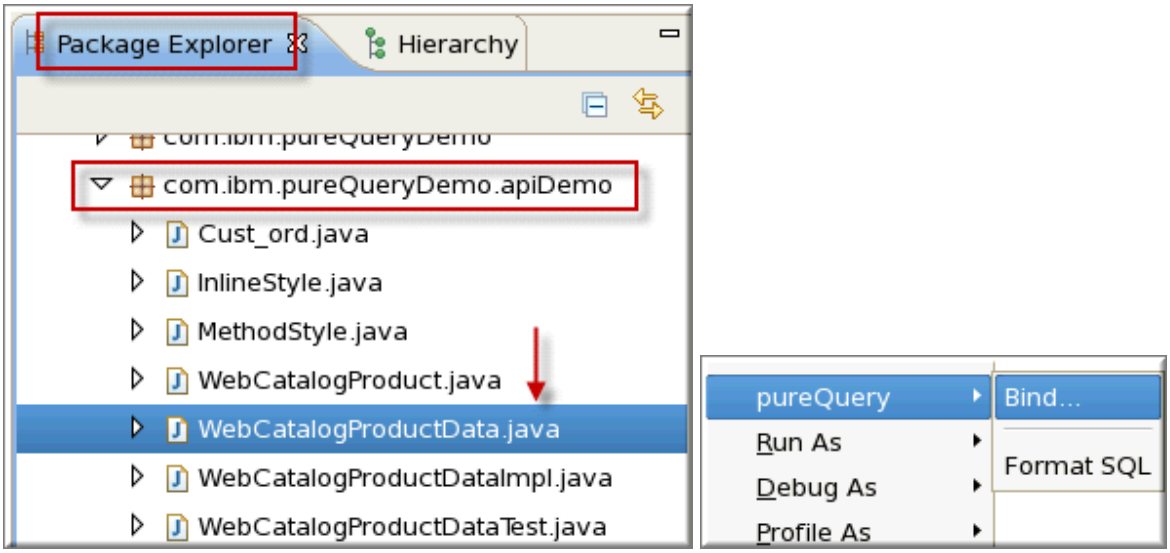
```
com.ibm.pureQueryDemo.apiDemo.WebCatalogProductData = -collection PDQCOL
```

__36. After you save this file (right click, then Save) it will show a message indicating that the project will be rebuilt since options specified in this file are applicable to interfaces generated. Click <Yes>.



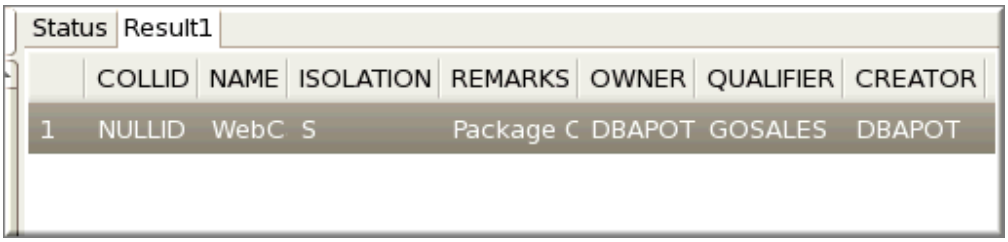
__37. Open `WebCatalogProductData.java` file. Delete any character and re-type same character and save the file.

__38. Re-bind the interface by right clicking on it and selecting `pureQuery` ⇨ `Bind`. Select `GSDB` database when prompted.

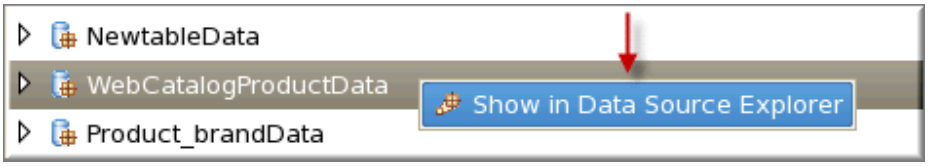


__39. Go to the *Package Explorer* and double click `Runtime04.sql` file in an editor. When prompted, select `GSDB` database. Click on `Script` ⇨ `Run SQL`.

__40. View the output of the command in *Results* view.



__41. Go to *pureQuery Outline* view and go to the `SQL` tab (Located at the bottom of the pane) and right click on `WebCatalogProductData` package and right click to click on `Show in Database Explorer`.



** **End of Lab 5: Explore pureQuery Runtime**

Lab 6 - pureQuery Explain

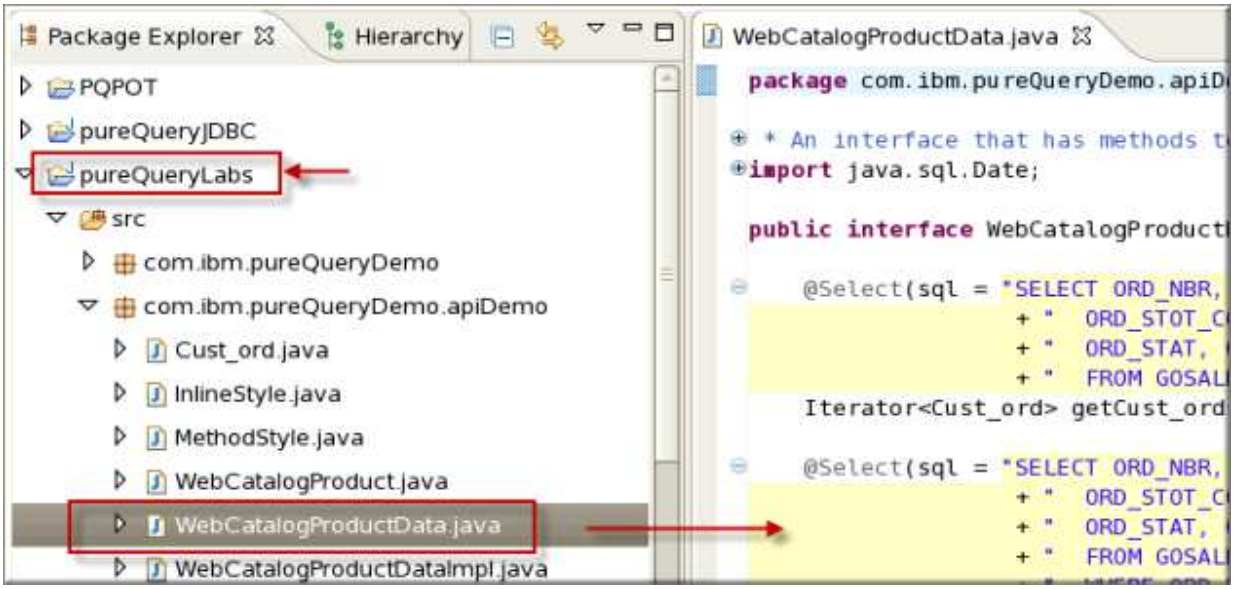
Introduction:

With *Data Studio Developer*, you can see the explain plan of the SQL statements which are embedded in your Java programs. Most importantly, you neither have to leave the *Data Studio Developer* nor reformat and copy SQL statements to any other tool.

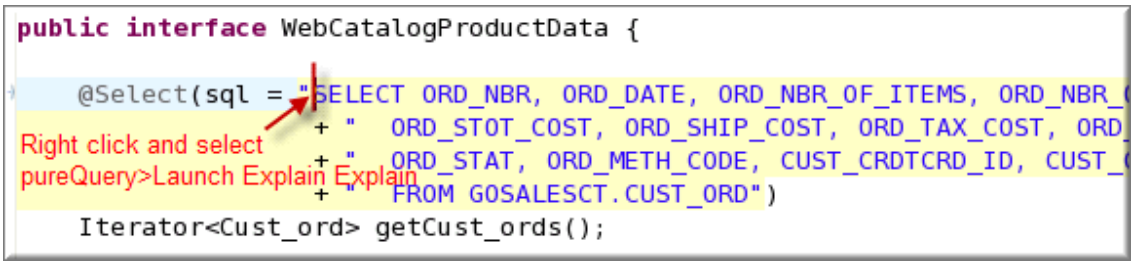
6.1 Explain Plan for SQLs in Java Programs

__1. Go to menu `File` ⇨ `Close All` to close all open files.

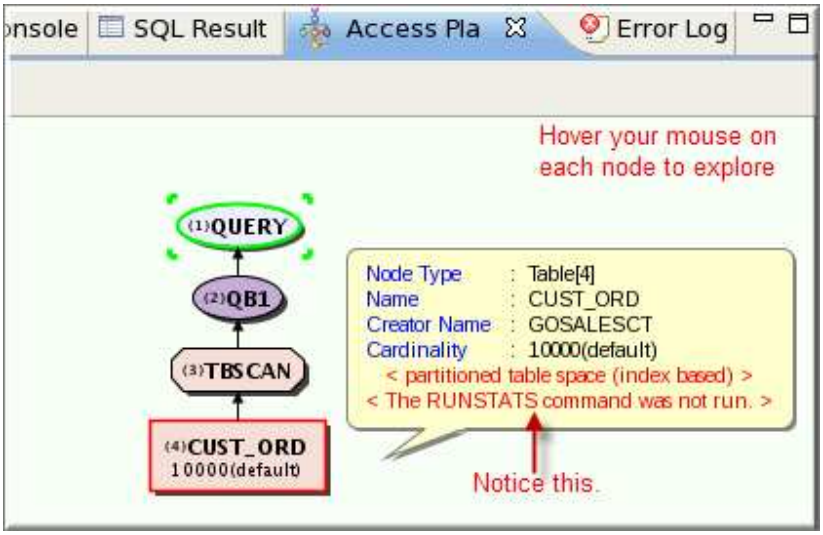
__2. In your *Package Explorer*, expand `apiDemo` package in `pureQueryLabs` project and double click on `WebCatalogProductData.java` to open it in an editor.



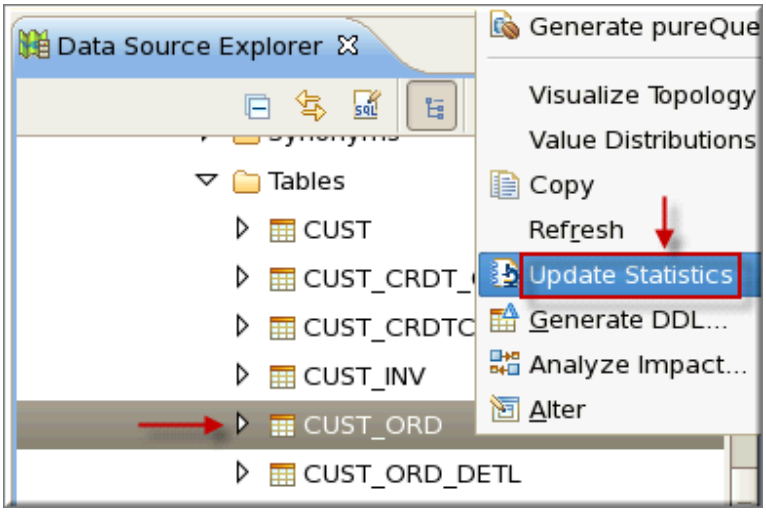
__3. Click anywhere on the first SQL statement and right click to select `pureQuery` ⇨ `Launch Visual Explain`.



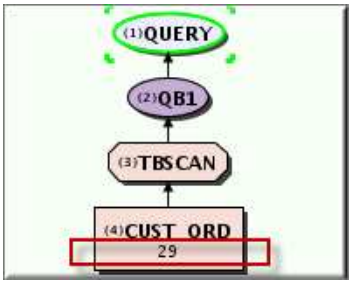
__4. When Visual Explain screen shows up, click <Finish> to launch it. Look at the explain plan in *Access Plan Diagram* in bottom right corner of the Java perspective.



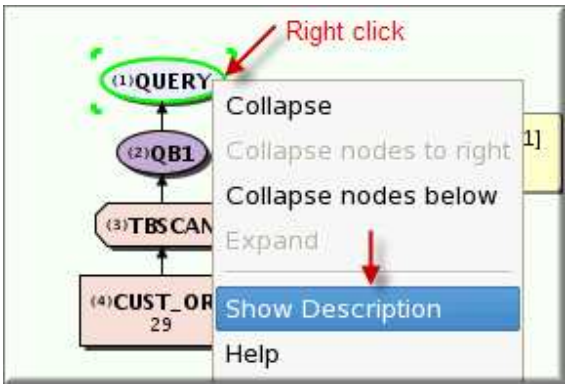
__5. In *Data Source Explorer*, right click on the GOSALESCT ⇨ CUST_ORD table and select Update Statistics option to create statistics.



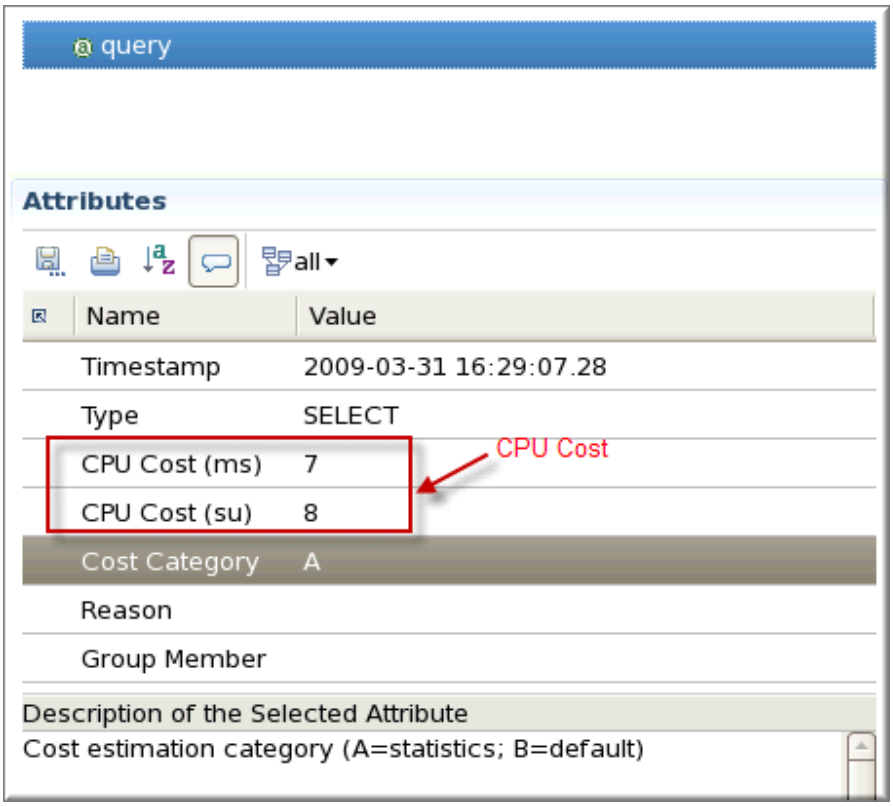
__6. Re-take explain plan of the same query again and see the difference in the access path. What did you notice in this access path?



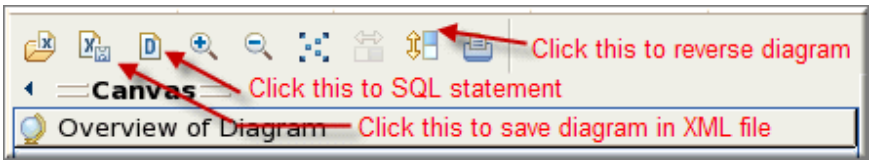
__7. Right click on the `Query` node in the visual explain plan and select `Show Description`.



__8. Go through different description provided for the `Query` node.



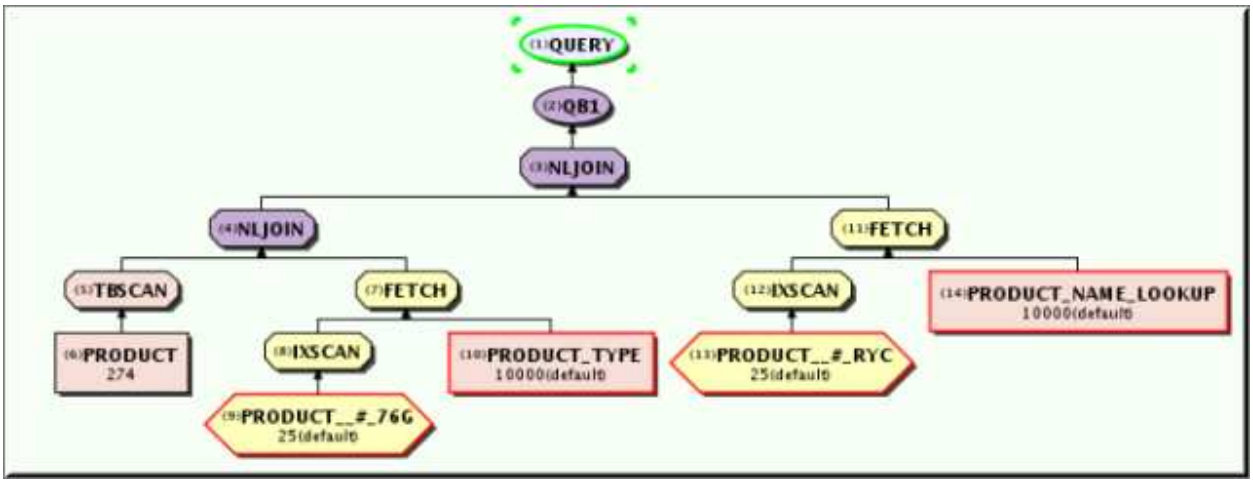
__9. In the *Overview Diagram,* click on `View the SQL statement` to see the SQL statement. You can save an explain plan in an XML file for viewing it later or for sending it to the DBA.



