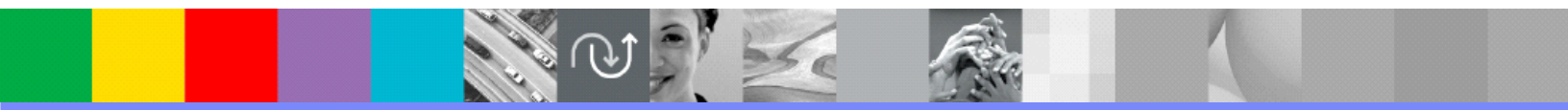
An IBM Proof of Technology

IBM Data Studio pureQuery For DBAs and Application Developers (v2.1)

Lab Exercises



PoT.IM.08.1.059.05

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

Lab 1 Introduction to Data Studio Developer

In this lab, you will open IBM® Data Studio and learn how to open the Java™ and Data 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 and profile a stored procedure.

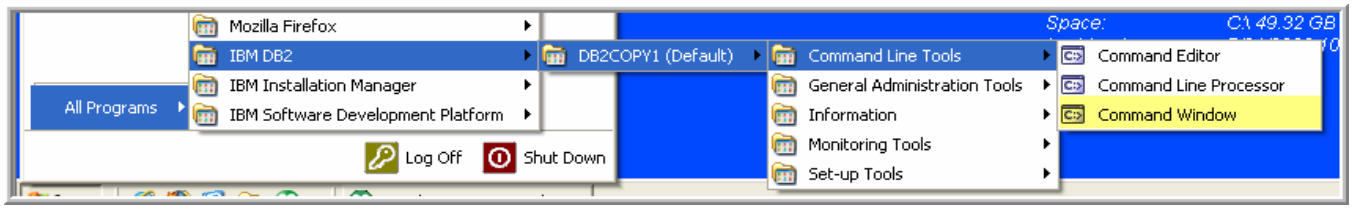
1.1 Required Initial Setup

__1. Open up DB2 command window.

Click on *DB2 Command Window* icon on the desktop. NOTE: This is NOT just a Windows Command prompt window. It is a DB2 command line processor window for executing DB2 commands.

OR

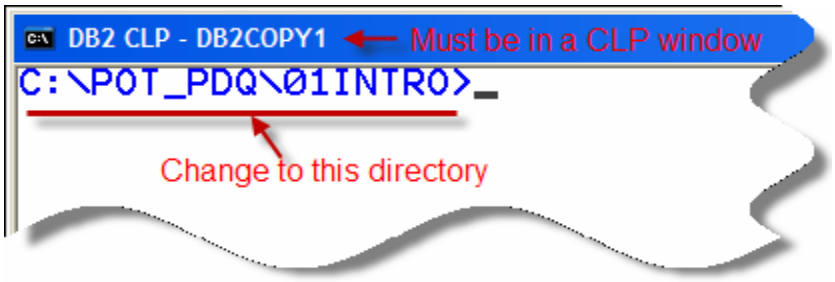
Click on `Start ⇨ All Programs ⇨ IBM DB2 ⇨ DB2COPY1 (Default) ⇨ Command Line Tools ⇨ Command Window`



Note: Please do not try to run DB2 commands in regular Window Command window as it will fail with an error that “Command line environment in not initialized”. Please make sure that you open a command window as shown above.

You will see DB2 command window as shown below and change directory to `C:\POT_PDQ\01INTRO`.

__2. Review and run `INTRO01.CMD` command to create a *GSDB* database and the necessary tables and other objects for the lab exercises for this PoT. When the script finishes, continue with the next section.



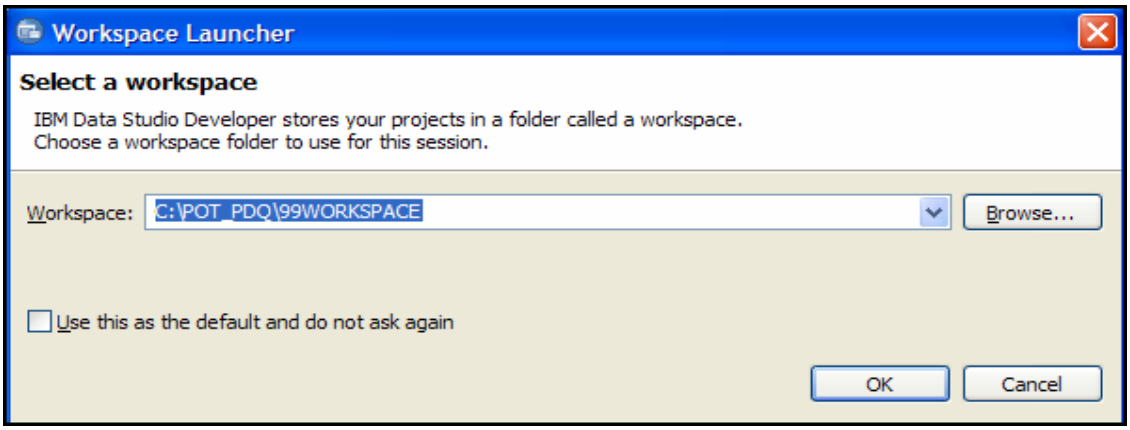
1.2 Open Data Studio Developer

__3. Open the IBM Data Studio Developer by clicking on this icon.

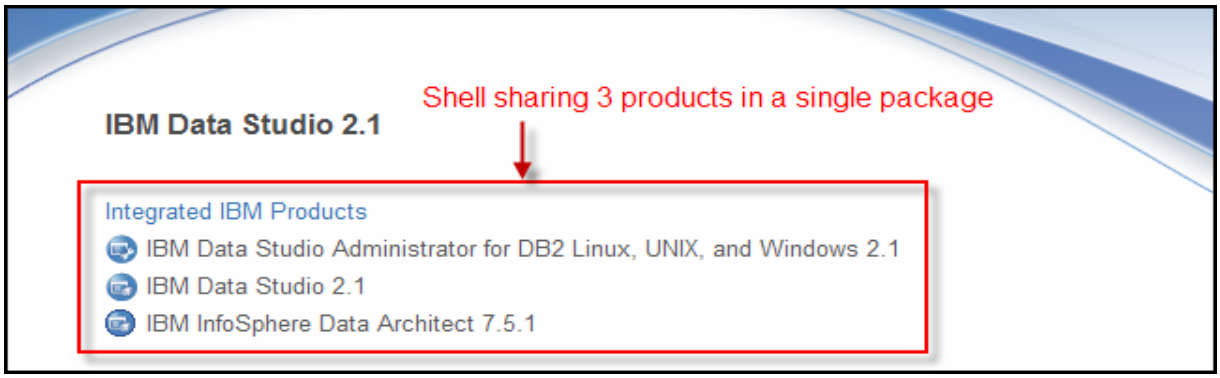
OR…You can open the IBM Data Studio Developer by clicking:

`Start ⇨ All Programs ⇨ IBM Data Studio ⇨ Data Studio Developer`

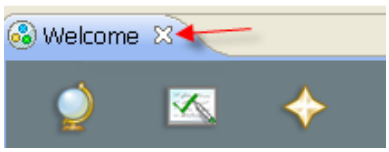
__4. Make sure that your workspace points `C:\POT_PDQ\99WORKSPACE` directory. Click <OK> and wait for the Data Studio Developer to launch.



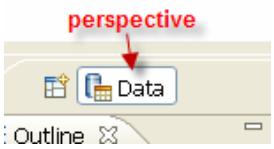
__5. You will see splash screen showing 3 products shell sharing with each other using a single package.



__6. If you get to a welcome screen, close it.



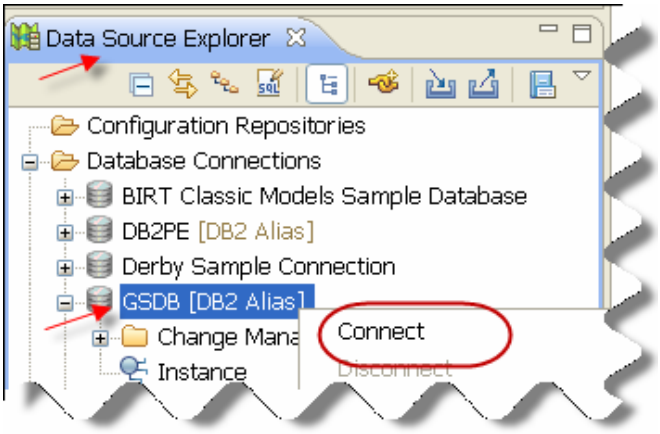
You will now be in the Data Studio Developer in a perspective called `Data`. You can see the perspectives on the top right corner of your screen.



1.3 Connect to a database and explore the tables

__7. Now we will connect to the *GSDB* database we just created by running that DB2 script.

__a. In the *Data Source Explorer*, right click on *GSDB* in the *Database Connections* folder. Choose: `Connect`



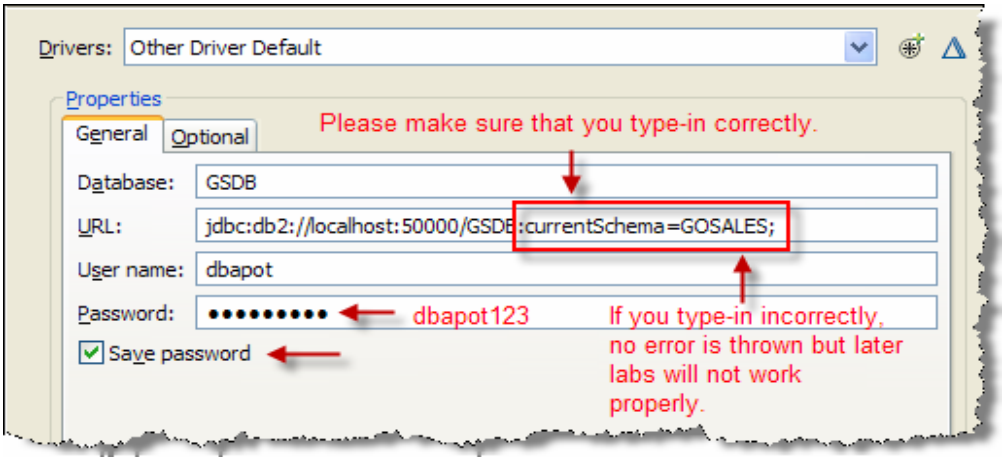
__b. Enter the following in the *Driver Properties*:

User name: *dbapot*

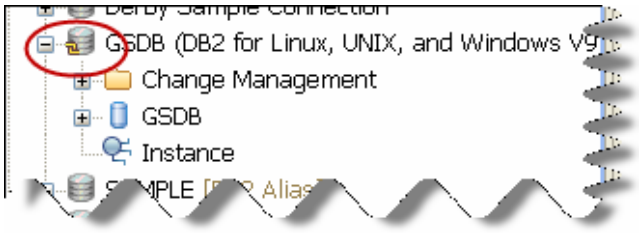
Password: *dbapot123*

Check: Save password

URL: Add: `:currentSchema=GOSALES;` at the end. (Include semicolon also)

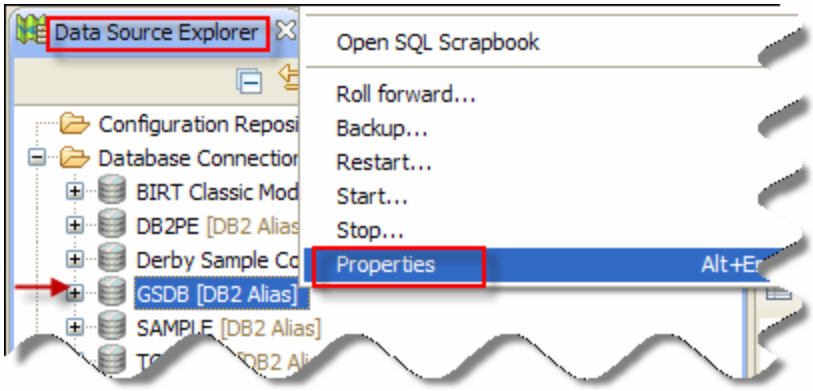


__c. We now have a connection and the database icon changes to reflect that.

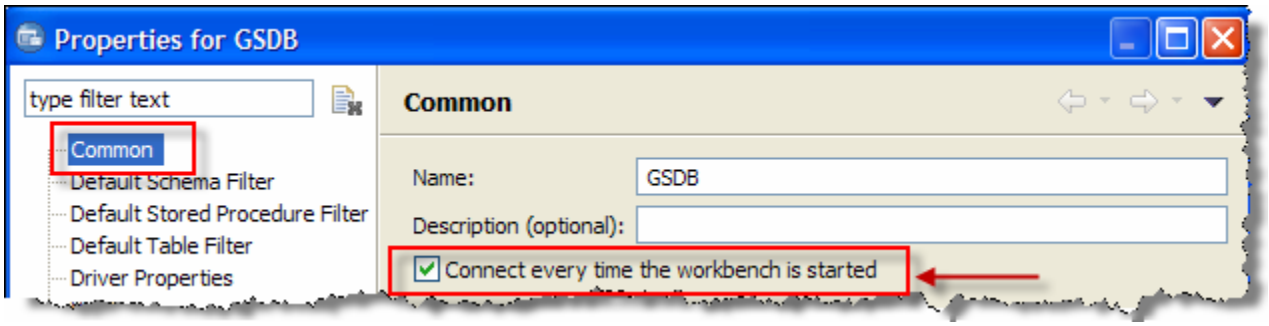


__d. We can now use Data Studio Developer to explore the GSDB database objects.

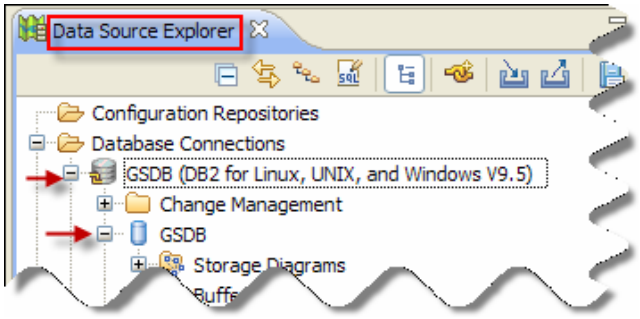
__8. Select GSDB database in *Data Source Explorer* and right click on it to select `Properties`.



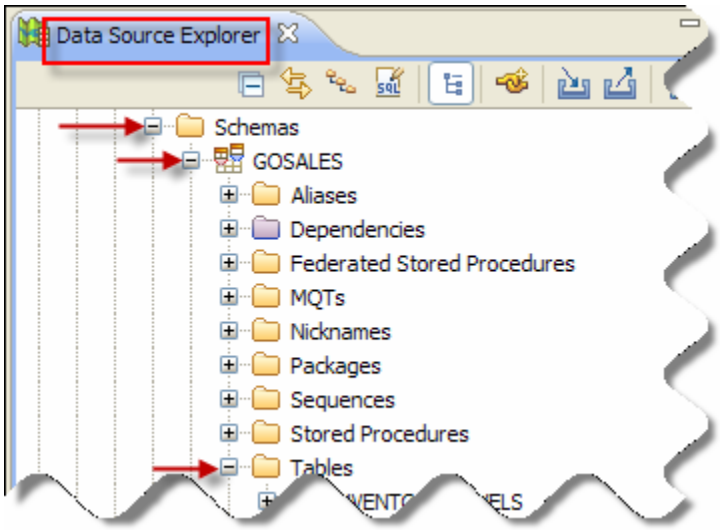
__9. Click on the option `Connect every time the workbench is started` and hit `OK`.



__10. In the *Data Source Explorer*, expand `Connections` ⇨ `GSDB` ⇨ `GSDB`



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

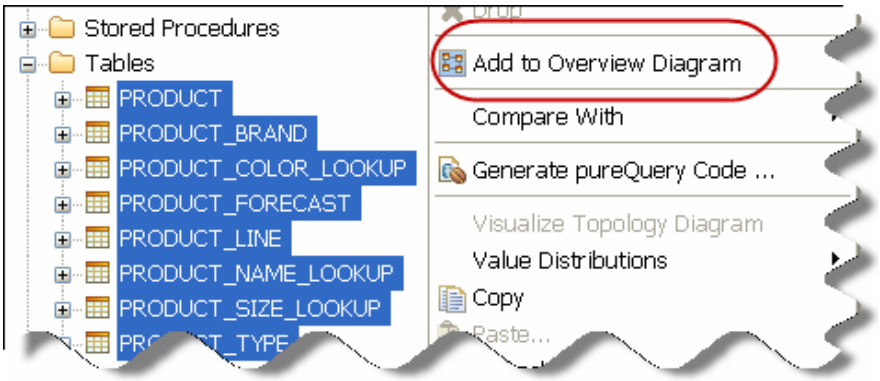


See a visual relationship between tables

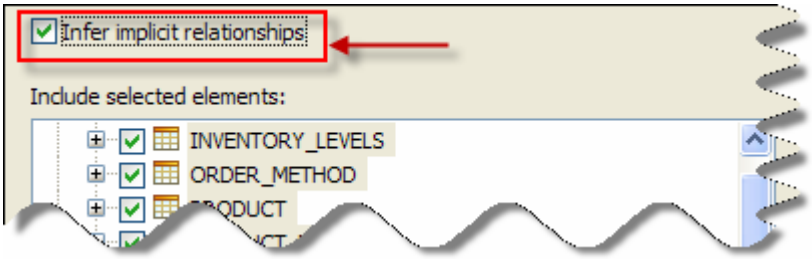
__11. Do the following to see a relationship between tables.

__a. Click on the *PRODUCT* table

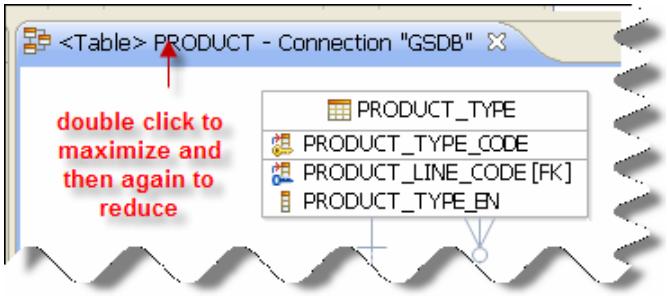
__b. Hold shift key and click on the *PRODUCT_TYPE* table. By doing so, you will select all tables as shown below. Now right click (to show the context menu) and choose: `Add to overview diagram`.



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



__d. 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.



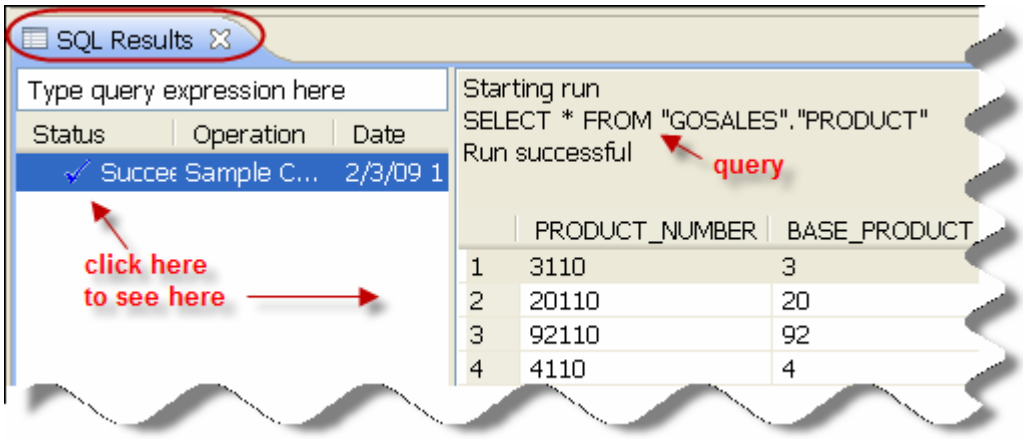
__12. After viewing this, double click on the title to minimize the window.

__13. Now close this overview diagram window (click on the X in the tab). 

Sample some data from a table.

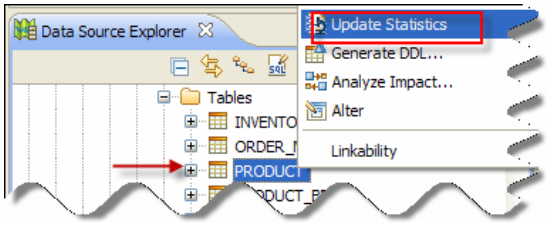
__14. In the *Data Source Explorer, Tables* folder, find the table *PRODUCT*. Right click on *PRODUCT*, then choose: `Data ⇨ Sample Contents`

__15. 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

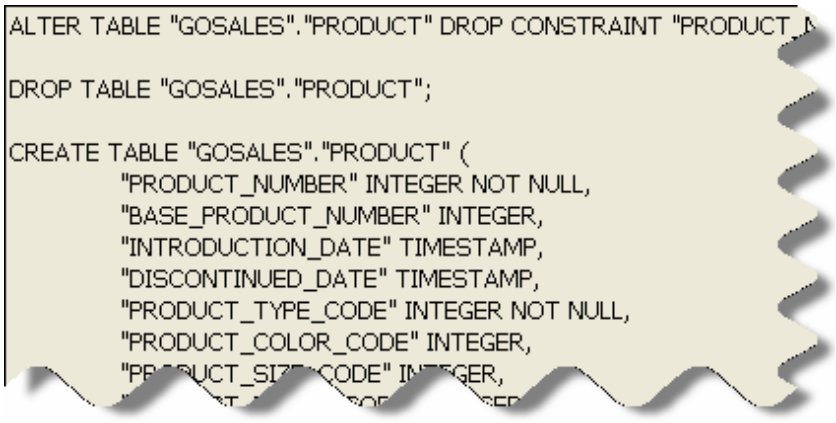
In the *Data Source Explorer, Tables* folder, right click on table *PRODUCT*, then choose: `Update Statistics.` Review the *SQL Results* view to see how this was accomplished.



Generate DDL for a table

__16. In the *Data Source Explorer, Tables* folder, right click on table *PRODUCT*, then choose: `Generate DDL.`

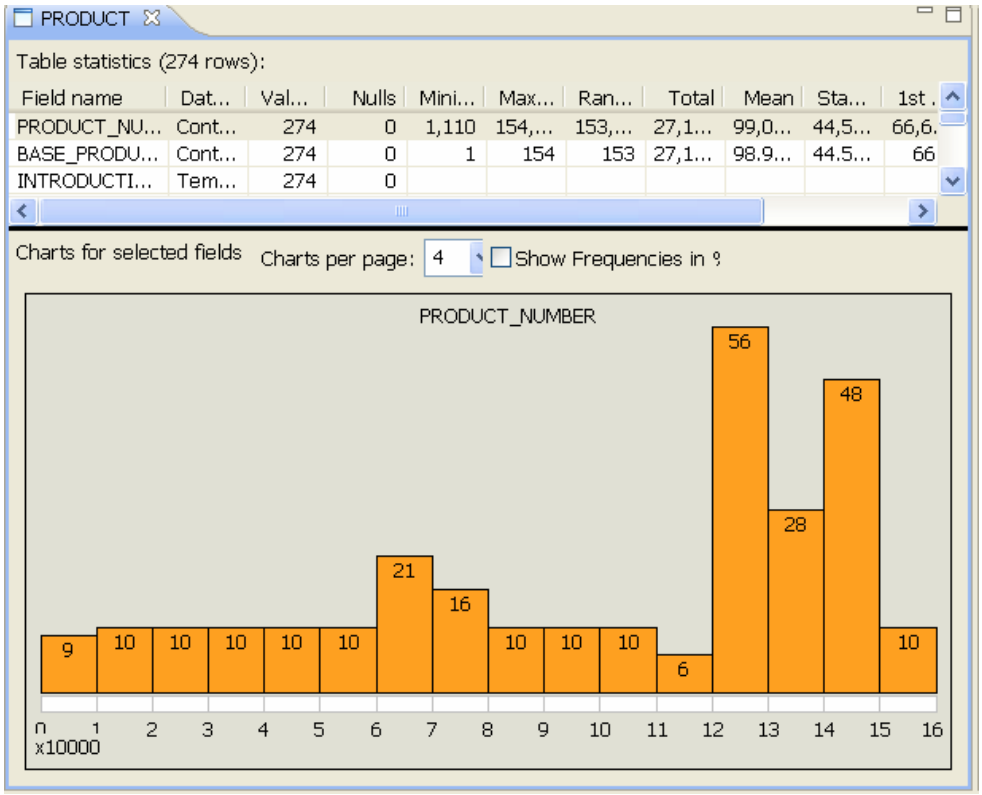
The output looks like this. Notice you can save this to run later if you like.



Look at value distributions

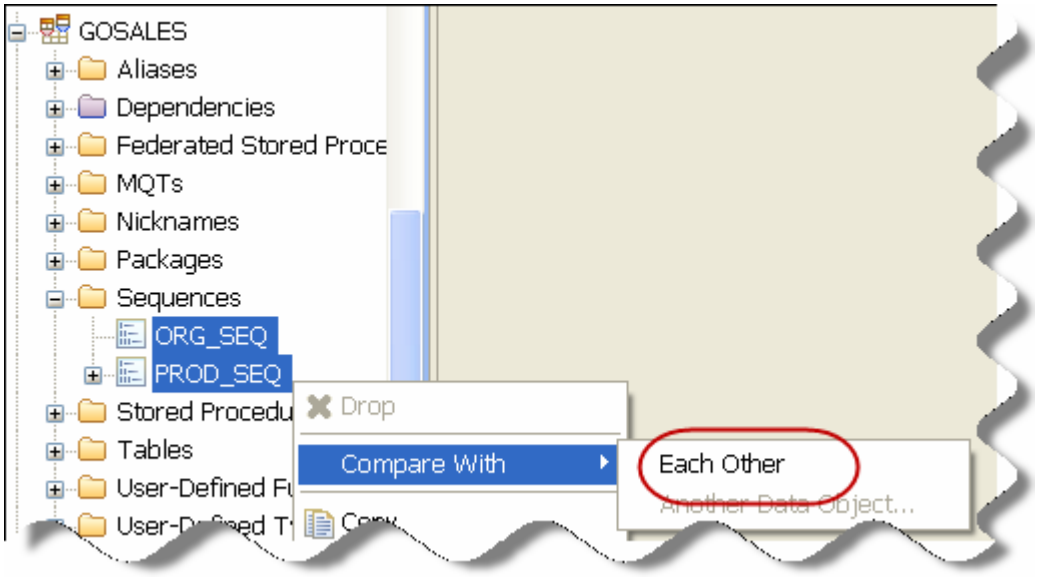
__17. In the *Data Source Explorer, Tables* folder, right click on table *PRODUCT*, then choose: `Value Distributions ⇨ Multivariate`

The output looks like this:

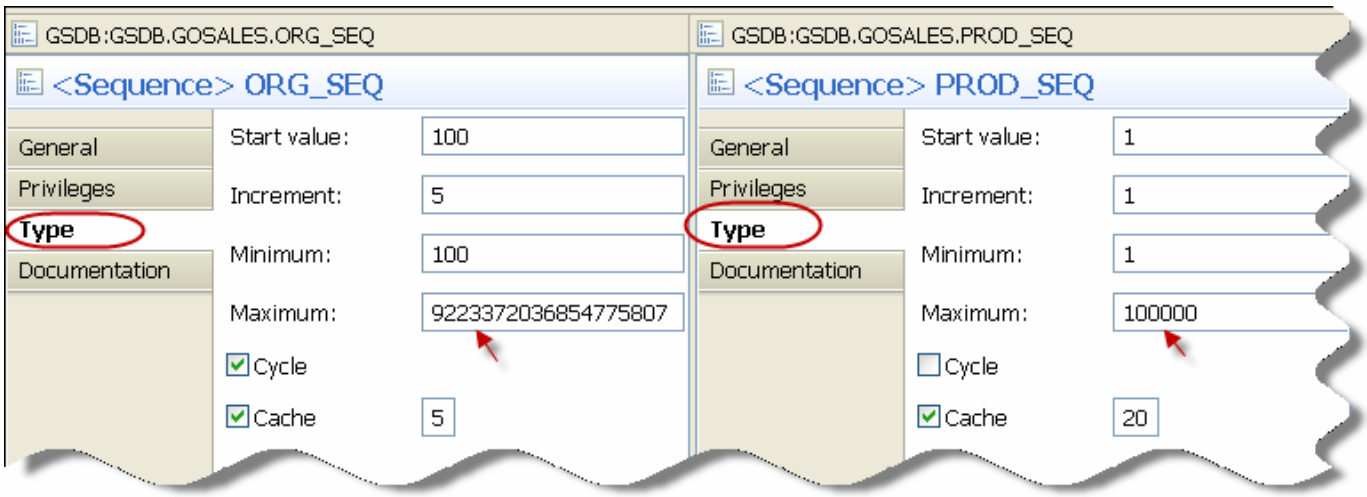


Compare objects

__18. In the *Data Source Explorer, Sequences* folder, right click on both: *ORG_SEQ* and *PROD_SEQ* Then choose: `Compare With ⇨ Each Other`

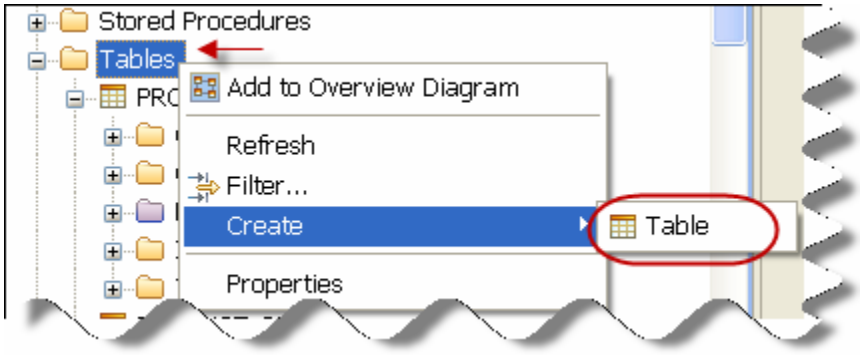


__19. Go to the *Type* screen and notice that the max values for the sequences are very different. Can you figure out why *ORG_SEQ* can be so large? (Hint: generate DDL for them both and see which data types they are defined to.)

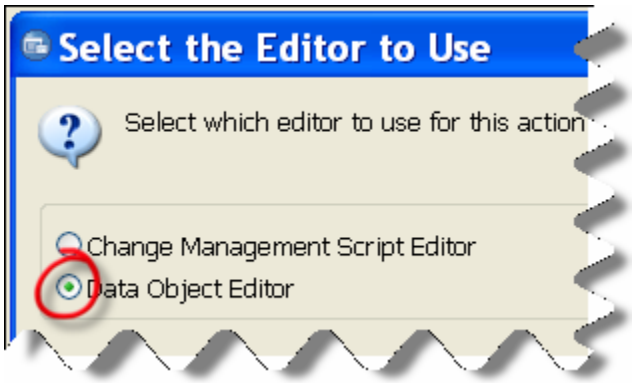


1.4 Creating a table

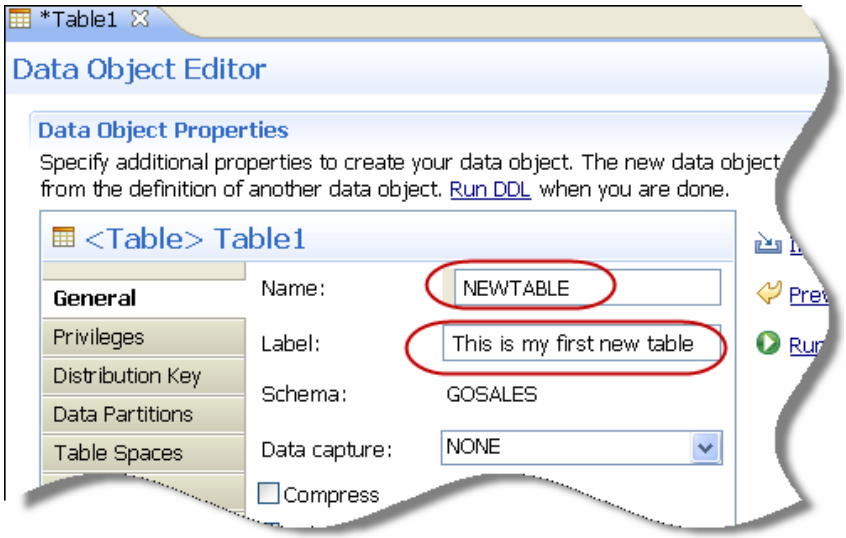
__20. Make sure you stay in the *GOSALES* schema in the *Data Source Explorer*, right click on the *Tables* folder itself then: `Create` ⇨ `Table`



__21. This invokes a choice of editors. For now we will be using the data *object* editor. The *Change Management Script Editor* is for another lab.

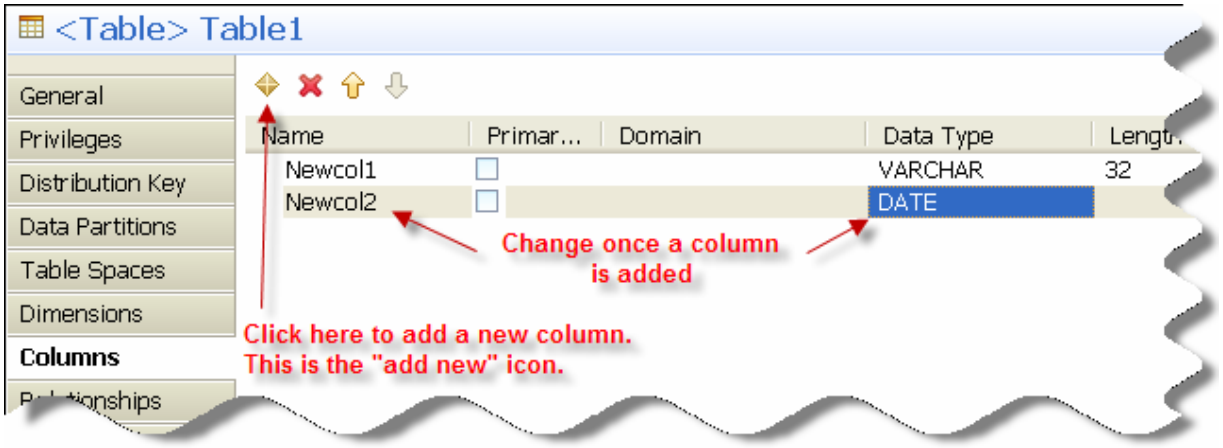


__22. Use the General screen of the *Data Object Editor* to create a table called: *NEWTABLE*

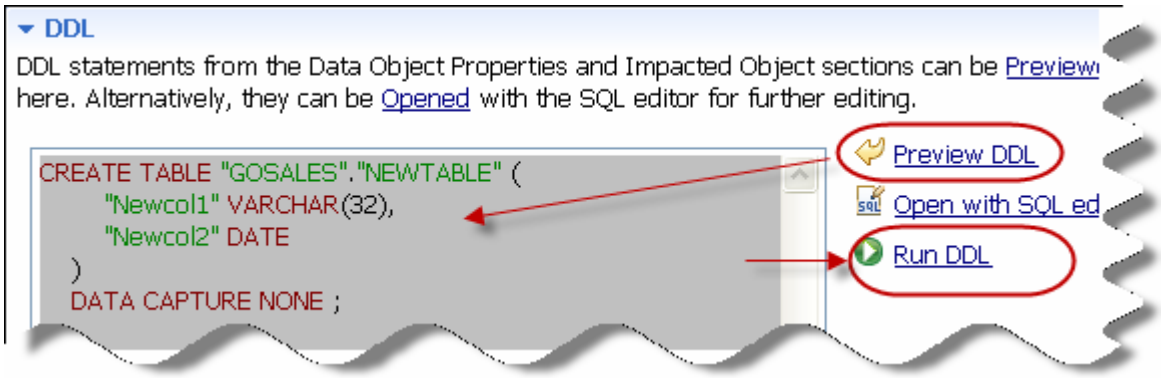


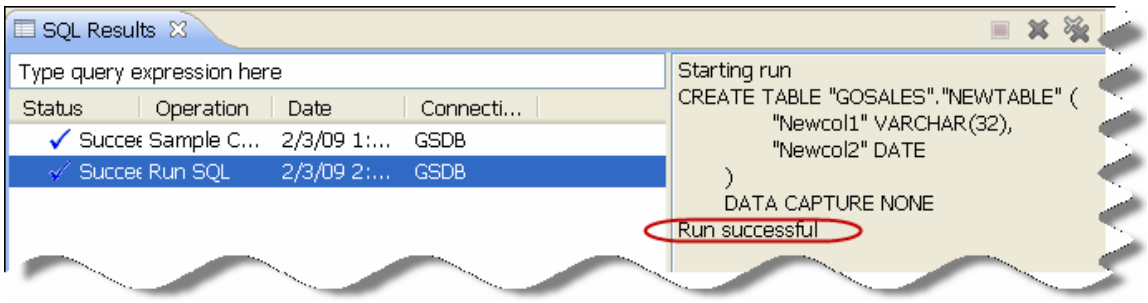
__23. Explore through each screen of the editor to get a feel for how it prompts you to create the table. Every option is available for you to take full advantage of the DB2 CREATE TABLE definition.

__24. 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:



__25. `Preview` and then `Run DDL`. This same technique can be used to create any database object.





1.5 Debug a Stored Procedure

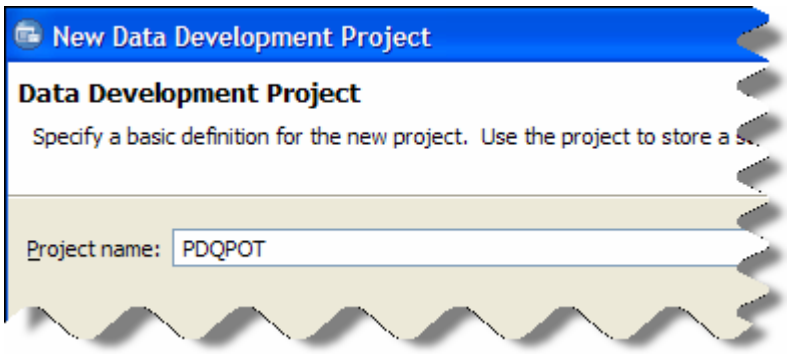
In this section, you will create a data project. A data project is used to save SQL scripts, stored procedure code, UDF code and so on. Data projects are usually used for application development, so many DBAs may never create a data project. Still, if you are a DBA that would develop and debug stored procedures, or if you develop SQL for any reason, you should give this a look.

Create a Project

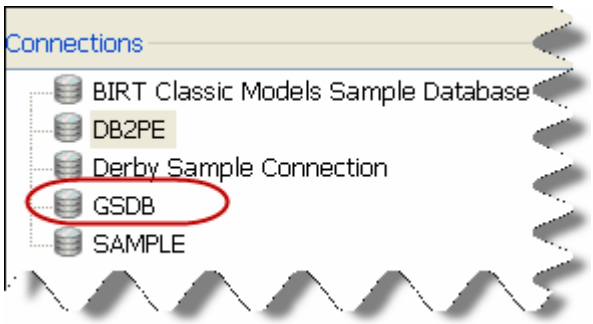
__26. Create a new data project. In the drop down menu at the top of the Data Studio Developer, select *File*, then: `New ⇨ Project`.

__27. Expand the *Data* folder, the choose: *Data Development Project*

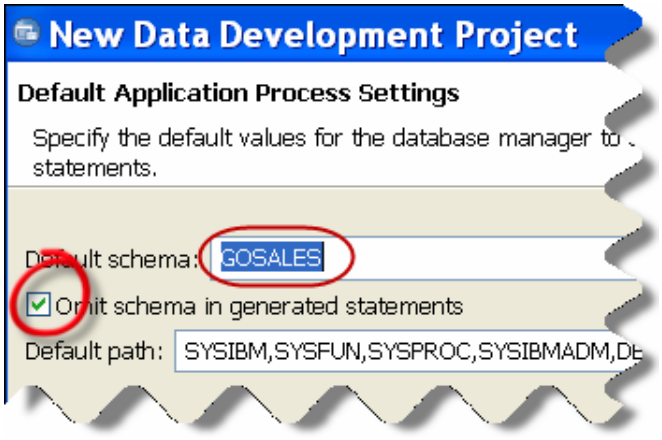
a. In <Next> screen, specify *PDQPOT* as the project name and *GOSALES* as the schema name and click `<Next>`.



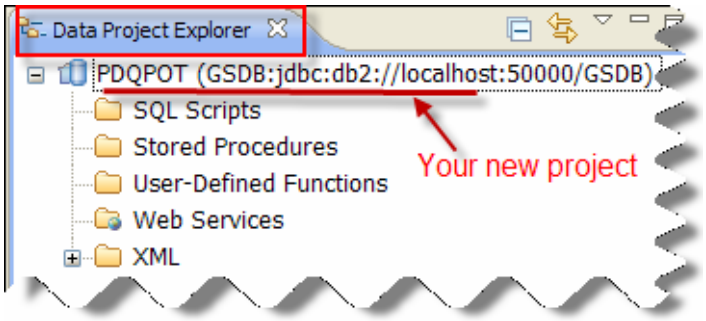
b. Choose the *GSDB* database. `<Next>`



c. Choose *GOSALES* as the default schema. `<Finish>`

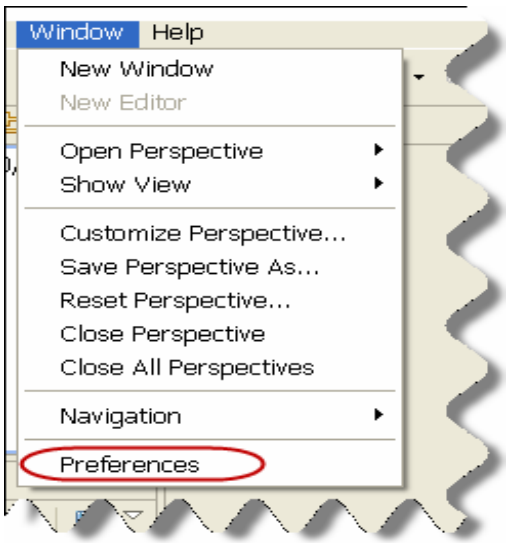


__28. In the top left quadrant of your *data* perspective, you will see your *Data Project Explorer*. Here is where your projects are managed. We'll be using it next to put our stored procedure code.

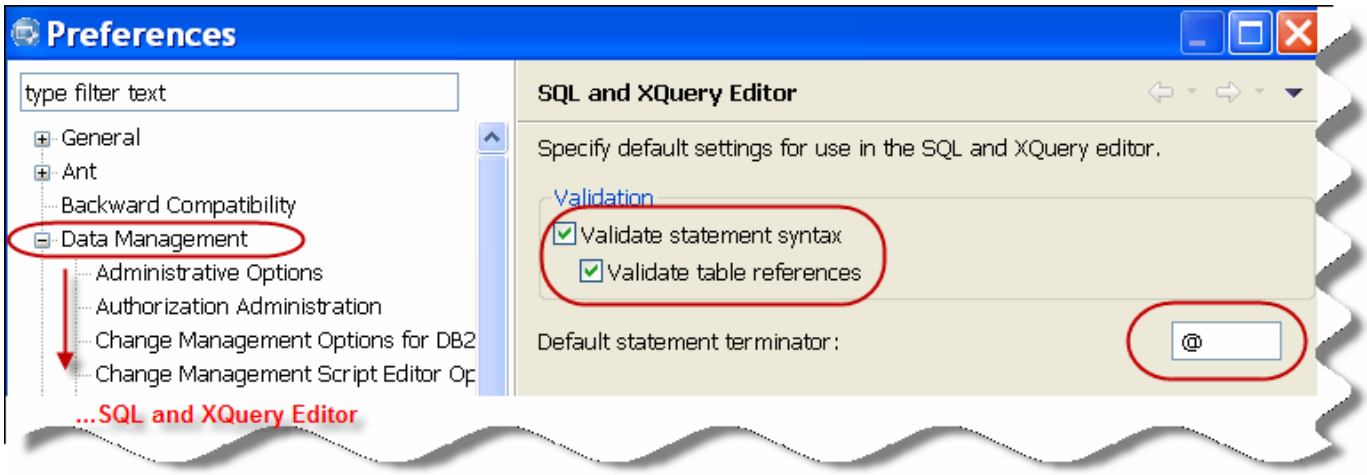


Customize your Editor Settings

__29. Go to the Data Studio Developer drop down menu bar and find *Window*, then choose: `Preferences`



__30. 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.

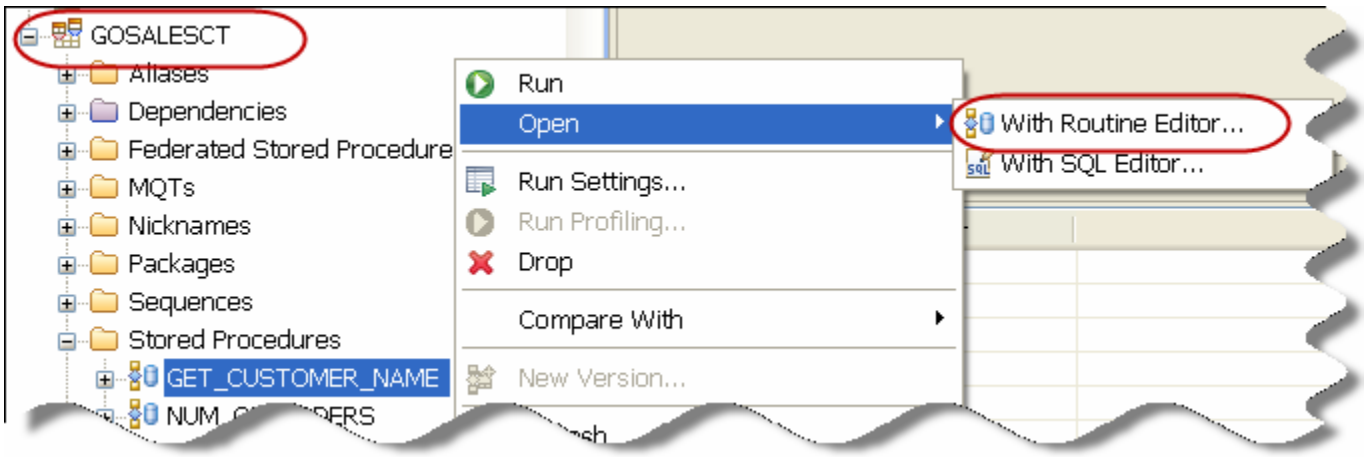


__31. Click: `<Apply> <OK>`

Bring stored procedure code into your project

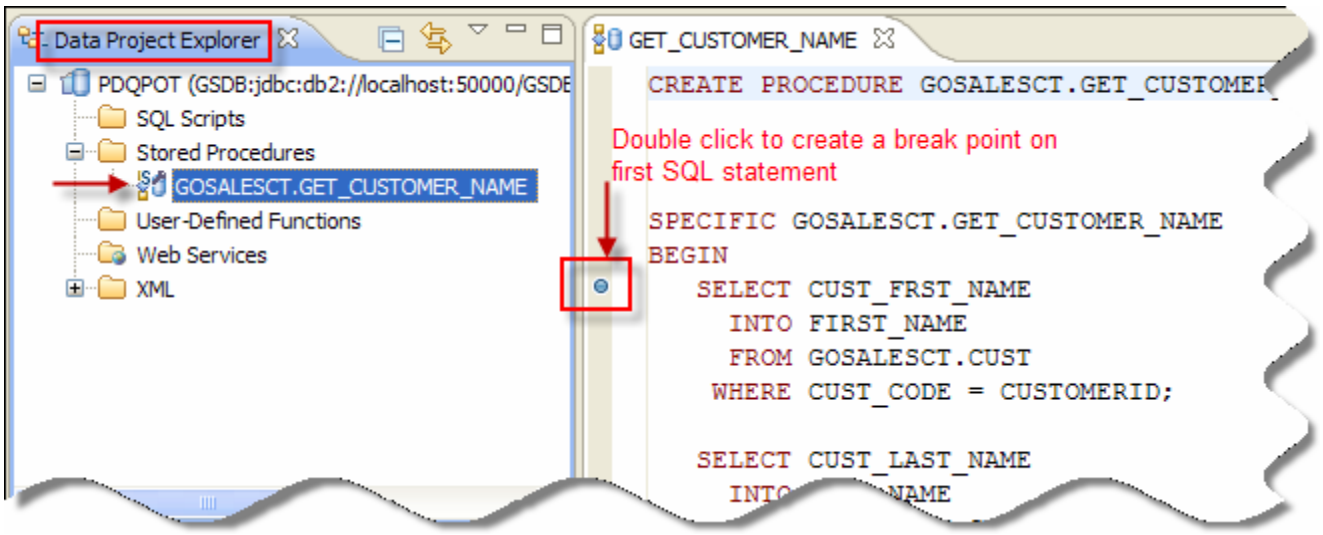
__32. In your *Data Source Explorer*, find the *GSDB* database schema called *GOSALESCT.*

__33. In this schema, find stored procedure *GET_CUSTOMER_NAME.* Right click on it then choose*: Open ⇨ With Routine Editor…*



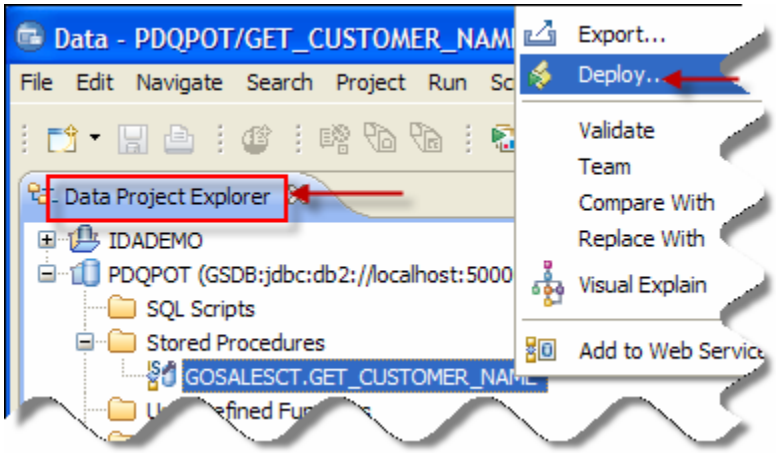
__34. Choose the project you just created (`PDQPOT`) as the target to place this code and click `<Finish>.`

__35. Notice in the *Data Project Explorer*, the SQL PL for this stored procedure has been placed in your project *PDQPOT*. Also, the SQL PL editor has this code loaded and is ready for you to start work with this code.



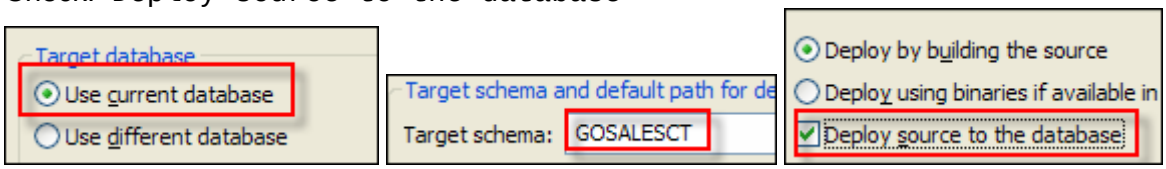
Debug your stored procedure

__36. 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`.

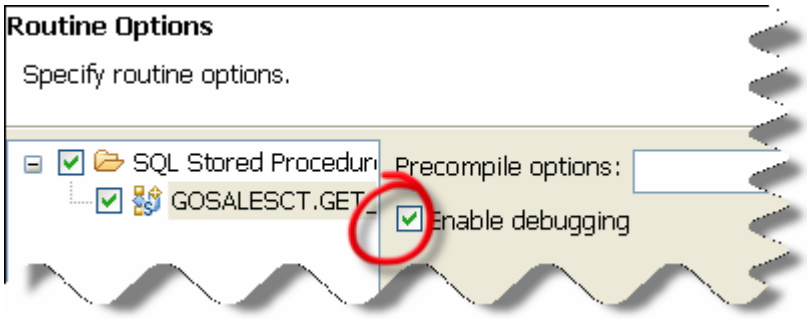


__37. In the *Deploy Routines* assistant, the *Deploy Options* screen, do the following:

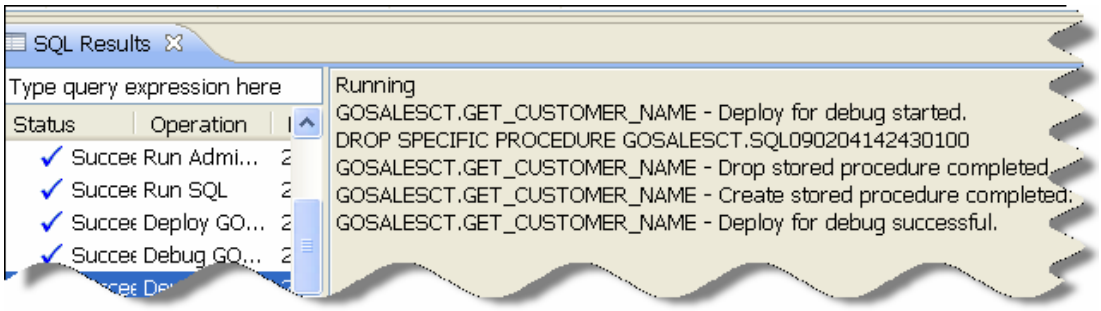
Select: `Use current database.`
Select: `Target Schema: GOSALESCT.`
Check: `Deploy source to the database`



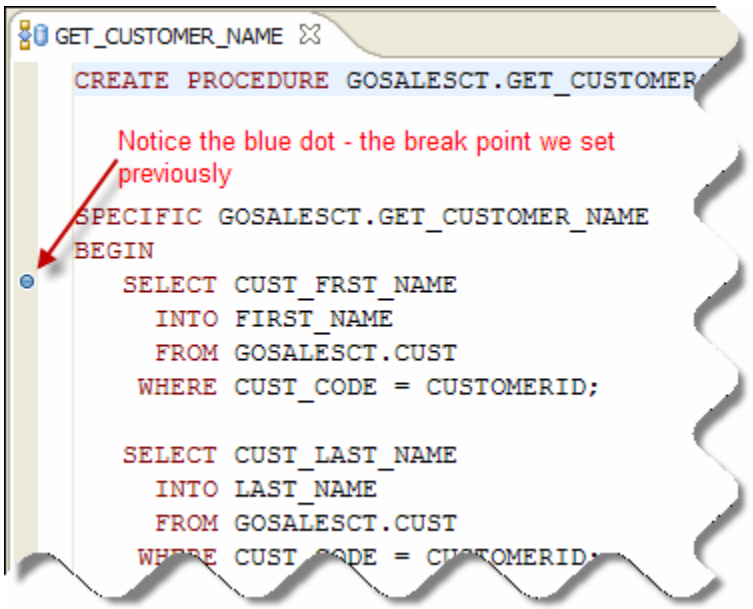
__38. In the next *Routine Options* screen, make sure you check: `Enable debugging.` Then `<Finish>.` You will now be able to debug this stored procedure from the *Data Project Explorer*.



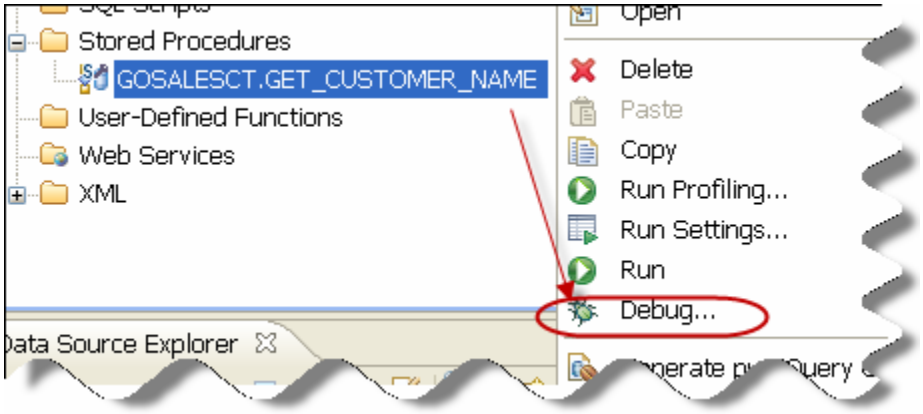
__39. The stored procedure code is now in your database, with the debug capability turned on. You should be able to see the successful deploy in your *SQL Results* view.



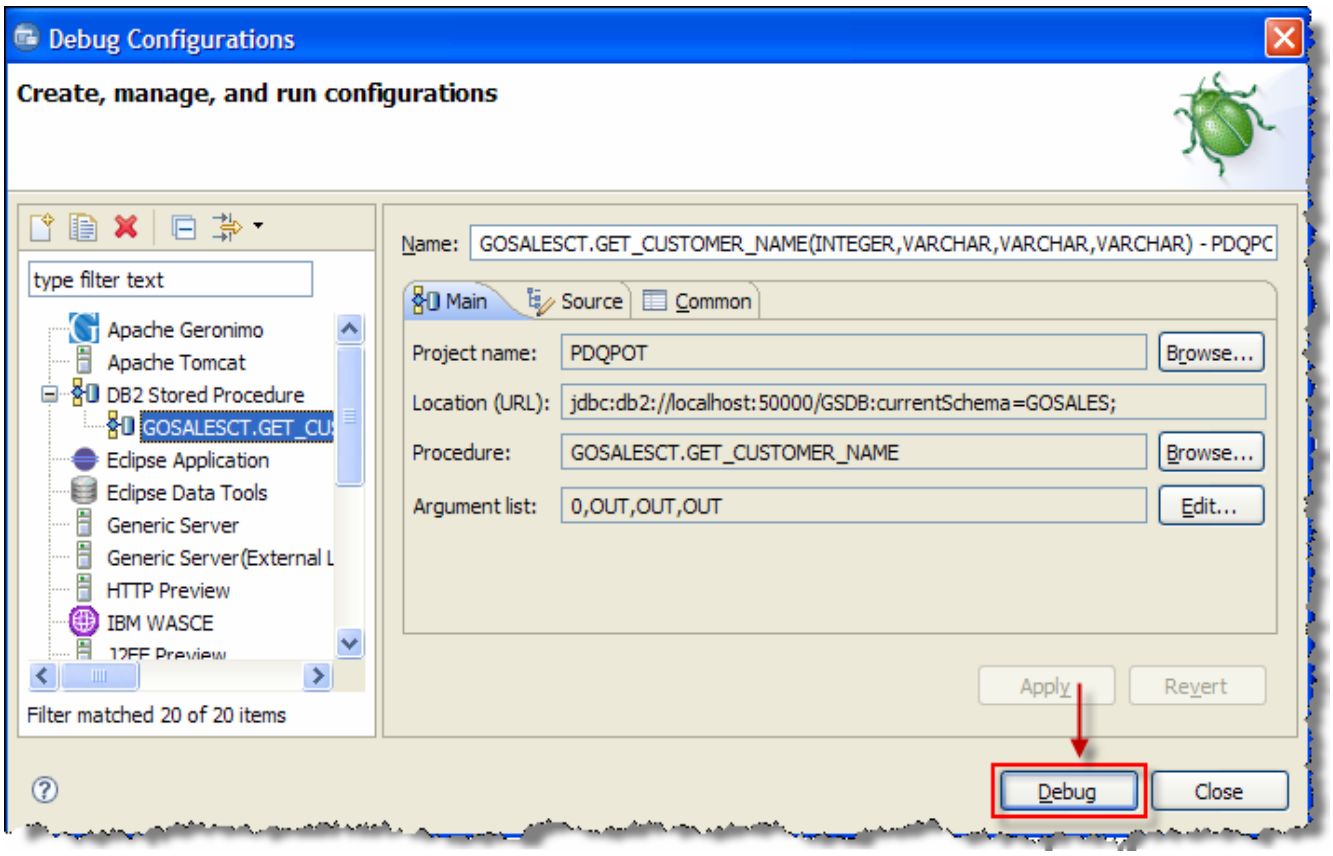
__40. Next, set any breakpoint you might need in the SQL editor itself. Do this by double clicking on the yellow boarder to the left of your code. A blue breakpoint dot will appear.



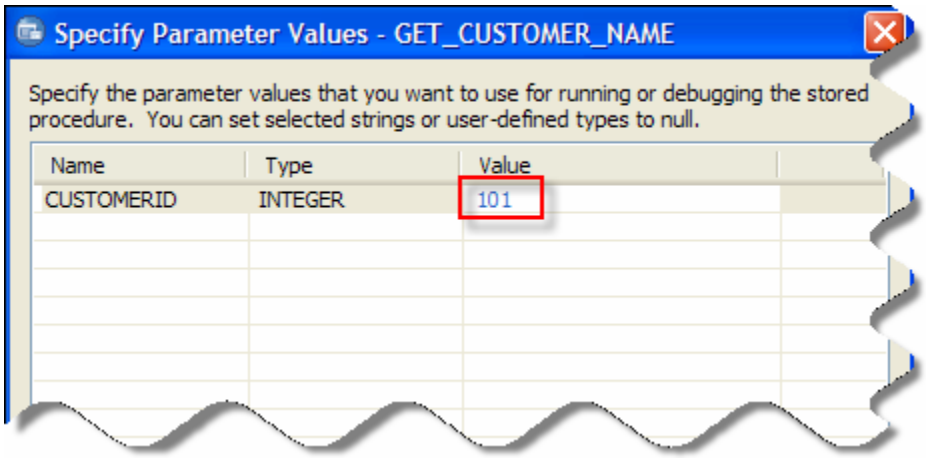
__41. Next, right click on *GOSALESCT.GET_CUSTOMER_NAME* stored procedure in *Data Project Explorer* view and choose: `debug`.



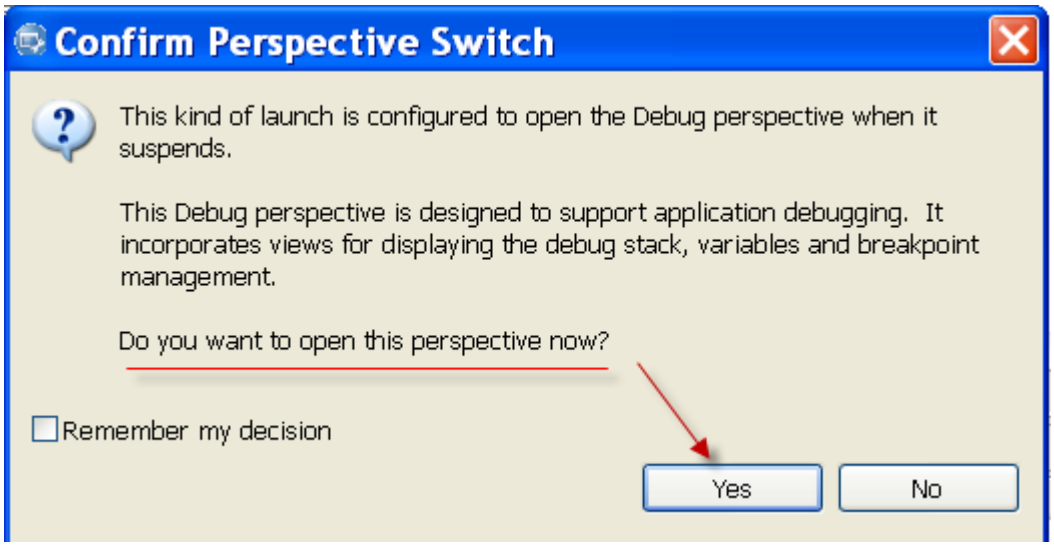
__42. In `<Next>` window, accept the default and click on `Debug` button.



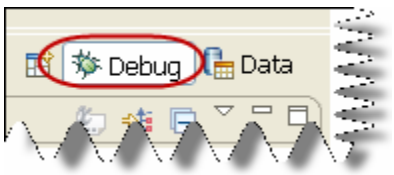
__43. Since this stored procedure expects an input parameter, you will see a window asking for a value for the *CUSTOMERID* parameter. Specify a value of *101* and click on <OK>.



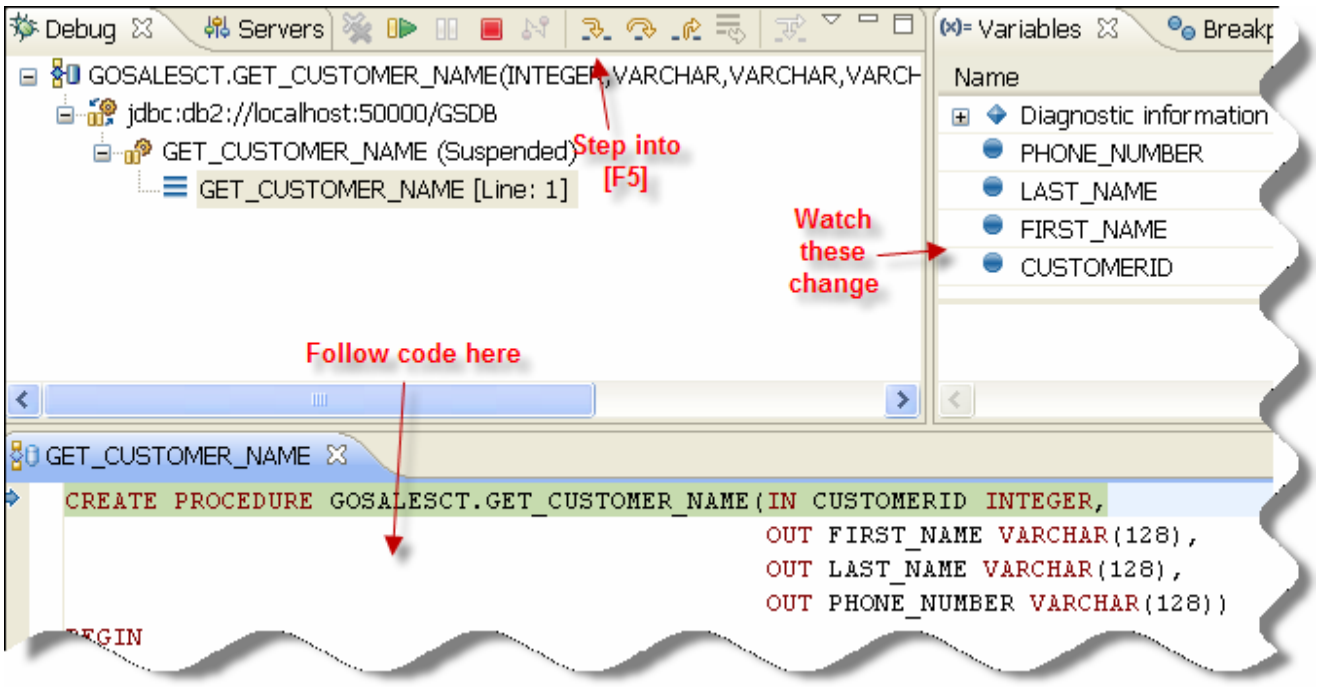
__44. Data Studio Developer is now doing something you haven't seen before. It is switching perspectives. It will now open the *Debug* perspective if you let it. Say *Yes* to open this perspective.



__45. 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.



__46. Go to *Debug* view in the *Debug Perspective* and either press *F5* 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 image a very large many-lined stored procedure you can debug.