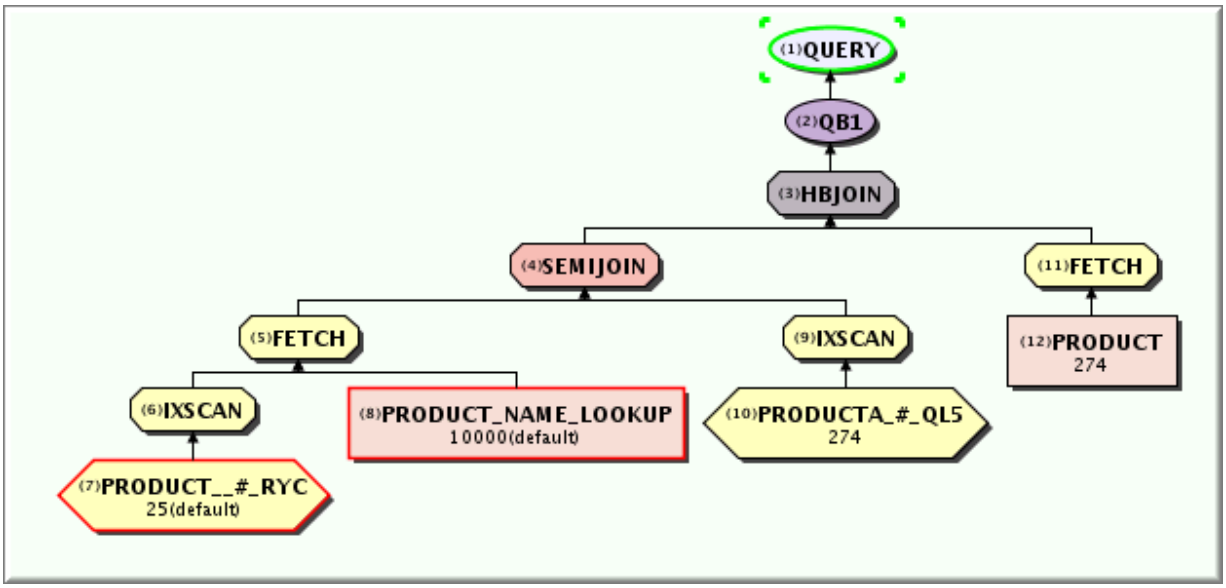
__10. Go back to the `WebCatalogProductData.java` program and try looking at the explain plan for each of the SQL statement. For example, look at the explain plan for the SQL associated with the method `getWebCatalogProductByType`. Try to figure out the tables, which are missing statistics. Create statistics on the tables and re-create the explain plan.

```
@Select(sql = "SELECT P.PRODUCT_NUMBER, Q.PRODUCT_NAME, Q.PRODUCT_DE
               + "   P.PRODUCTION_COST, P.PRODUCT_IMAGE"
               + "   FROM GOSALES.PRODUCT AS P, GOSALES.PRODUCT_NAME_
               + "        GOSALES.PRODUCT_TYPE AS R"
               + "   WHERE P.PRODUCT_NUMBER = Q.PRODUCT_NUMBER AND Q.
               + "     AND R.PRODUCT_TYPE_CODE = P.PRODUCT_TYPE_CODE
Iterator<WebCatalogProduct> getWebCatalogProductByType(String param1
```

Try taking explain plan of this query



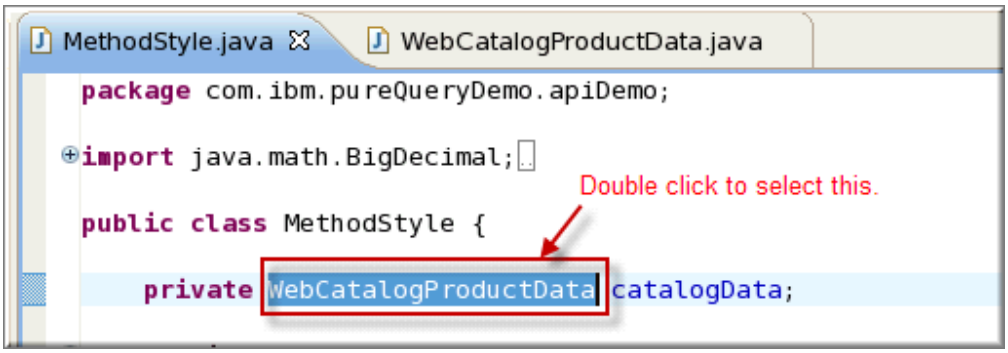
__11. Look at the explain plan for the query associated with the `getWebCatalogProduct` method.



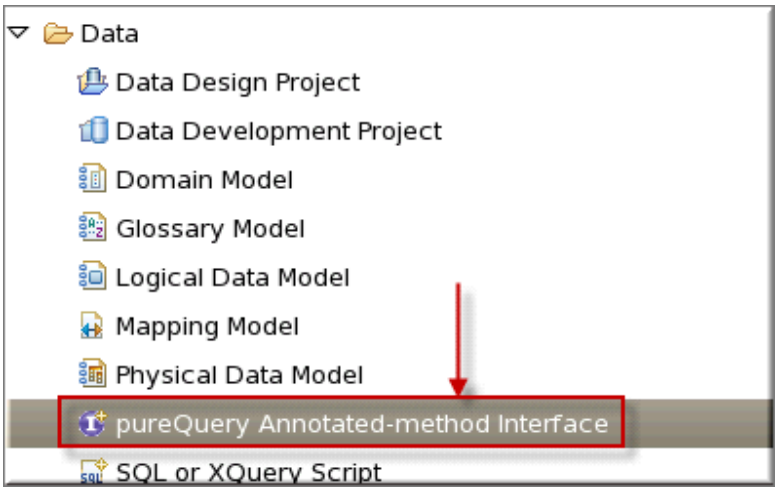
6.2 Explain Plan for new methods

__12. Go to the *Package Explorer* and click `apiDemo` package to select it.

__13. Go back to the `MethodStyle.java` program and double click on `WebCatalogProductData` to select it.

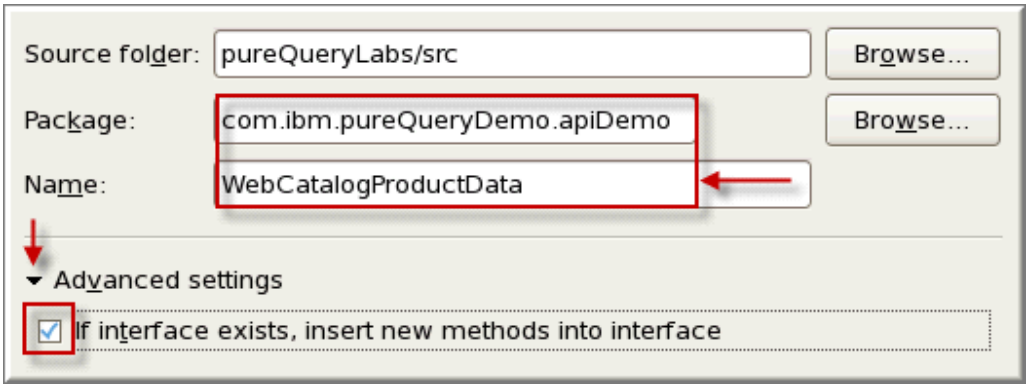


__14. Hit `CTRL-N` to open a new wizard. Expand Data and select `pureQuery Annotated-method Interface`. Click on `<Next>`.

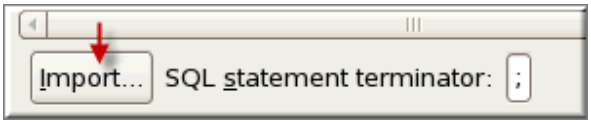


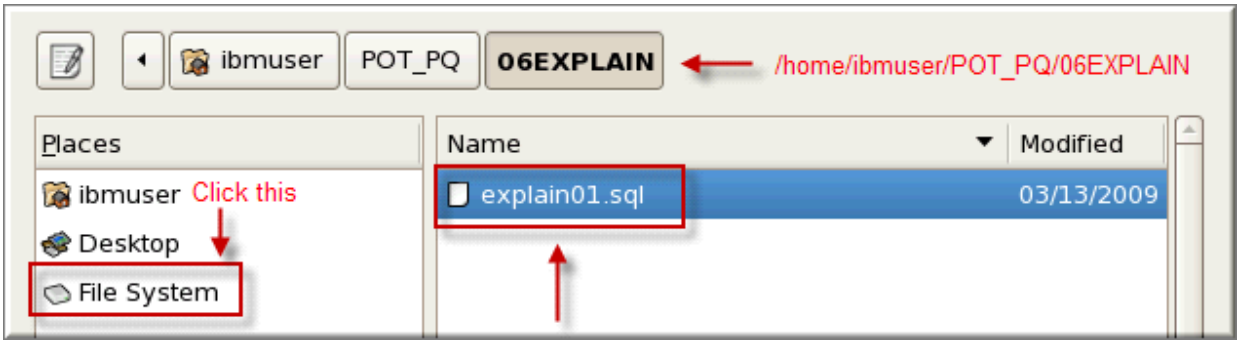
__15. Make sure to expand the `Advanced Settings` and click on `If Interface exists, insert new methods into interface`.

__16. The other two values should already be selected for you. If not, type-in those values. Click `<Next>`

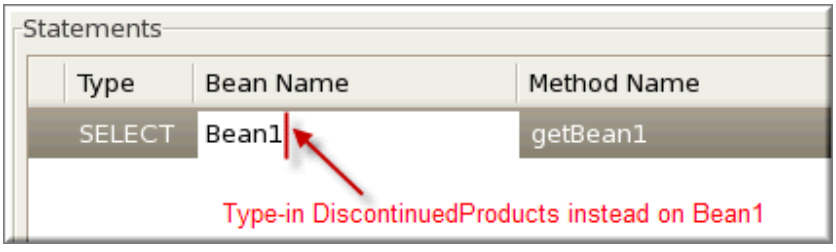


__17. Click on `Import` button and change directory to `/home/ibmuser/POT_PQ/06EXPLAIN` and select `explain01.sql` to import the SQL statements from this file.



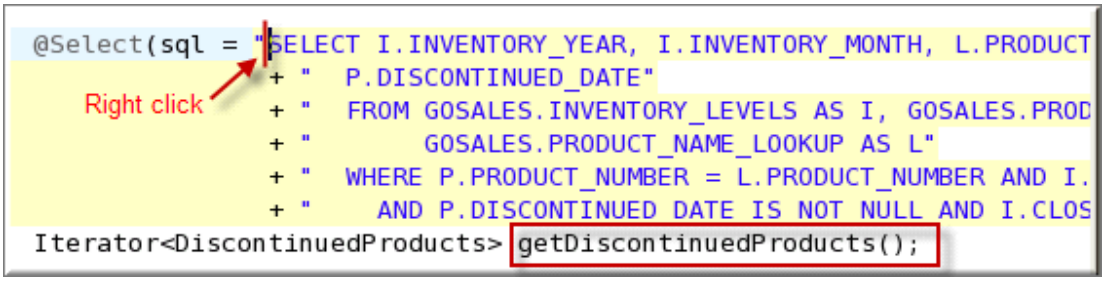


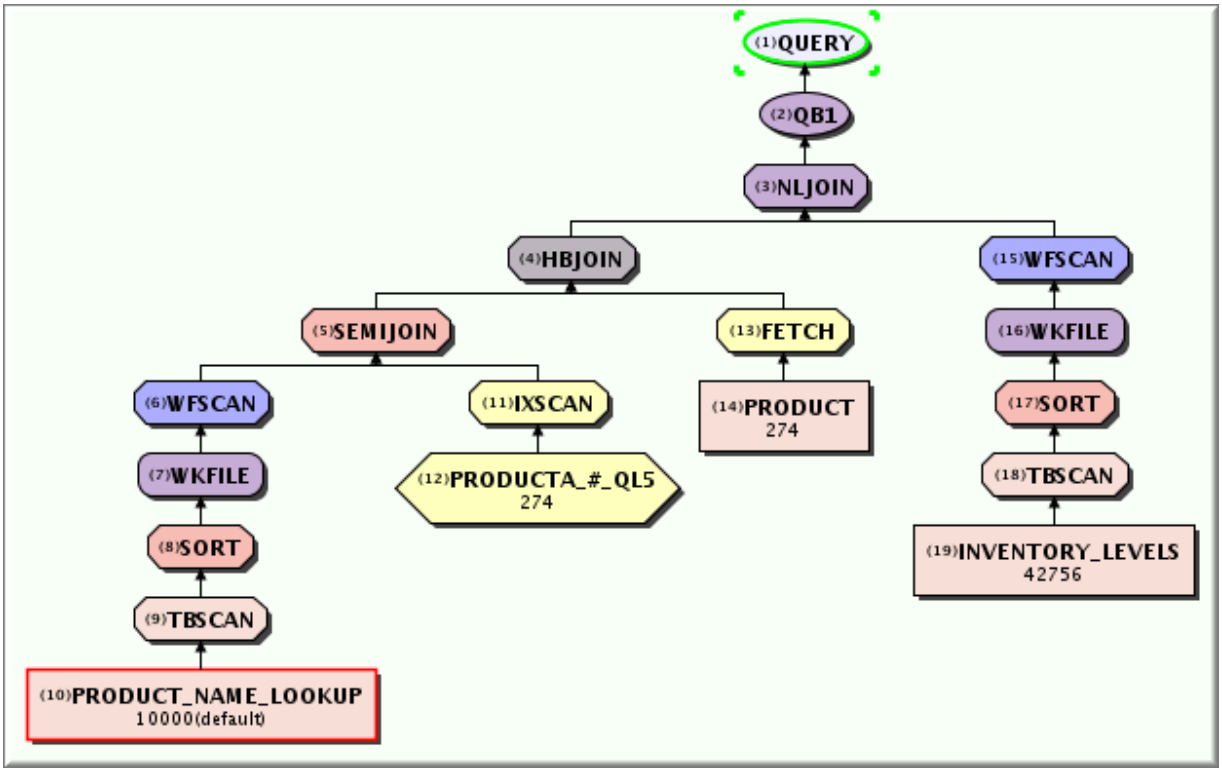
__18. In the dialog box that appears, click on `Bean1` under Bean Name and type name `DiscontinuedProducts` as the name of the bean. Hit TAB and the method name is generated automatically.



__19. Click <Finish> to generate a bean to hold the results.

__20. The method `getDiscontinuedProducts` is added to `WebCatalogProductData` interface. Click anywhere on the SQL statement and right click to select `pureQuery` ⇨ `Launch Visual Explain` and hit <Finish> to generate an explain plan. The explain plan may look like following.





** **End of Lab 6**: **pureQuery Explain**

Lab 7 - Optimize an existing JDBC Application using pureQuery

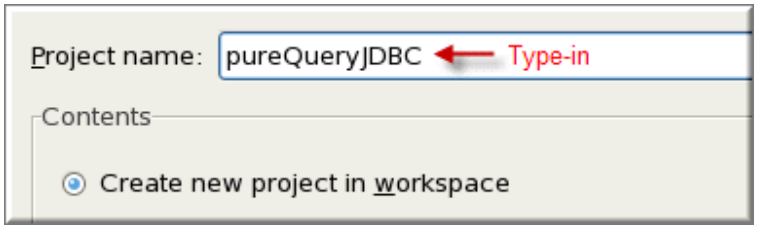
IBM Data Studio Developer pureQuery feature allows you to optimize existing JDBC™ applications. The pureQuery features allow you to optimize custom or packaged JDBC applications to execute SQL statements statically without a need to change the application in any way.

7.1 Create a Java Project

__1. Close all open files by clicking <CTRL><SHIFT><W>. If you have a command window open from the previous lab, close it.

__2. Switch your perspective to Java. Click menu Window ⇨ Open Perspective ⇨ Other… Click on Java and click OK.

__3. Create a new Java project. Click File ⇨ New ⇨ Java Project. Specify project name as pureQueryJDBC and click <Finish> to create the project.