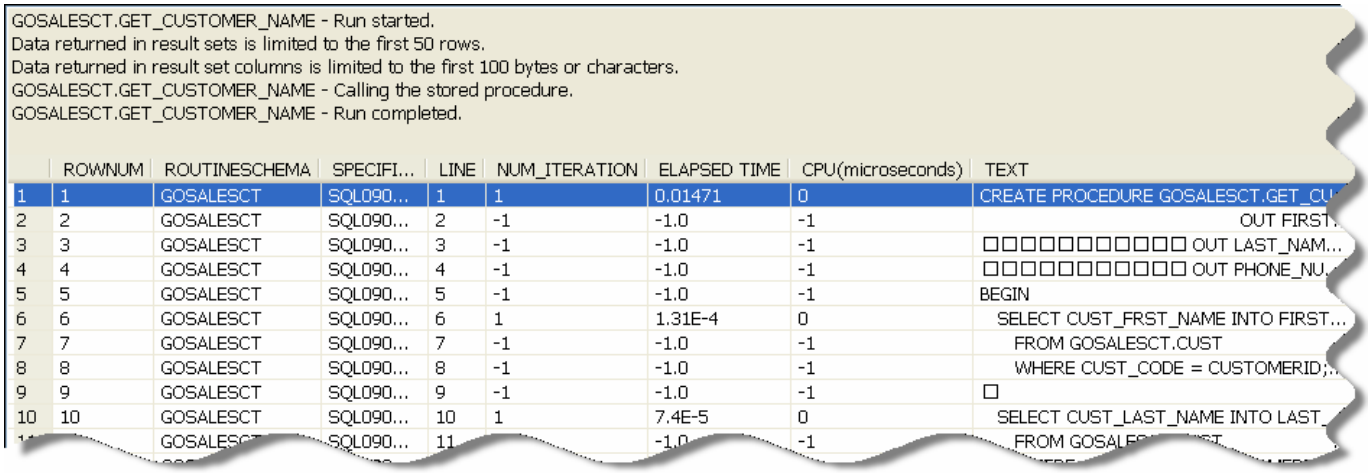
__47. If you run all the way through the code and the session is terminated, then just right click on the terminated session itself and choose: `relaunch.`

__48. Close the debug perspective when done.

Profile a stored procedure

__49. Profiling allows you to see each line of code and how many times it is executed for a particular running of that code. To see this right click on your stored procedure in the *Data Project Explorer* and then: `Run Profiling`

__50. Choose all defaults and all options. After it runs, double click on the *SQL Results* view to see your profiling in full screen mode.



GOSALESCT.GET_CUSTOMER_NAME - Run started.
Data returned in result sets is limited to the first 50 rows.
Data returned in result set columns is limited to the first 100 bytes or characters.
GOSALESCT.GET_CUSTOMER_NAME - Calling the stored procedure.
GOSALESCT.GET_CUSTOMER_NAME - Run completed.

	ROWNUM	ROUTINESCHEMA	SPECIFI...	LINE	NUM_ITERATION	ELAPSED TIME	CPU(microseconds)	TEXT
1	1	GOSALESCT	SQL090...	1	1	0.01471	0	CREATE PROCEDURE GOSALESCT.GET_CU
2	2	GOSALESCT	SQL090...	2	-1	-1.0	-1	OUT FIRST
3	3	GOSALESCT	SQL090...	3	-1	-1.0	-1	□□□□□□□□□□ OUT LAST_NAM...
4	4	GOSALESCT	SQL090...	4	-1	-1.0	-1	□□□□□□□□□□ OUT PHONE_NU
5	5	GOSALESCT	SQL090...	5	-1	-1.0	-1	BEGIN
6	6	GOSALESCT	SQL090...	6	1	1.31E-4	0	SELECT CUST_FRST_NAME INTO FIRST...
7	7	GOSALESCT	SQL090...	7	-1	-1.0	-1	FROM GOSALESCT.CUST
8	8	GOSALESCT	SQL090...	8	-1	-1.0	-1	WHERE CUST_CODE = CUSTOMERID;
9	9	GOSALESCT	SQL090...	9	-1	-1.0	-1	□
10	10	GOSALESCT	SQL090...	10	1	7.4E-5	0	SELECT CUST_LAST_NAME INTO LAST
11		GOSALESCT	SQL090...	11		-1.0	-1	FROM GOSALES

Notice you can run a stored procedure with profiling to break down each line of code to see how many times it ran, how long it took and so on.

**** End of pureQuery lab 01: 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.

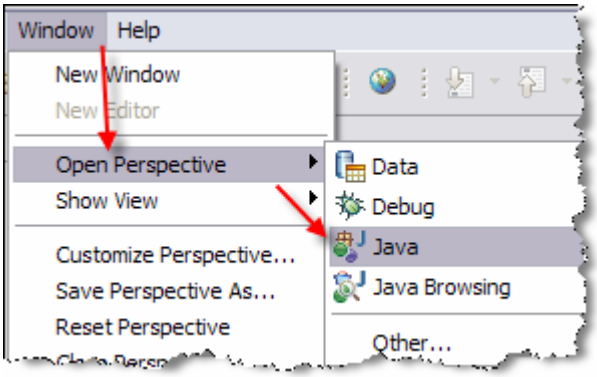
2.1 Creating a new Java Project

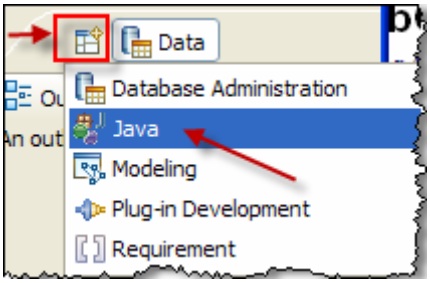
__1. Make sure that the Data Studio is open in the *Java Perspective.*

If Data Studio is **not** in the *Java Perspective* you can open one from the top left window of the Data Studio window and choose:

`Window ⇨ Open Perspective ⇨ Java`

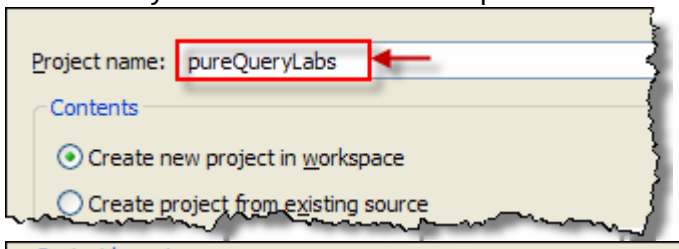
You should now see:

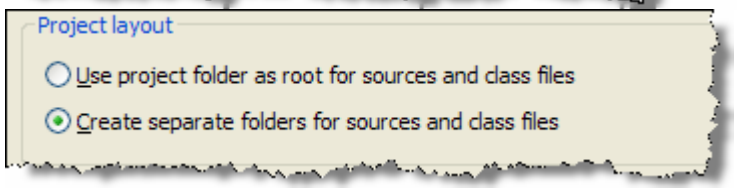




__2. Now create a new Java project by going to:

- `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`:

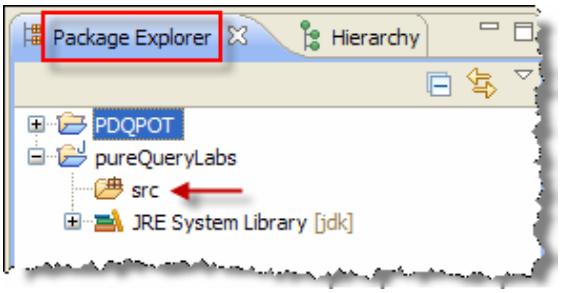




Please make sure that you type the name of the project correctly. The later labs will fail if there is a typo in the name.

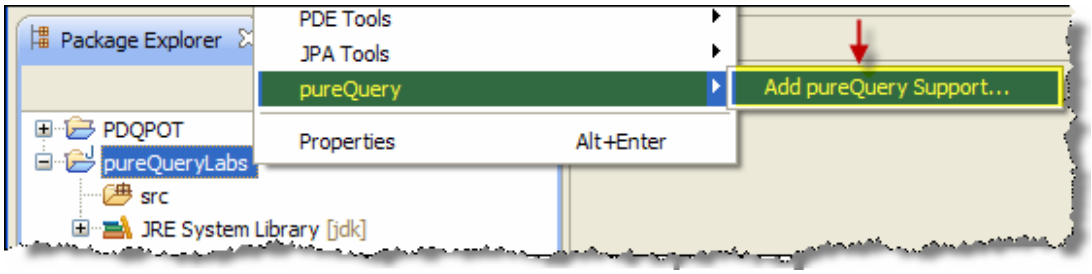
- Click `<Next>.`
- Click `<Finish>` again and it may come up with a dialog box saying that this type of project is associated with Java perspective. Click `<Yes>.`

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

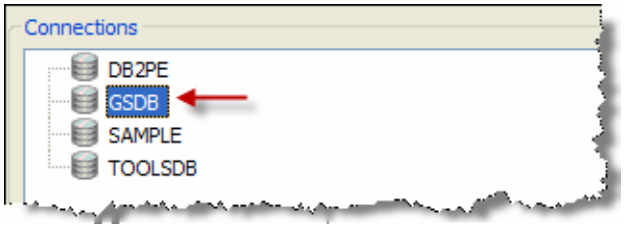


2.2 Enable Java project for pureQuery

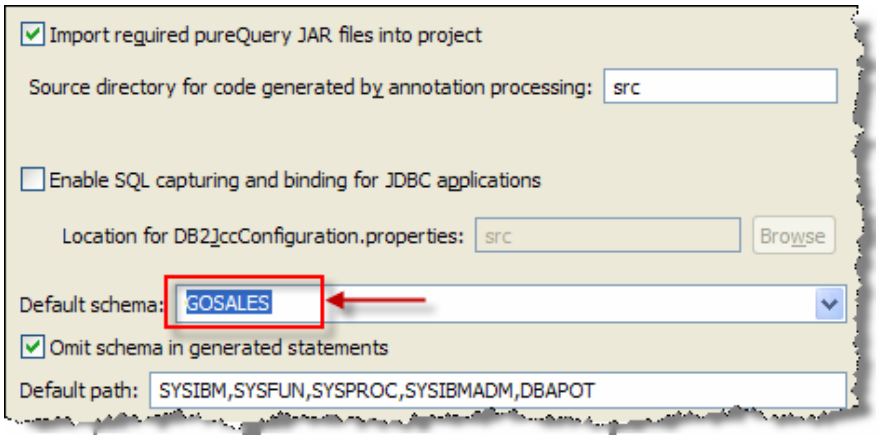
__3. Right click on pureQueryLabs java project in Package Explorer view and click on `pureQuery` and click again on "`Add pureQuery Support`".



__4. Select GSDB database and click `<Next>`

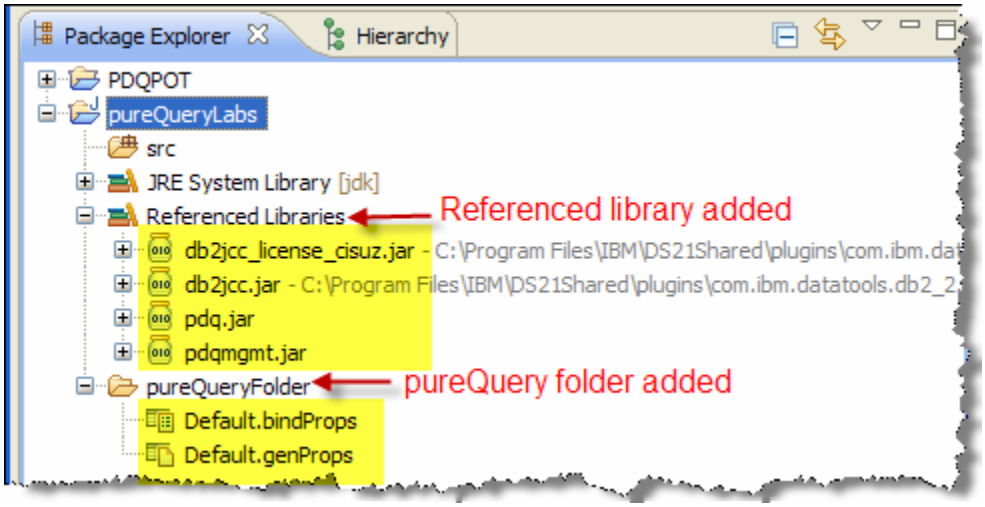


__5. Check default schema `GOSALES` and click `<Finish>.`



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

__6. 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.

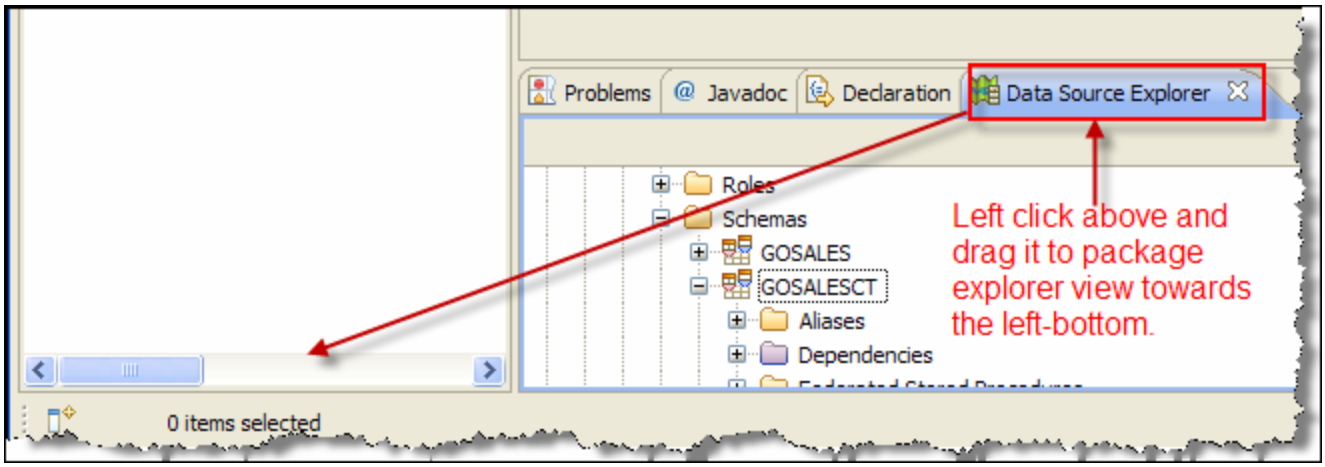


2.3 Enable Data Explorer View in Java project

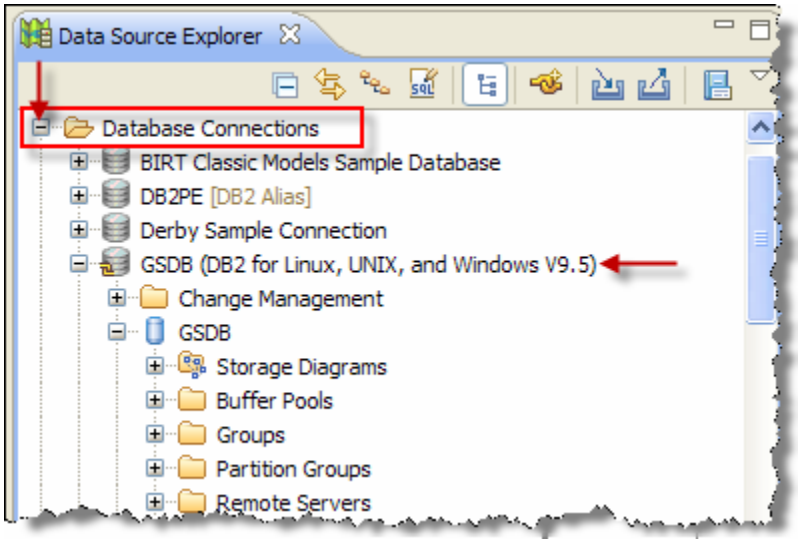
__7. Now we want to add the *Data Source Explorer* View to this perspective. To do this we will drag and drop this view to the package explorer.

__8. **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.

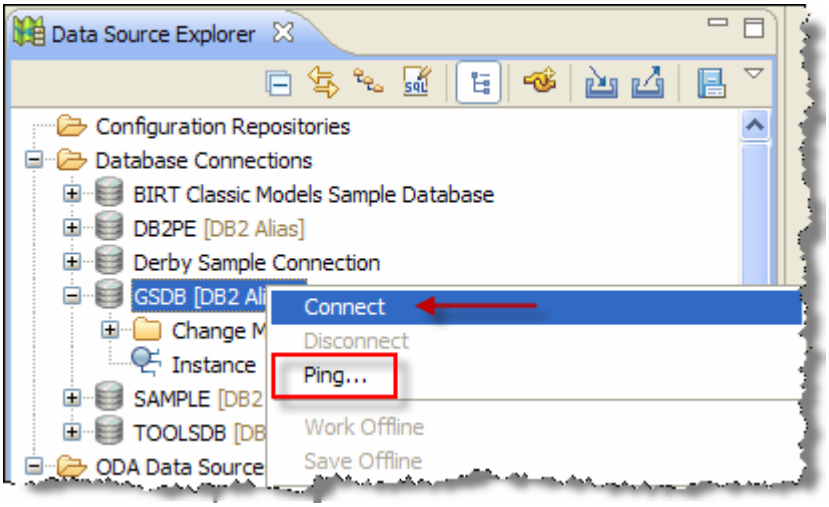
__9. Drag and drop the *Data Source Explorer* View right below the Package Explorer for a better view of the connections. The *Data Source Explorer* View will show all the available connections, if any, of your default database.



__10. Expand the database connections to see all that are available to you



__11. 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.



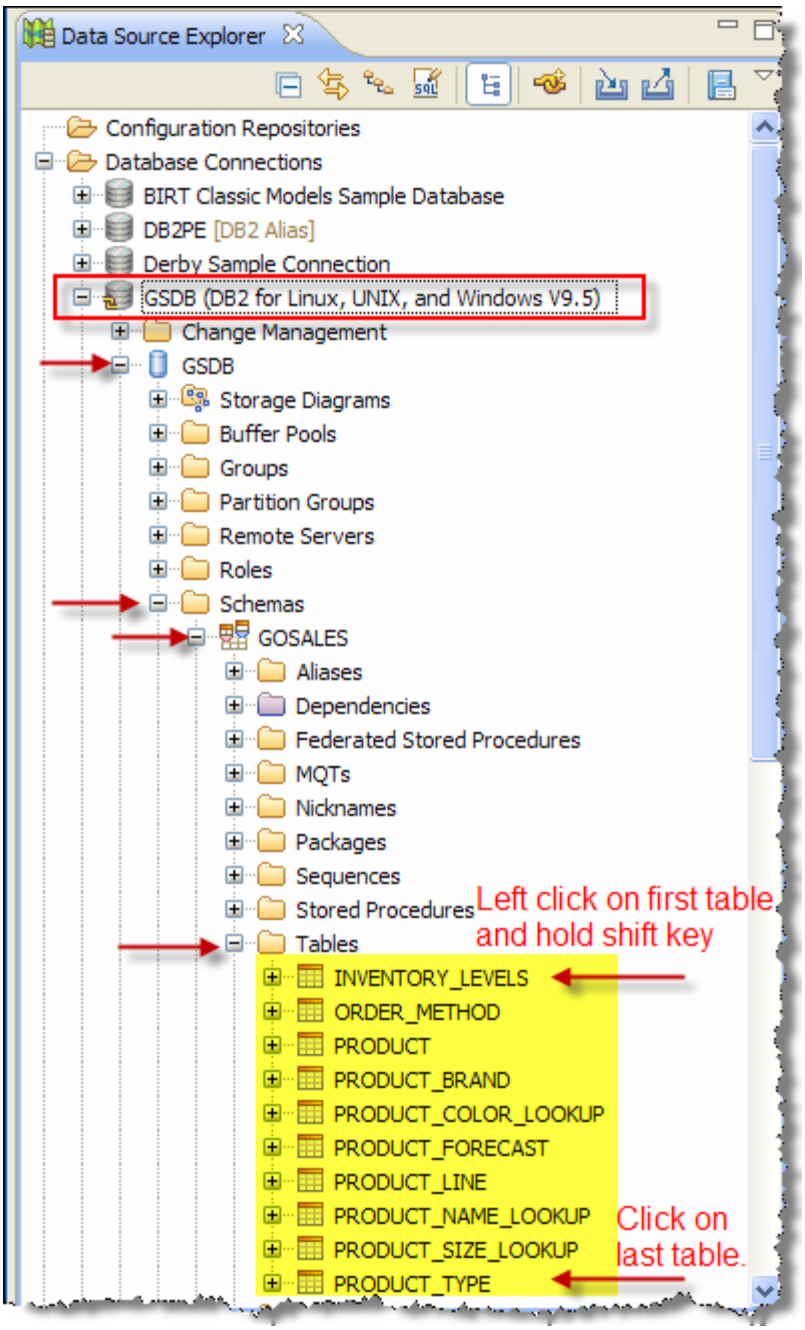
You are now ready to begin working with pureQuery code from the GSDB database.

**** End of lab 02: Create pureQuery Project**

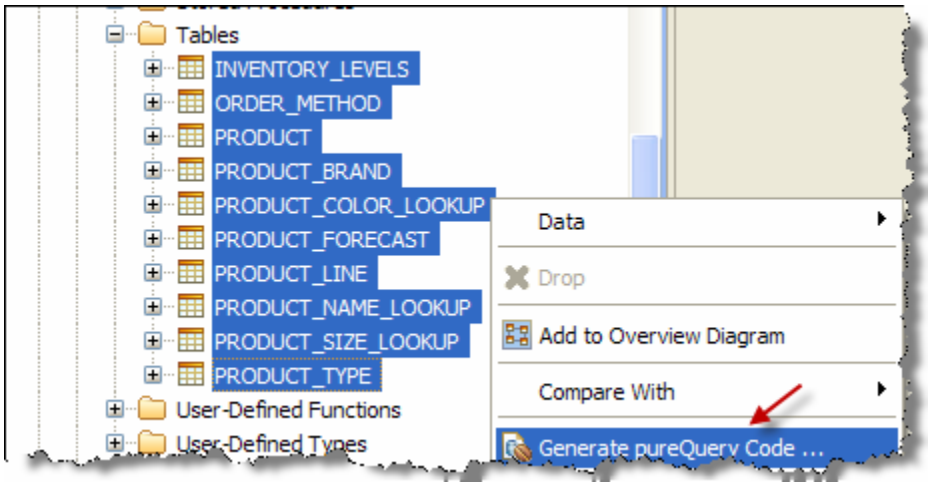
Lab 3 Explore pureQuery Tools

3.1 Generate pureQuery code from database tables

__1. Expand database GSDB, then expand Schemas, then GOSALES, and finally Tables. Select all tables. (Click on the first table, hold the shift key and click on the last table).

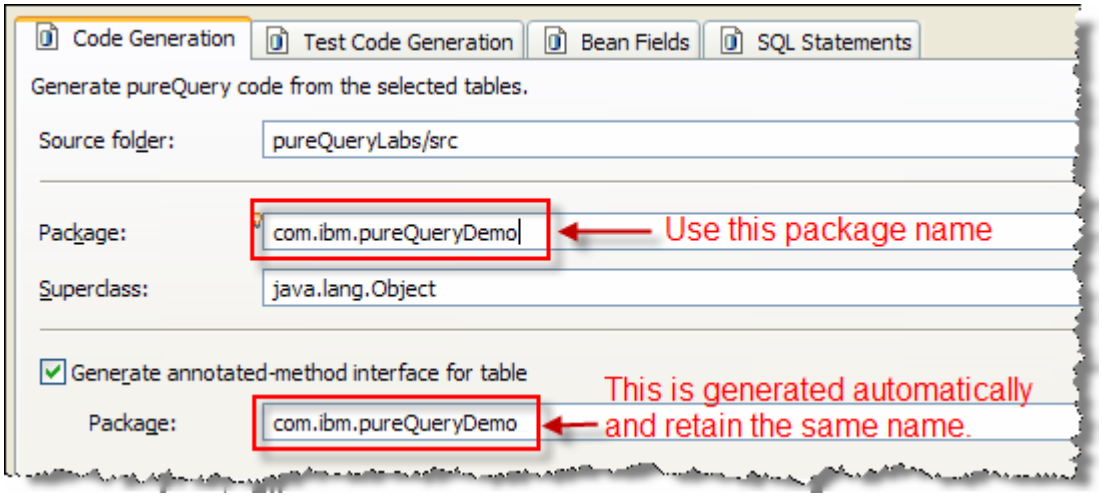


__2. Right click on any one of the selected tables and click on `Generate pureQuery Code`.



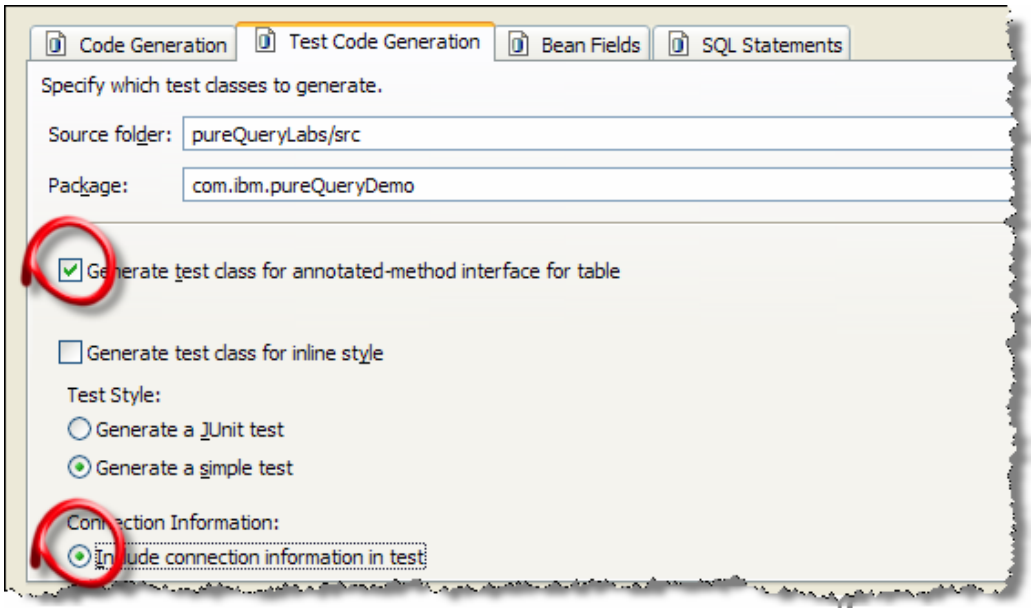
__3. The Generate pureQuery Code for a Table window will open.

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

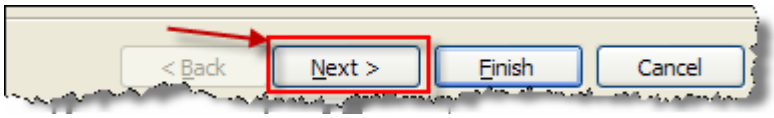


- While you are on this tab, browse through tabs on `Test Code Generation`, `Bean Fields` and `SQL Statements`. You can make high level selections that will be applicable to all tables we selected in our previous step. Here we will select options for `Test Code Generation` so that they are applicable to all tables selected.

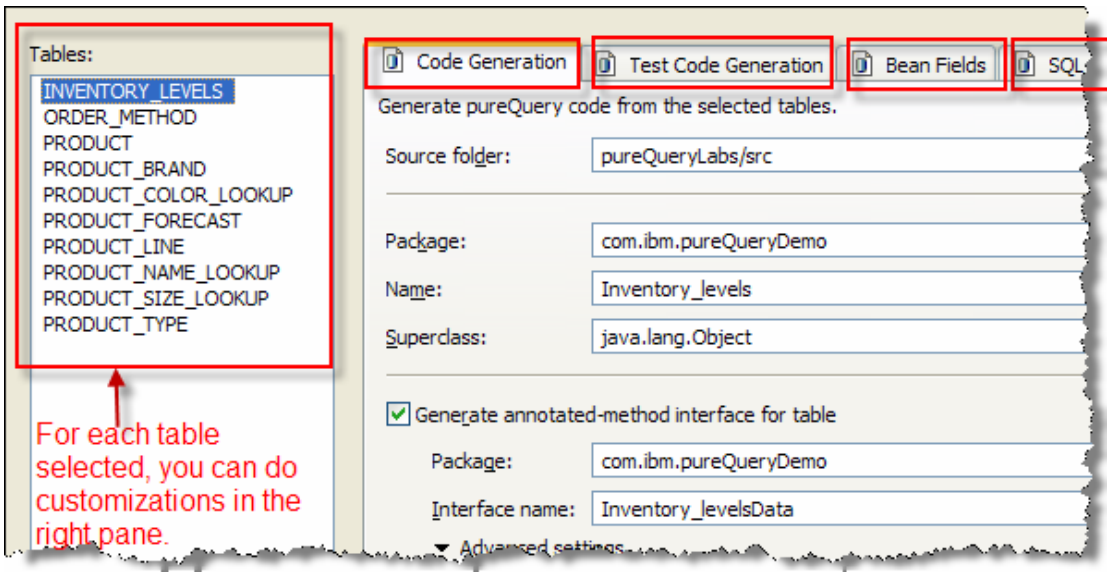
- Check radio button to `Generate test class for annotated-method interface for table` and make sure that you have also selected option for `Include connection information in test`.



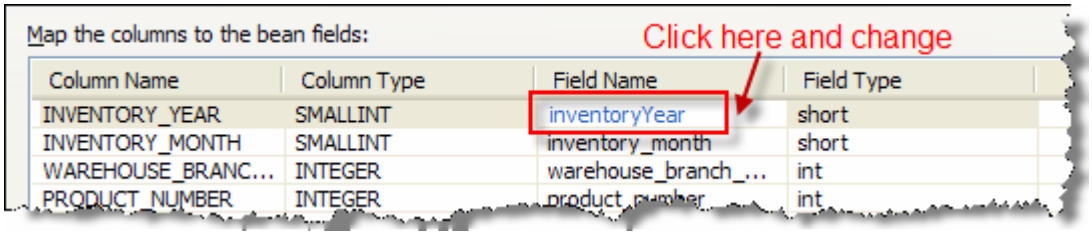
- Click `<Next>` button to go to the next screen.



- 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 *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`.



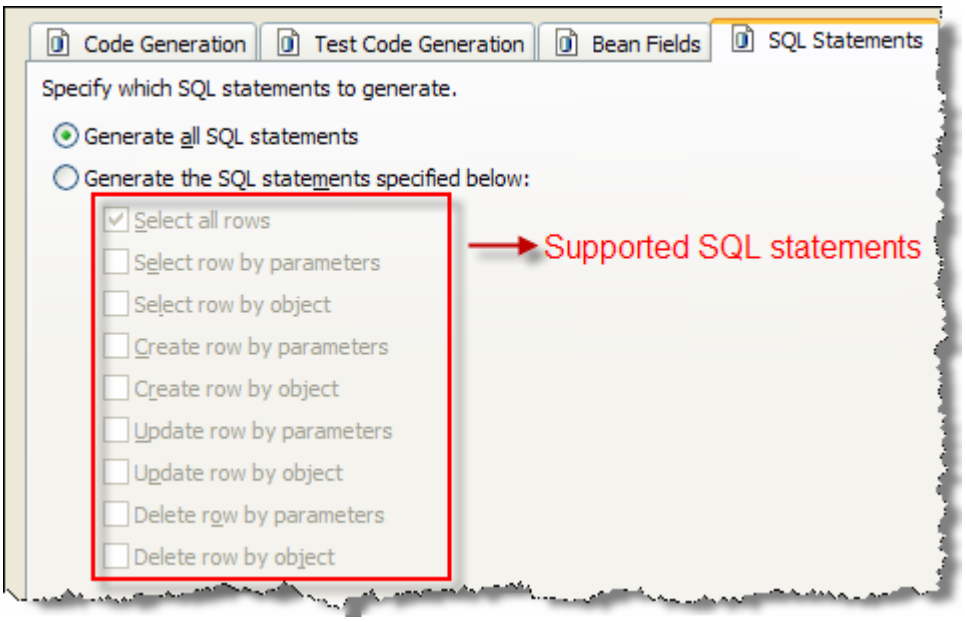
- The `inventoryYear` is mapped to the database column `INVENTORY_LEVEL.INVENTORY_YEAR` by adding the following annotations to the variable and to the setter and getter methods. (The following is an example of that mapping which will happen after you are done doing this step):

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

…and…

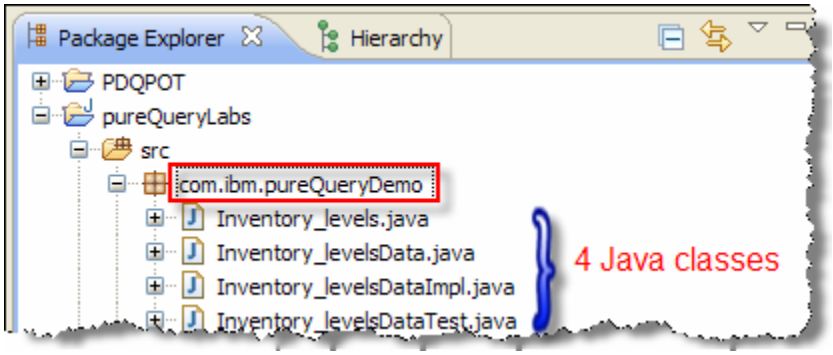
```
@Column(name="INVENOTORY_YEAR") public String getInventoryYear() {
return inventoryYear;
}
```

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



- Now click `<Finish>`.

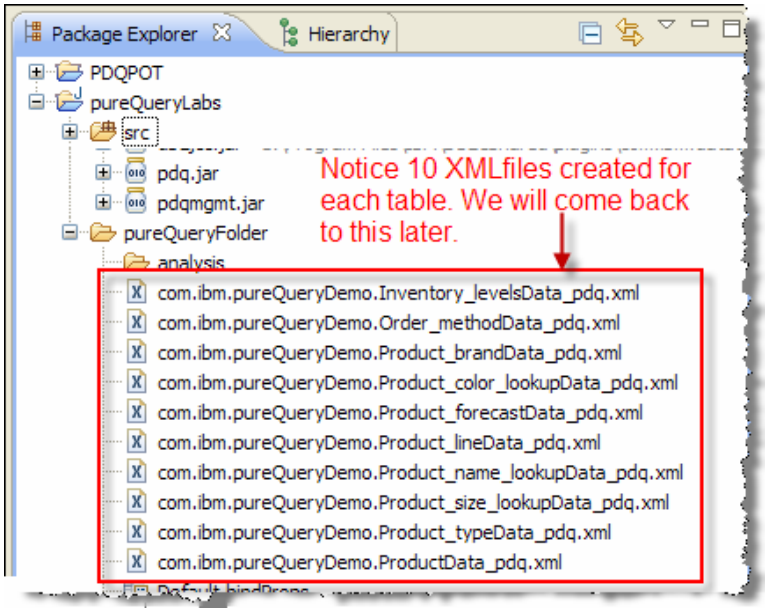
__4. Notice that a java package `com.ibm.pureQueryDemo` has been created with 40 classes for 10 tables selected in the previous step. There are 4 classes for each of the table.