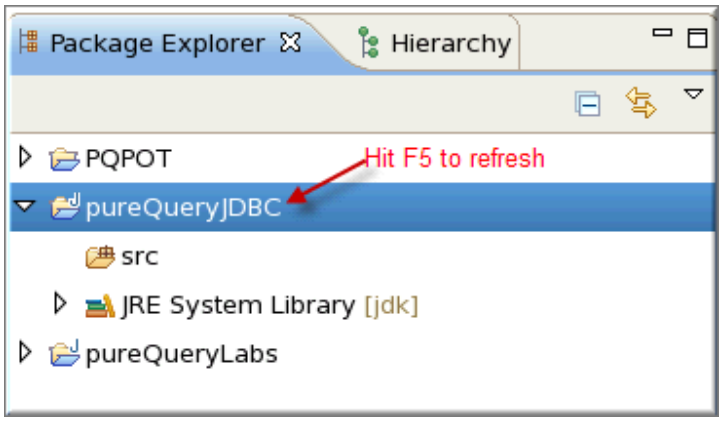


Note: Please make sure that you type the name of the project **exactly** as given above.

__4. Double click on the *Terminal* icon on the desktop to open up a command window.

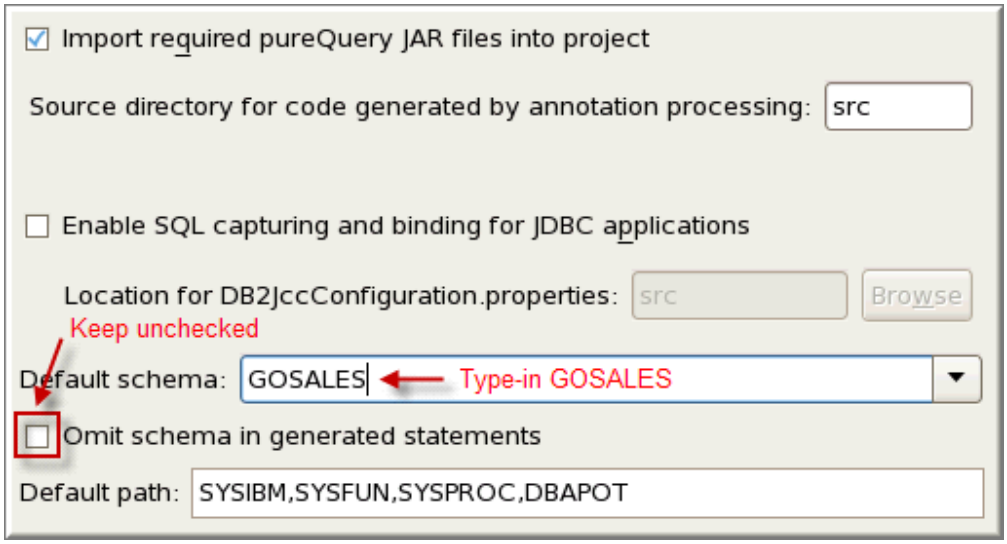
__5. Change your directory to: 07JDBC and run jdbc01 command to copy jdbc02.java in the pureQueryJDBC java project.

__6. In *Package Explorer* view, select `pureQueryJDBC` project and hit `F5` to refresh.



__7. Right click on `pureQueryJDBC` project, select `pureQuery` and `Add pureQuery Support …`

__8. Select `GSDB` database and click `<Next>`. Use default schema `GOSALES`. Click `<Finish>` to add pureQuery support to the project.



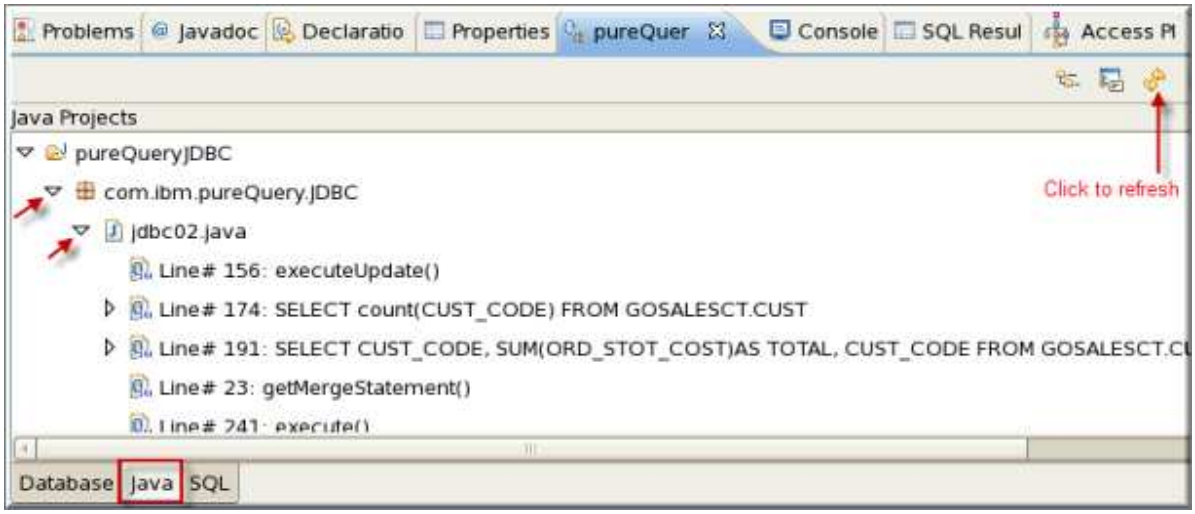
__9. The project `pureQueryJDBC` is now enabled for the pureQuery. Click `pureQueryJDBC` project in the *Package Explorer* to select it. We need this to select so that it shows up in the *pureQuery Outline* view in the next step.

7.2 SQL Profiling when source is available

In this section, you will run SQL profiling for an existing Java application.

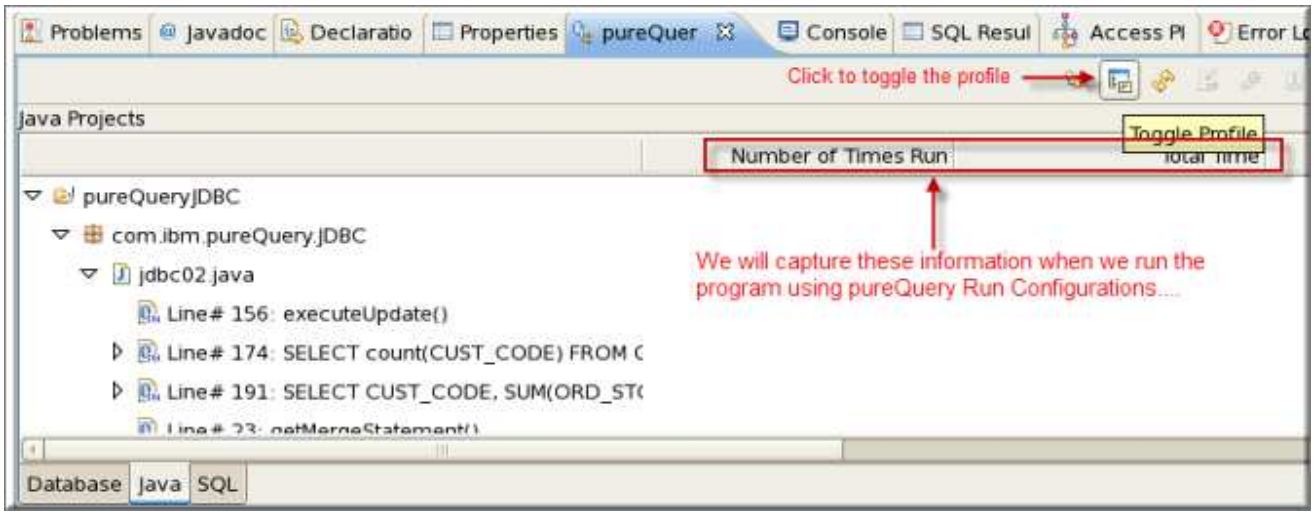
__10. Double click on `jdbc02.java` to open it in the editor.

__11. Go to the *pureQuery Outline* view and select `Java` tab at the bottom on the view. Click on refresh button and expand `jdbc02.java` under `JDBC` package.

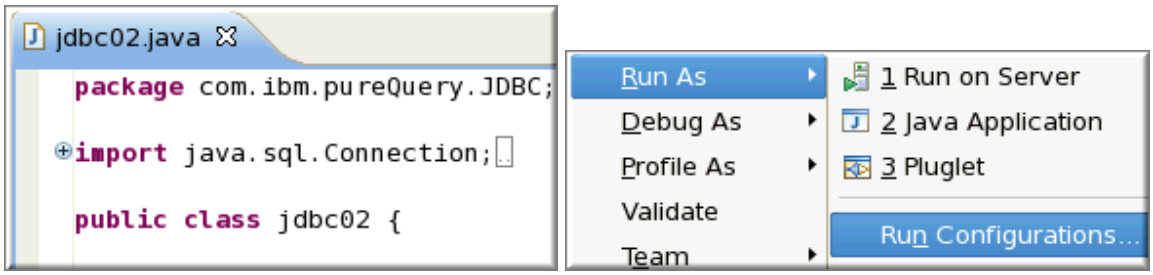


__12. You will notice the line numbers where SQL is getting executed. This information is captured even though you have not run the program.

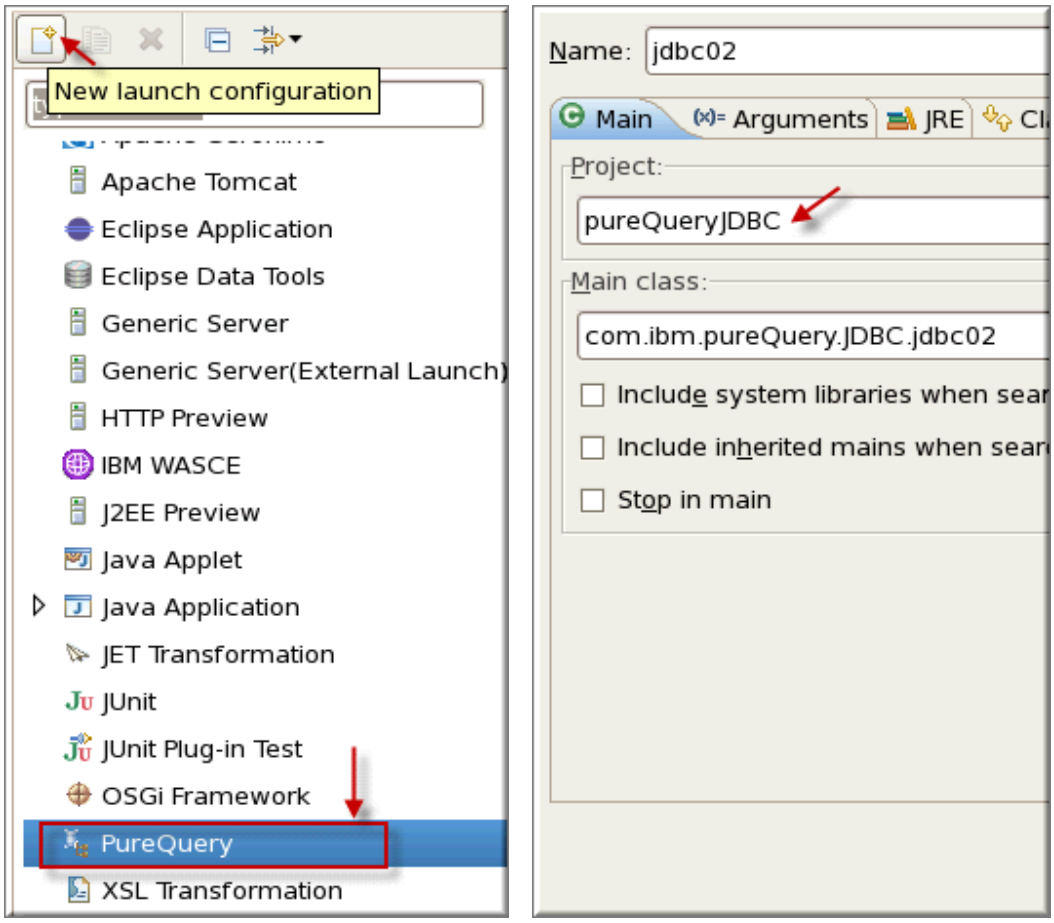
__13. Click on Toggle Profile button on the view to see the view for the SQL profiling. At this time, we will not see SQL profile data since we did not run the program.



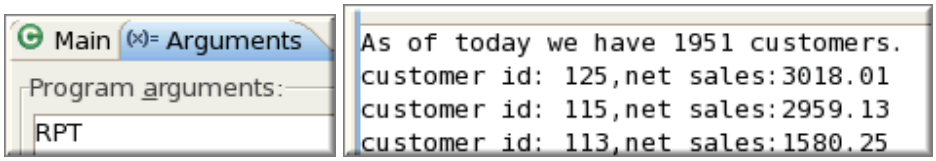
__14. Right click within `jdbc02.java` program and select `Run As` ⇨ `Run Configurations…`



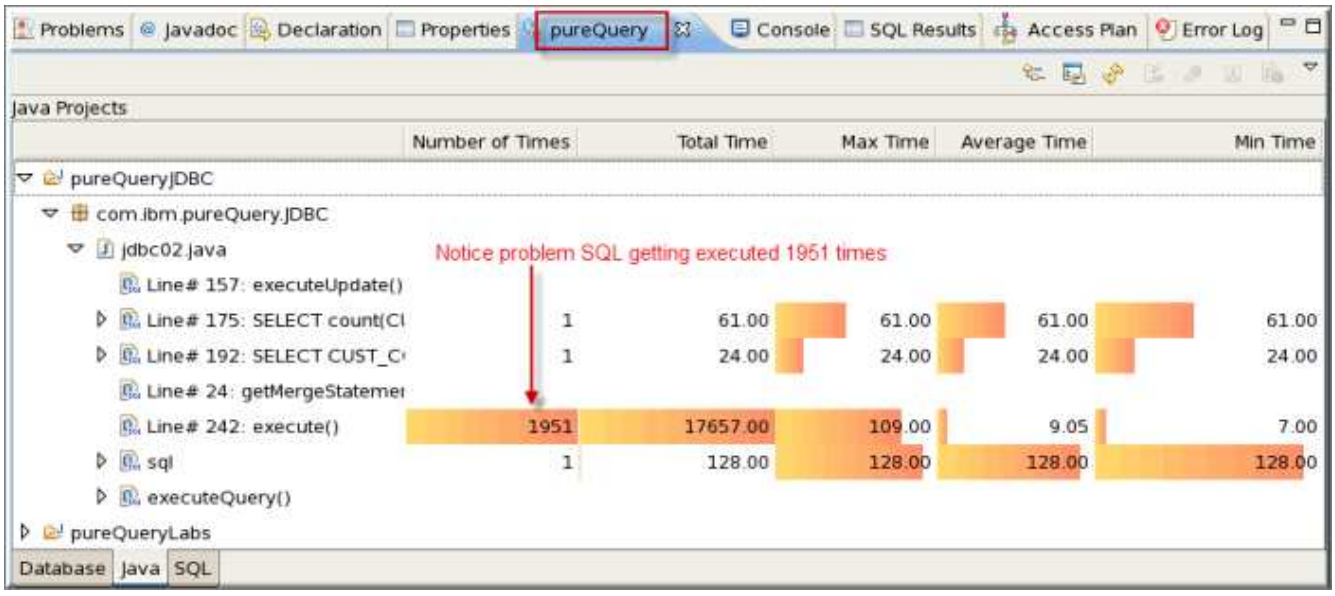
__15. Click on pureQuery to select it first. Click on `New launch configuration`. You will see right hand side window populated with `jdbc02.java` information.



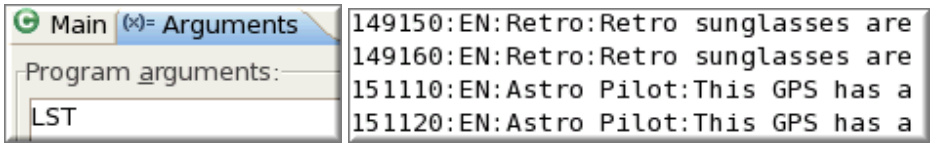
__16. Click on the `Arguments` tab and specify `RPT` and click `Run` to run the program. You will see following console.