__5. 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.

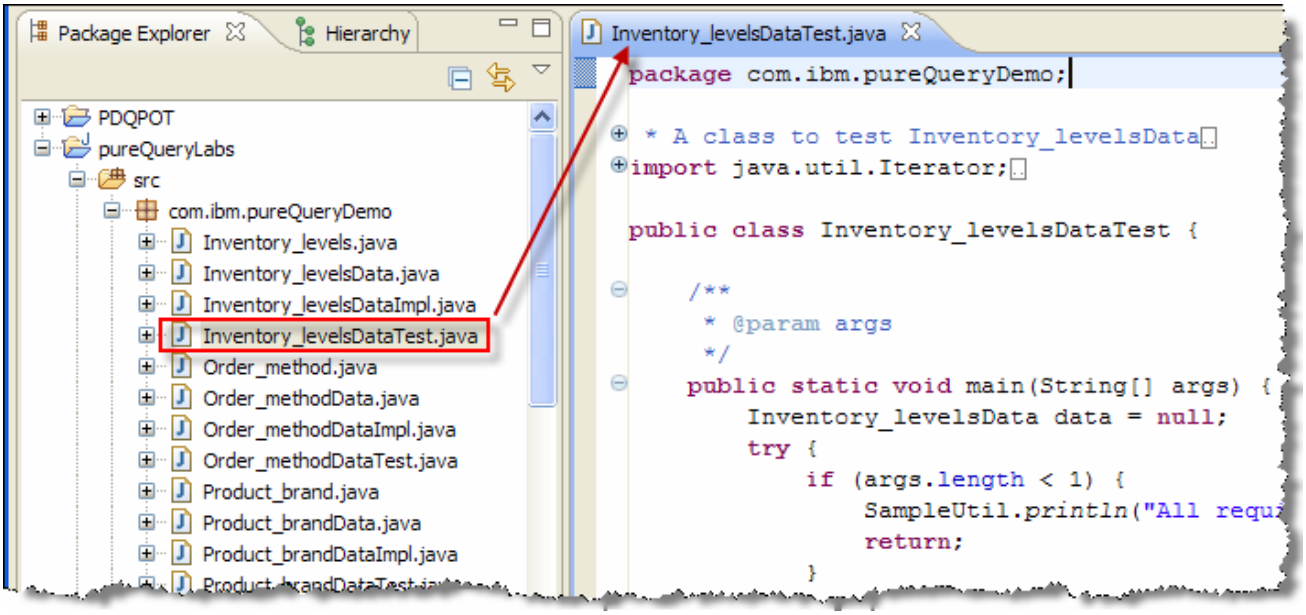
__6. Notice several XML files created for each table. We will come back to these later.



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.

__7. Select the `Inventory_levelsDataTest.java` file:



__8. 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 provided.");
    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. Did you notice `currentSchema` as part of the connection string? This is here since we specified it as a part of connection URL.

```
data = SampleUtil.getData(Inventory_levelsData.class,
        "jdbc:db2://localhost:50000/GSDB:currentSchema=GOSALES;",
        "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:

```
((Data) data).setAutoCommit(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`.

```
Iterator<Inventory_levels> getInventory_levelss = data
        .getInventory_levelss();

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");
```

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

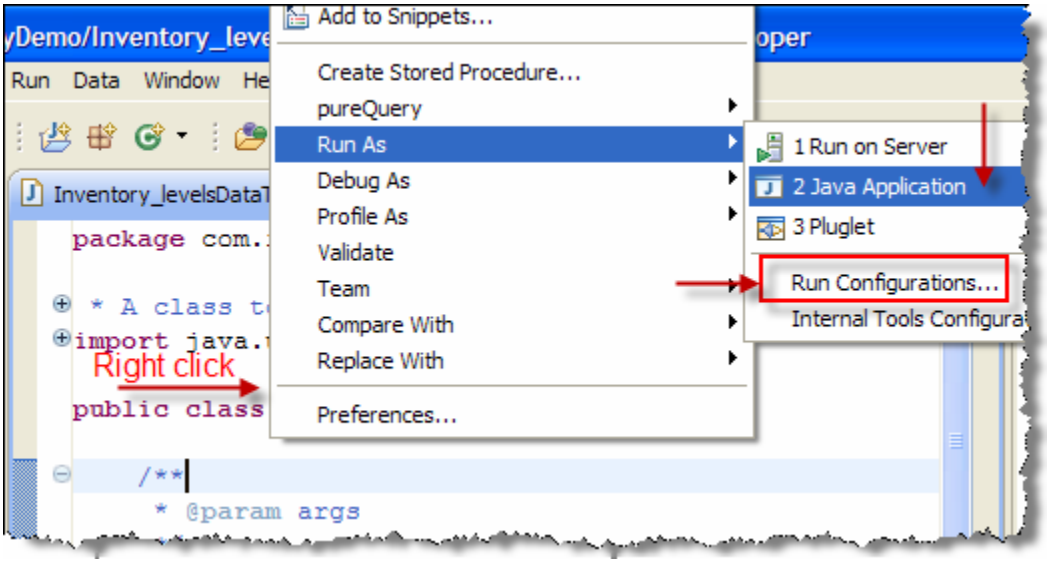
```
getInventory_levels = data.getInventory_levels(bean);
SampleUtil.println("Results for createInventory_levels(bean)");
SampleUtil.printClass(getInventory_levels);
```

- All the transactions are committed in the last statement.

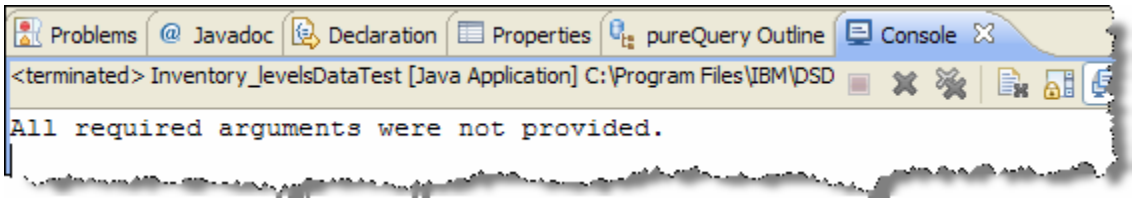
```
((Data) data).commit();
```

__9. 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.`

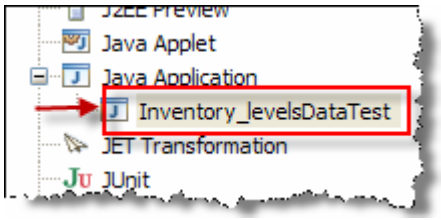


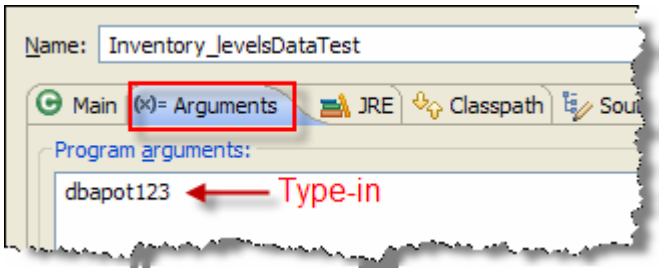
__10. You will notice "`All required arguments were not provided.`" in the console. It was expected since we did not specify the argument while running this test class.



__11. Again right click anywhere on the 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.

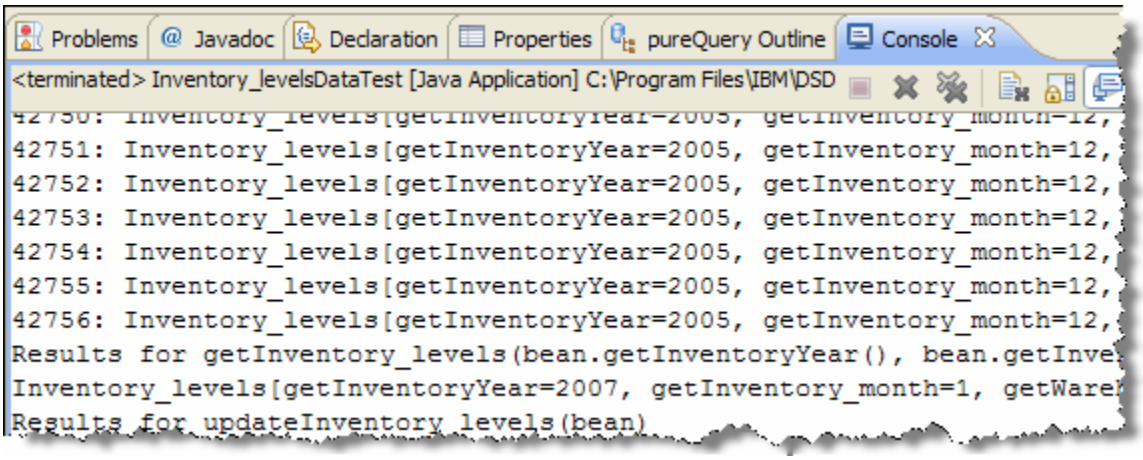
1. Type in password `dbapot123`:





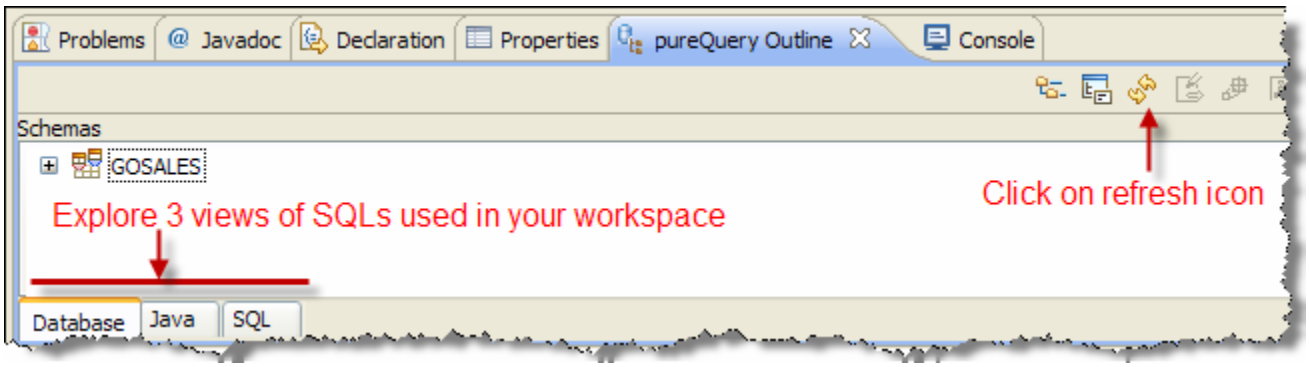
- Click <Run>

- You will see the results on the Console:

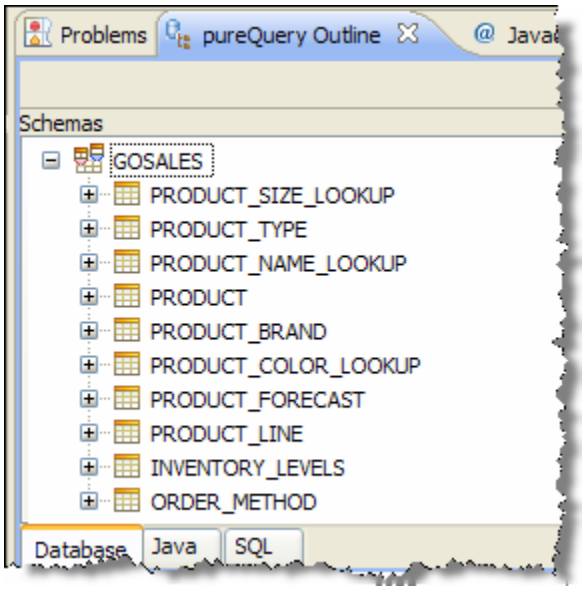


3.3 Explore pureQuery outline view

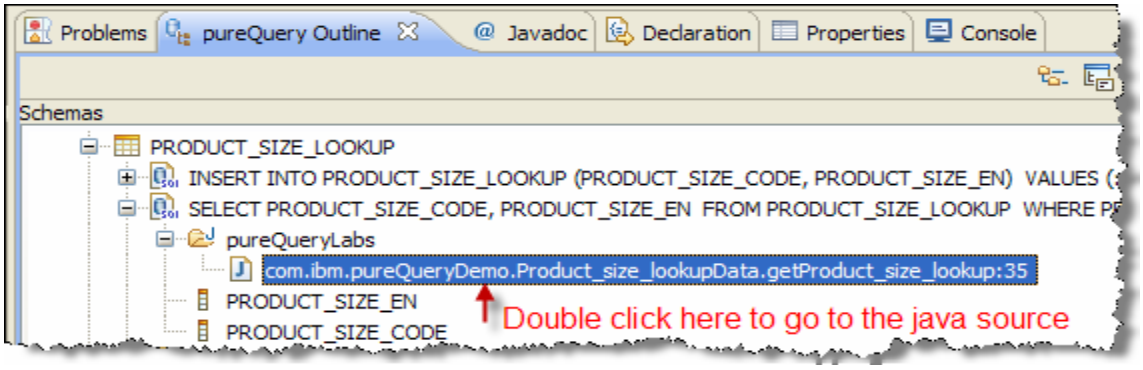
__12. You can view SQL statements used in a class (or projects in a workspace) with the help of pureQuery outline view. Click on *pureQuery Outline* view in the bottom pane.



__13. After refreshing the outline view, you will see a view as shown below.

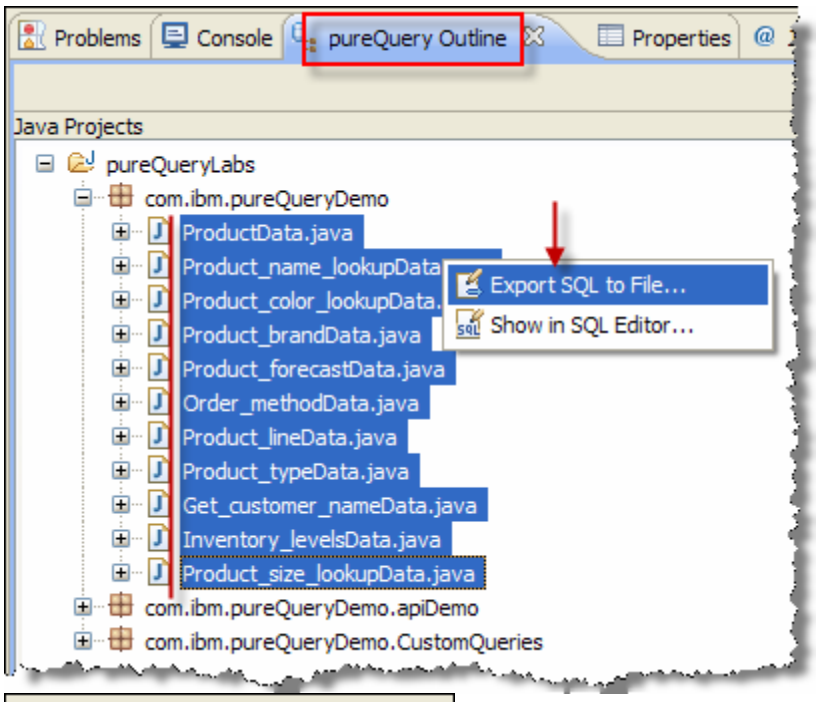


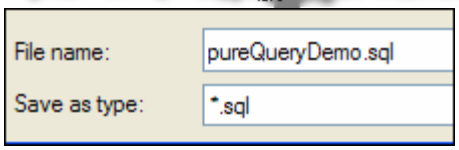
__14. Explore each view and you can easily see the relationships between java classes, SQL statements and database using different views.

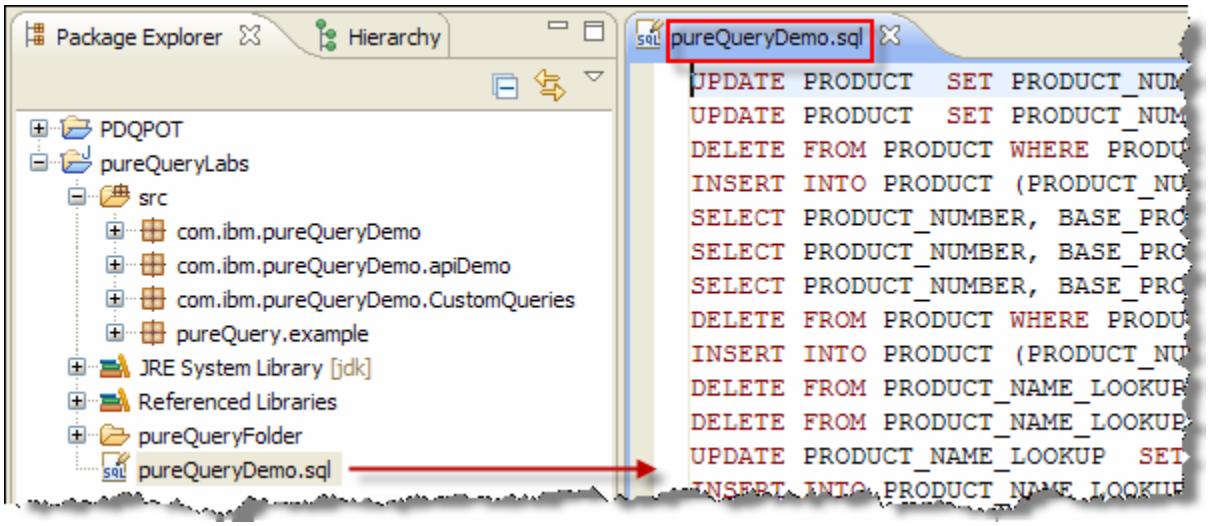


__15. Explore `Java` view 

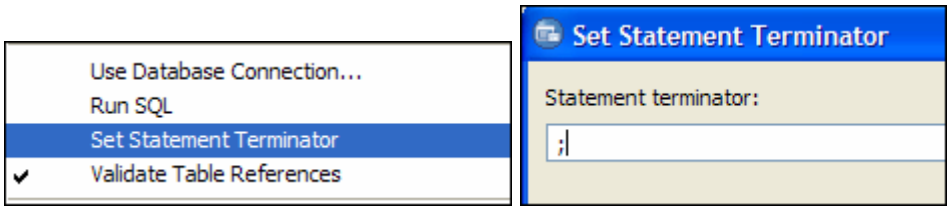
to see SQL statements used in Java classes. Expand `pureQueryDemo` package and select all java interface data access classes and right click. Select `Export SQL to File...` Save the file using any name you like and open it in an editor.



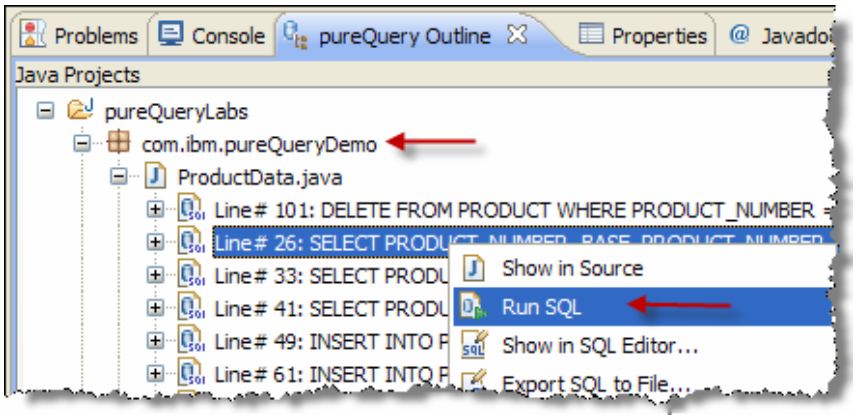




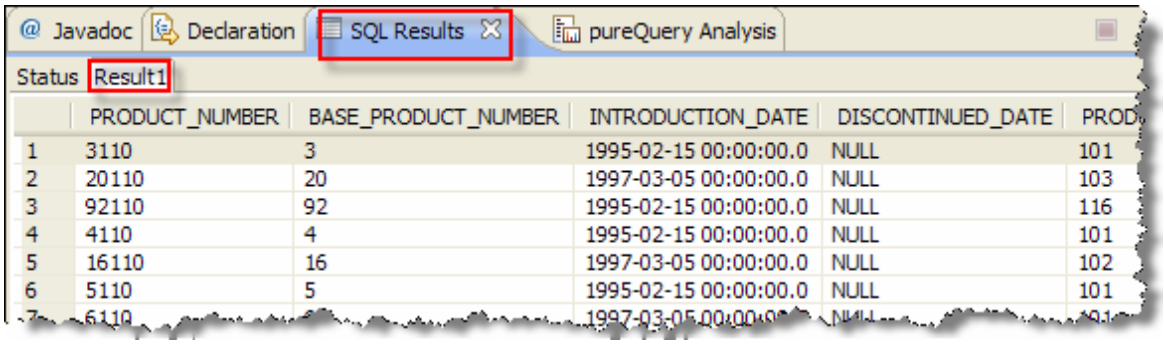
__16. Right click anywhere in the SQL file and choose option `Set Statement Terminator` and specify semicolon.



__17. In same `Java` view, expand `ProductData.java` and select `SELECT` statement and right click on it. Select `Run SQL`.

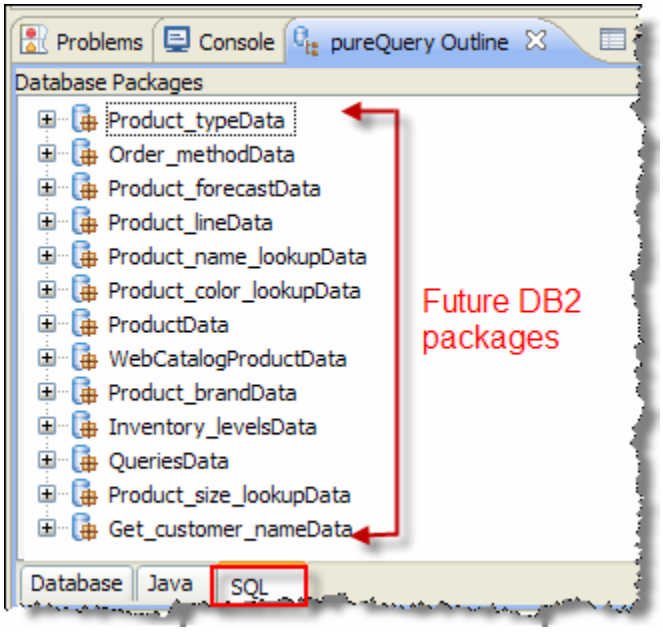


__18. The results can be viewed on `SQL Results` window.



	PRODUCT_NUMBER	BASE_PRODUCT_NUMBER	INTRODUCTION_DATE	DISCONTINUED_DATE	PROD
1	3110	3	1995-02-15 00:00:00.0	NULL	101
2	20110	20	1997-03-05 00:00:00.0	NULL	103
3	92110	92	1995-02-15 00:00:00.0	NULL	116
4	4110	4	1995-02-15 00:00:00.0	NULL	101
5	16110	16	1997-03-05 00:00:00.0	NULL	102
6	5110	5	1995-02-15 00:00:00.0	NULL	101
7	6110		1997-03-05 00:00:00	NULL	101

__19. Explore SQL view to explore SQL statements in DB2 packages. Please note that these packages are not yet created.

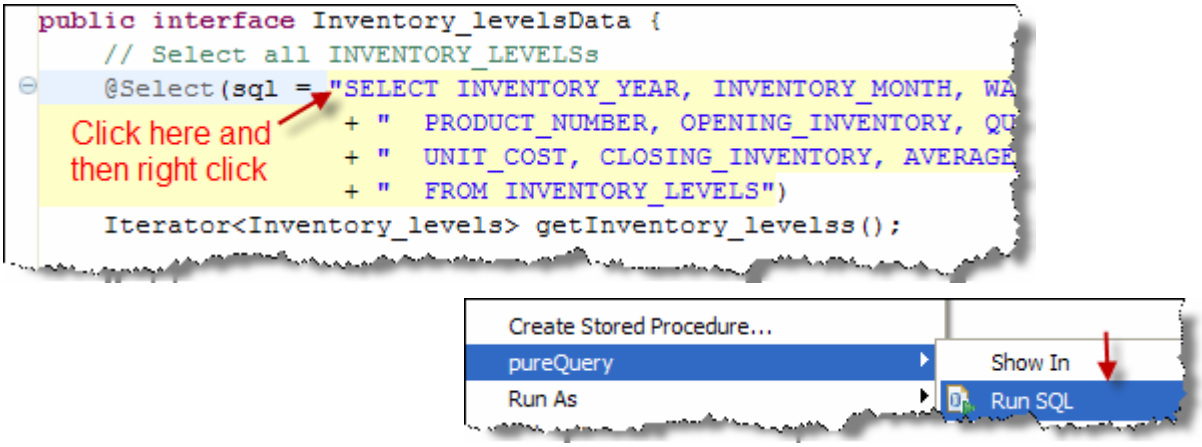


3.4 Explore pureQuery context assist capabilities

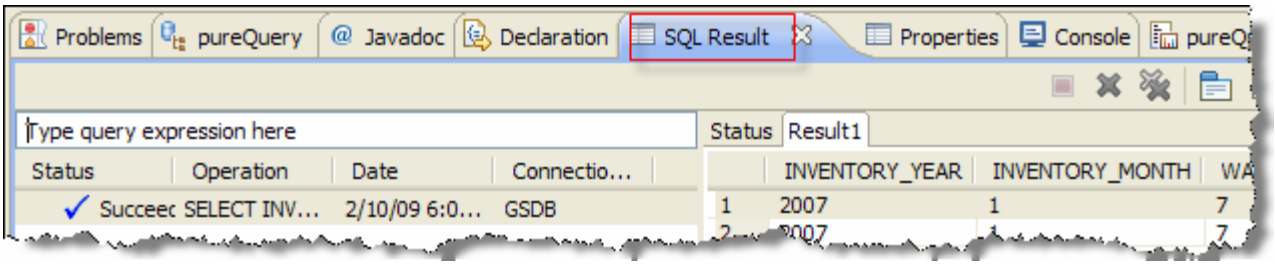
The pureQuery tools integrate the SQL editor inside the Java Editor providing developers a boost in productivity. Developers can now run SQL statements embedded in their Java programs as well as have SQL errors reported while typing the SQL statement inside the Java Editor.

__20. In the *Package Explorer*, double click on `Inventory_levelsData.java` to open the interface.

__21. Click at the beginning of the first SQL and then right click. Select `pureQuery` ⇨ `Run SQL`

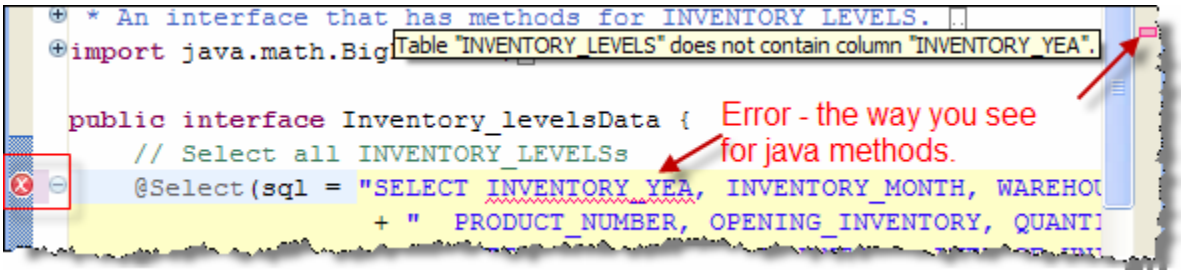


See the results of your query in the Data Output View in the bottom of the Data Studio:



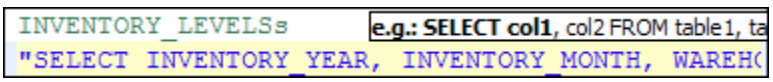
__22. While typing a SQL statement, errors will be underlined in red, just as in Java.

- 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`:



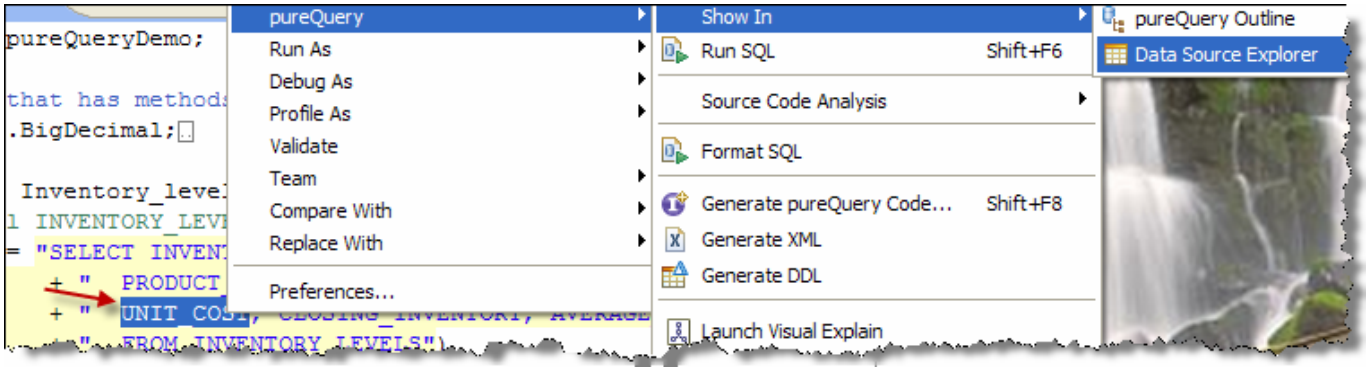
__23. Using SQL Content Assist within the Java Editor:

- 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`.

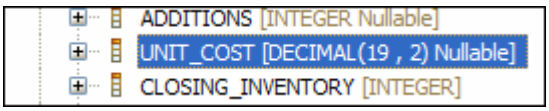


__24. If a developer wants to know the data type of a specific column or whether the column is *nullable* he/she can easily check with the help of the pureQuery tool.

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

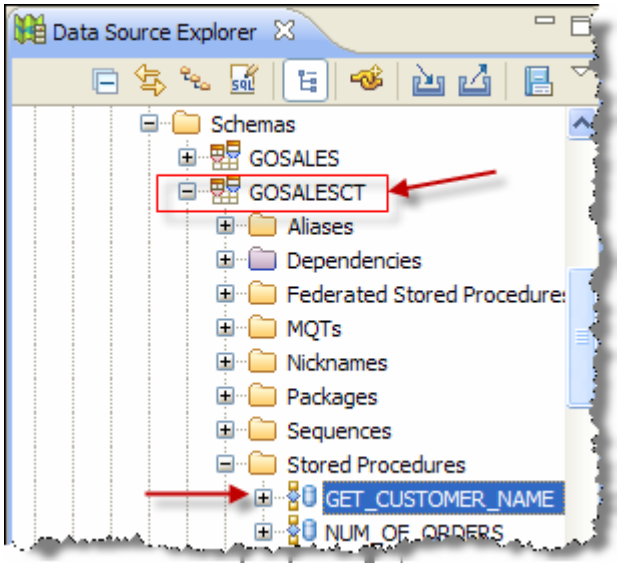


__25. You will see the *Data Source Explorer* expanding the `INVENTORY_LEVELS` table and showing the information for the columns with the `UNIT_COST` highlighted. The developer now knows that the `UNIT_COST` column is of type `DECIMAL(19,2)`and is `Nullable`:



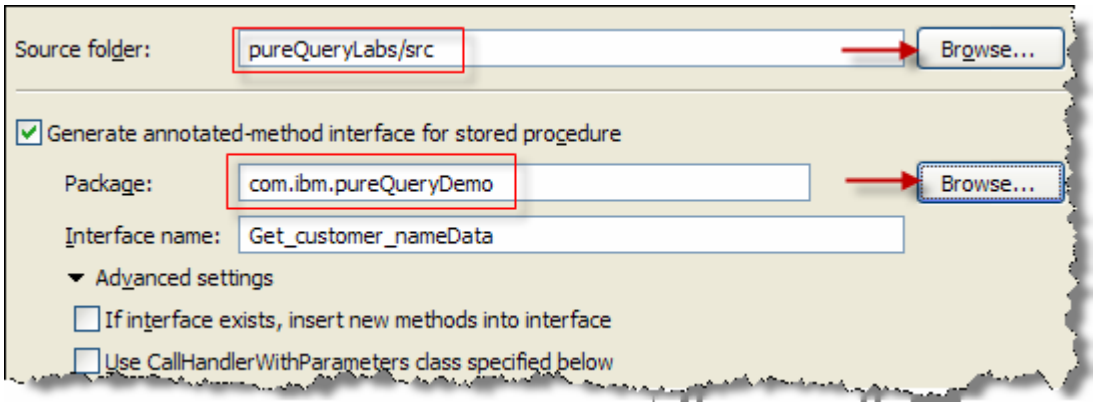
3.5 Generate pureQuery code for a SQL Procedure

__26. Expand the `Stored Procedure` folder under the `GOSALESCT` Schema (This is different schema than `GOSALES`) in the *Data Source Explorer*:

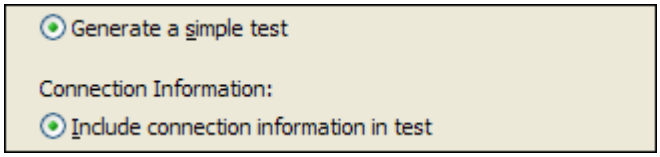


- Right-Click on the stored procedure *`GET_CUSTOMER_NAME`* and select *`Generate pureQuery Code…`*

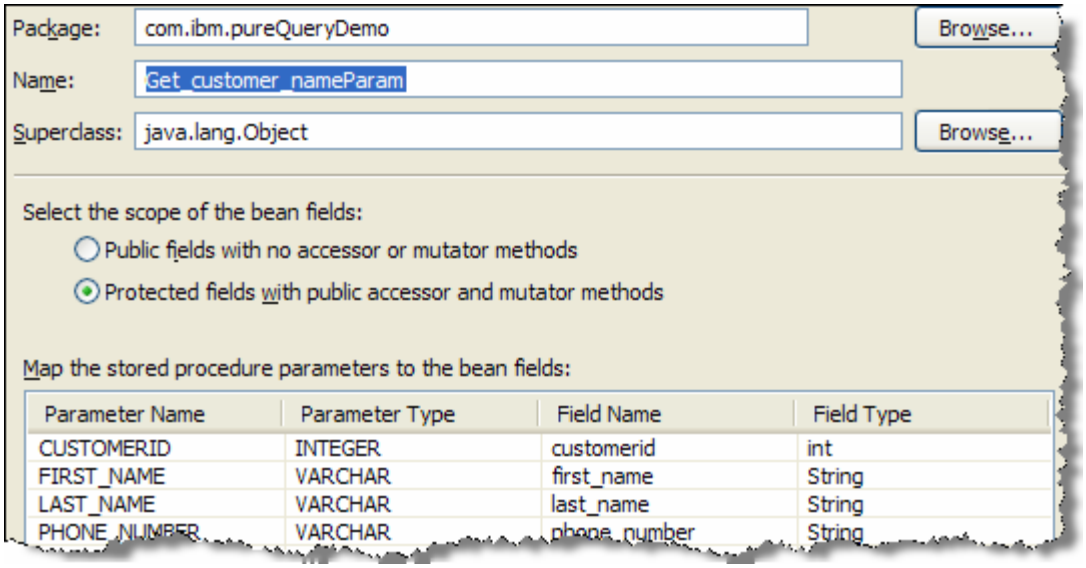
- Fill the pop-up window as below.



- 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.

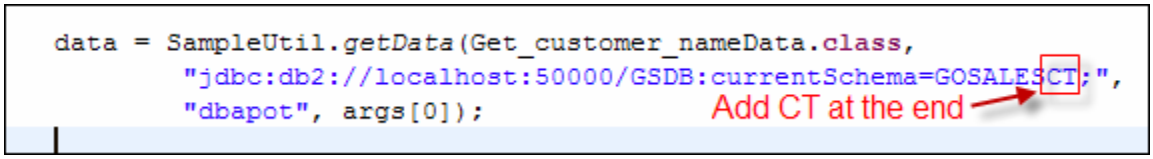


- In this screen, we are given a chance to discover result sets if stored procedure returns some result. Since our selected stored procedure does not return any result set, Click `<Finish>`.
- You will notice that 4 java classes have been created for this stored procedure.

- `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

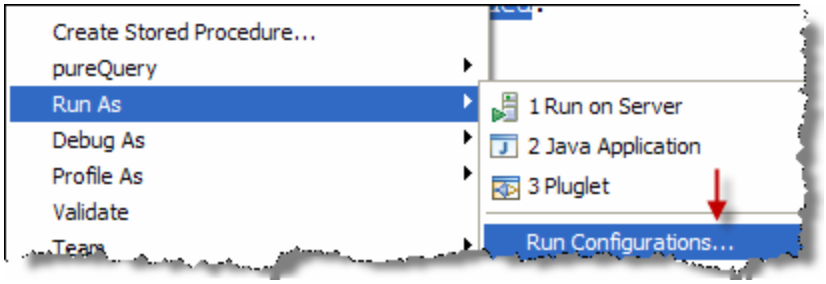
3.5.1 Calling a stored procedure

__27. Double click `get_customer_nameDataTest.java` in package explorer and this will open the Java test program in the editor view. We will need to do a simple change to modify schema name from `GOSALES` to `GOSALESCT`. Complete change as shown below.

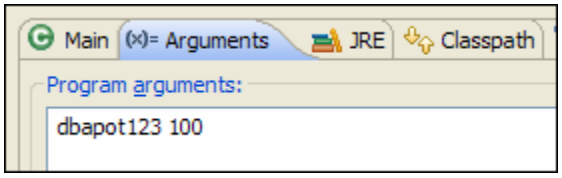


__28. Right click anywhere in Java source file and choose `Run` ⇨ `Java Application`. You will see console output stating that `All required arguments were not provided`. But doing so, you have created an instance of this application that can now be modified to specify input parameters.

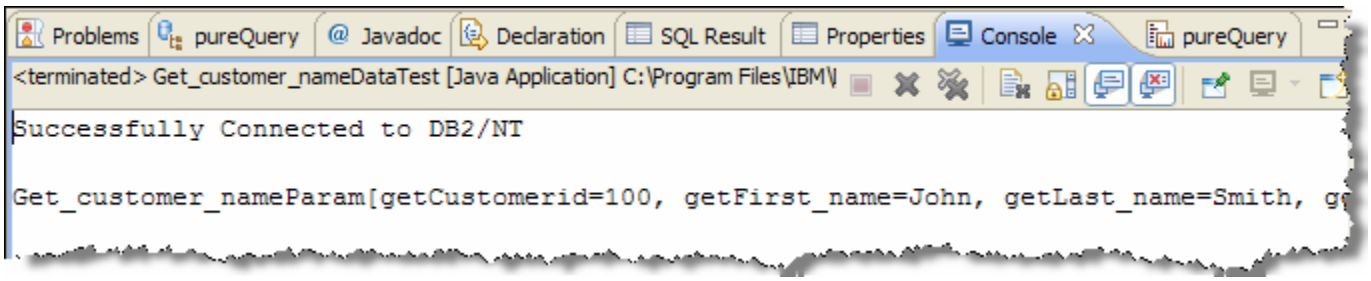
__29. Right click on same java source again and choose `Run As` ⇨ `Run Configurations..` which will open up `Run Configurations` window.



__30. Go to the *Arguments* tab and specify `dbapot123` 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>.



__31. You will see the results in the Console.

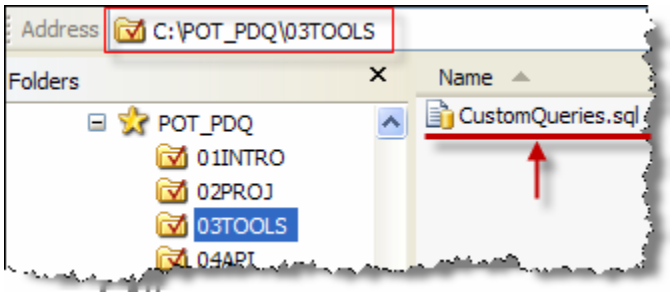


__32. Please review the Java source file `Get_customer_nameData.java` to see method `callGet_CUSTOMER_NAME` which was annotated with a CALL statement to the stored procedure. The implementation of this method was auto-generated and is shown in `Get_customer_nameDataImpl.java`.

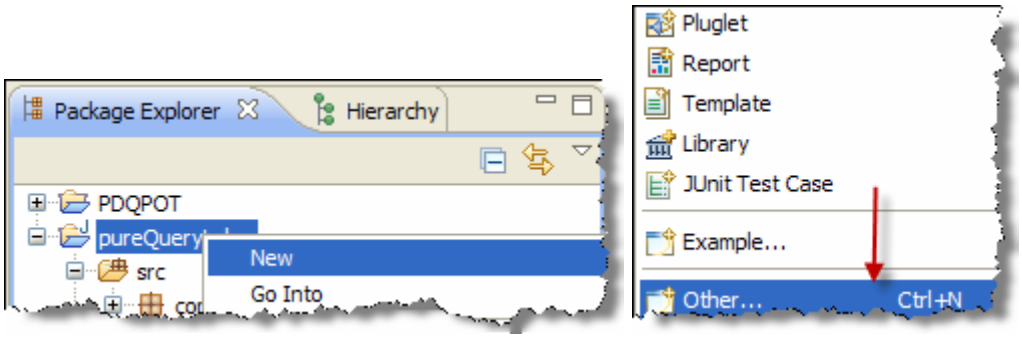
```
public interface Get_customer_nameData {

    // Call GOSALESCT.GET_CUSTOMER_NAME
    @Call(sql = "Call GOSALESCT.GET_CUSTOMER_NAME( :customerid,
    StoredProcedureResult callGET_CUSTOMER_NAME(Get_customer_name
```

3.6 Generate pureQuery code from SQL Scripts

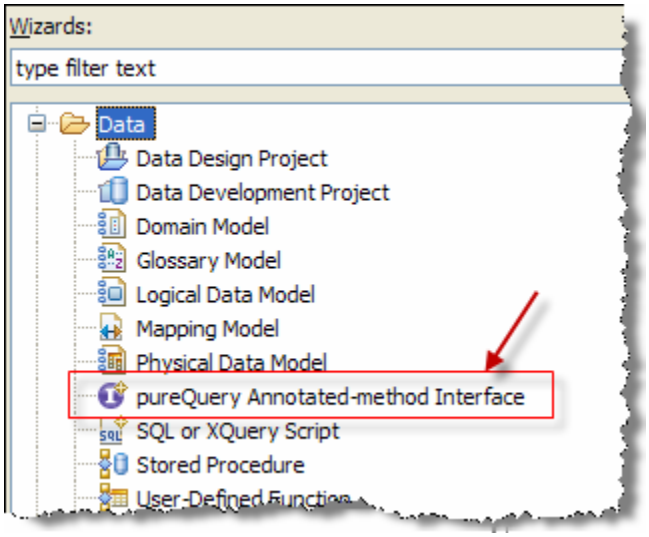
__33. You can generate pureQuery code from SQL defined in a file. Navigate to `C:\POT_PDQ\03TOOLS` folder using your *Windows Explorer* and double click on `CustomQueries.sql` file to view the SQL statements.



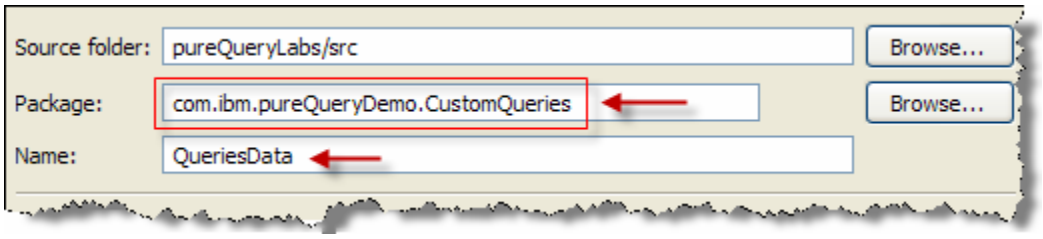
__34. Right click on `pureQueryLabs` project in the package explorer and click on `New` ⇨ `Other` …



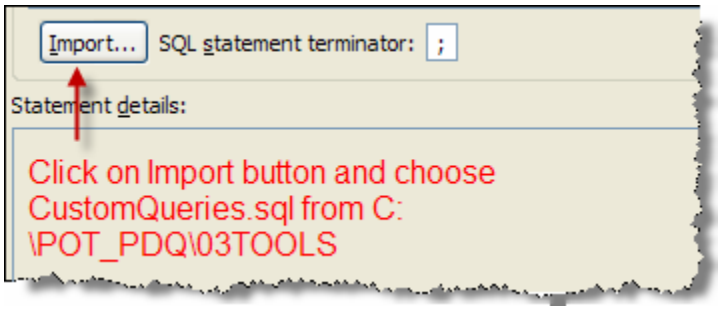
__35. Expand the Data section and select the `pureQuery Annotated-method Interface` from next window click on `<Next>`.



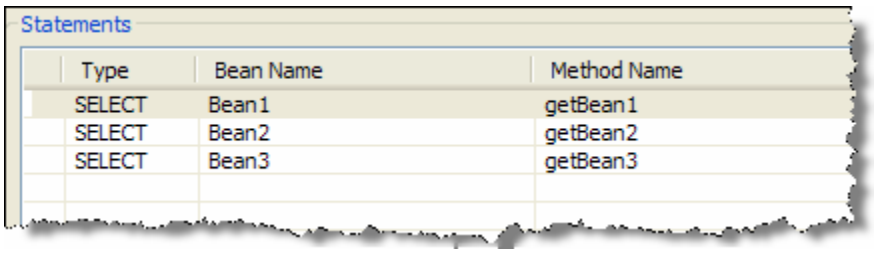
__36. Select `pureQuery Annotated-method Interface` from next window click on `<Next>`. Type in name of the package as `com.ibm.pureQueryDemo.CustomQueries` and name as `QueriesData` and click on `<Next>`.



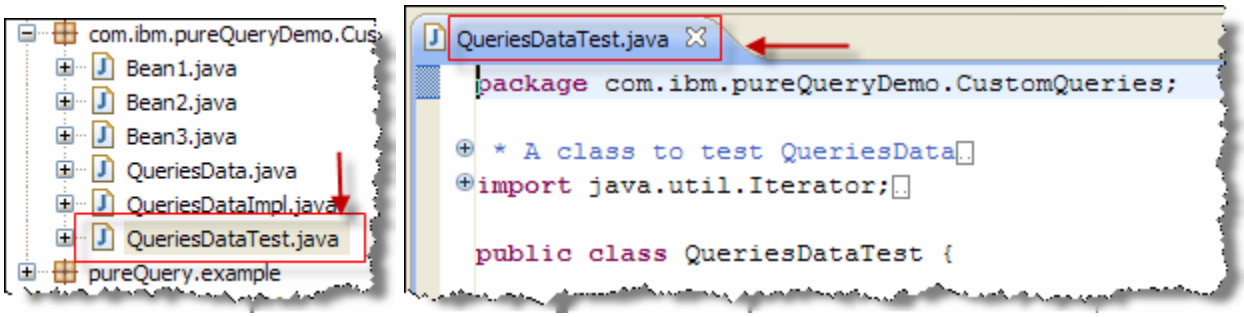
__37. In SQL statements window, click on Import button and select file `CustomQueries.sql` from folder `C:\POT_PDQ\03TOOLS`.



__38. You will see 3 `SELECT` statements imported in this window with default bean names as Bean1, Bean2 and Bean3. Click on `<Next>` and then on `<Finish>` button to generate pureQuery code for these 3 SQL statements.

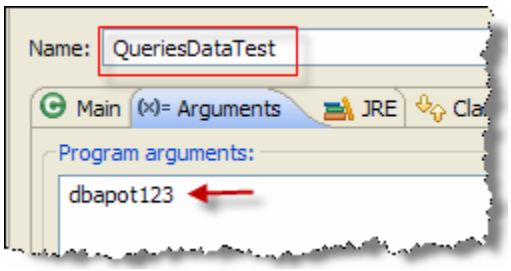


__39. After generating pureQuery code, double click on `QueriesDataTest.java` in package explorer.

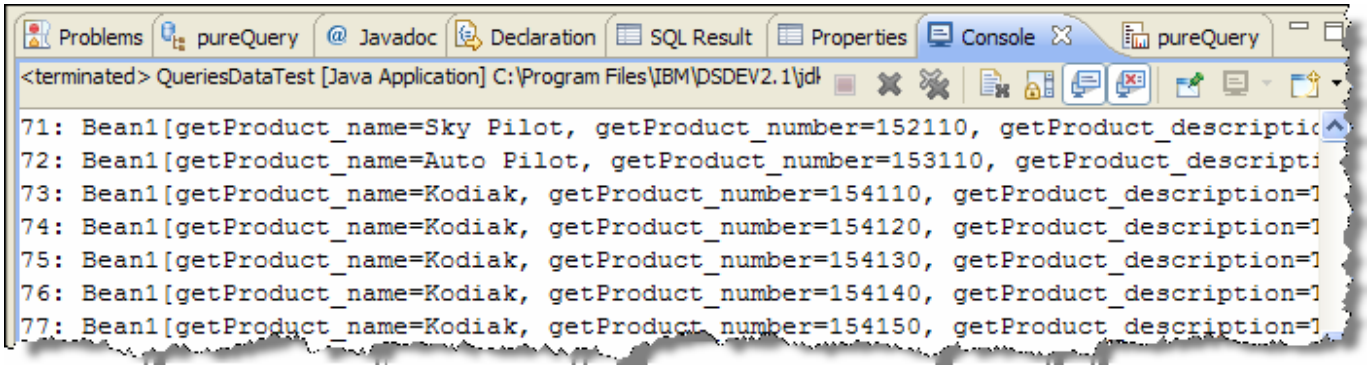


__40. Right click anywhere in the QueriesDataTest.java and choose `Run ⇨ Java Application`. You will see console output stating that `All required arguments were not provided`.

__41. Right click on same java source again and choose `Run As ⇨ Run Configurations..` which will open up `Run Configurations` window. Type-in `dbapot123` in the Arguments tab.



__42. Click on `Run` and you should see output from the `GetBean1` method.



**** End of Lab3 – Explore pureQuery Tools**

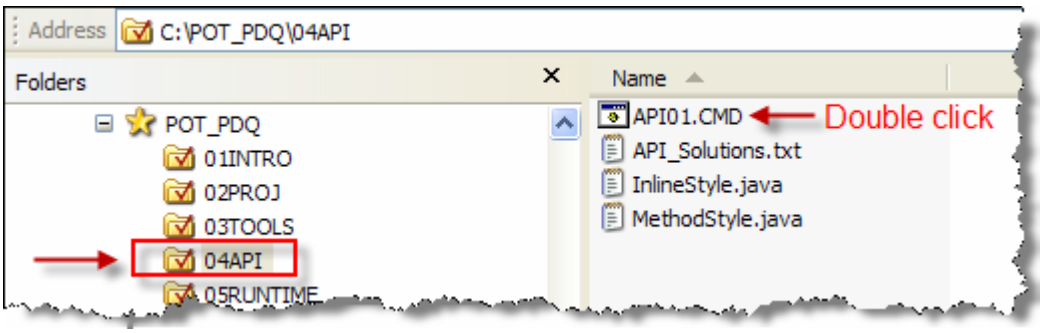
Lab 4 Explore pureQuery API

Prerequisites:

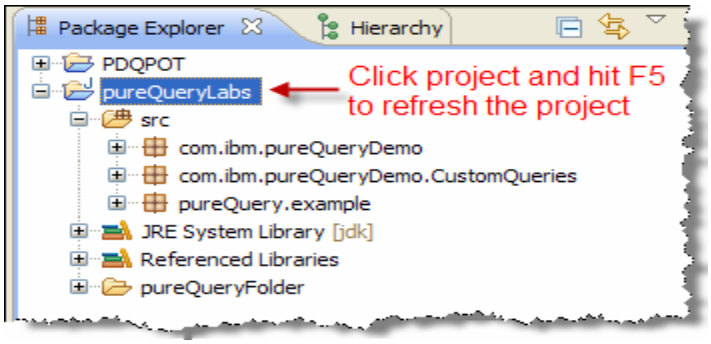
We need to copy the java source files to the workspace.

__1. Go to Windows Explorer and navigate to `C:\POT_PDQ\04API` directory.

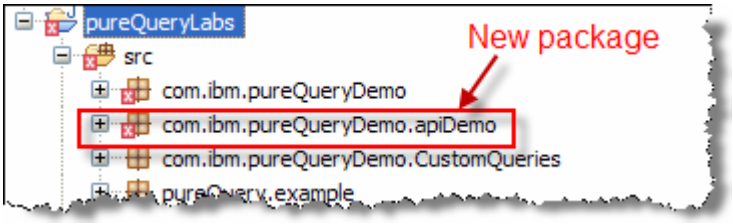
__2. Double click on script `API01.CMD`. This script copies java programs to the `pureQueryLabs` project and we will use `MethodStyle.java` and `InlineStyle.java` to explore Method and Inline style APIs in this lab.



__3. Refresh the Java project so that the files copied in the previous step are reflected in *Package Explorer*.

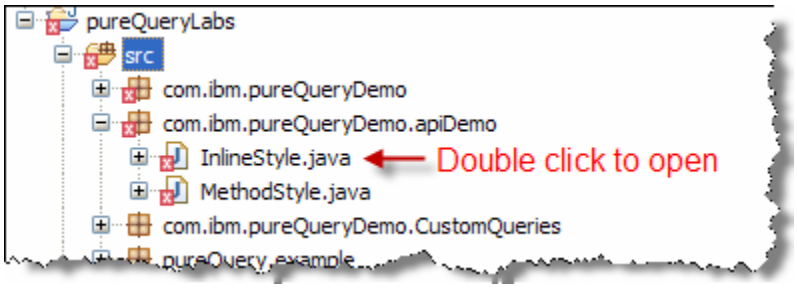


__4. After you hit F5, you should see `apiDemo` package showing up in the *Package Explorer*. We created this package by double clicking on `API0.CMD` in the previous step. You also notice a small cross icon on both the packages indicating errors. Do not worry about these errors and fixing them is part of this lab exercise.

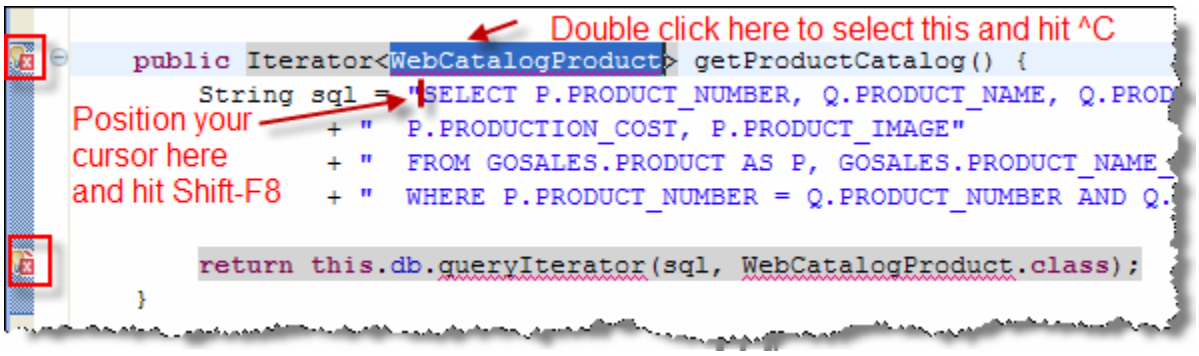


4.1 Practice Code Generation

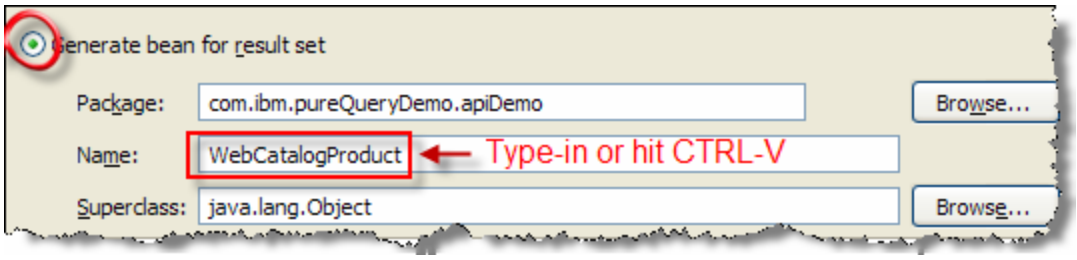
__5. Expand `apiDemo` package and open `InlineStyle.java` by double clicking on it. We will fix some errors by creating beans from SQL statements.

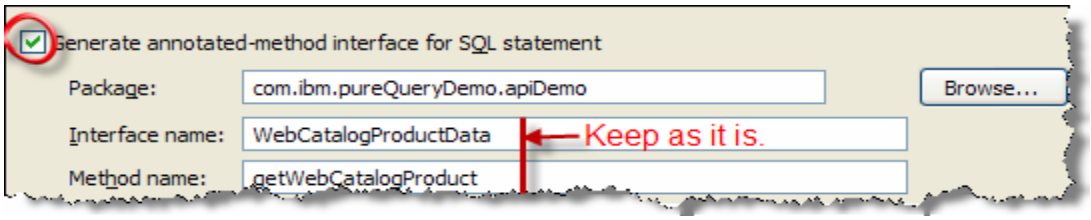


__6. Go to the 1st SQL statement or the first error marked in the file. We are referencing a missing `WebCatalogProduct` bean here. This bean maps to the SQL statement and we will generate it from the SQL. Double click on `WebCatalogProduct` to select it and hit `CTRL-C` to copy this in clipboard. Position your cursor at the beginning of the `SELECT` statement in the next line and hit `Shift-F8` to open pureQuery code generation dialog.



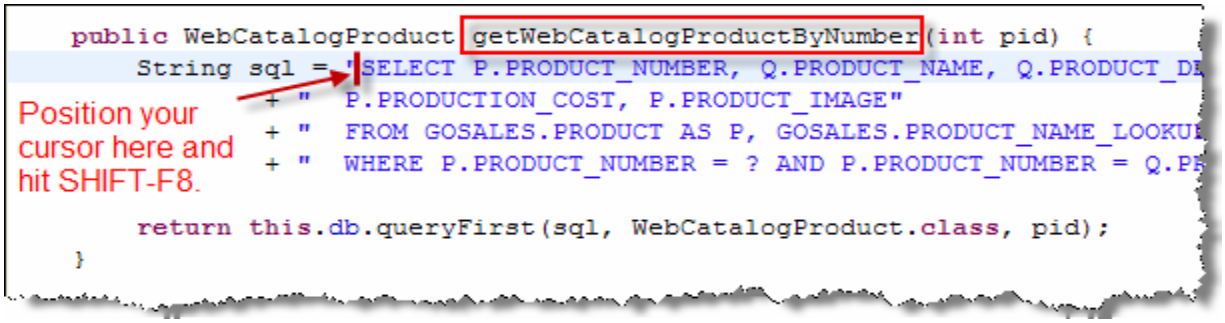
__7. Type-in `WebCatalogProduct` bean name as shown and make sure that `Generate bean for result set` is selected and `Generate annotated-method interface for SQL statement` is also checked. Click <`Finish`> to generate bean for the SQL statement.





__8. The above action will generate `WebCatalogProduct.java` bean and will open it up for you. Review and close this and go back to the `InlineStyle.java` program.

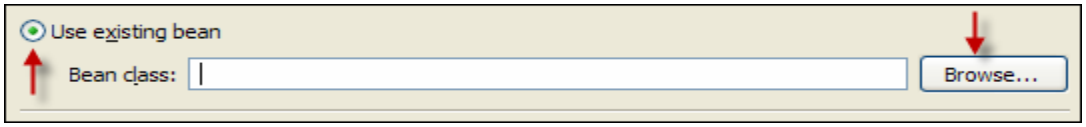
__9. Go to the 2nd SQL statement and position your cursor at the start of the `SELECT` statement and hit `SHIFT-F8`.



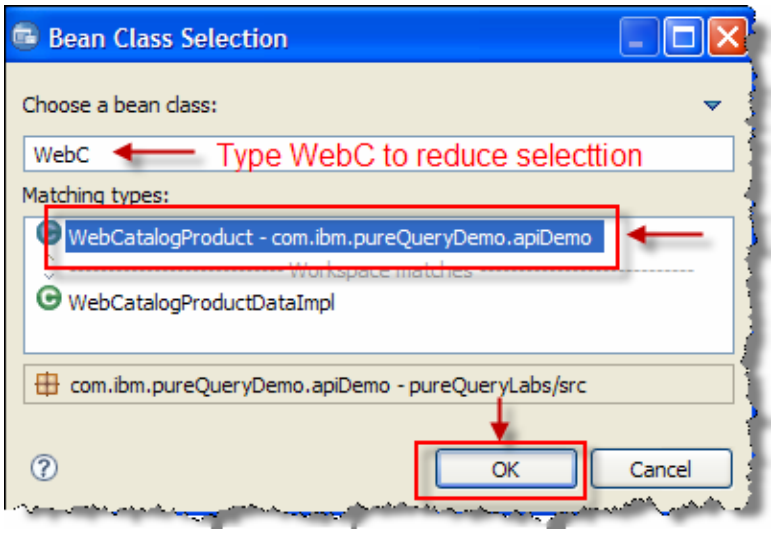
__10. In the `pureQuery Code Generation` screen, our choices will be different than the previous step.

- 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.

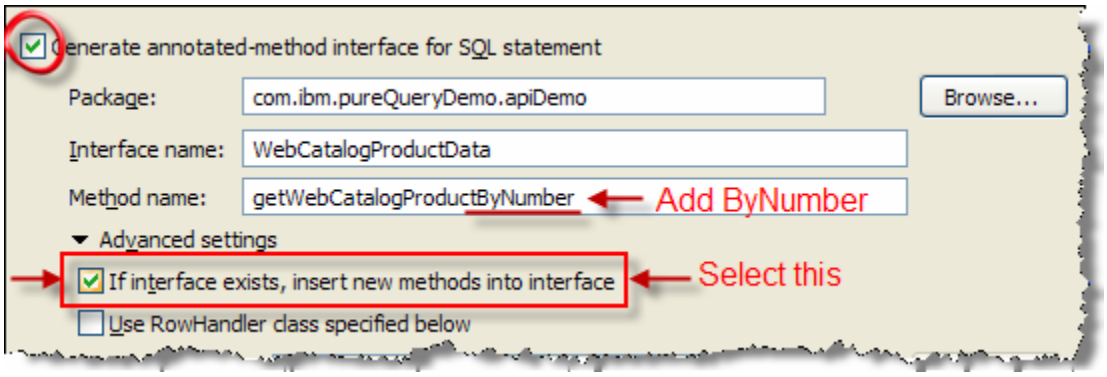
__11. Click on `Use existing bean` and click on `Browse` button to select the bean class.



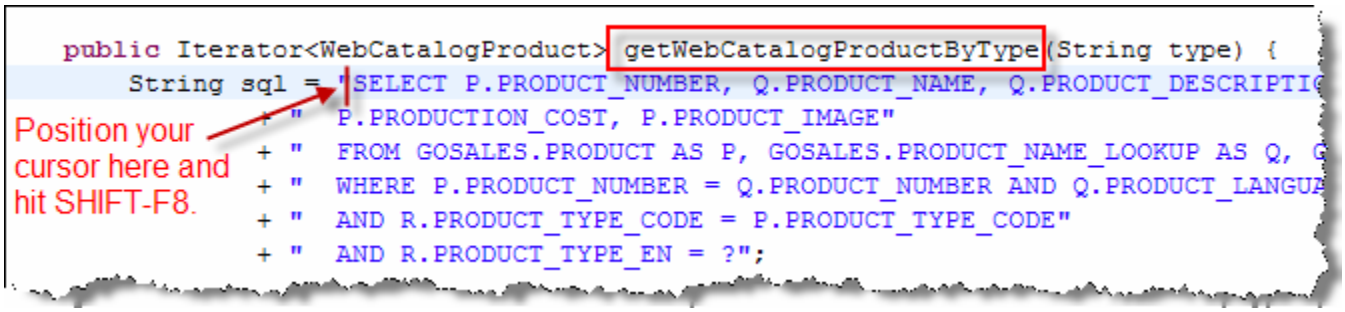
__12. Type-in `WebC` to reduce the number of beans and select `WebCatalogProduct` bean and hit `OK`.



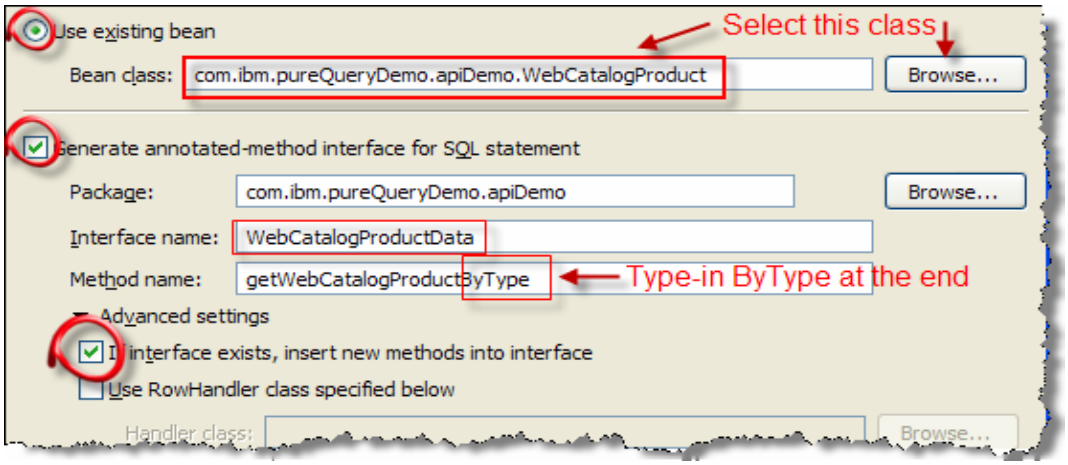
__13. When you come back to the same screen from the previous step, you should give the method name as `getWebCatalogProductByNumber`. You need to just add `ByNumber` at the end of already given name of `getWebCatalogProduct`. We already created the interface in the previous step and we are using the same one, so it is also necessary that you check `If interface exists, insert new methods into the interface`. 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.



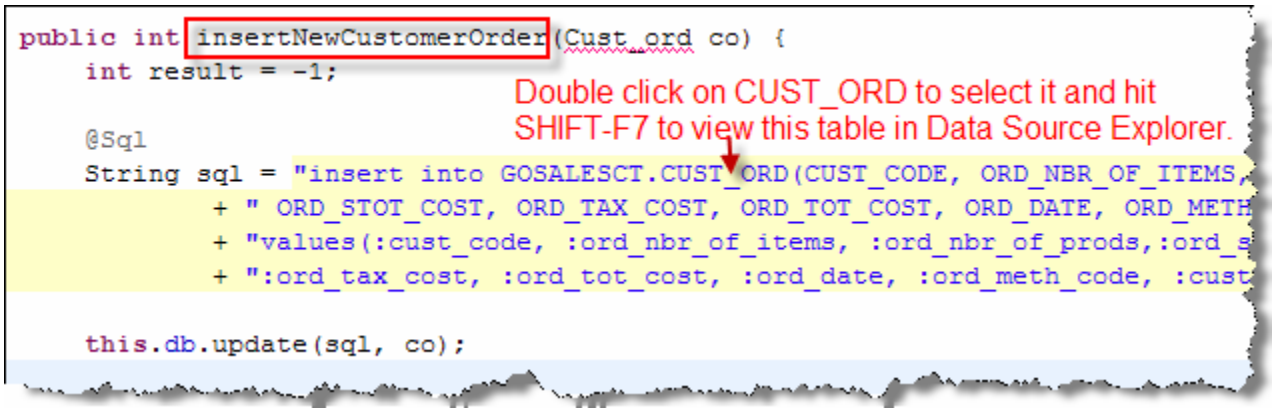
__14. Go to the 3rd SQL statement in `InlineStyle.java` and position your cursor at the start of the `SELECT` statement and press `SHIFT-F8` to open pureQuery Code Generation screen.