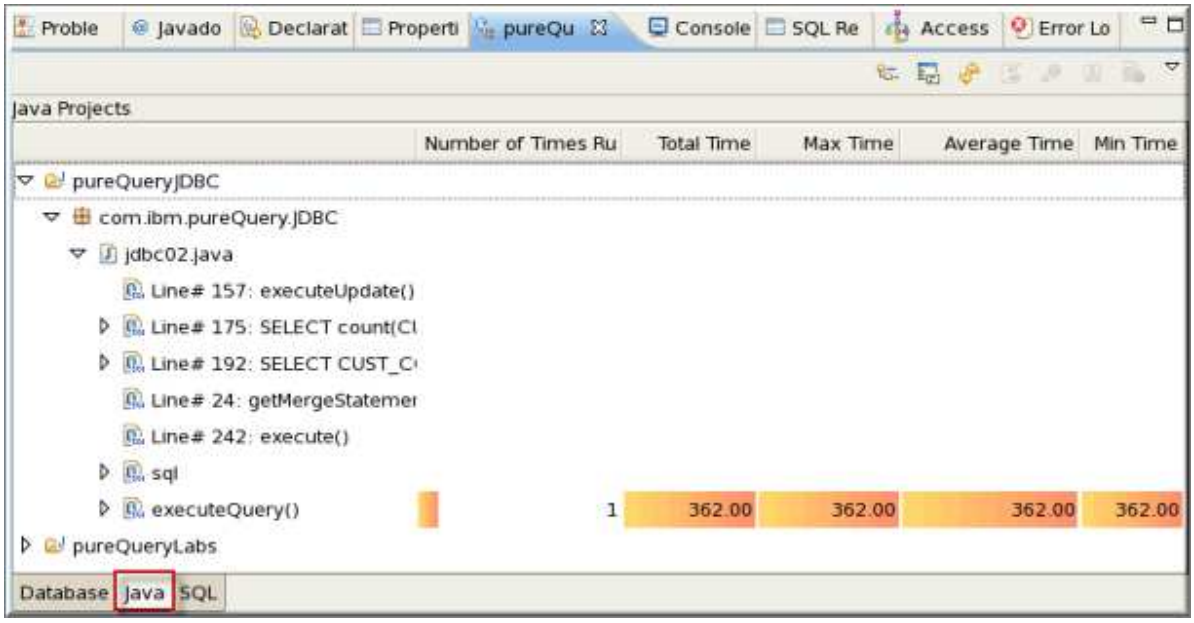
__17. Go to the *pureQuery Outline* view and hit the refresh button. You will see a view similar as shown.



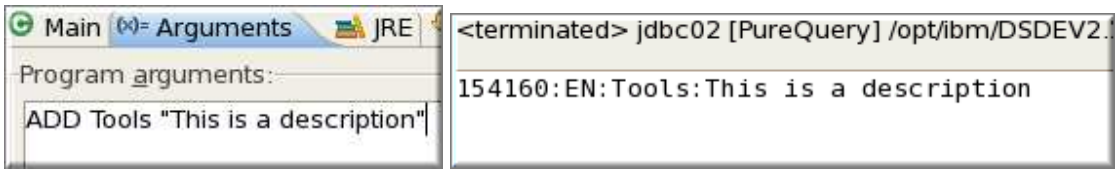
__18. Click on menu `Run` ⇨ `Run Configurations….` Click on `Arguments` tab and specify `LST` option. Click Run to run the program.



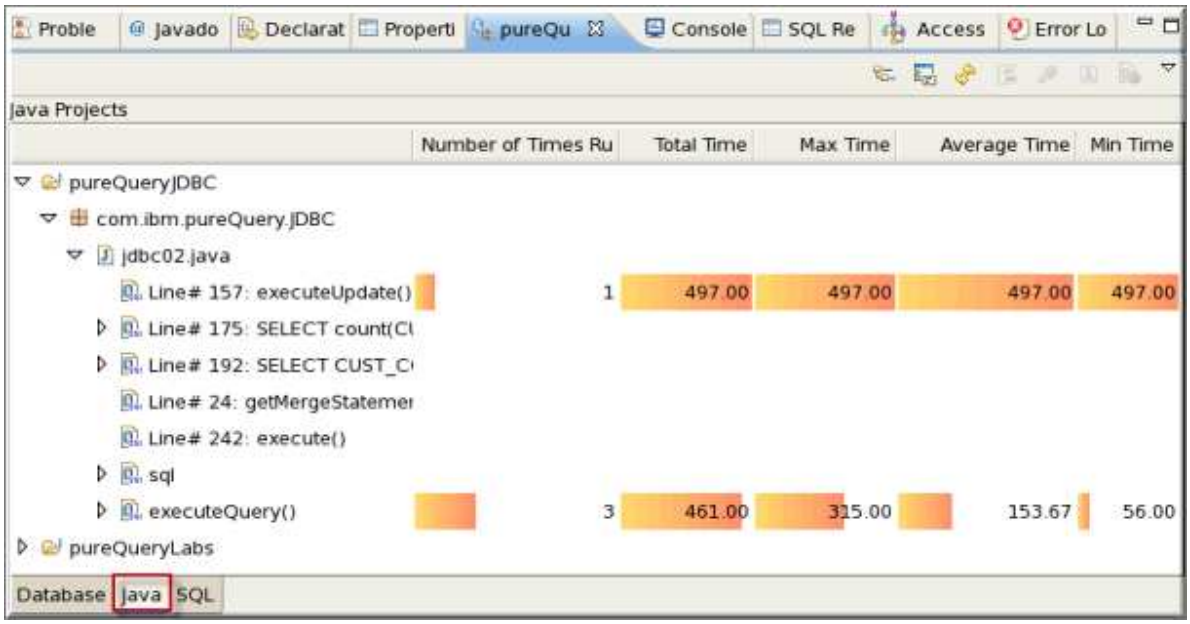
__19. Go to the *pureQuery Outline* view and hit the refresh button. You will see a view similar as shown.



__20. Click on menu `Run ⇨ Run Configurations….` Click on `Arguments` tab and specify `ADD TOOLS "This is a description"`. Click Run to run the program.



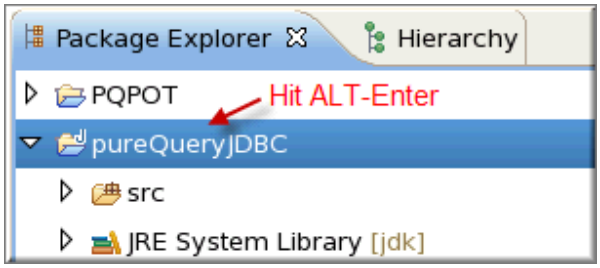
__21. Go to the *pureQuery Outline* view and hit the refresh button. You will see a view similar as shown.



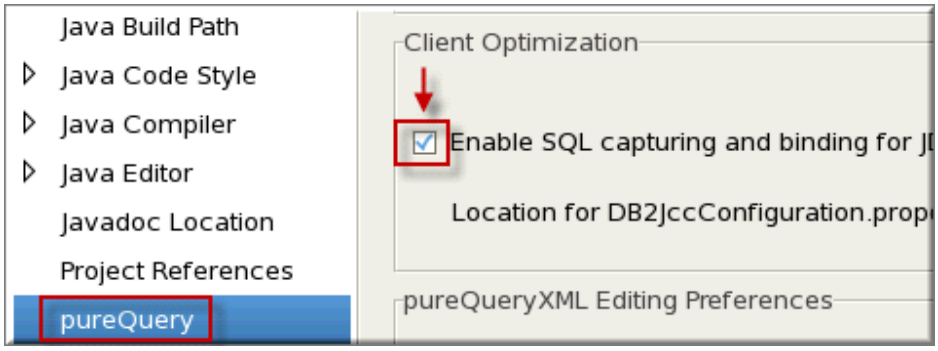
7.3 Optimization when source is available

In this section, we will capture metadata to enable optimization using pureQuery.

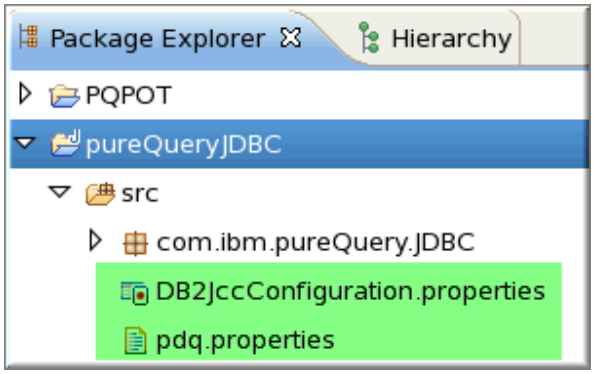
__22. Click `pureQueryJDBC` project and hit `ALT-ENTER` to open properties.



__23. Click on pureQuery and select the check box for `Enable SQL capturing and binding for JDBC applications`.



__24. After we enable SQL capture, you will notice addition of 2 files in the Java project as shown.

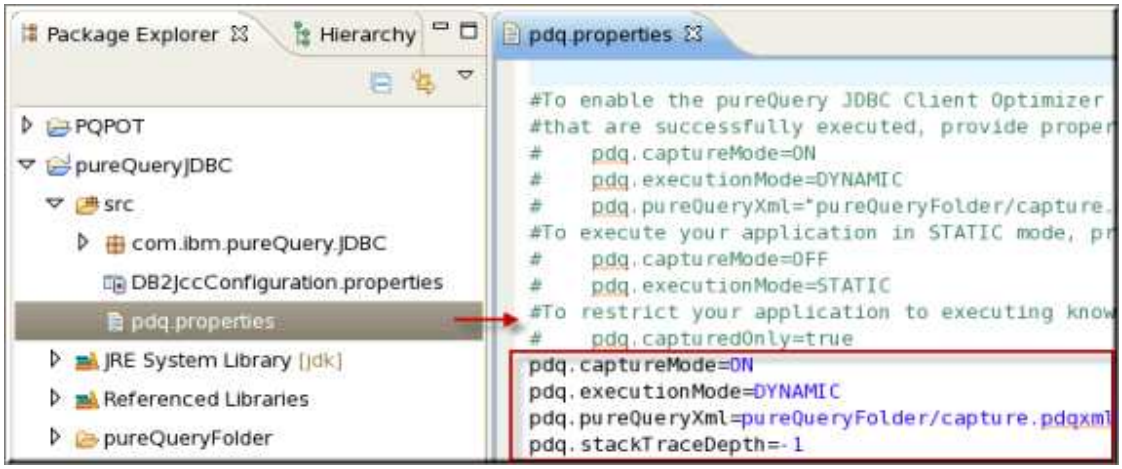


__25. We will be running this program using scripts given in `/home/ibmuser/POT_PQ/07JDBC` so that you do not have to keep on modifying the program arguments for each and every step. This has been done for your convenience. This program uses JDBC calls to do `SELECT`, `MERGE` and `DELETE` against `GOSALES.PRODUCT_NAME_LOOKUP` table. We will run the program by using different test cases to capture SQL metadata through the command line.

__26. Close all open files by clicking `<CTRL><SHIFT><W>.`

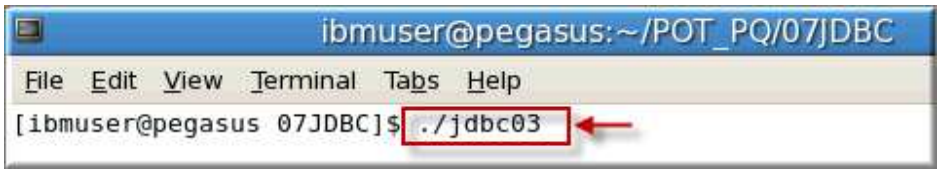
7.3.1 Capture metadata

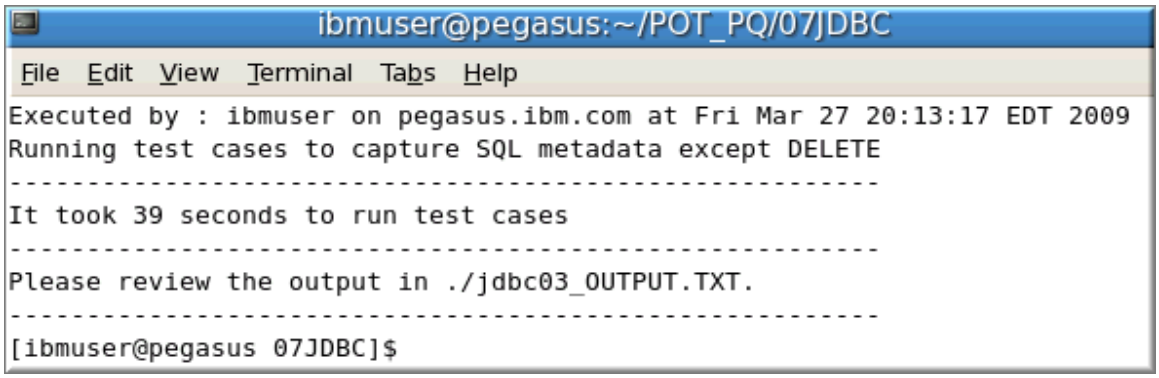
__27. Double click on `pdq.properties` to open it in an editor. It has `pdq` properties set to capture the SQL metadata.



__28. Go to the *Command Window* and make sure that you are in `/home/ibmuser/POT_PQ/07JDBC` directory and review the contents of `jdbc03`.

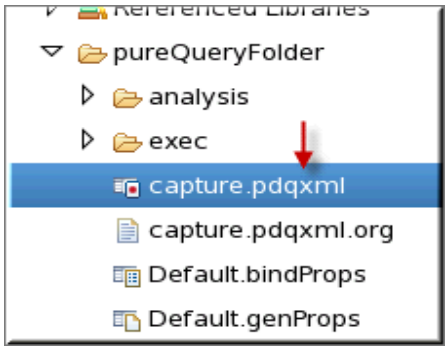
__29. Run script `jdbc03` to run the JDBC application with different test cases and to capture SQL metadata.

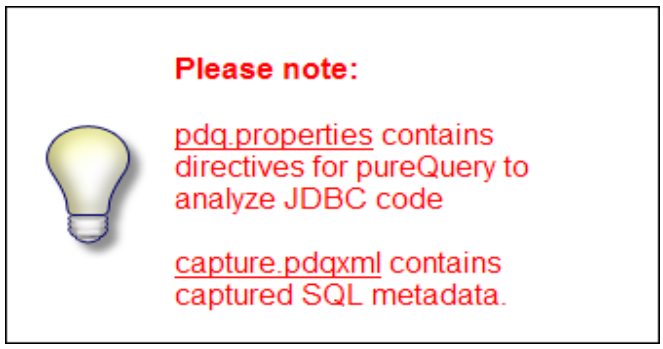




__30. Review `jdbc03_OUTPUT.TXT` file. This contains output from our custom JDBC application.

__31. Go to the *Package Explorer* and notice `capture.pdqxml` in `pureQueryFolder`. This file will contain SQL metadata when we run our custom JDBC application in the next step. (Press F5 to refresh your *Package Explorer* if you do not see this file.)



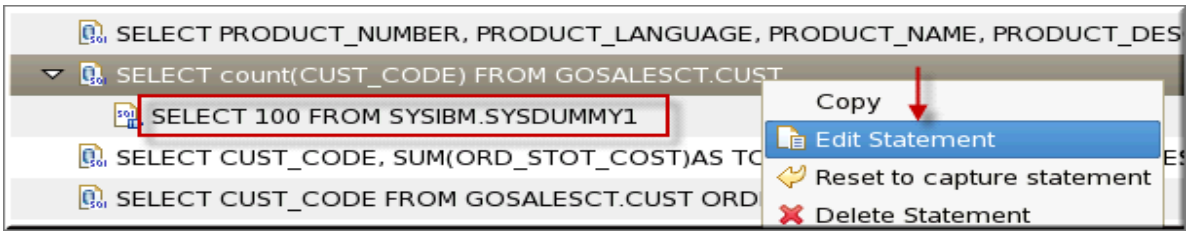


7.3.2 Browsing the captured metadata

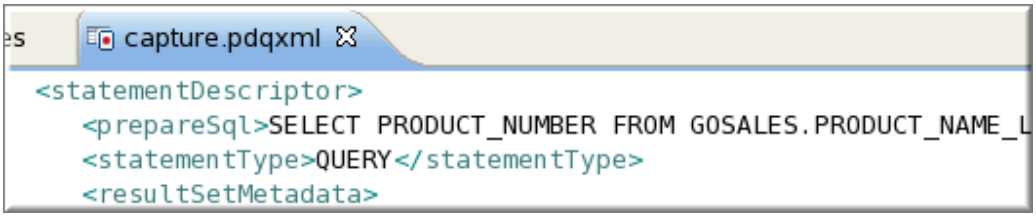
__32. In the *Package Explorer* and hit F5 to refresh it. Expand folder `pureQueryFolder` and double click `capture.pdqxml`. You can view this file in two modes 1. In Edit mode and 2. In View Source mode.

- In `Edit` mode. You can view each SQL statement captured and you can choose if you want to Bind that statement or not. You also have an ability to edit a SQL statement to replace original SQL statement with a new optimized without modifying the application. You can change the statement only to the extent where new SQL statement is equivalent to the original SQL if its input and output are identical. The new SQL statement is stored as a child node of the original statement.

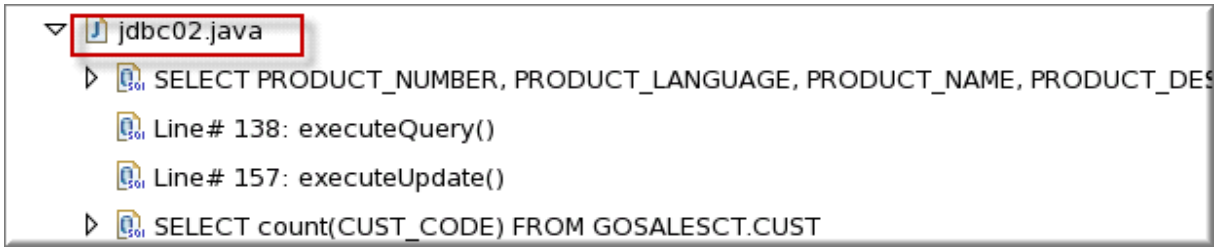
- Go ahead and modify `SELECT count(CUST_CODE)` statement to `SELECT 100 FROM SYSIBM.SYSDUMMY1` where you replaced original statement with a fixed return value of 100.

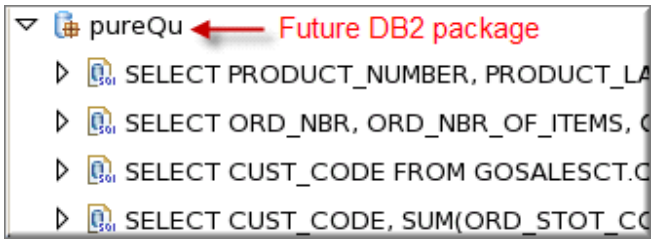


- In `View Source` mode

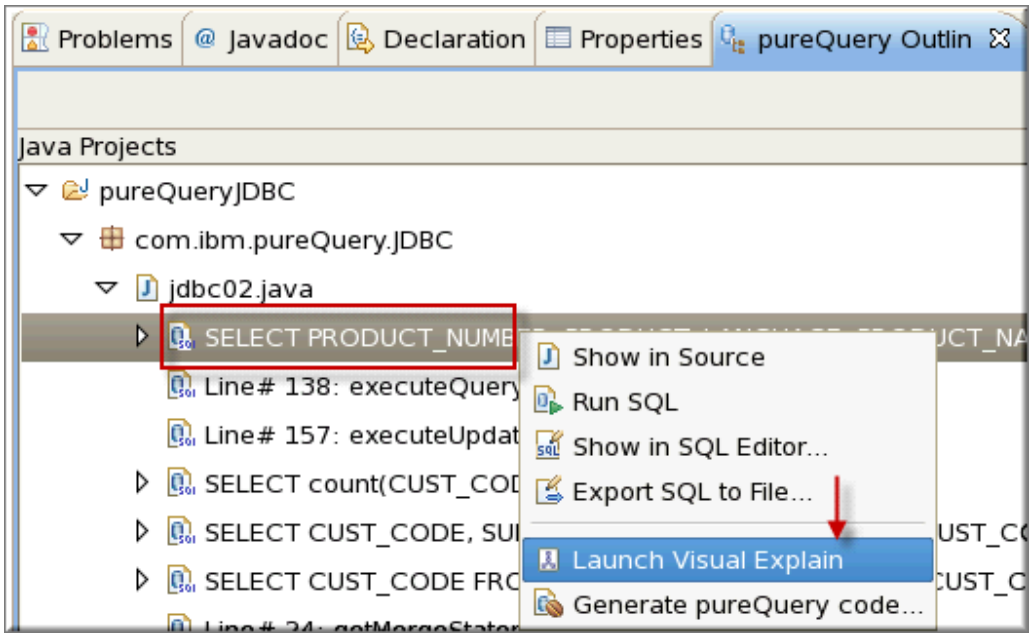


__33. Navigate to the *pureQuery Outline* view. Explore the contents of the `Java` and `SQL` tabs to browse same information in different views. You will notice actual SQL statements now.

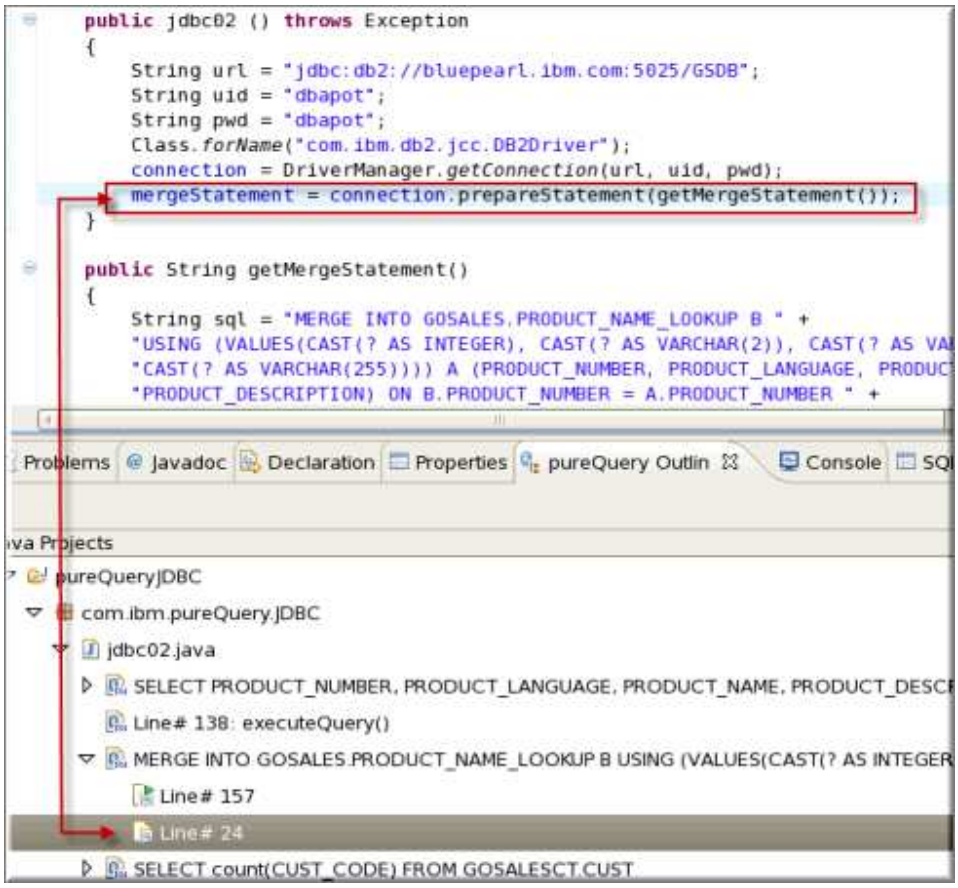




__34. You can do a number of activities on the SQL shown in the `pureQuery Outline` view. Right click on any SQL in any view to explore different actions. Try seeing explain plan for the `SELECT` statement used in the JDBC application.



__35. You can double click the SQL statement or stack trace element to pen the source code in an editor window. The cursor will be positioned on the source line where the statement is being executed.



7.3.3 Configuring captured metadata

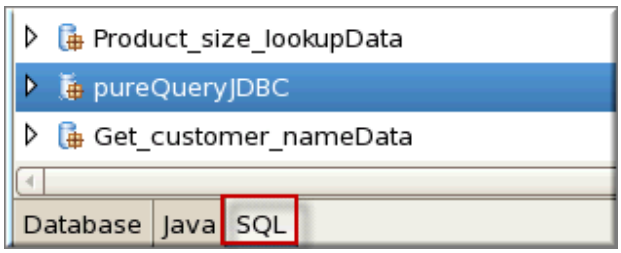
__36. Before captured metadata can be bound to a database in the form of a package, you will need to define the package properties. In the *Package Explorer*, make sure you are positioned in the `pureQueryJDBC` project, expand the folder `pureQueryFolder` and open the file `Default.genProps` by double clicking on it. Through this configuration file you can define package properties such as:

- Package name prefix
- Database collection id (or schema name containing package)
- If packages are versioned
- Maximum number of SQL statements that are to be included within a single package before a new one is created.

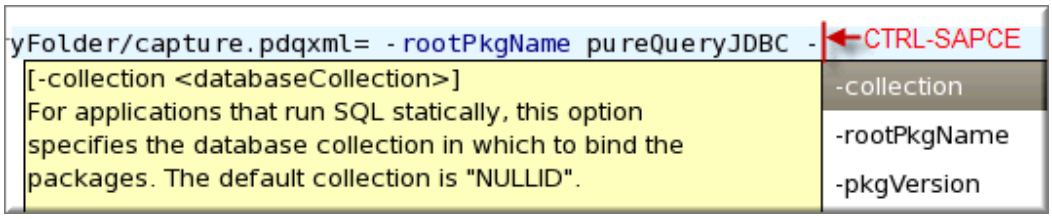
__37. In the pureQuery outline view, go to the SQL view. This view provides a preview of the packages that will be created based on the current configuration settings. You will notice the name of the package is [`pureQu`] and this name is selected since [`-rootPkgName pureQu`] is defined in the `Default.genProps` configuration file.

__38. Go and change this name to `pureQueryJDBC` in `Default.genProps` and save the configuration file. A warning will be displayed indicating that the configuration properties have been changed and a rebuild may be necessary. Click <Yes>.

__39. Go back to the SQL tab in the *pureQuery Outline* view and now you should see the package changed from [pureQu] to [pureQueryJDBC].

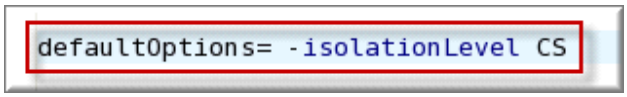


__40. Go back to the `Default.genProps` file and if you hit <CTRL><SPACE> at the end of the line, you can see other options available.



__41. Do not use any other property at this time and close the editor window containing `Default.genProps` file without saving it.

__42. Double click on the `Default.bindProps` file in `pureQueryFolder` in package explorer. To change the default options, enter `defaultOptions=` at the bottom of the file and hit <CTRL><SPACE> to invoke content assist and review the available options. Choose `–isolationLevel` and set the value to CS.

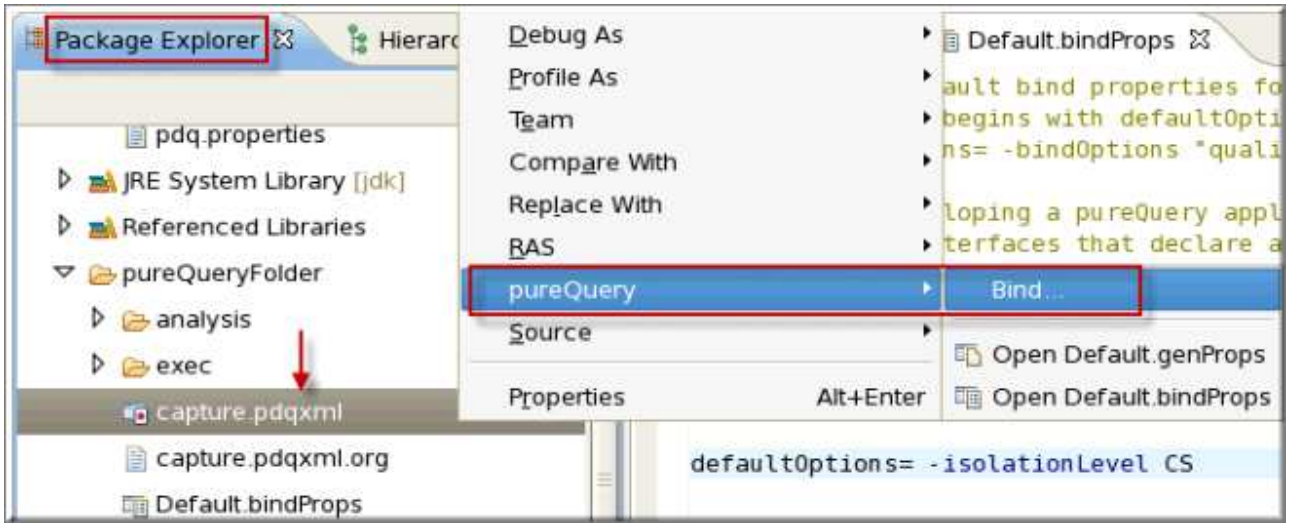


__43. Save the file (CTRL-S) and now you are ready to bind the captured SQL statements to DB2.

7.3.4 Binding captured SQL statements

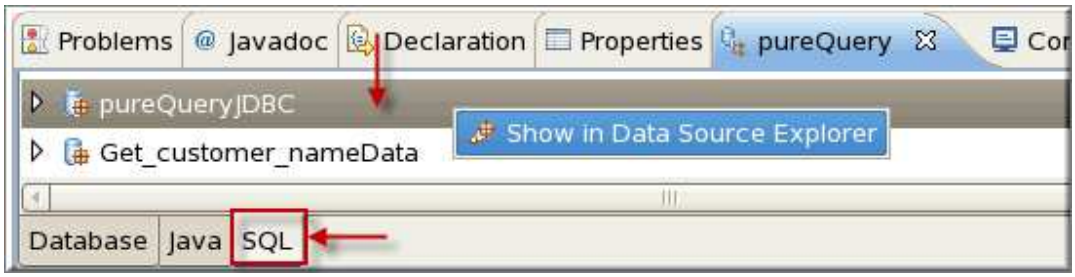
__44. In the *Package Explorer* navigate to the `pureQueryFolder` and select `capture.pdqxml` by clicking on it.

__45. Right click on it and select `pureQuery` ⇨ `Bind` … The bind wizard is displayed prompting for a database connection. Select `GSDB` database to bind the captured SQL statements from `capture.pdqxml` file to DB2 database.

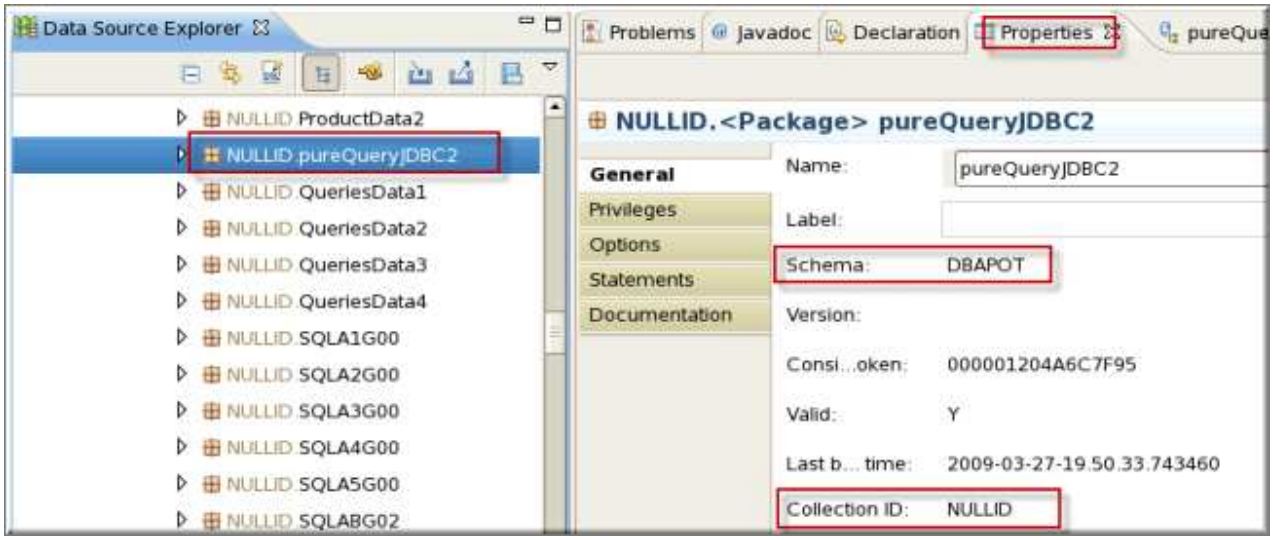


__46. Check *Console* view for the message.

__47. Navigate to the *pureQuery Outline* view and go to the `SQL` tab. Right click on `pureQueryJDBC` package and select `Show in Data Source Explorer`.