__15. 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.

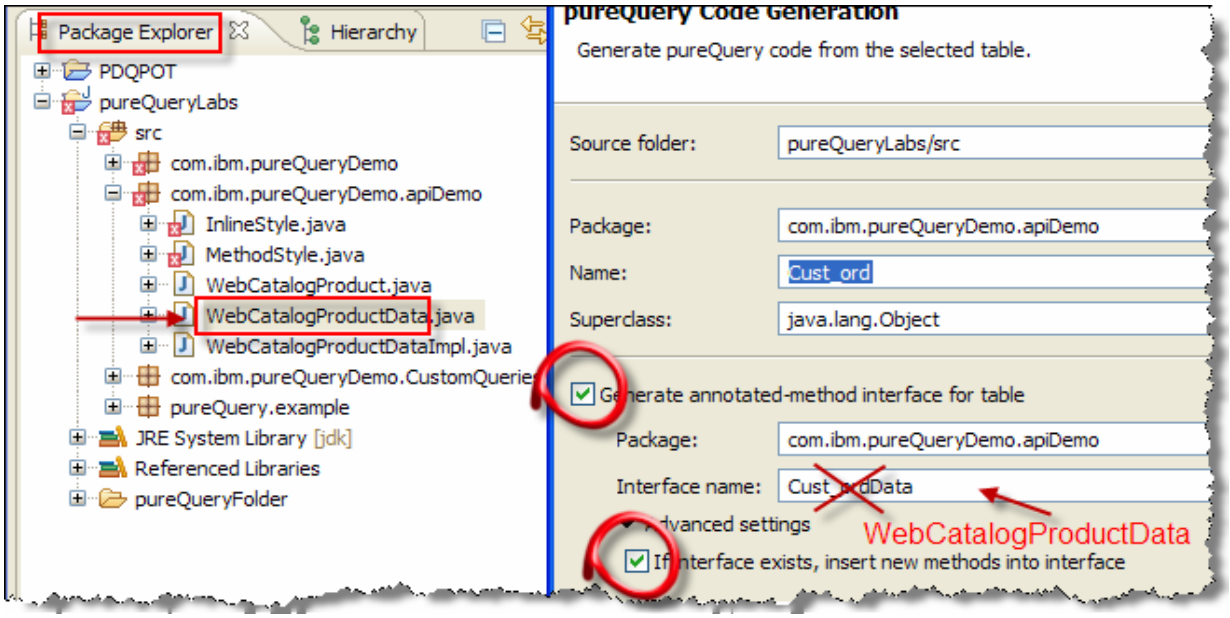


__16. Go to the 4th SQL statement in `InlineStyle.java` and position your cursor at the start of the `INSERT` statement and press `SHIFT-F7` to view this table in *Data Source Explorer*.

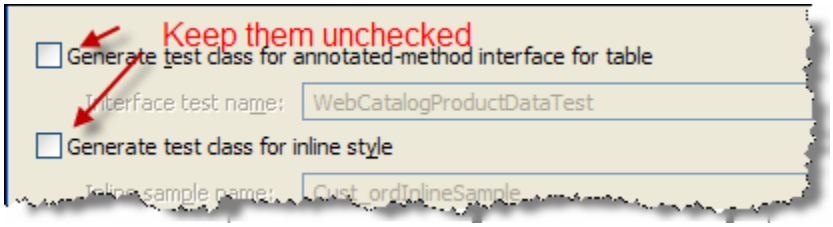


__17. Go to the `Data Source Explorer` and right click on table `CUST_ORD` and click on `Generate pureQuery Code`.

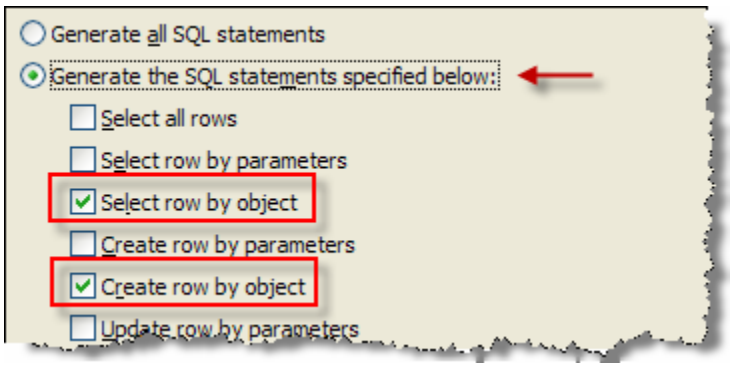
__18. In `pureQuery Code Generation` screen, check `Generate annotated-method interface for table` and `If Interface exists, insert new methods into interface` check boxes. Replace `Interface name` from the default value of `Cust_OrdData` to `WebCatalogProductData`. Click `<Next>` to go to the next screen.



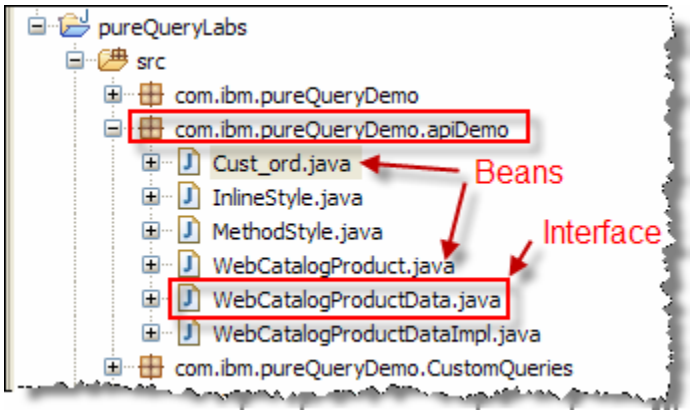
__19. In the next screen, keep both the check boxes unchecked for creating test classes. Click <Next> two times to go to the last screen of SQL Statements.



__20. In SQL Statements screen, check option for Generate the SQL statements specified below and check 2 options for Select row by object and Create row by object and click <Finish> to append methods to fetch and create the customer order in an existing WebCatalogProductData data interface.



__21. After completing above steps to create two missing beans and five annotation methods, we should see error free InlineStyle.java and MethodStyle.java with additional file 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.

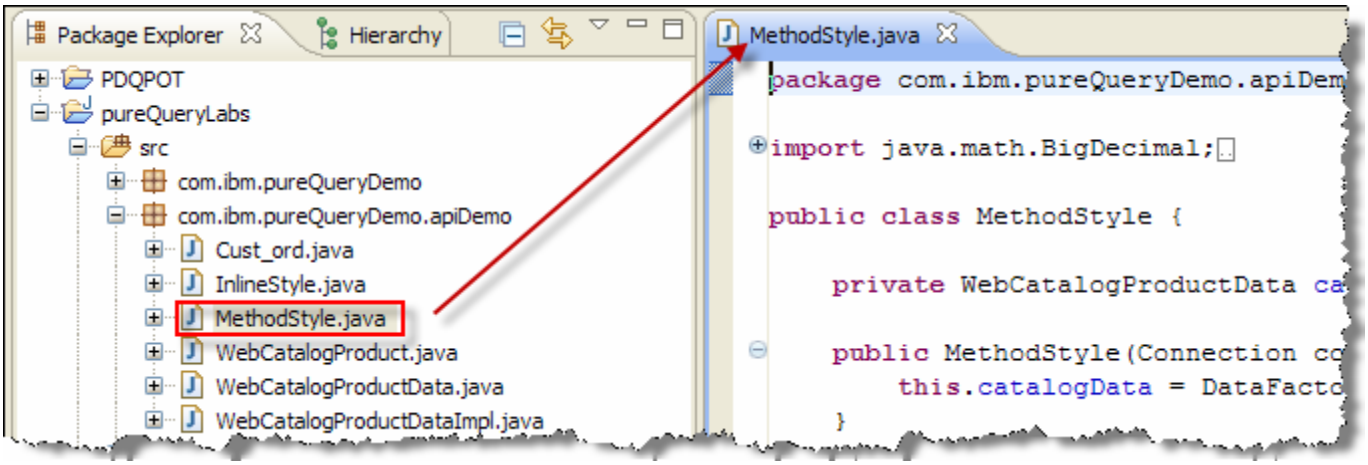
__22. Close all open files in the editor and open `InlineStyle.java`, `MethodStyle.java` and `WebCatalogProductData.java` and review them. The `InlineStyle.java` contains inline SQL statements for which we created annotation methods in previous steps. Both the java programs provide same output but by using inline and annotation APIs.

4.2 Using Method-style Program

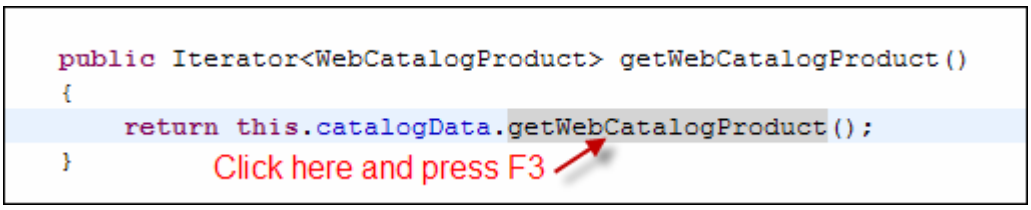
Introduction:

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.

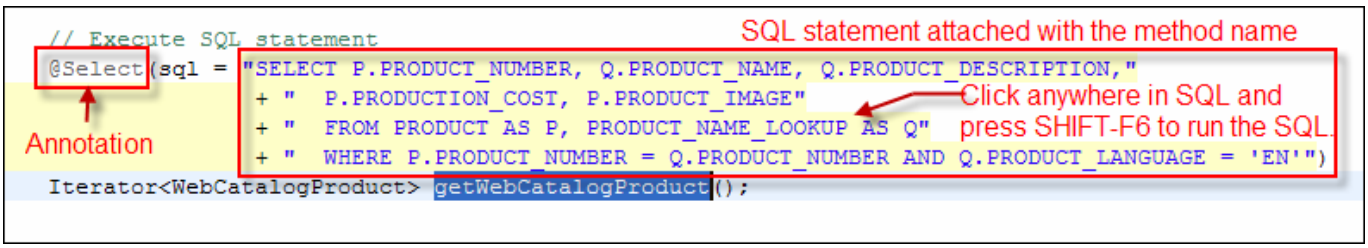
__23. Open the `MethodStyle.java` class by double-clicking the file and review the methods.



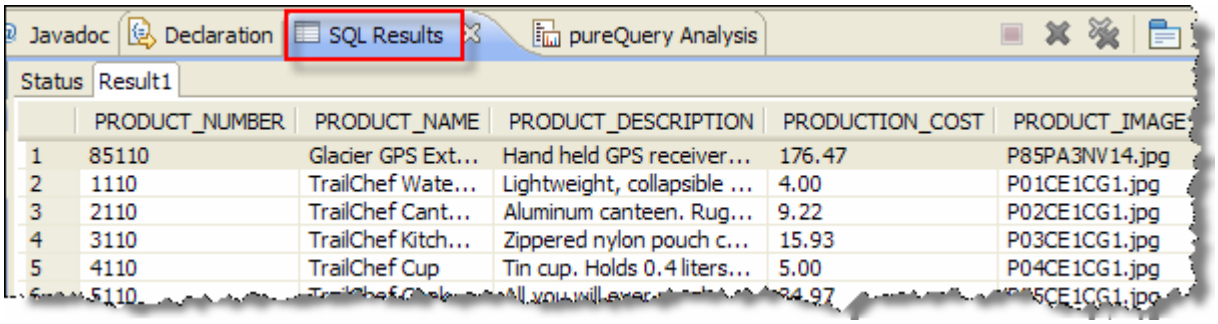
__24. Review method `getWebCatalogProduct`. Click on the method name inside the body of the method and hit F3 which will take you to the definition of the method in the Interface class.



__25. Review the method in the interface class. The method name `getWebCatalogProduct` is annotated with the SQL statements.



__26. 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.

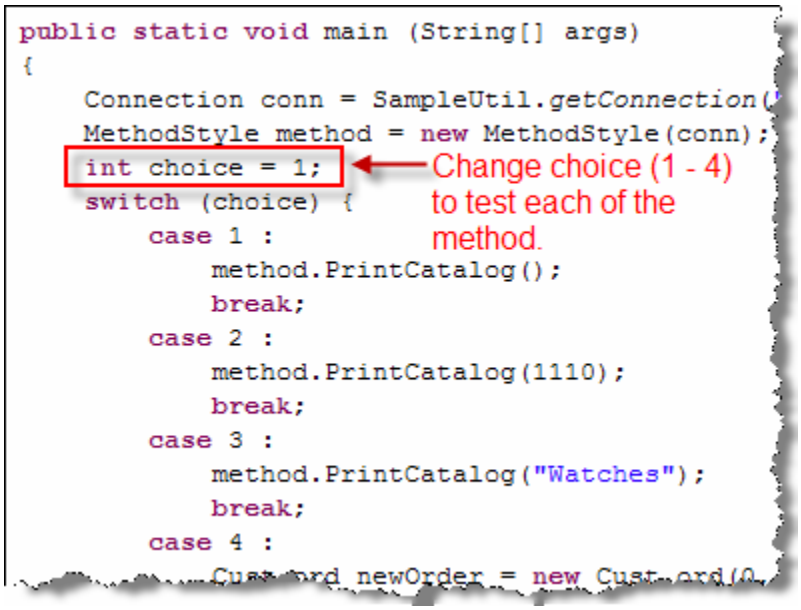


	PRODUCT_NUMBER	PRODUCT_NAME	PRODUCT_DESCRIPTION	PRODUCTION_COST	PRODUCT_IMAGE
1	85110	Glacier GPS Ext...	Hand held GPS receiver...	176.47	P85PA3NV14.jpg
2	1110	TrailChef Wate...	Lightweight, collapsible ...	4.00	P01CE1CG1.jpg
3	2110	TrailChef Cant...	Aluminum canteen. Rug...	9.22	P02CE1CG1.jpg
4	3110	TrailChef Kitch...	Zippered nylon pouch c...	15.93	P03CE1CG1.jpg
5	4110	TrailChef Cup	Tin cup. Holds 0.4 liters...	5.00	P04CE1CG1.jpg

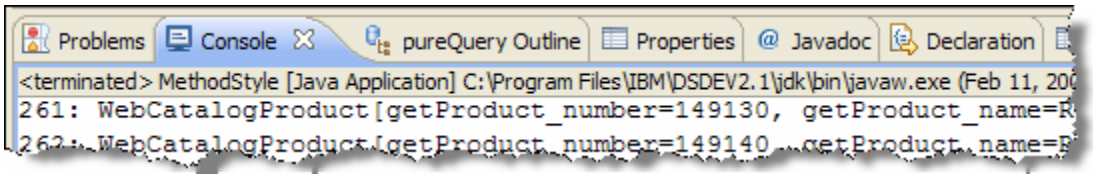
__27. The implementation of method `getWebCatalogProduct` is in `getWebCatalogProductDataImpl.java`. This implementation file gets generated whenever any change is made to the interface file by adding or removing the methods.

__28. Open `getWebCatalogProductDataImpl.java` and review the generated code.

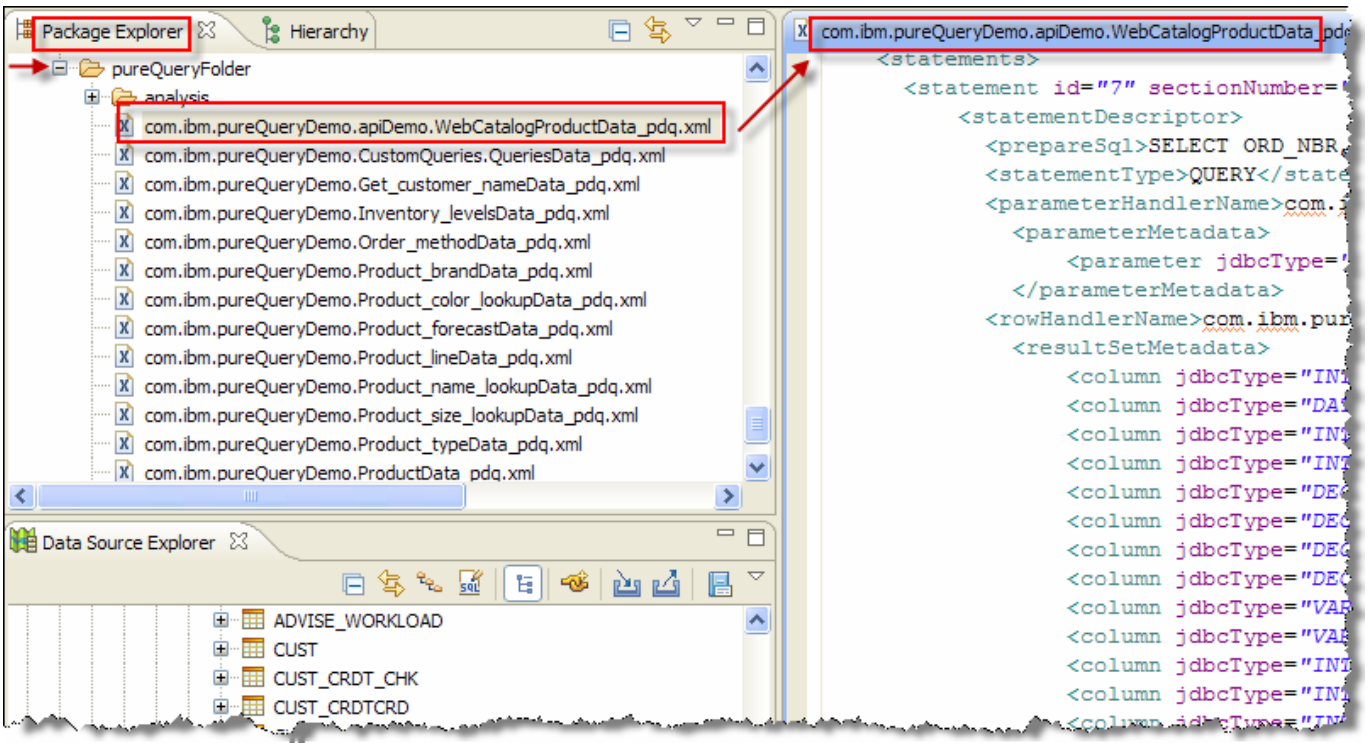
__29. Go back to *MethodStyle.java* and review the `main` method. We will test each of the method through the main routine.



__30. Right click anywhere in the program and click on `Run As ⇨ Java Application.` You will see the console output as shown:



__31. 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.

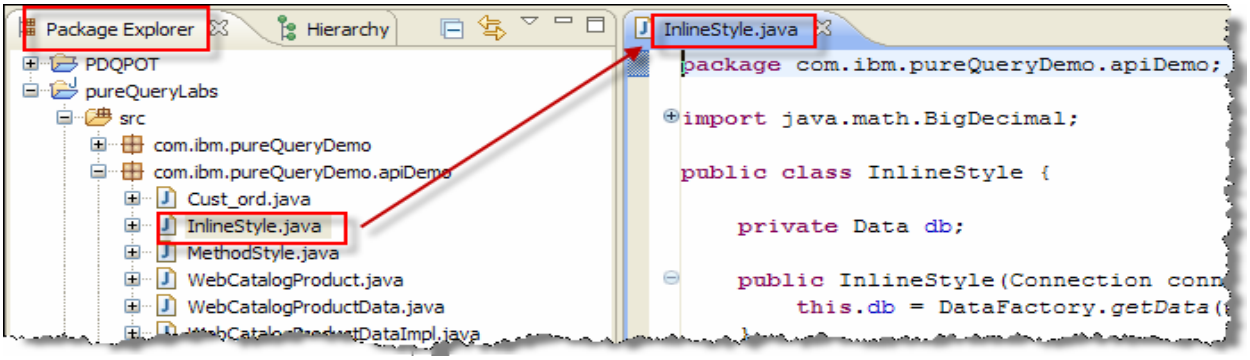


4.3 Using an Inline-style Program

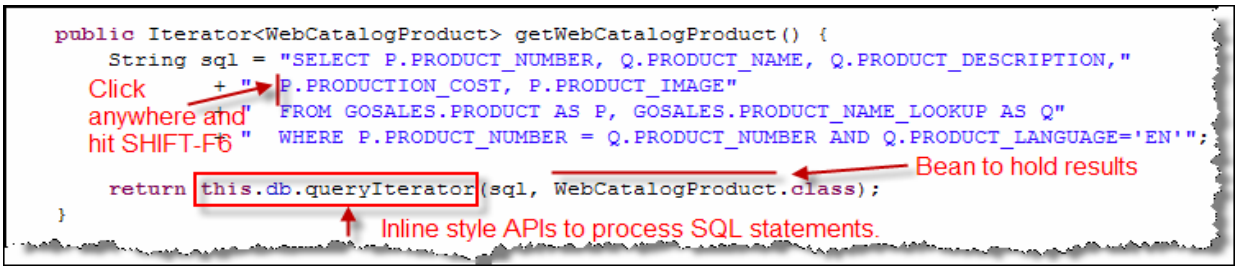
Introduction:

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.

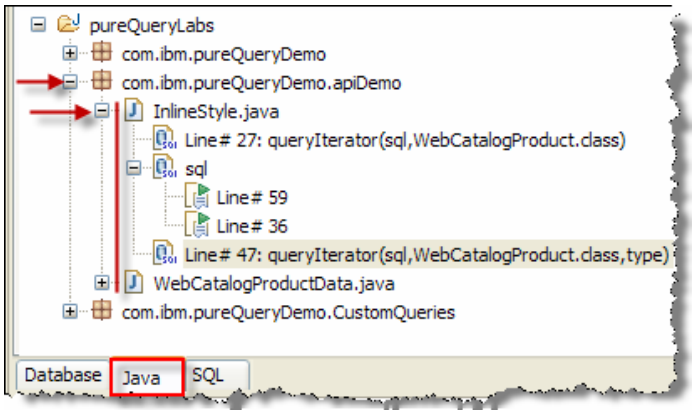
__32. Open the `InlineStyle.java` class by double-clicking the file and review the methods.



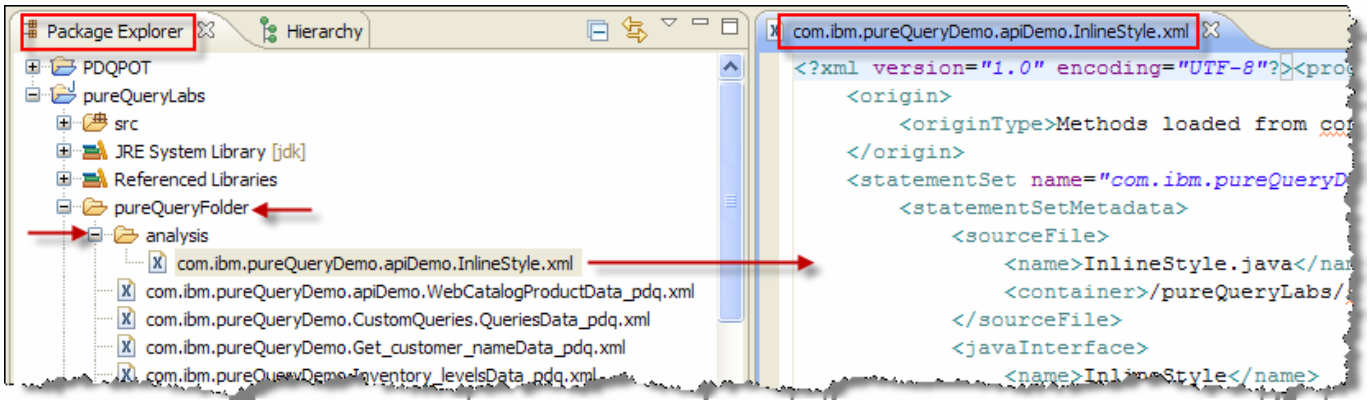
__33. Review all 4 SQL statements and associated in-line style pureQuery APIs to process the SQL statement.



__34. 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.

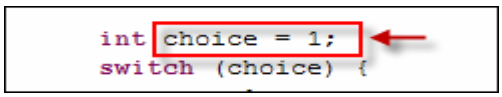


__35. Expand `pureQuery Folder` in `pureQueryLabs` project in the *Package Explorer*. Expand `InLineStyle.xml` under the analysis folder. The methods used in `InLineStyle.java` are saved in this XML file. We will come back to this later.



__36. 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.

__37. Change the value of choice parameter from 1 to 4 and run the program each time.

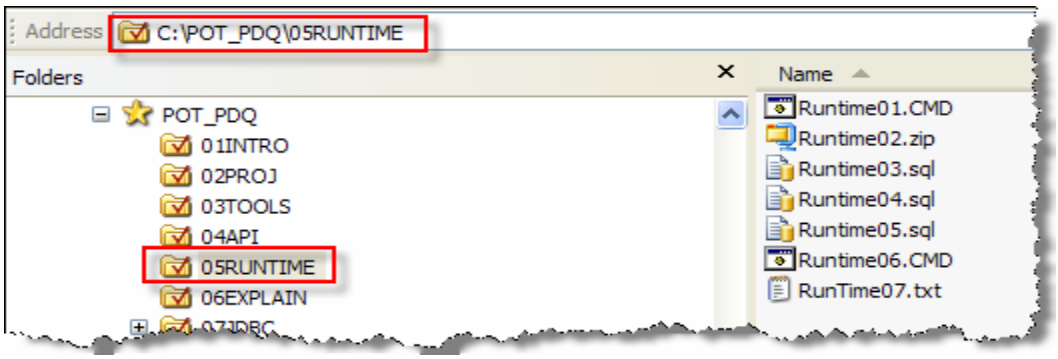


**** End of lab 4: Explore pureQuery API**

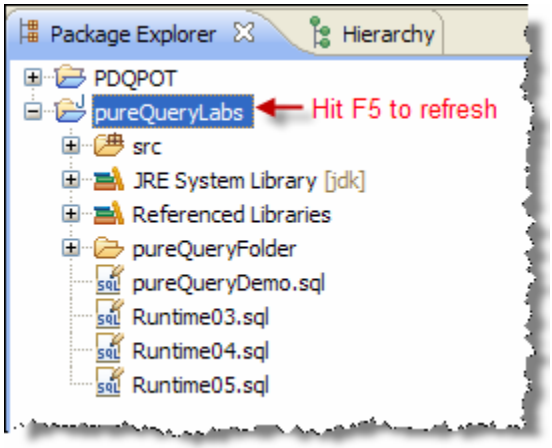
Lab 5 Explore pureQuery Runtime

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. Go to the *Windows Explorer* and locate `C:\POT_PDQ\05RUNTIME` and double click on `Runtime01.CMD` file. (This will refresh your project as if you completed labs 03 and 04)

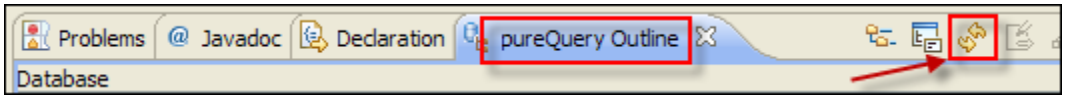


__2. Click on `pureQueryLabs` project in the Package explorer and hit `F5` to refresh the project.



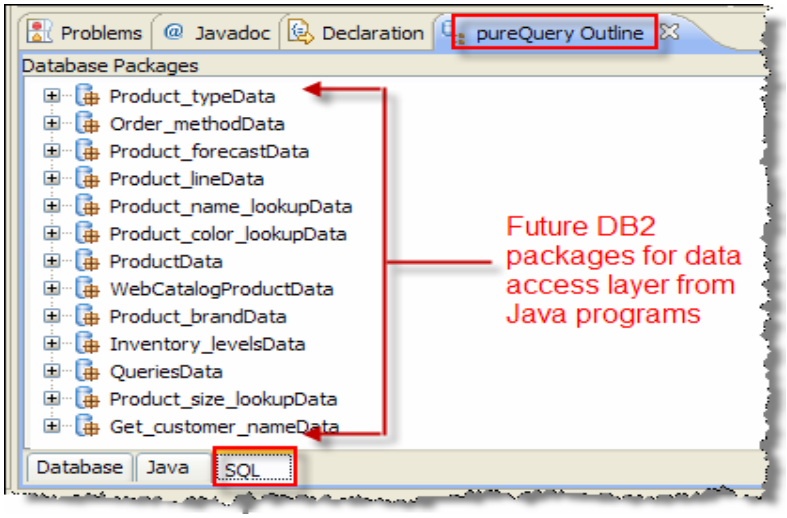
5.1 Explore pureQuery Outline View

__3. Go to the *pureQuery Outline* view and click on refresh icon to rebuild it.

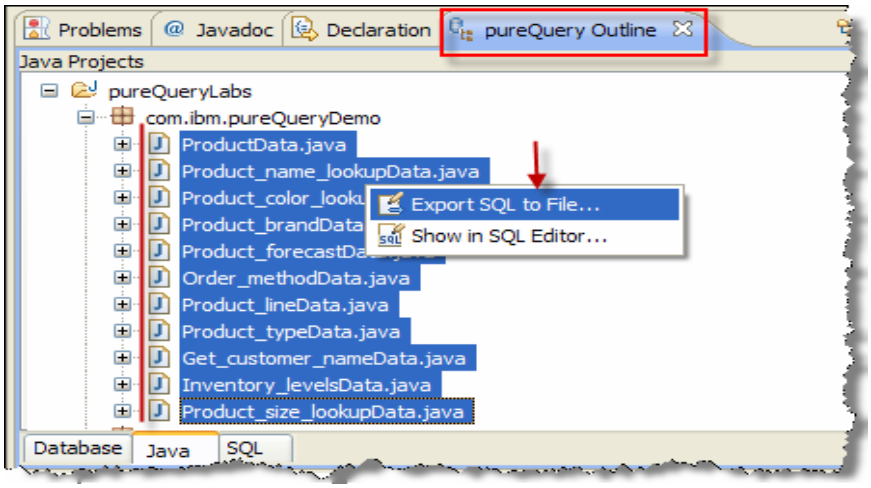


Note: If you do not see *pureQuery Outline* view, right click on *pureQueryLabs* project in *Package Explorer* and click on `pureQuery` ⇨ `Show pureQuery Outline`.

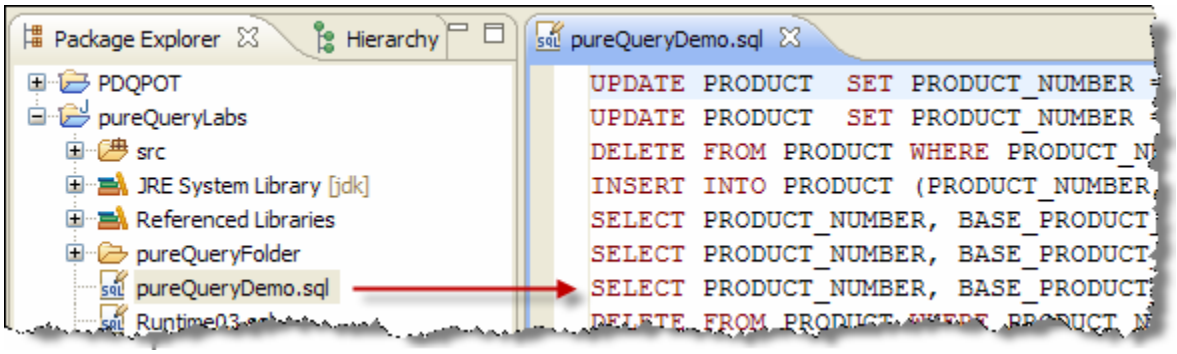
__4. In *pureQuery Outline* view, click on the SQL tab at the bottom. You will see a list of the future DB2 packages that are ready for bind or deploy.



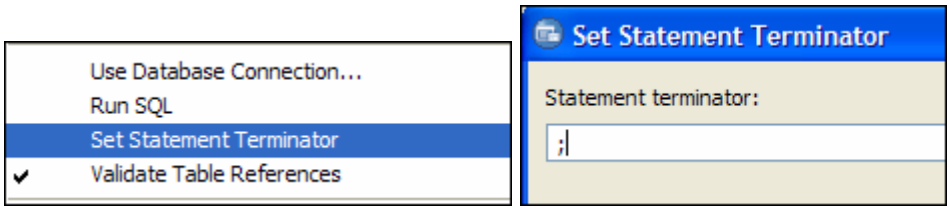
__5. Click on `Java` tab at the bottom of the *pureQuery Outline* view and select all java data access classes. Right click and export SQLs to a file to view them.



__6. Specify file name as `pureQueryDemo.sql` and extract all SQL statements from data access classes and view them in an editor.



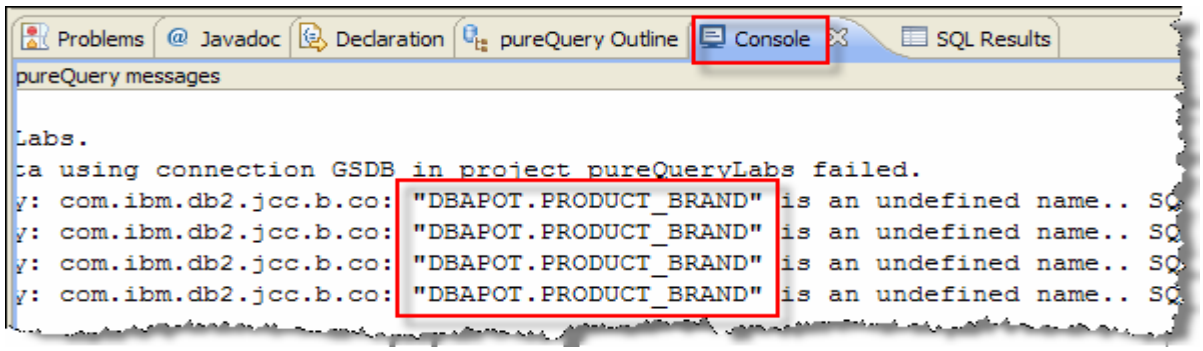
__7. Right click anywhere in the SQL file and choose option `Set Statement Terminator` and specify semicolon.