__48. Click on the `Properties` view to see the package characteristics.



Note: If you do not see Properties view, go to `Window` ⇨ `Show View` ⇨ `Other` … Expand `General` and click on `Properties.`

7.3.5 Run Application and Monitor Through Omegamon

Let us recap what we have done so far:

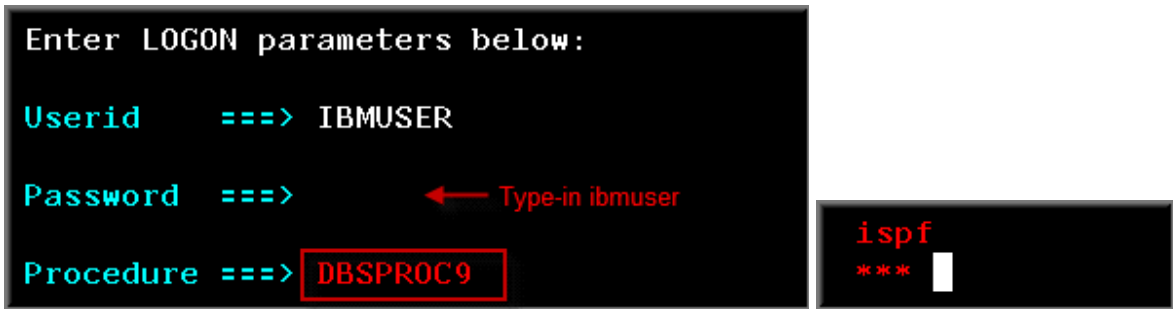
- Enabled the Java project for pureQuery JDBC.
- Captured the SQL statements by setting properties in `pdq.properties`
- Browsed the SQL statements and their associated metadata. Configured `Default.genProps` to set the `rootPkgName`.
- Bound the package

__49. Now, you will run application by modifying `pdq.properties` to set executionMODE to DYNAMIC or STATIC and monitor it through Omegamon.

__50. Double Click on the `TSO` icon on your desktop and type `L TSO` and hit right `CTRL` key.

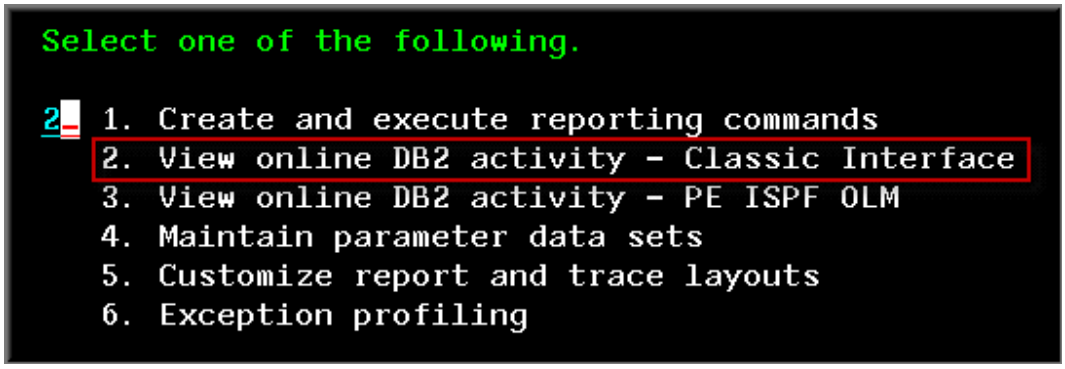
__51. Specify login ID as `IBMUSER` and hit right `CTRL` key to go to the login screen.

__52. Type-in password as IBMUSER and hit right CTRL key. When you see ISPF at the bottom of the next screen, hit right CTRL key.

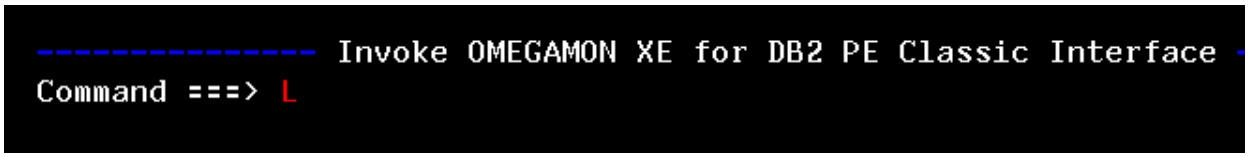


__53. Type-in [=m.13] at the command line and hit right CTRL key.

__54. Type option 2 and hit right CTRL key to launch classic Omegamon interface.

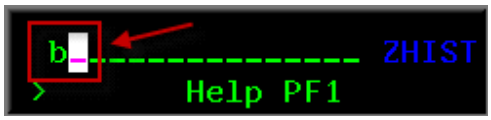


__55. Type-in L in the next screen to login to the classic Omegamon. Keep default options as it is. Press right CTRL key in the next copyright message screen.

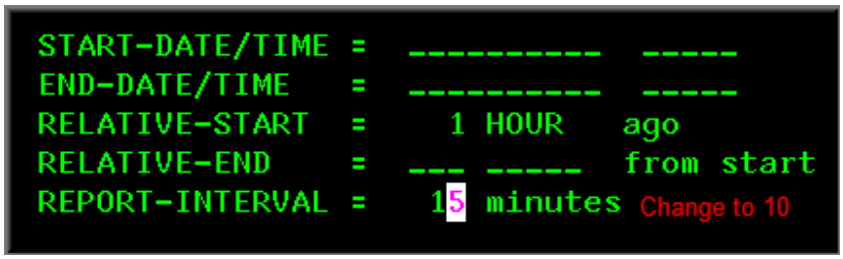


__56. In the next screen, you will see several Omegamon menu options. Type-in H option and hit right CTRL key.

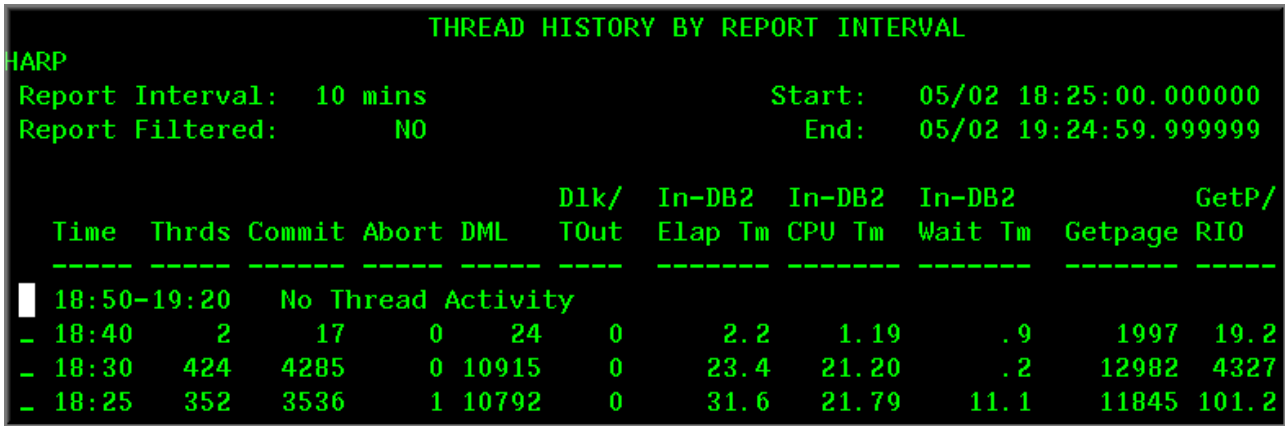
__57. Type-in B option and hit right CTRL key to go to the thread history.



__58. Change report interval to 10 minutes and hit right CTRL key.

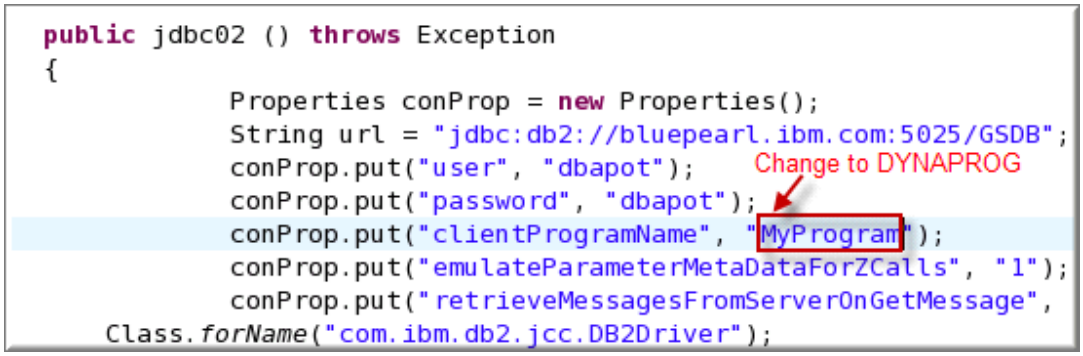


__59. Hit CTRL key in the next screen and you will see Thread History By Report Interval Screen.



__60. You are now ready to run Java program in DYNAMIC and STATIC mode few times and notice the performance difference between 2 using Omegamon. You will switch back and forth between Data Studio and this screen as you run your Java.

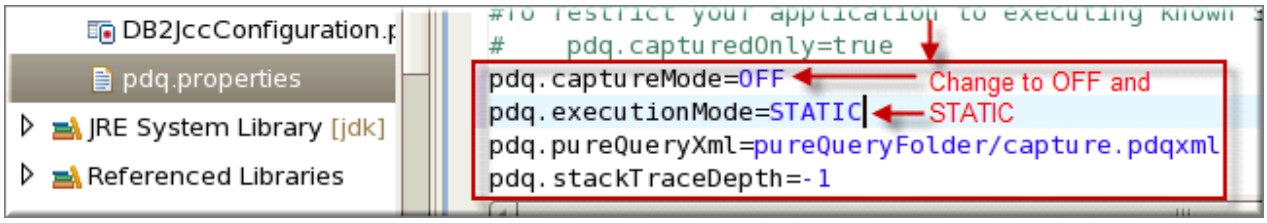
__61. Now, switch back to the Data Studio and go to the `jdbc02.java` program and modify the `clientProgramName` property name to `DYNAPROG` so that we can distinguish between different runs using correlation id. Hit `CTRL-S` to save the file



__62. Right click anywhere in the program and click `Run As` ⇨ `Run Configurations`.

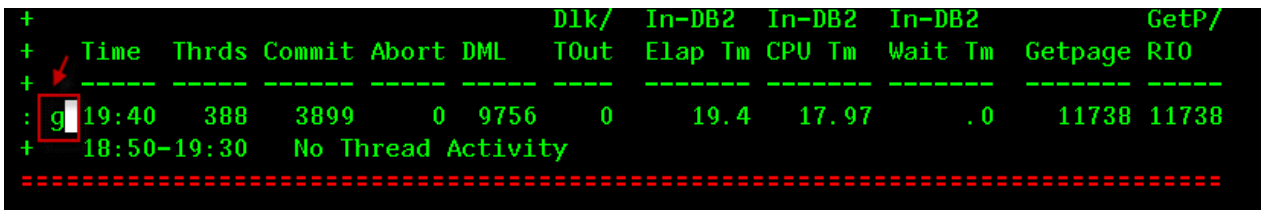
__63. Select `jdbc02` in `PureQuery` option and click on the `Run` button and verify the run status from the console view.

__64. Open `pdq.properties` file and change the value of `captureMode` from `ON` to `OFF` to disable statement capturing. Change the default value for the `executionMode` from `DYNAMIC` to `STATIC` to enable static execution.

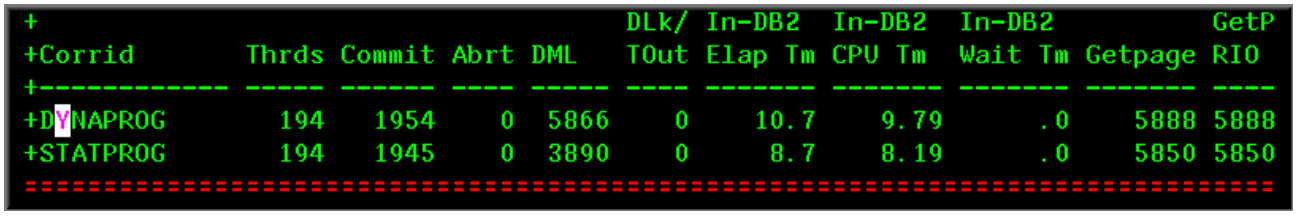


__65. After running the program, go back to the `jdbc02.java` and modify `clientProgramName` property to `STATPROG`. Hit `CTRL-S` to save the file and run the program again as you did in the previous steps.

__66. Go back to the Omegamon screen and type `G` against the most recent time interval and hit right `CTRL` key.



__67. The next screen should look like the following. If you only see DYNAPROG, wait for few minutes for STATPROG to appear as our time interval is set to 10 minutes. It may also so happen that you might see both in different time interval depending upon the actual time of the capture.



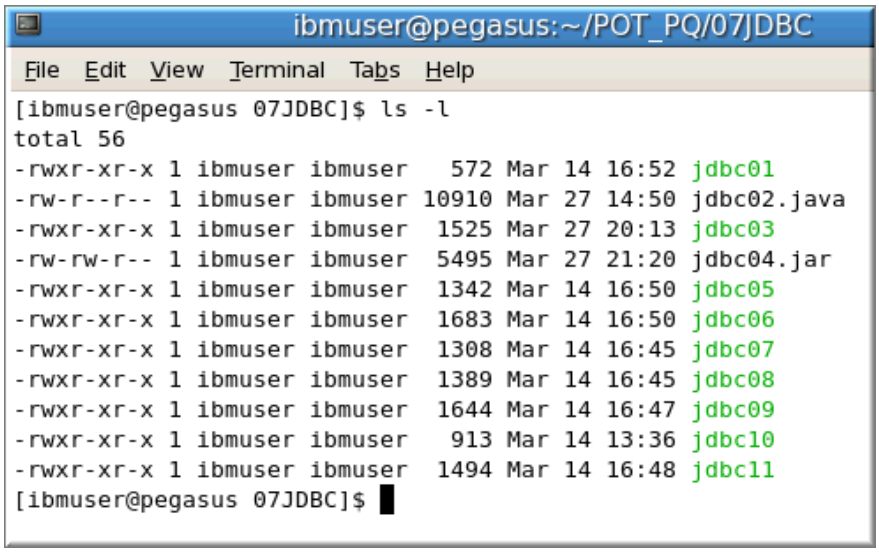
```
+                                        DLk/ In-DB2  In-DB2  In-DB2          GetP
+Corrid       Thrds Commit Abrt DML      TOut Elap Tm CPU Tm  Wait Tm Getpage RIO
+------------ ----- ------ ---- -------- ---- -------- -------- -------- ------- ----
+DYNAPROG       194   1954    0   5866     0     10.7     9.79       .0    5888 5888
+STATPROG       194   1945    0   3890     0      8.7     8.19       .0    5850 5850
================================================================================
```

__68. Notice the difference in DML count between DYNAMIC and STATIC. This is mainly due to NO PREPARE for STATIC SQL. Also notice the time difference between static and dynamic. [Please note: This is not a performance measurement system but still you see the improvement in the execution time. When you incrementally add all of these savings for thousands of dynamic SQL, your savings in CPU utilization could be significant.]

7.4 Optimization when source is not available

In this section you will use the pureQuery command line utilities to capture, configure and bind SQL that is issued in a Java application for which you do not have the source. We will use same program assuming that we do not have source.

Go back to the Command Window and you will run commands from this shell.



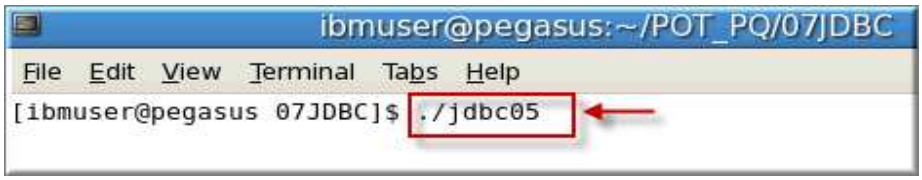
__69. For this lab, 5 administration scripts have been created for you.

jdbc04.jar	This is our custom application JAR file
jdbc05	This script runs the custom application as it is
jdbc06	This script runs the application for different test cases and captures SQL statements and puts them in capture.pdqxml
jdbc07	This script configures `capture.pdqxml` file for binding purposes.
jdbc08	This script binds the SQL statements from capture.pdqxml to the database
jdbc09	This script runs the custom application in STATIC mode.

Note: These above mentioned scripts are not part of Data Studio. These scripts have been provided to you in this PoT as samples for you to customize your profile in your JDBC applications.

7.4.1 Run custom JDBC application as it is

__70. At your command prompt, run `jdbc05` to execute custom JDBC program as shown below.



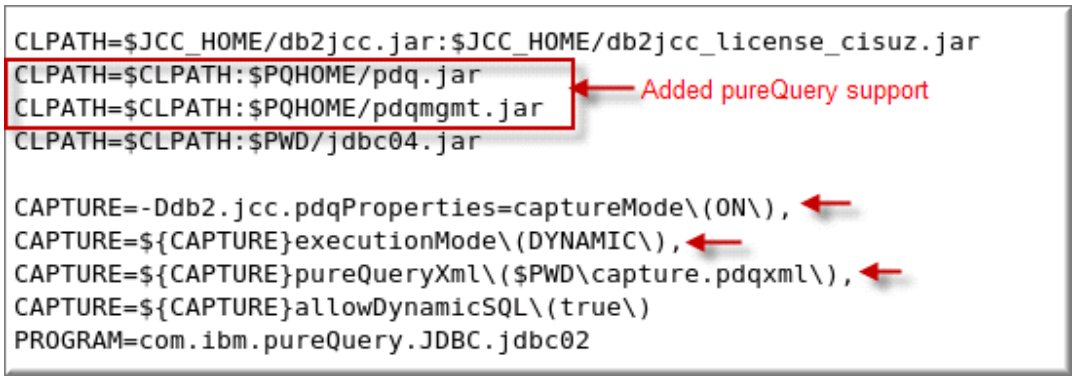
7.4.2 Capture SQL metadata

__71. To capture the SQL statements that are being issued by our custom application, we will modify a few runtime environment settings for the application.