5.2 Bind packages for a pureQuery project

__8. From the *Package Explorer* view right-click on the `pureQueryLabs` Java project and select `pureQuery` ⇨ `Bind pureQuery Application.` Select GSDB database when prompted.

__9. Look at the output in the `Console` view and you will notice that most of the package creation failed for SQL error -204 indicating undefined name. This is the most common error since our connection user id is not the owner of the tables.



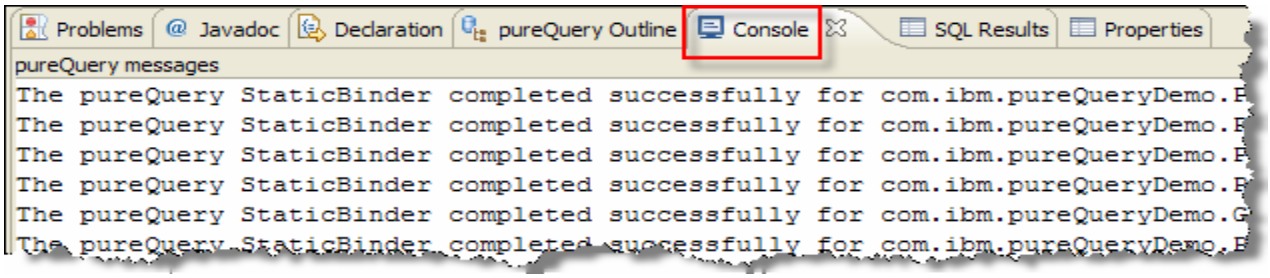
__10. Go to the *Package Explorer* view and navigate to the `pureQueryFolder` of the `pureQueryLabs` project. Double click `Default.bindProps` file to open it in an editor.

- Add `defaultOptions` and specify `QUALIFIER` option.

```
defaultOptions= -bindOptions "QUALIFIER GOSALES"
```

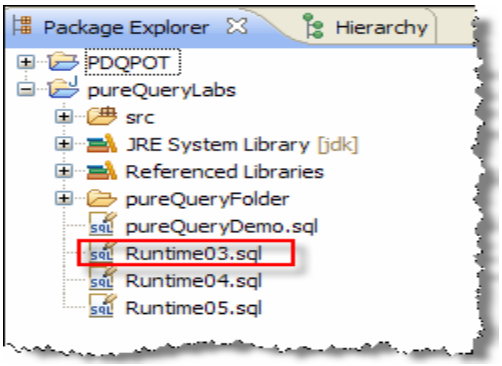
- Save the file (right click, <Save>)

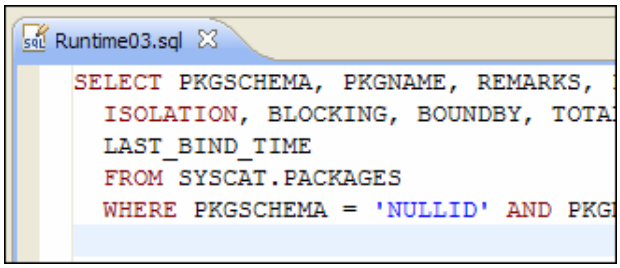
__11. Go back to the *Package Explorer* view and right-click on the `pureQueryLabs` Java project and select `pureQuery` ⇨ `Bind pureQuery Application.` Select `GSDB` database and wait for `BIND` process to finish. This time, the `BIND` should complete successfully.



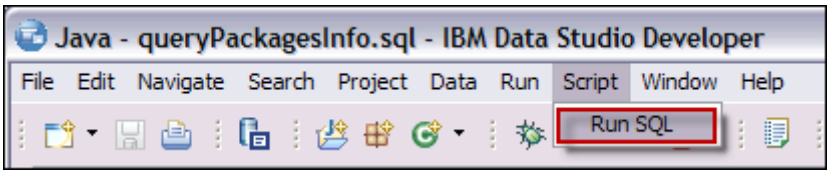
__12. View the info of the packages through SQL:

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

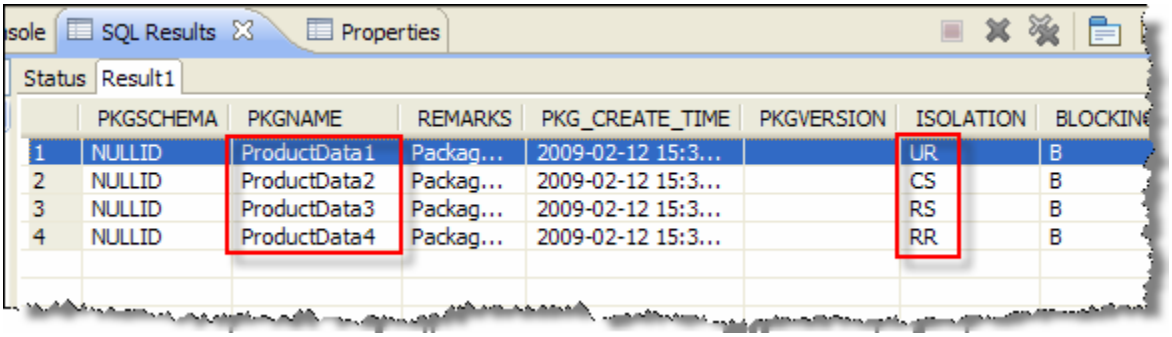




- Go to main menu and click on `Script` ⇨ `Run SQL`.



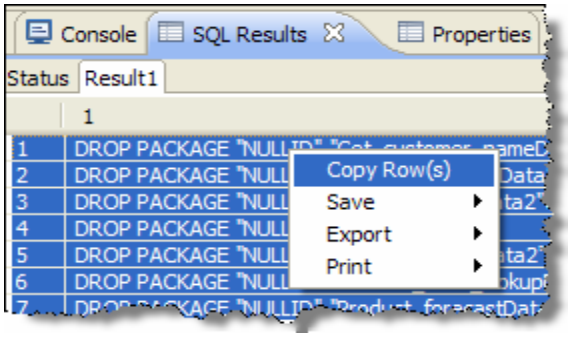
- You should now see the following results:



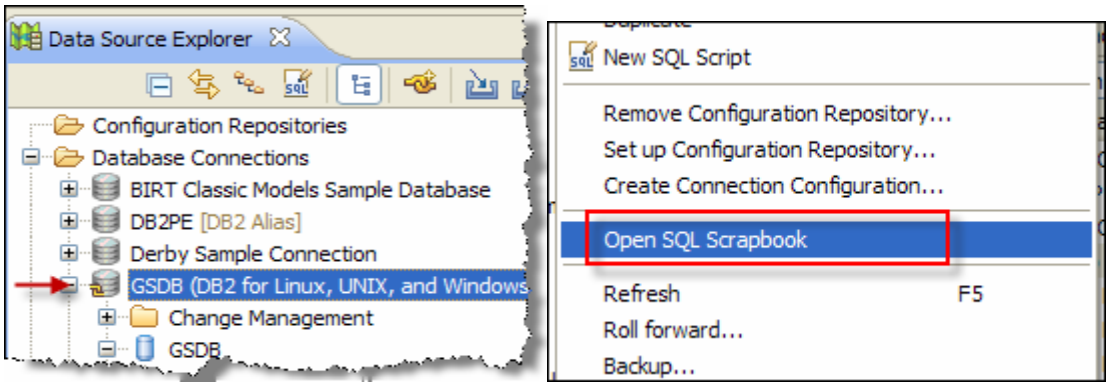
__13. Did you notice 4 packages created for the ProductData interface? This happened since we did not specify the `ISOLATION LEVEL` in `Default.bindProps` file. Go back to this file and add `ISOLATION LEVEL` and hit `CTRL-S` to save the file.

```
defaultOptions= -bindOptions "QUALIFIER GOSALES" -isolationLevel CS
```

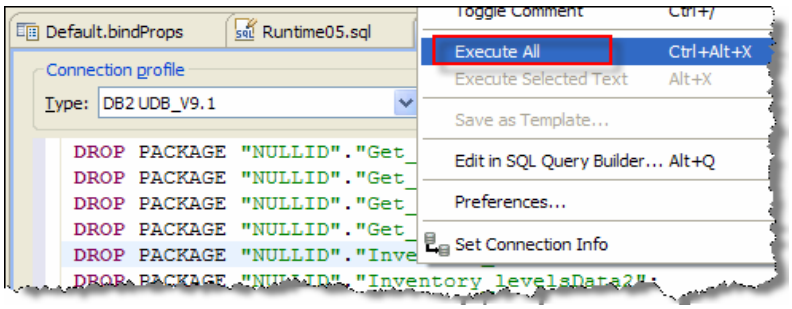
__14. We will need to drop these packages before we bind the application. Double click on `Runtime05.sql` to open it in an editor. Right click anywhere on the file and select `Run SQL` to generate the `DROP` statements. Go to the *SQL Results* view and hit `CTRL-A` to select all rows and right click to `Copy row(s).`



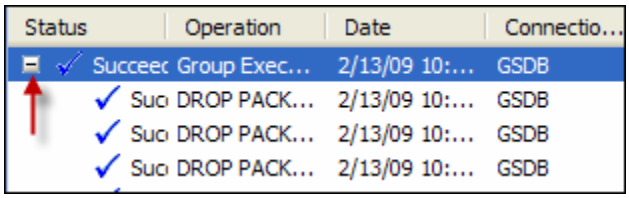
__15. Right click on `GSDB` database in *Data Source Explorer* and click on `Open SQL Scrapbook`.



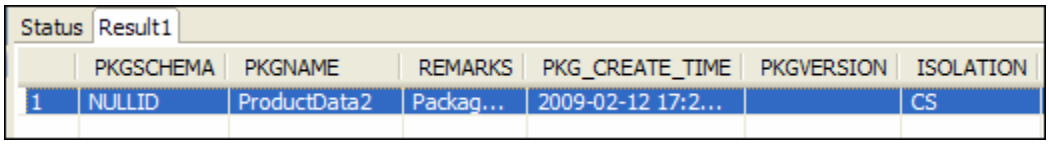
__16. Hit `CTRL-V` to paste `DROP` statements. Right click anywhere and select `Execute All` to drop the packages.



__17. Check the status of the DROP command status in SQL Results view.



__18. Bind the data access classes from pureQueryLabs project to the database by right clicking on the project and select pureQuery ⇨ Bind pureQuery Application. 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

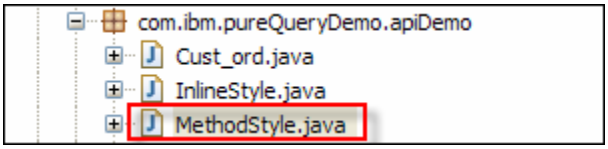
After binding the packages for data access classes, the SQL in the java application continues to run in dynamic mode unless we turn on the switch also known as executionMode.

__19. There are many ways to turn executionMode to STATIC or DYNAMIC. For example:

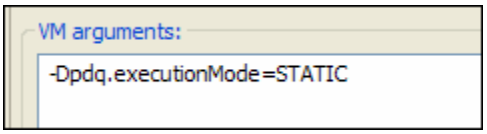
Scope	Method	Description and how to set
Global	JVM	Set the value as a JVM system property and is applicable to all of the pureQuery XML files in the application that you start with the java command.
Global	Property file	Use pdq.properties and it is applicable to all connections for an application
Connection 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.

__20. We will use the JVM option to set this property.

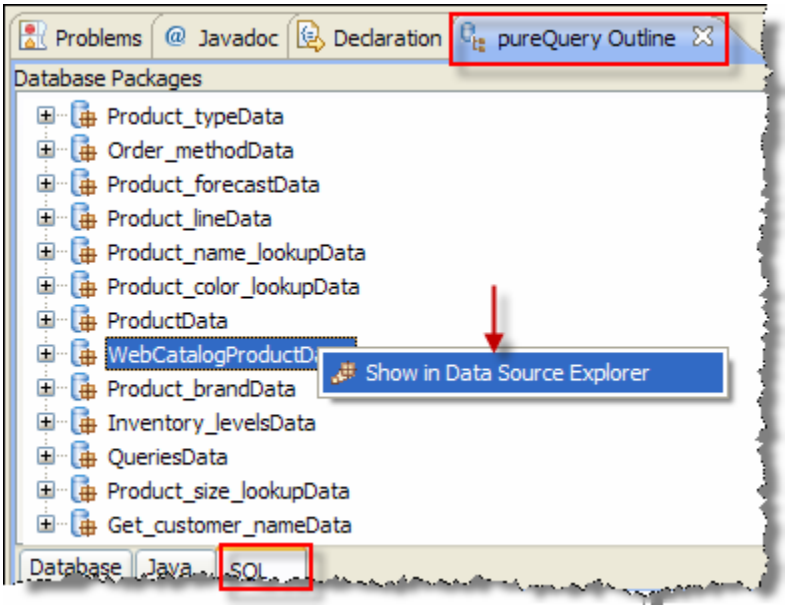
- Expand apiDemo package and double click MethodStyle.java to open it in an editor.



- 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.

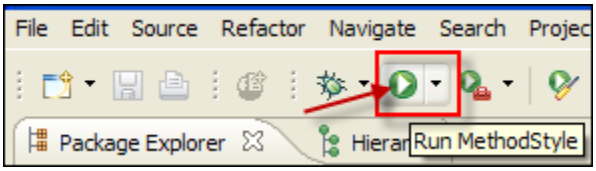


- Notice the output on the Console is the same as if you were running dynamic SQL.
- But how did you know if it ran using DB2 package or not? Go to the *pureQuery Outline* view and click on `SQL` tab at the bottom. Go to the `WebCatalogProductdata` package and right click on it. Click on `Show in Data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

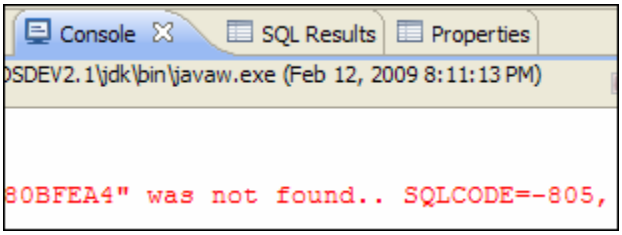


- In the Data Source Explorer view, right click on `WebCatalogProductdata` package and click on `Drop`. Choose `Data Object Editor` when prompted and drop the package.

- After dropping the package, run it again.

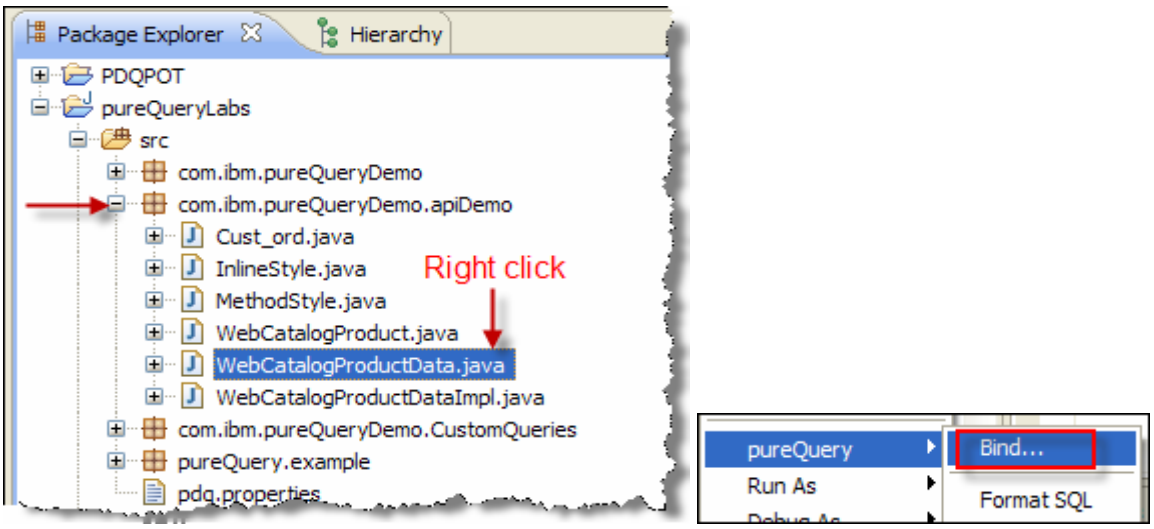


- You will see SQL -805 error indicating that the `WebCatalogProductdata` package was not found.

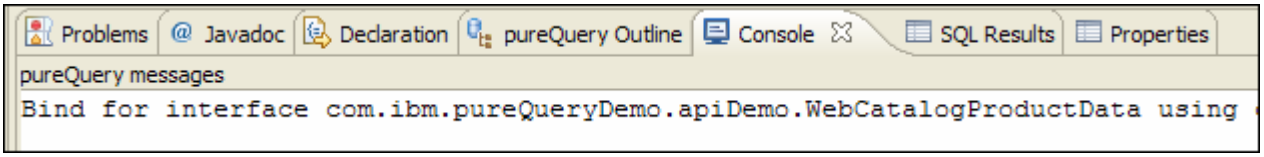


5.4 Bind a single Interface using pureQuery Tools

In the previous step, we dropped `WebCatalogProductdata` DB2 package manually to see if we could run `MethodStyle`. We noticed SQL -805 error confirming that the `WebCatalogProductdata` package was not found. Expand `apiDemo` package and right click on the `WebCatalogProductdata` data interface in the *Package Explorer*, and click on `pureQuery` ⇨ `Bind…`



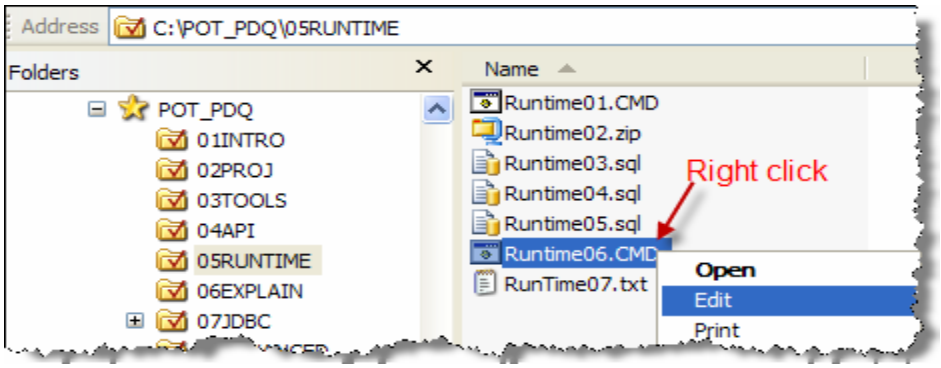
__21. Select `GSDB` database when prompted and click `<Finish>`. You should see this interface bound to the database.



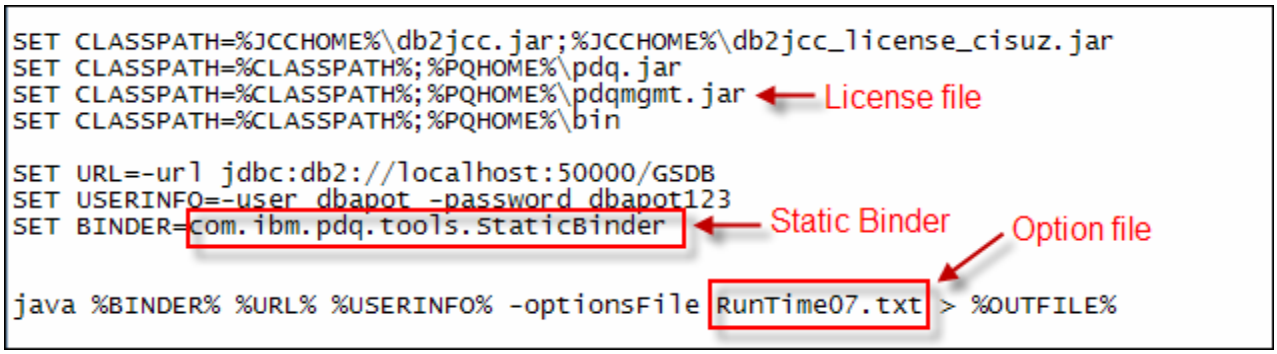
5.5 Bind Packages through Command Line

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.

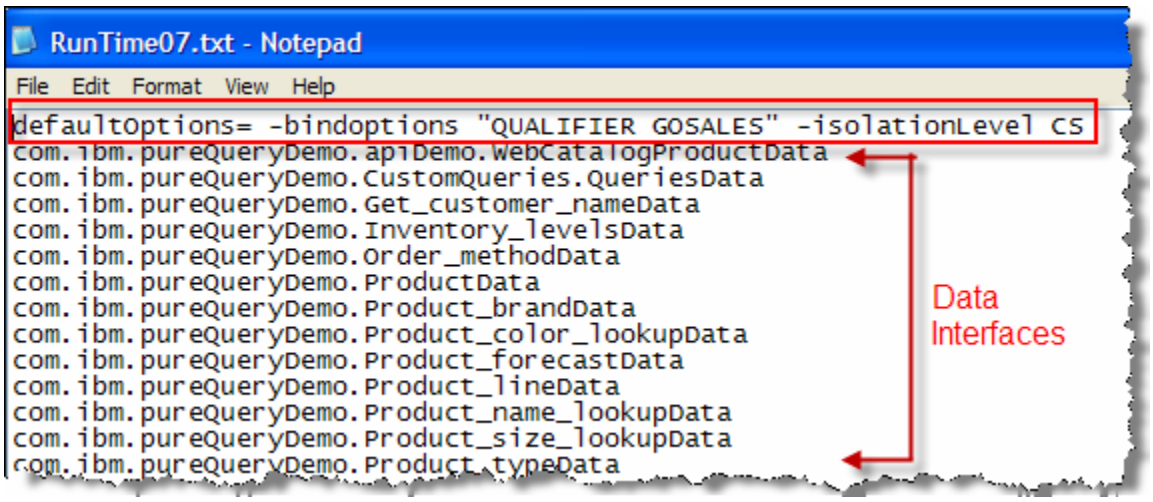
__22. In *Windows Explorer*, navigate to `C:\POT_PDQ\05Runtime` folder and right click on `RunTime06.CMD`. (Note: Do not double click to run it yet.)



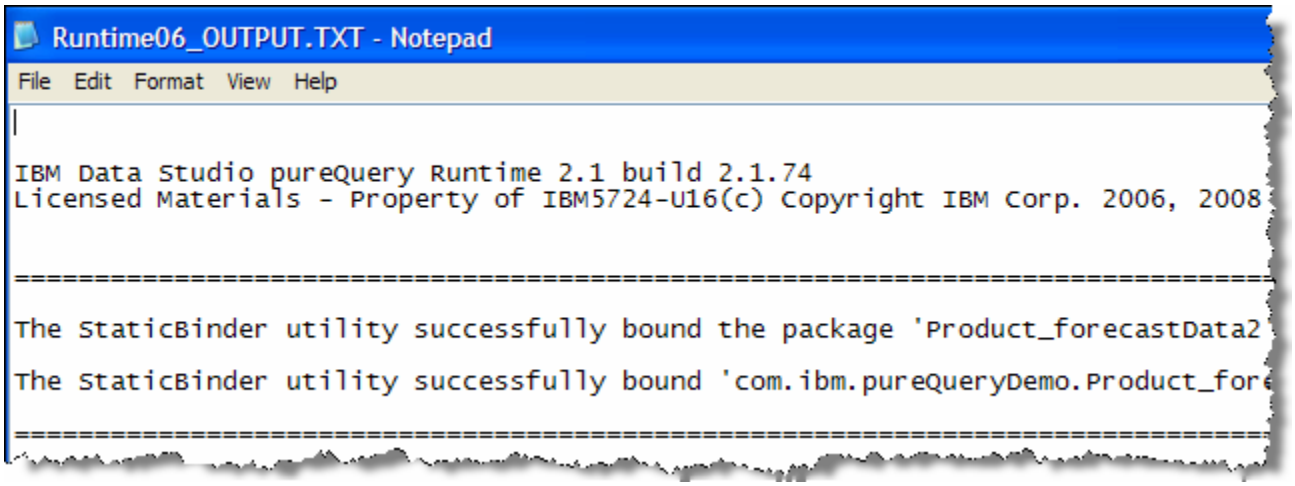
__23. Review the contents of this file and notice how a Static Binder is invoked.



__24. Close this file and double click on `RunTime07.txt` option file where data interface classes are listed.



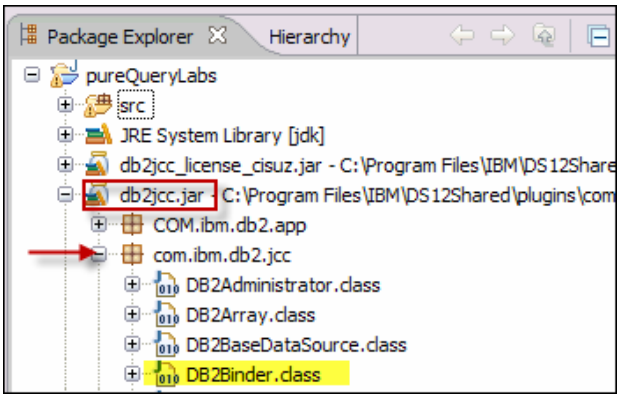
__25. Now double click on RunTime06.CMD and after it has completed the work, double click on `Runtime06_OUTPUT.TXT` and review the output.



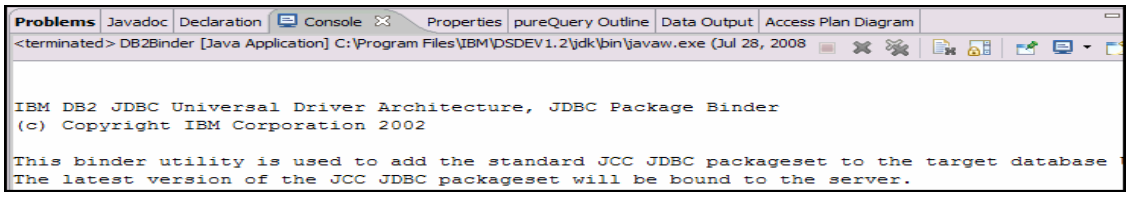
5.6 DB2Binder command to REBIND a package

__26. Under the `pureQueryLabs` project expand the `db2jcc.jar` file.

__27. Expand the `com.ibm.db2.jcc` package and notice the `DB2Binder` class.



__28. Right-click the `DB2Binder` class and select `Run As ⇨ Java Application` and you will see the help message.



__29. In *Windows Explorer*, navigate to the `C:\POT_PDQ\05RUNTIME` folder. Right click on the file `RUNTIME08.CMD` and select `Edit` to review the file. This is an example script that can be used and customized to `REBIND` DB2 packages.

```
SET JCCHOME=%CD%\driver
chdir /d %OLDDIR%

SET CLPATH="%JCCHOME%\db2jcc.jar";"%JCCHOME%\db2jcc_license_cisuz.jar"

SET URL=-url jdbc:db2://localhost:50000/GSDB
SET USERINFO=-user dbapot -password dbapot123
SET BINDER=com.ibm.db2.jcc.DB2Binder


"%JAVA_HOME%\bin\java" -cp %CLPATH% %BINDER% %URL% %USERINFO% ^
                       -collection NULLID -action REBIND -generic ^
                       -package WebCatalogProductData2 > %OUTFILE%
```

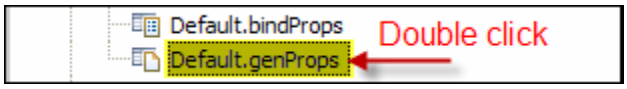
__30. Go ahead and close `RUNTIME08.CMD`. Double click it to run and review the `RUNTIME08_OUTPUT.TXT` to notice that the interface has been rebound to DB2.

```
File  Edit  Format  View  Help
Binder performing action "REBIND" to "jdbc:db2://localhost:50000/GSDB" under
Package "WebCatalogProductData2": Rebind succeeded.
DB2Binder finished.
```

5.7 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.

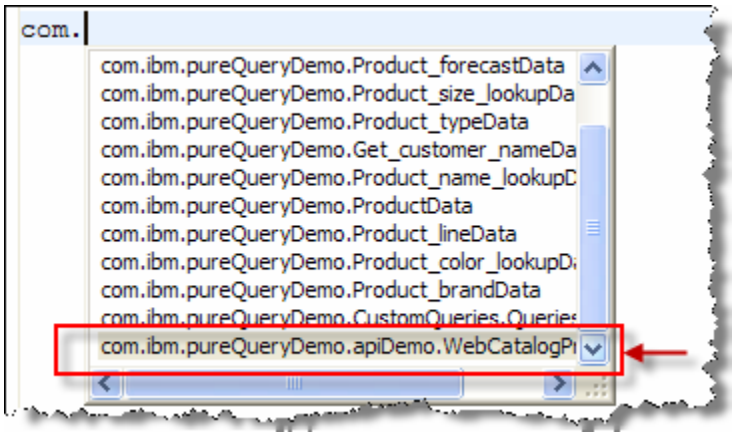
__31. Double-click on `Default.genProps` in `pureQueryFolder` of `pureQueryLabs` project.



__32. Add following this line in `Default.bindProps` file.

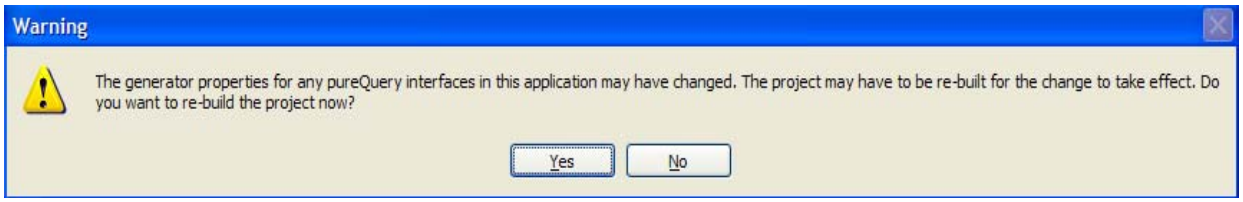
```
defaultOptions= -isolationLevel CS
```

__33. Add following line in `Default.genProps` file 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.



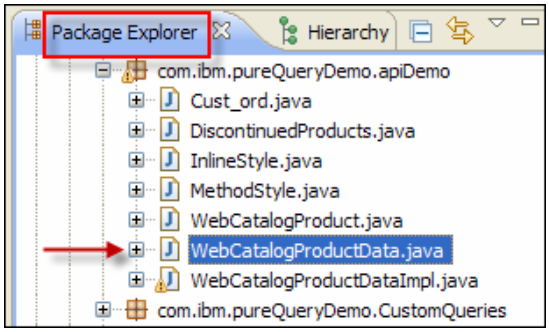

```
com.ibm.pureQueryDemo.apiDemo.WebCatalogProductData = -collection PDQCOL
```

__34. 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>`.

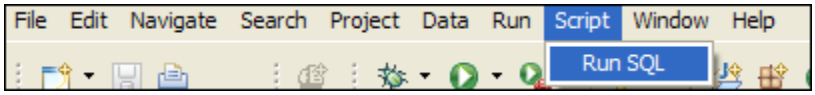


__35. Open `WebCatalogProductData.java` file. Delete any character and re-type same character and save the file.

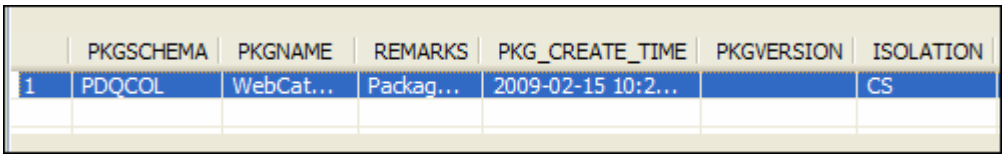
__36. Re-bind the interface by right clicking on it and selecting `pureQuery` ⇨ `Bind`. Select `GSDB` database when prompted.