- Include the required DB2 JCC driver and pureQuery JAR files.
- Enable pureQuery capabilities in the JDBC driver.

__72. Review script `jdbc06`.

__73. Notice following highlighted changes that were added to capture the SQL statements from the custom JDBC application program.

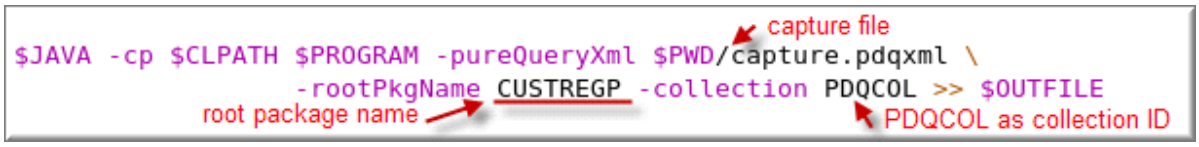


The `jdbc06` script is very similar to `jdbc05` that was used in the previous section. The Java classpath was updated to include the required pureQuery runtime JAR files and a `db2.jcc.pdqProperties` property was passed to the JVM. Through this property, we signaled to the DB2 JCC driver to start capturing the SQL and create a `capture.pdqxml` file to store the metadata. Now, run the script to capture the SQL.

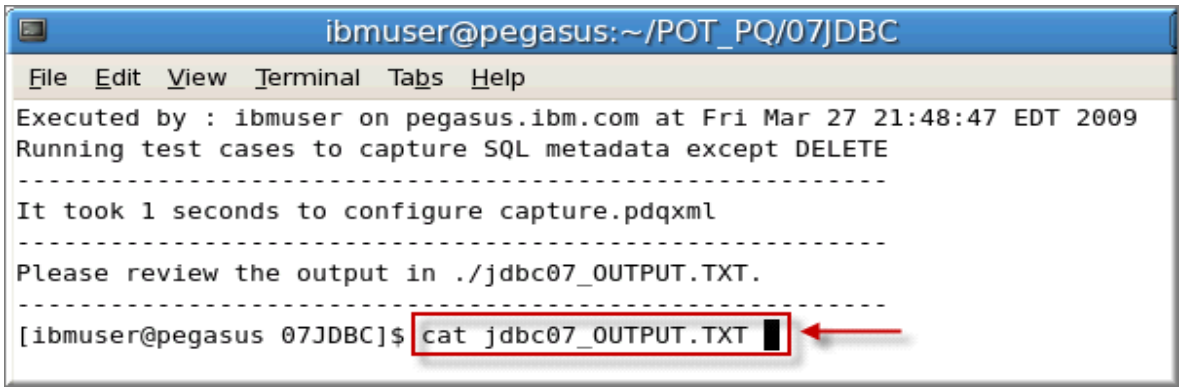
Note: In a real life scenario, one would exercise all known use cases to capture as much SQL as possible. However, here we are not using DELETE on purpose to show other things in lab later.

7.4.3 Configuring SQL metadata

__74. The command line utilities support batch configuration and binding of the captured metadata. These utilities are implemented as Java classes packaged together in pureQuery runtime JAR files. View `jdbc07` file and this script invokes `Configure` utility and it assigns a package prefix and a collection ID or schema name for the package.



__75. Go ahead and run `jdbc07` to make changes in the `capture.pdqxml` file.

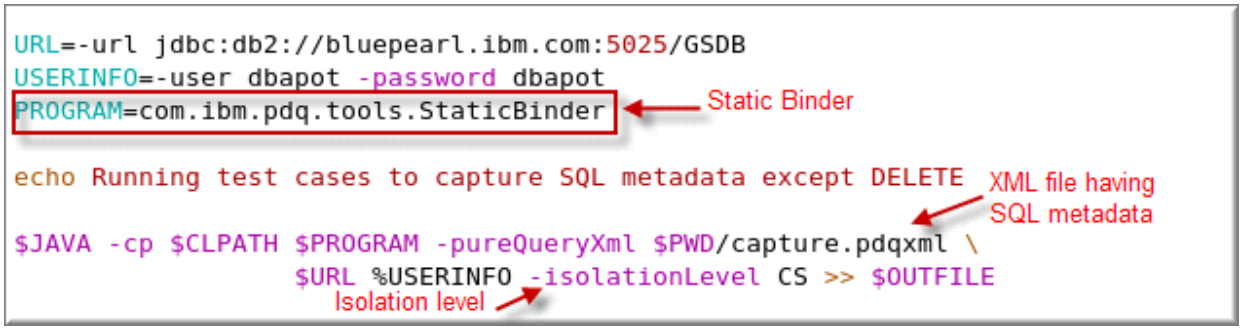


__76. The `Configure` command provides many other options and you can see the help by running the following command in your command shell window. (Run script `jdbc10` and see the output file)

7.4.4 Bind SQL metadata

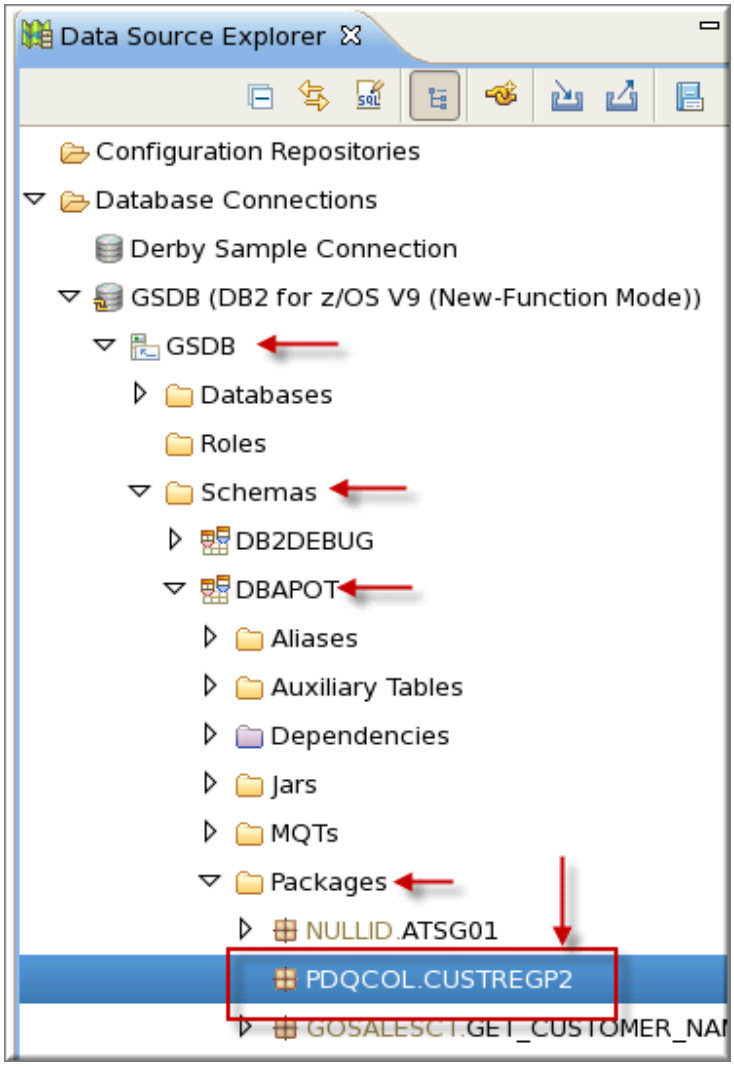
__77. Review script `jdbc08`.

__78. The bind utility processes the previously captured and configured metadata and creates one or more packages in the database. You are invoking the `StaticBinder` utility to bind SQL and its metadata from `capture.pdqxml` file.



__79. Go ahead and run `jdbc08` from the command prompt and you can see the package created through Data Studio.

```
Executed by : ibmuser on pegasus.ibm.com at Fri Mar 27 22:11:47 EDT 2009
Running test cases to capture SQL metadata except DELETE
-------------------------------------------------------------
It took 3 seconds to bind packages to DB2
-------------------------------------------------------------
Please review the output in ./jdbc08_OUTPUT.TXT.
-------------------------------------------------------------
```



__80. There are many options available with the `StaticBinder` and you can run following command to see them. (Run script `jdbc10` and see the output file)

7.4.5 Run Packaged Application in STATIC SQL mode

__81. Review script `jdbc09` and notice options to run this custom JDBC application in STATIC mode.

```
CAPTURE=-Ddb2.jcc.pdqProperties=captureMode\(OFF\),
CAPTURE=${CAPTURE}executionMode\(STATIC\),
CAPTURE=${CAPTURE}pureQueryXml\($PWD/capture.pdqxml\),
CAPTURE=${CAPTURE}allowDynamicSQL\(true\)
PROGRAM=com.ibm.pureQuery.JDBC.jdbc02
```

__82. You will notice that we have specified `captureMode OFF` and `executionMode` has been specified as `STATIC` and `dynamicSQL` are still allowed.

__83. Go ahead and run `jdbc09`.

Note: How would you know if you are really using SQLs in static mode or not?

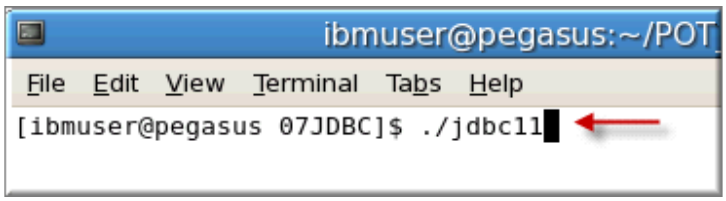
[Hint: Drop package `PDQCOL.CUSTREGP2` and run one of the above command. You should get SQL -805 error indicating that the package was not found.] After your test, run `JDBC08` command again to bind the package.

__84. Review script `jdbc11` and notice the option `allowDynamicSQL` modified from `true` to `false` and will try to delete one of the product that we registered before.

Remember: We did not capture the DELETE statement.

```
CLPATH=$JCC_HOME/db2jcc.jar:$JCC_HOME/db2jcc_license_cisuz.jar
CLPATH=$CLPATH:$PQHOME/pdq.jar
CLPATH=$CLPATH:$PQHOME/pdqmgmt.jar
CLPATH=$CLPATH:$PWD/jdbc04.jar

CAPTURE=-Ddb2.jcc.pdqProperties=captureMode\(OFF\),
CAPTURE=${CAPTURE}executionMode\(STATIC\),
CAPTURE=${CAPTURE}pureQueryXml\($PWD/capture.pdqxml\),
CAPTURE=${CAPTURE}allowDynamicSQL\(false\)   <-- No dynamic SQL allowed
PROGRAM=com.ibm.pureQuery.JDBC.jdbc02

                                             Try delete and it should fail
echo Running app using static SQL and trying DELETE which was not captured

$JAVA -cp $CLPATH $CAPTURE $PROGRAM DEL Hammer >> $OUTFILE 2>&1
```

__85. Go ahead and run script `jdbc11`



__86. The problem you have just seen is an indication that the application issued an SQL statement that was not captured previously. The driver has thrown an exception because it was configured to run all SQL statements statically but encountered a SQL statement for which no metadata was previously captured. There are several solutions to the problem.

- Repeat the process (capture, configure and bind) to capture the missing SQL and re-run the application in static SQL mode.

  __i. You can re-run your application in capture mode and exercise the use cases that were missed during the previous capture iteration using either one of the following two property settings:

  ```
  captureMode(ON), executionMode(DYNAMIC)
  captureMode(ON), executionMode(STATIC), allowDynamicSQL(TRUE)
  ```

  __ii. The process is defined to as incremental capture. After a subsequent re-configuration and rebind operation the JDBC application should be able to execute its SQL statically.

- Run the application in static SQL execution mode but allow dynamic SQL execution to avoid application failures.

  __i. This solution avoids the issue of not having captured all SQL statements by allowing for the execution of SQL in dynamic mode. The only difference between this solution and the previous one is that no incremental capture is performed.

  captureMode(OFF), executionMode(STATIC), allowDynamicSQL(TRUE)

  __ii. The driver will execute an SQL statement statically if it has been captured before execute it dynamically and capture it if execution succeeds.

__87. Complete the lab by applying one of the solutions shown above and verify successful execution of all 3 registration commands with property `executionMode(STATIC)`.

**** End of Lab 7: pureQuery for JDBC Applications**

Lab 8 - pureQuery Advanced Concepts

This lab demonstrates some of the advanced features of the pureQuery. The following topics are covered in this lab:

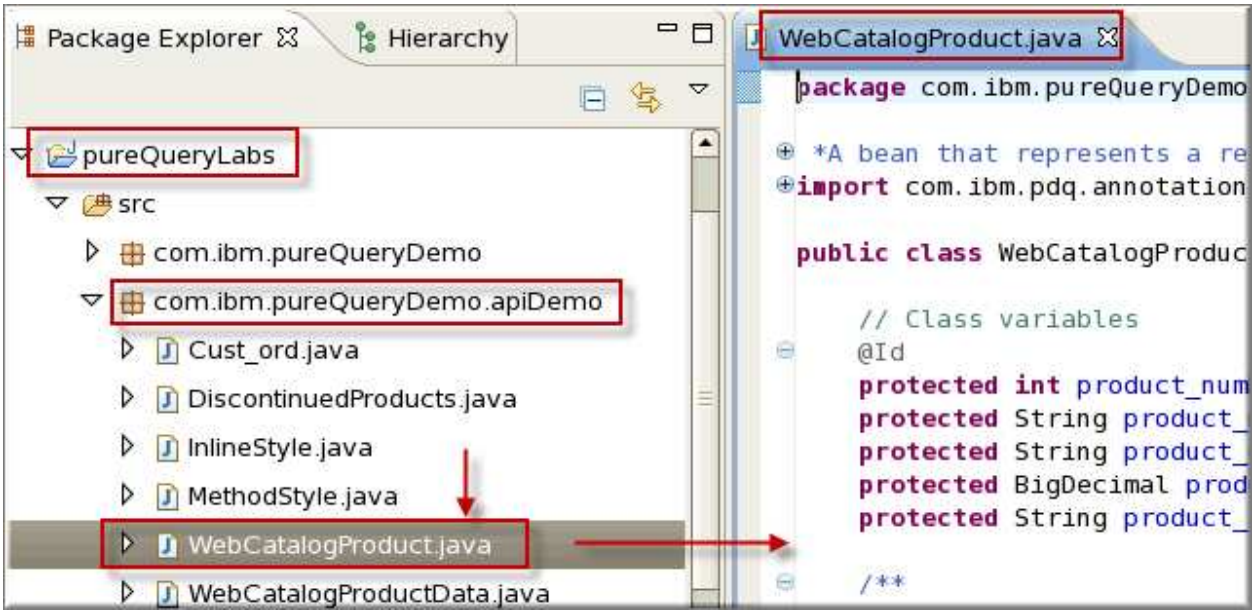
- Generate JPA compliant XML for annotated method SQL statements.
- Custom `ResultHandler` to return a XML data structure.
- Custom `ResultHandler` to map `ResultSet` into HTML output
- Custom `ResultHandler` to populate nested beans.
- Use of `Hook` callback as a built in performance monitor.

8.1 Generate JPA compliant XML

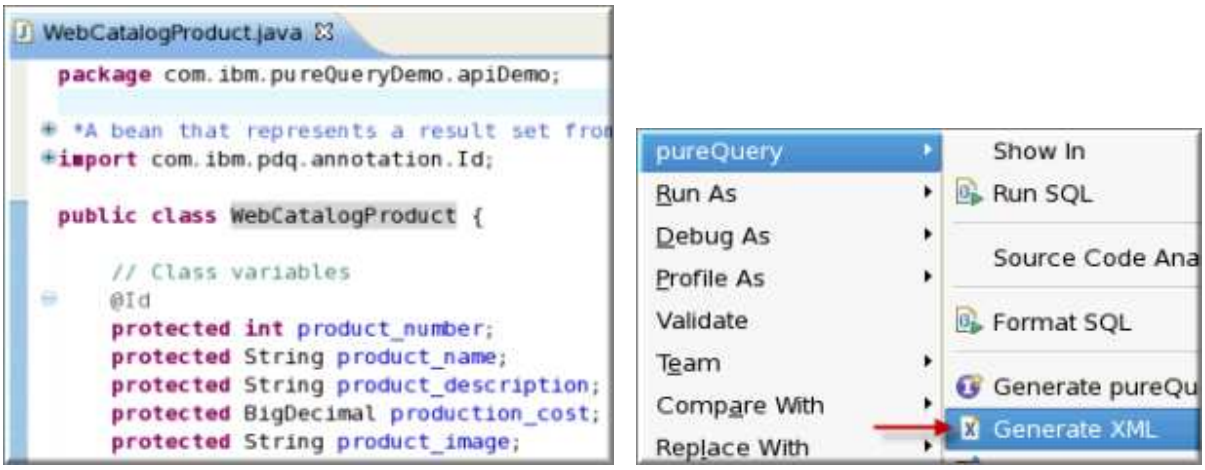
In this section you will explore how annotated method SQL works in an XML file. Using annotated method SQL in an XML file allows you to organize / isolate SQL accessor methods into separate interface files. It also allows easy deployment of static SQL as well as allowing application metadata to be gathered, stored and registered.

__1. Close all open files by clicking `<CTRL><SHIFT><W>`. If you have a command window open from the previous lab, close it.

__2. In the *Package Explorer*, expand `apiDemo` package in the `pureQueryLabs` project. Double click on `WebCatalogProduct.java` to open it.



__3. To generate XML for the `WebCatalogProduct` bean, right click anywhere within `WebCatalogProduct.java` and select `pureQuery` ⇨ `Generate XML`



__4. The attributes from the bean `WebCatalogProduct` are exported to the `orm.xml` file and it is opened for you. Verify bean attributes in the `orm.xml` file.

```
<orm:entity class="com.ibm.pureQueryDemo.apiDemo.WebCatalogProduct">
<orm:attributes>
    <orm:id name="product_number">
        <orm:column name="PRODUCT_NUMBER"/>
        <orm:generated-value/>
    </orm:id>
    <basic name="product_number">
        <orm:column name="PRODUCT_NUMBER"/>
    </basic>
    <orm:basic name="product_description">
        <orm:column name="PRODUCT_DESCRIPTION"/>
    </orm:basic>
```

Note: The orm.xml file is created in the `pureQueryFolder` under the `pureQueryLabs` project. Did you know that why we exported attributes of the bean first before we go to the next step?

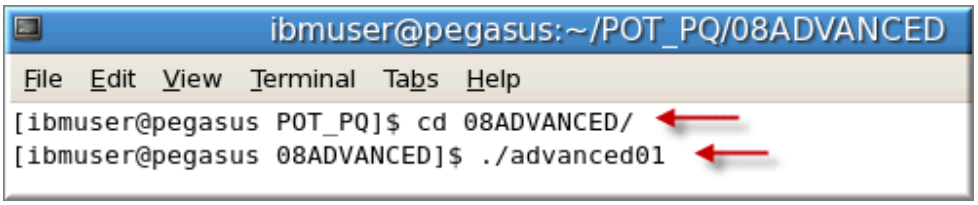
__5. Open interface file `WebCatalogProductData.java` and right click anywhere within it and select `pureQuery` ⇨ `Generate XML`. The SQLs defined in the interface are exported in the `orm.xml` file. This is a JPA compliant XML file and is also known as named query methods.

```
<?xml version="1.0" encoding="UTF-8"?><orm:entity-mappings xmlns="http://ja
        <orm:query><![CDATA[SELECT P.PRODUCT_NUMBER, Q.PRODUCT_NAME, Q.PROD
    </orm:named-native-query><orm:named-native-query name="com.ibm.pureQuer
        <orm:query><![CDATA[SELECT P.PRODUCT_NUMBER, Q.PRODUCT_NAME, Q.PROD
    </orm:named-native-query><orm:named-native-query name="com.ibm.pureQuer
        <orm:query><![CDATA[SELECT P.PRODUCT_NUMBER, Q.PRODUCT_NAME, Q.PROD
```

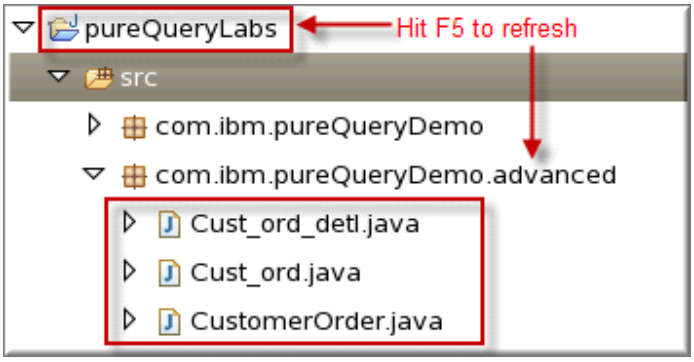
8.2 Examples of the `ResultHandler`

__6. Double click on the *Terminal* icon on the desktop to open up a command window.

__7. Change your directory to: `08AVANCED` and run `advanced01` command to copy Java source files from this directory to the Java project `pureQueryLabs` Java project.



__8. Select `src` folder in `pureQueryLabs` project in your package explorer and hit F5 to refresh the view.

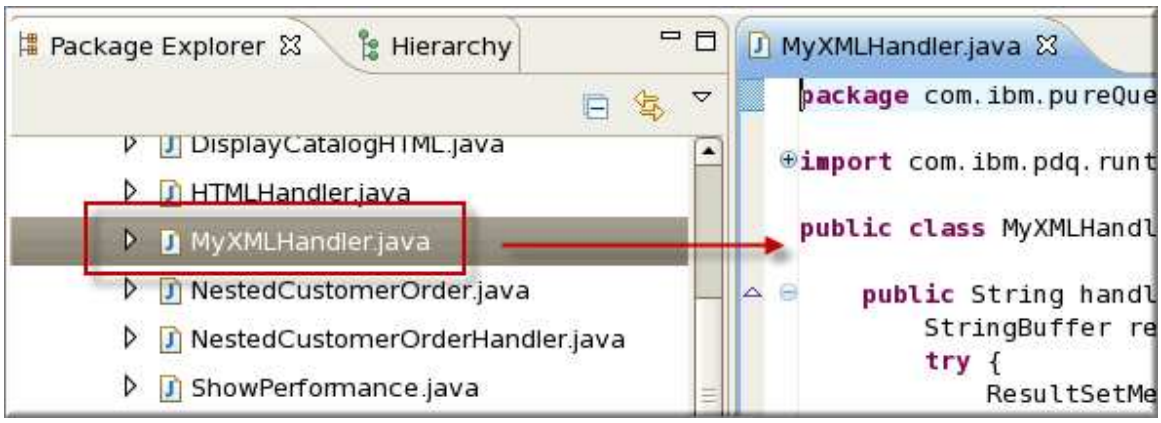


8.2.1 XML handler

The `pureQuery` allows you to define your result set handler to customize results in any way suitable to you. The only method in the `ResultHandler` API is `handle(java.sql.ResultSet arg0)`, a generic method that given a `ResultSet` will produce a new Java object of class `<T>`. Therefore, in order for us to create a custom `ResultHandler`, we must implement the `handle(…)` method.

In the following example we will output to the console the *Product Number, Name, Description, Cost* and *Image* for a `PID=1110`. We will use the `ResultHandler` to format our output as XML.

__9. Open the `MyXMLHandler.java` class. We will not edit this file.



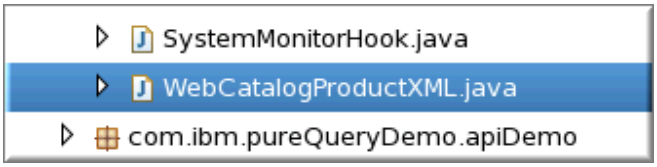
Notice that the `MyXMLHandler` class implements `ResultHandler` of generic type `String`:

```
public class MyXMLHandler implements ResultHandler<String> {
```

The `handle(…)` method is executed when the `query(…)` method of the Data API is invoked. Within the `handle(…)` method, we form XML by formatting the column names as XML Elements and the column data as the XML Text:

```
while (rs.next()) {
    for (int col = 1; col <= m.getColumnCount(); col++) {
        result.append("<" + m.getColumnName(col) + ">");
        result.append(rs.getObject(col));
        result.append("</" + m.getColumnName(col) + ">");
        result.append("\n");
    }
```

__10. Double click on the `WebCatalogProductXML.java` and study how `Resultset Handler` has been used in the Query method.



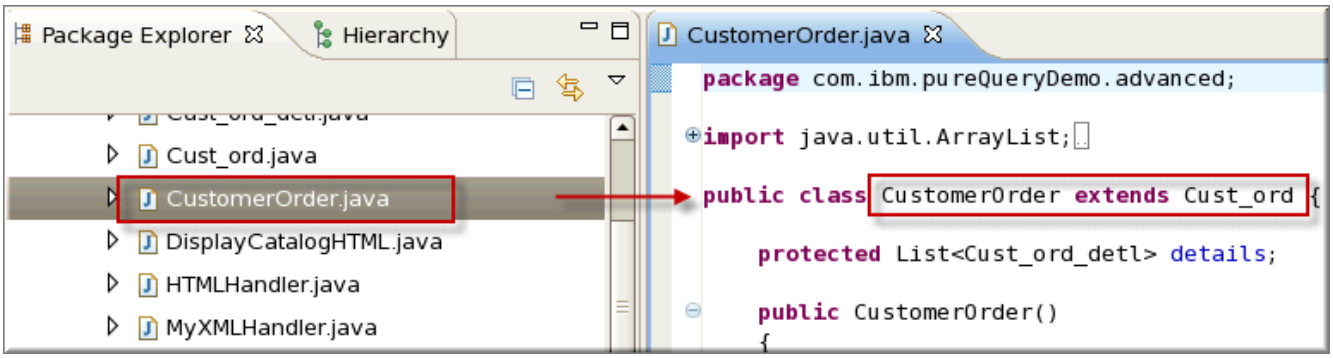

```
    return this.db.query(sql, new MyXMLHandler(), pid);
}
```

__11. Right click anywhere in the `WebCatalogProductXML.java` and click on `Run As ⇨ Java Application` to run the program. You will see an output shown below:

```
Successfully Connected to DB2

<PRODUCT_NUMBER>1110</PRODUCT_NUMBER>
<PRODUCT_NAME>TrailChef Water Bag</PRODUCT_NAME>
<PRODUCT_DESCRIPTION>Lightweight, collapsible bag to carry liquids easily.
<PRODUCTION_COST>4.00</PRODUCTION_COST>
<PRODUCT_IMAGE>P01CE1CG1.jpg</PRODUCT_IMAGE>
```

8.2.2 Nested bean handler

__12. Double-click on the `CustomerOrder.java` and this bean extends `Cust_ord` bean and contains `Cust_ord_detl` which contains details of the order. This bean represents one-to-many relationship between `Cust_ord` and `Cust_ord_detl`. We do not need to edit this file.



__13. Open custom result handler `NestedCustomerOrderHandler.java`. In it we declare a bean of `CustomerOrder` type and populate this through `handle` method which will be called by `data` APIs `query` method.

__14. Open `NestedCustomerOrder.java` and review following `data` API.

```
String sql = "SELECT ORDER.*, DETAIL.* "
           + "  FROM GOSALESCT.CUST_ORD AS ORDER, "
           + "       GOSALESCT.CUST_ORD_DETL AS DETAIL "
           + "  WHERE ORDER.ORD_NBR = ? "
           + "  AND ORDER.ORD_NBR = DETAIL.ORD_NBR ";

List<CustomerOrder> order = this.db.query(sql,
        new NestedCustomerOrderHandler(), orderNumber);
```

__15. Right click anywhere in `NestedCustomerOrder.java` and select `Run As ⇨ Java Application`.

__16. You should see results similar to one shown below.

```
Successfully Connected to DB2

CustomerOrder[getDetails=
Cust_ord_detl[getOrd_detl_code=1003, getOrd_nbr=100002, getOrd_ship_date=2004-01-19,
Cust_ord_detl[getOrd_detl_code=1004, getOrd_nbr=100002, getOrd_ship_date=2004-01-19,
Cust_ord_detl[getOrd_detl_code=1005, getOrd_nbr=100002, getOrd_ship_date=2004-01-19,
Cust_ord_detl[getOrd_detl_code=1006, getOrd_nbr=100002, getOrd_ship_date=2004-01-19,
Cust_ord_detl[getOrd_detl_code=1007, getOrd_nbr=100002, getOrd_ship_date=2004-01-19,
null, ]
```

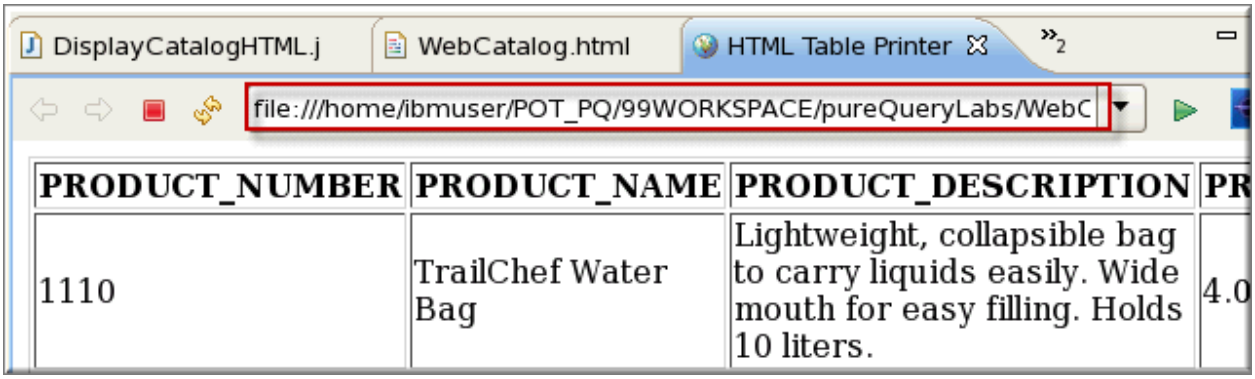
8.2.3 HTML table handler

__17. Open the `HTMLHandler.java` class in the editor. It demonstrates the use of custom `ResultHandler` to format the output of a `ResultSet` in an HTML. The handler can be used with nearly any database query to format it into a displayable HTML representation of the query results.

__18. Open `DisplayCatalogHTML.java`. Right click anywhere in `DisplayCatalogHTML.java` and select `Run As ⇨ Java Application`.

```
Successfully Connected to DB2

WebCatalog.html created.
```

__19. This will create an HTML file in the top level directory of the project. Right click on `pureQueryLabs` project and click on `<Refresh>`. Right click on `WebCatalog.html` file and select `Open With ⇨ Web Browser` and it will open in a browser, showing the HTML table.



8.3 Use of the Hook for built-in Performance Monitor

The pureQuery API allows you to provide an exit to receive control before and after each method invocation. This part of the lab uses that feature to implement a basic performance monitor. The `Hook` exit that we will use exploits a capability in the IBM JDBC driver called the `SystemMonitor`. It allows you to see how much time was spent in various parts of the processing like the driver, network and database server. As each pureQuery operation is performed, these exits invoke the monitor and print the results to the console. The exit could also be changed to print to a file.

__20. Open the `SystemMonitorHook.java` class in the editor. However, as mentioned above, simple changes could be made to write the output to an external. Notice that there are two methods: A method named `pre()` which will be invoked before any pureQuery operation. The other method named `post()` that will be invoked after each operation.

In this `Hook` class, the `pre()` method enables and starts the JDBC `SystemMonitor`. The `post()` method stops the monitor and prints the measurements.

__21. Open `ShowPerformance.java` program in an editor. To enable the `Hook` exits, we must register our `SystemMonitorHook` class with the `Data` object that will be used.

```
public ShowPerformance(Connection conn) {
    SystemMonitorHook monitorHook = new SystemMonitorHook();
    this.db = DataFactory.getData(conn, monitorHook);
}
```

__22. Right click anywhere `ShowPerformance.java` and click using the `Run As ⇨ Java Application`.

__23. You will see an output similar to the one shown below:

```
Successfully Connected to DB2

Performance of method: query(java.lang.String,com.ibm.pdq.runti
Application Time: 263 milliseconds
Core Driver Time: 92837 microseconds
Network Time: 25171 microseconds
server Time: 11805 microseconds
CustomerOrder[getDetails=
Cust_ord_detl[getOrd_detl_code=1003, getOrd_nbr=100002, getOrd_
Cust_ord_detl[getOrd_detl_code=1004, getOrd_nbr=100002, getOrd_
Cust_ord_detl[getOrd_detl_code=1005, getOrd_nbr=100002, getOrd_
Cust_ord_detl[getOrd_detl_code=1006, getOrd_nbr=100002, getOrd_
Cust_ord_detl[getOrd_detl_code=1007, getOrd_nbr=100002, getOrd_
null, ]
```

**** End of pureQuery Lab 8: pureQuery Advanced Concepts**

Appendix A. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Appendix B. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	System p
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	Tivoli	WebSphere	Workplace		

Adobe, Acrobat, Portable Document Format (PDF), and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. See Java Guidelines

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Other company, product and service names may be trademarks or service marks of others.