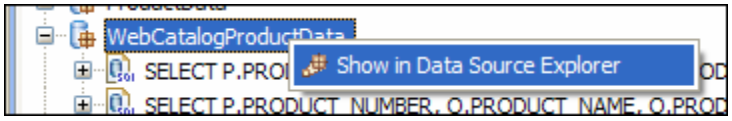
__37. Go to the *Package Explorer* and double click `Runtime04.sql` file in an editor. Click on `Script` ⇨ `Run SQL.`



__38. View the output of the command in *Results* view.



__39. 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

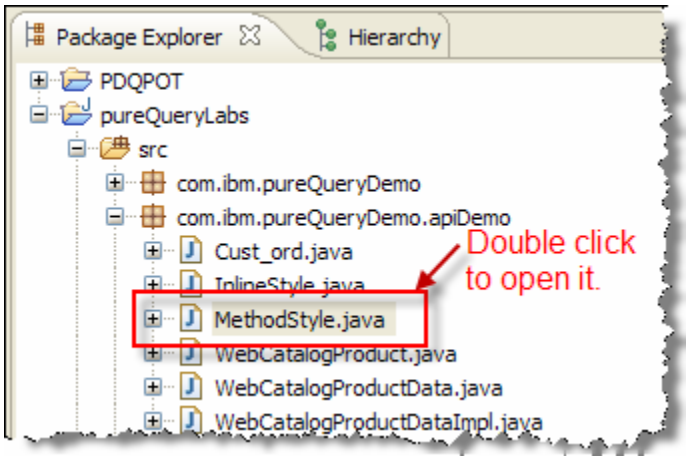
Introduction:

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

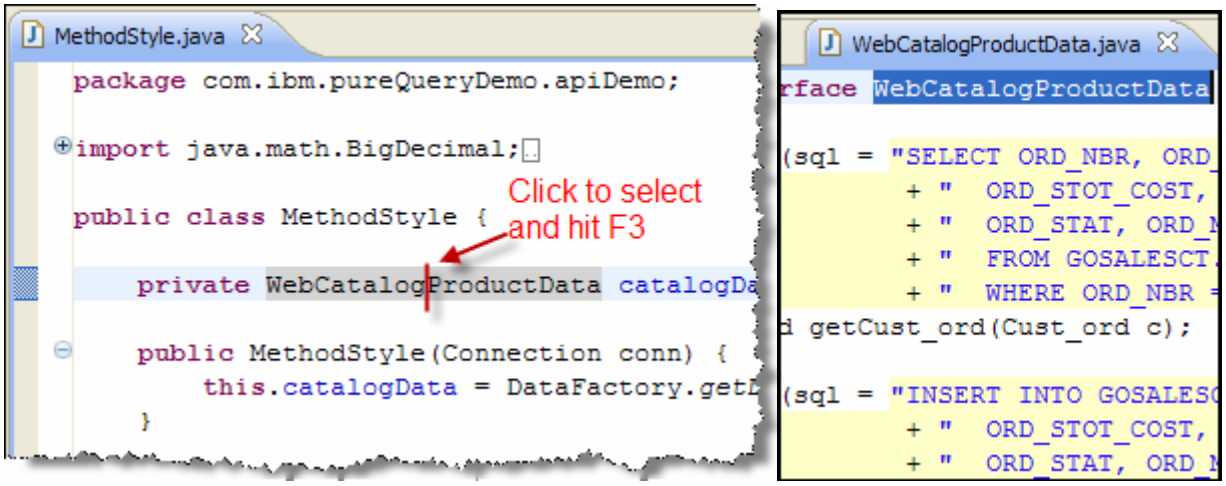
6.1 Explain Plan for SQLs in Java Programs

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

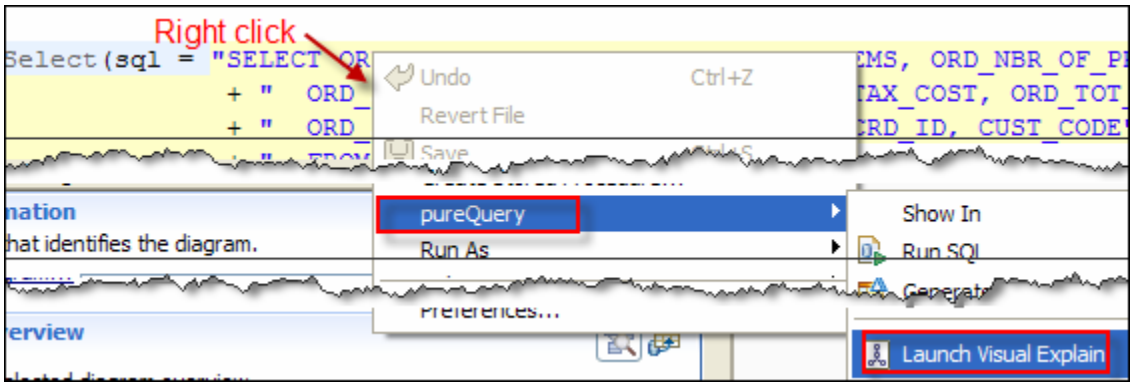
__2. In your *Package Explorer*, expand `apiDemo` package and double click on `MethodStyle.java` to open it in an editor.



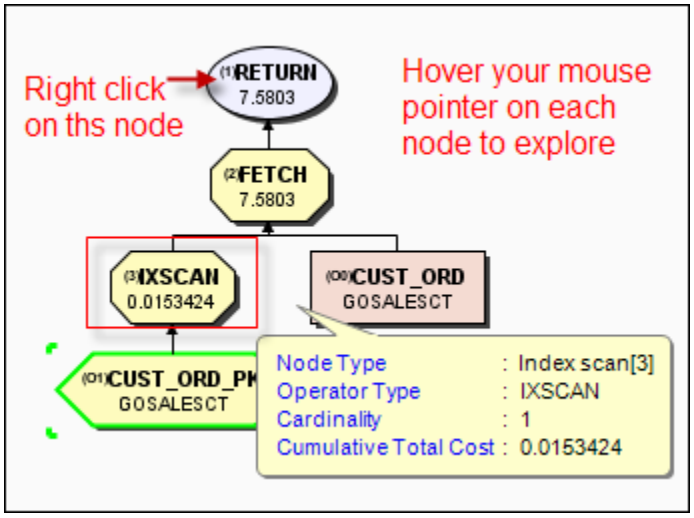
__3. Click anywhere in `WebCatalogProductData` and hit F3 to open the data interface file.



__4. Click anywhere on the first SQL statement and right click to select `pureQuery` ⇨ `Launch Visual Explain`.

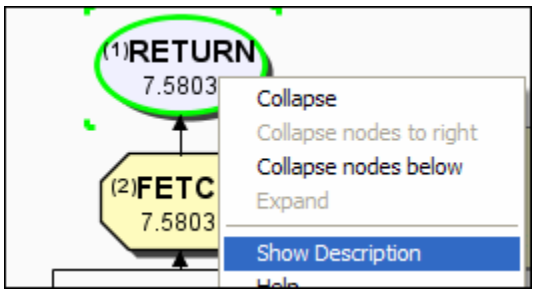


__5. When Visual Explain screen shows up, click <`Finish`> to launch it. Look at the explain plan in *Access Plan Diagram* in bottom right corner of the java perspective.

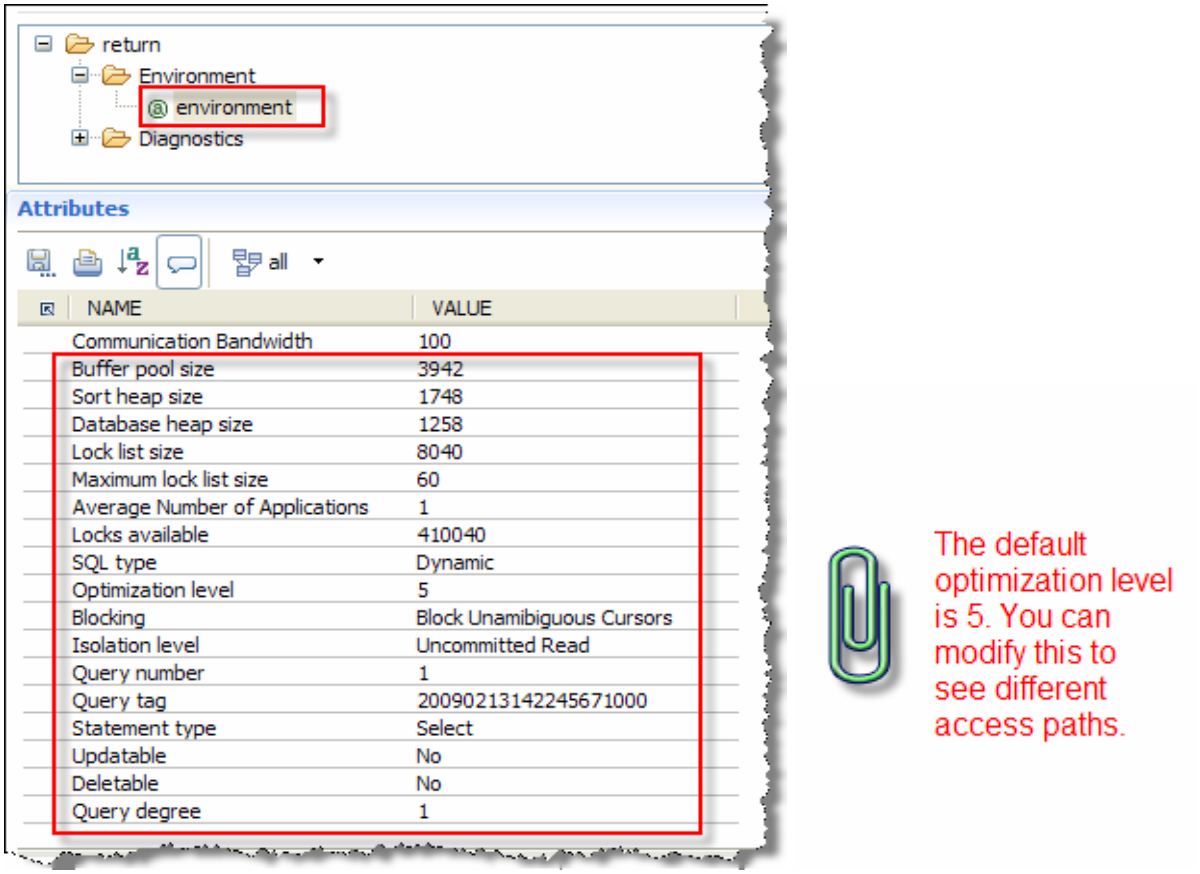


The number displayed below the operator at each node is the cumulative value of the timeron, which is also known as cost.

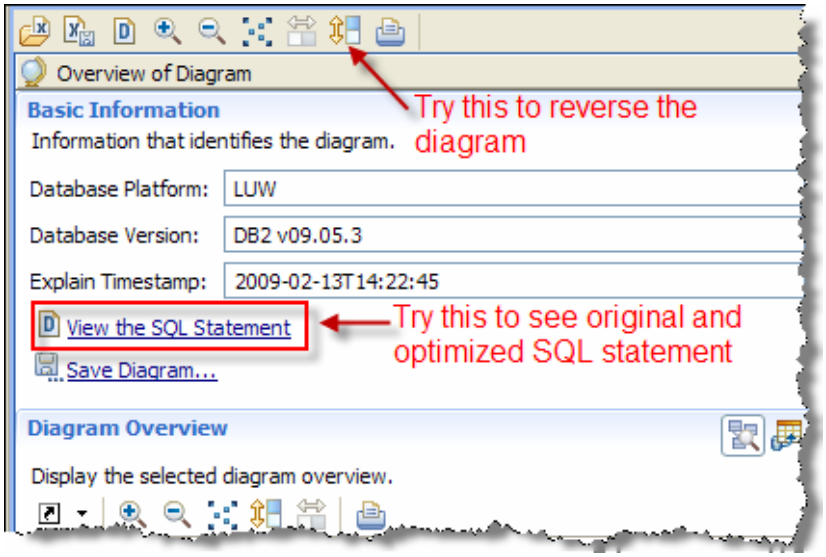
__6. Right click on a node and select `Show Description` to see details about a node.



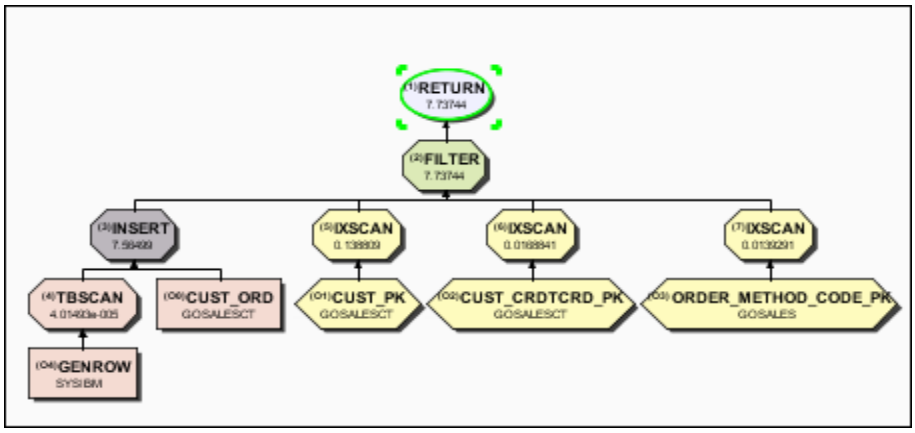
__7. For example, you will be able to see CPU, I/O and computed cumulative explain cost along with values of the database parameters that affect a explain plan as shown below:



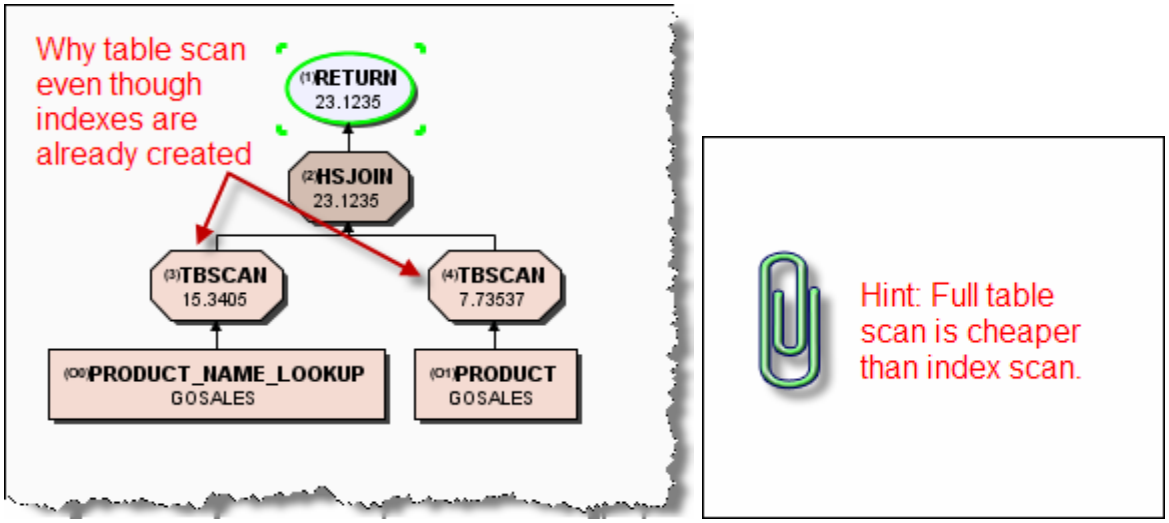
__8. In the *Overview Diagram,* click on `View the SQL statement` to see original and optimized SQL. You can save an explain plan in a XML file for viewing it later or sending it to the DBA.



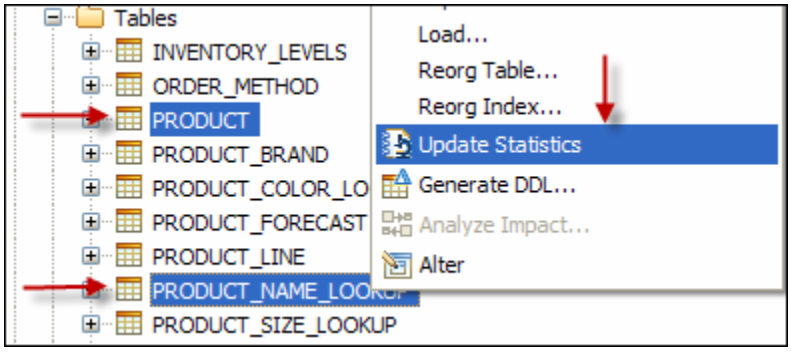
__9. Go back to the `WebCatalogProductData.java` program and see the explain plan for each of the SQL statement. For example, try to figure out why there are multiple index scans when inserting a row in `CUST_ORD` table.



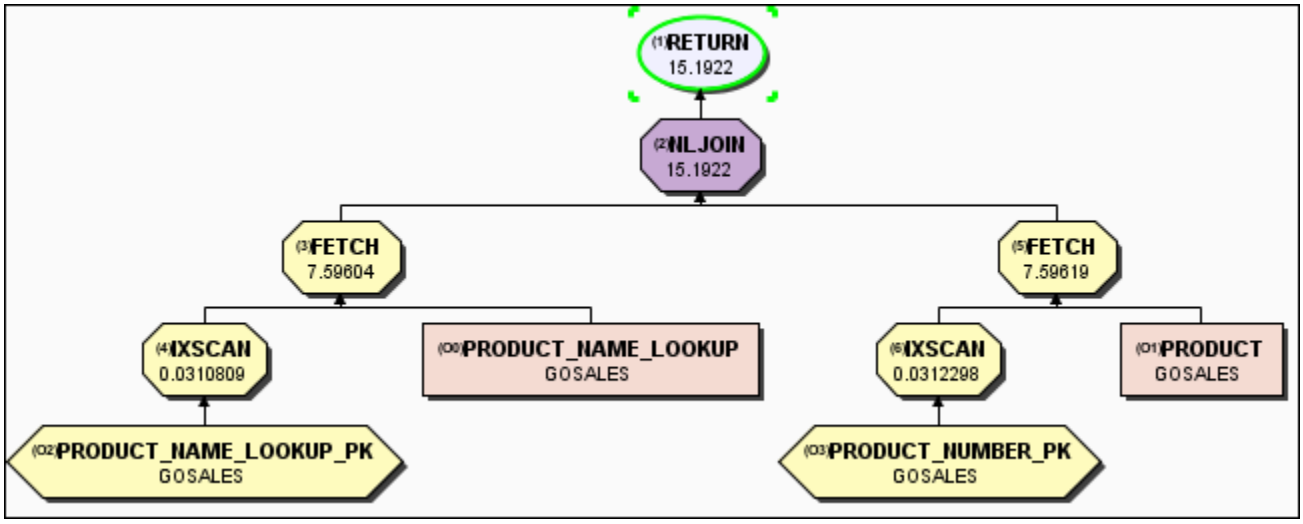
__10. Look at the explain plan for the query associated with the `getWebCatalog` method and why there are full table scans on `PRODUCT` and `PRODUCT_NAME_LOOKUP` when indexes exist on the columns accessed in the query.



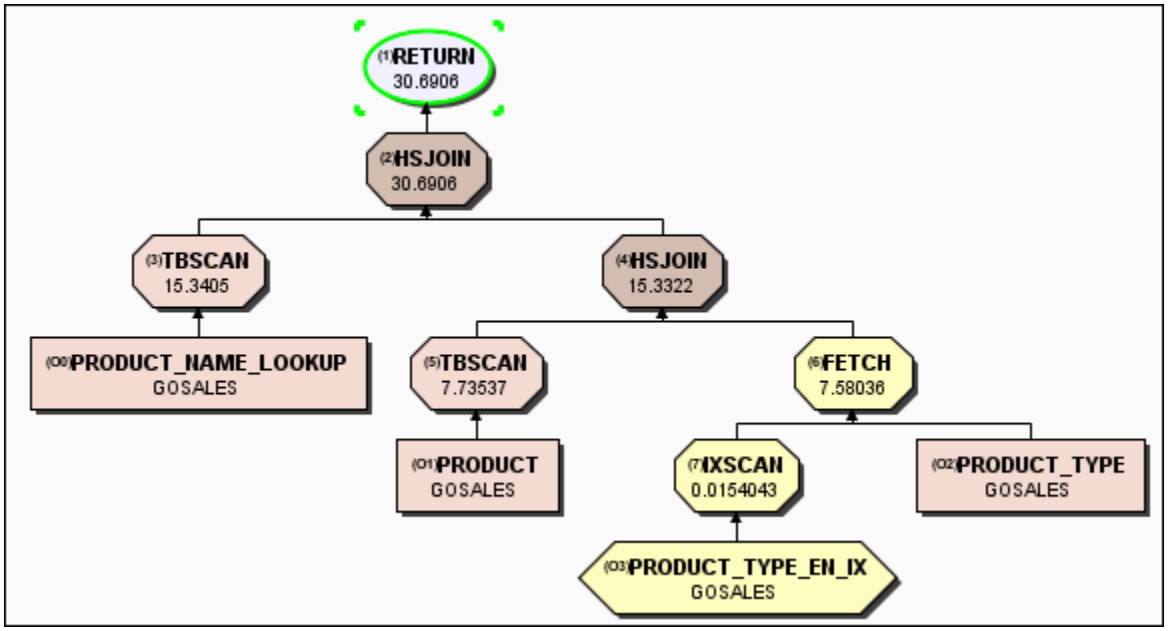
__11. Try updating statistics on both tables and regenerate the explain plan. If plan does not change, can you think of a reason that why it still did not use indexes.



__12. Look at the explain plan for the query associated with the method `getWebCatalogProductByNumber` and notice the nested loop join when we query by a number which is on index.

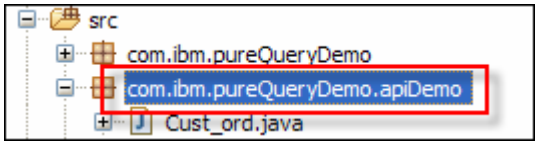


__13. Look at the explain plan for the query associated with the method `getWebCatalogProductByType` where the search is based upon the type (not on the index) and access path uses hash loop join.

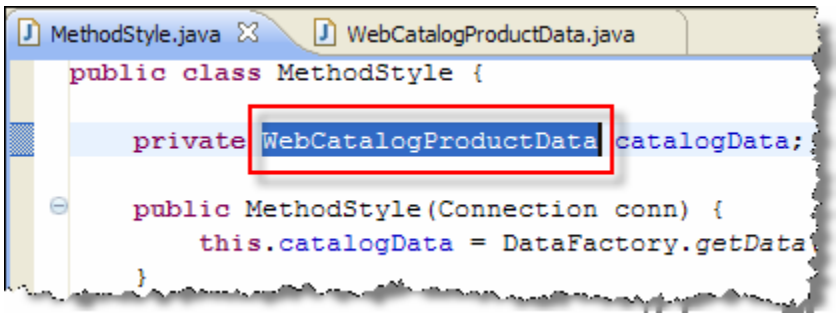


6.2 Explain Plan for new methods

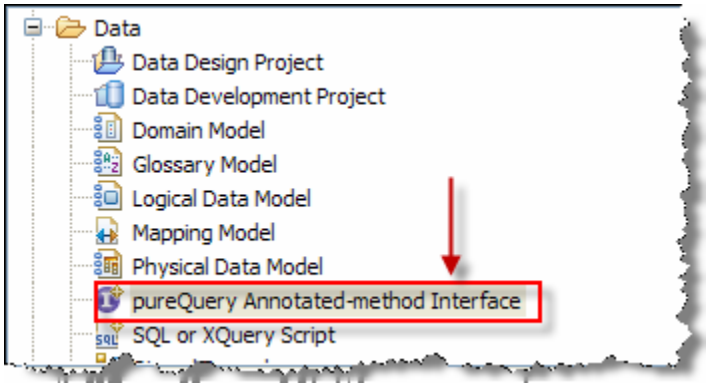
__14. Go to the *Package Explorer* and click `apiDemo` package to select it.



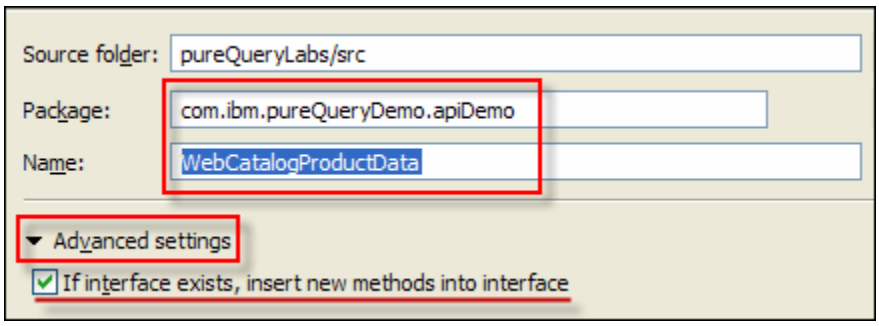
__15. Go back to the `MethodStyle.java` program and double click on `WebCatalogProductData` to select it.



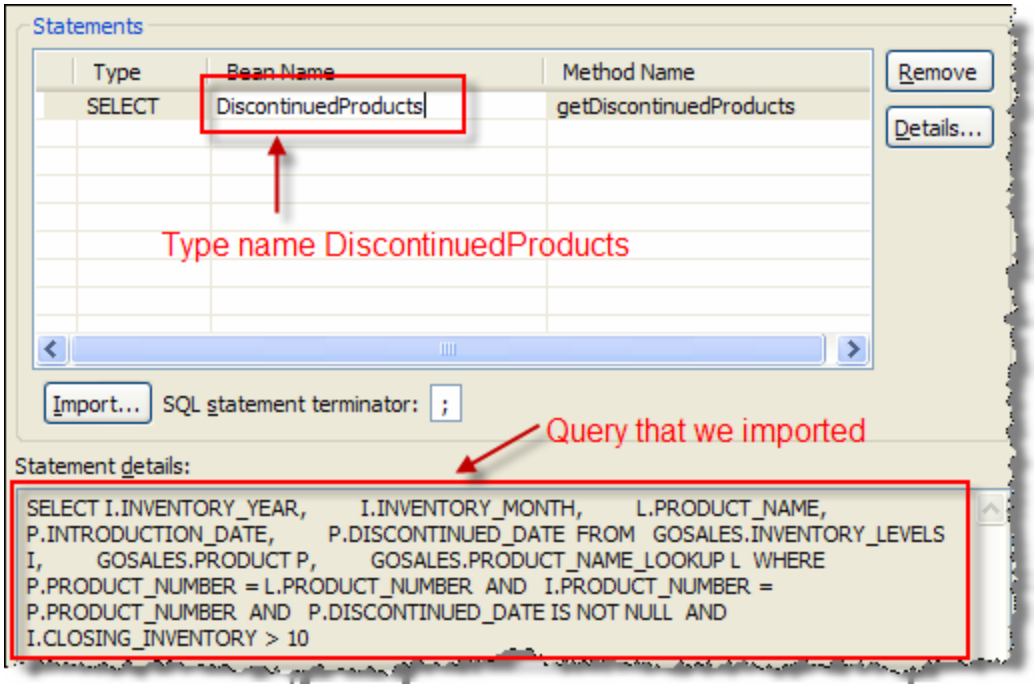
__16. Hit `CTRL-N` to open a new wizard. Expand Data and select `pureQuery Annotated-method Interface`. Click on `<Next>`.



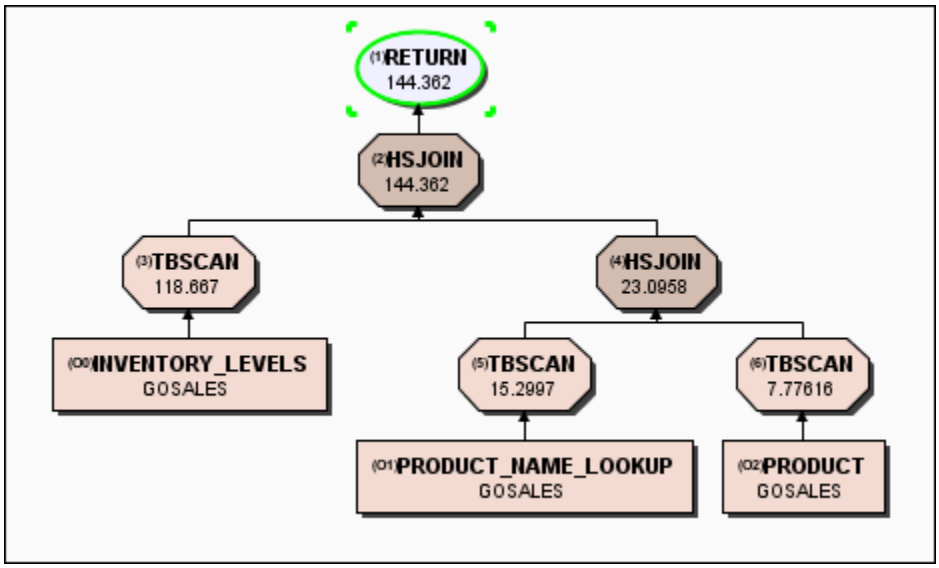
__17. Make sure to expand the `Advanced Settings` and click on `If Interface exists, insert new methods into interface`. The other two values should already be selected for you. If not, type-in those values. Click `<Next>`



Click on `Import` button and change directory to `C:\POT_PDQ\06EXPLAIN` and select `Explain01.sql` to import the SQL statements from this file. 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. Click <`Finish`> to generate a bean to hold the results and add method to an existing interface.



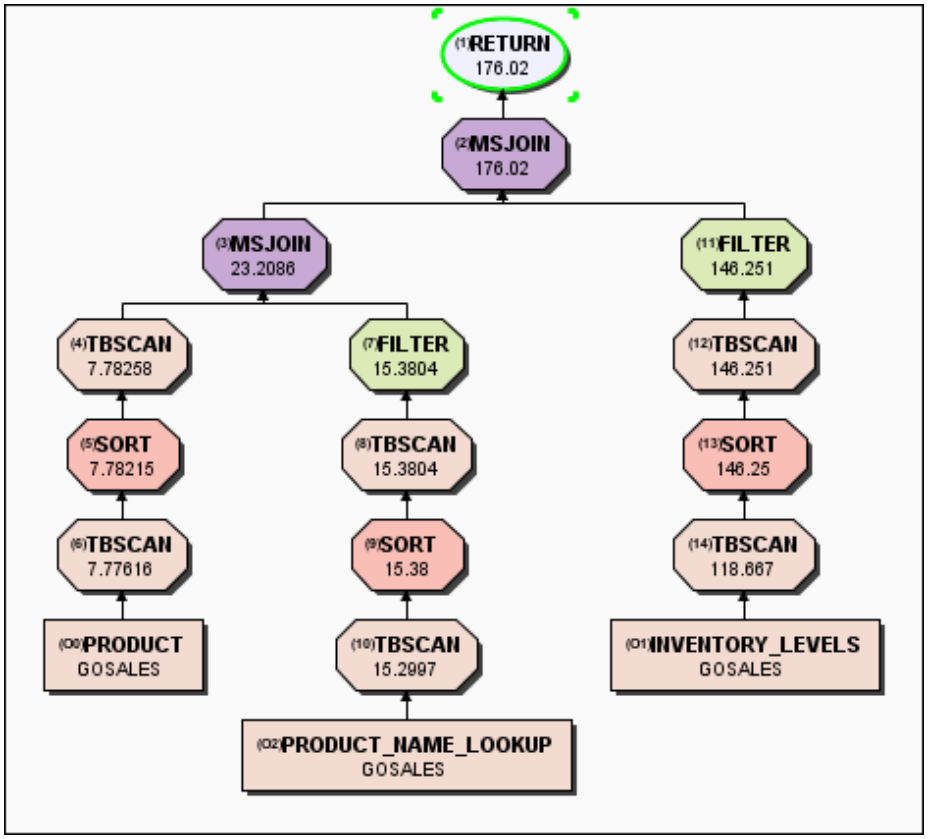
__18. The method `getDiscontinuedProducts` is added to `WebCatalogProductData` interface. Click anywhere on the SQL statement and right click to select `pureQuery` ⇨ `Launch Visual Explain` and hit <Finish> to generate an explain plan. The explain plan may look like following.



6.3 Explain Plan with Different Query Optimization

__19. Again right click on the SQL statement and select `pureQuery` ⇨ `Launch Visual Explain` and click `<Next>` and select *Current Query Optimization* to 1 and click on `<Finish>`.

__20. Examine the explain plan and compare it with the previous one.



You may notice the different access path by using different optimization profiles. Refer to the DB2 documentation for details.

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

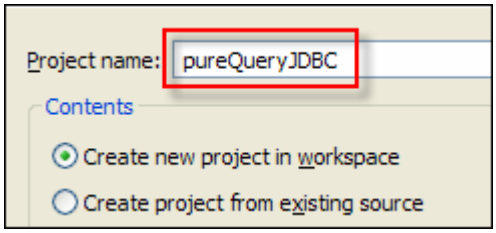
Lab 7 Optimize an existing JDBC Application using pureQuery

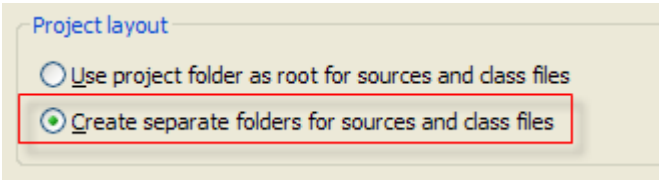
IBM Data Studio Developer pureQuery feature allows you to optimize existing JDBC applications. The pureQuery features allow you to optimize custom or packaged JDBC applications to execute SQL statements statically without a need to change the application in any way.

7.1 Create a Java Project

__1. Switch your perspective to Java. Click menu `Window ⇨ Open Perspective ⇨ Other…` Click on `Java` and click `OK`.

__2. Create a new Java project. Click `File ⇨ New ⇨ Java Project.` Specify project name as `pureQueryJDBC` and click `<Finish>` to create the project.





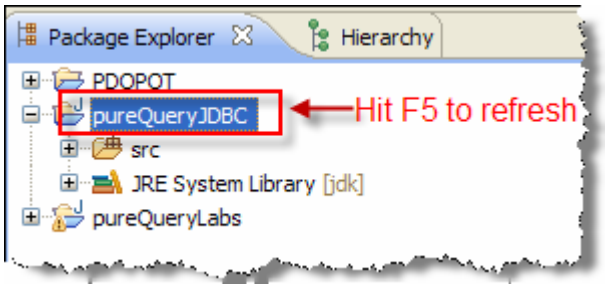
Note: Please make sure that you type the name of the project **exactly** as given above.

__3. Specify name of the project as `pureQueryJDBC` and select radio button for `Create separate folders for source and class files`. Click `<Finish>` on this screen.

__4. In Windows explorer go to `C:\POT_PDQ\07JDBC` directory.

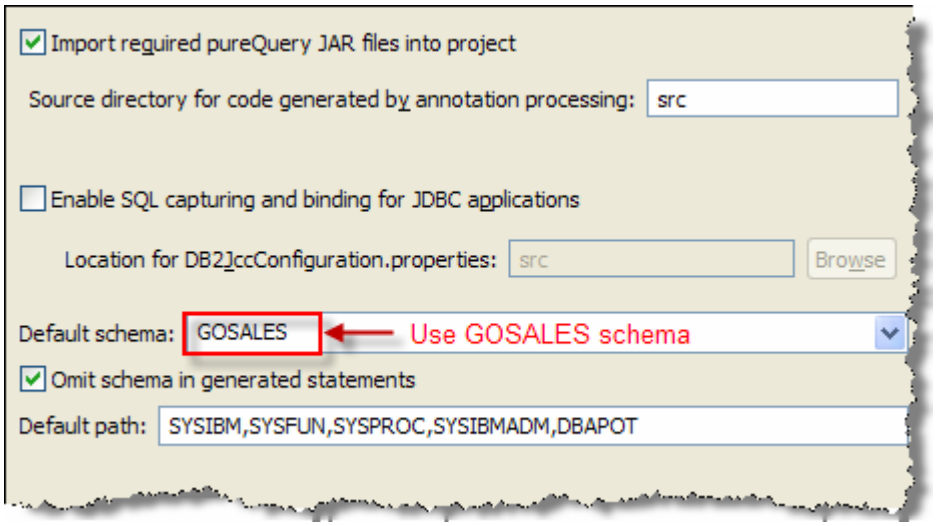
__5. Double click on `JDBC01.CMD` to copy `JDBC02.java` in the `pureQueryJDBC` java project.

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

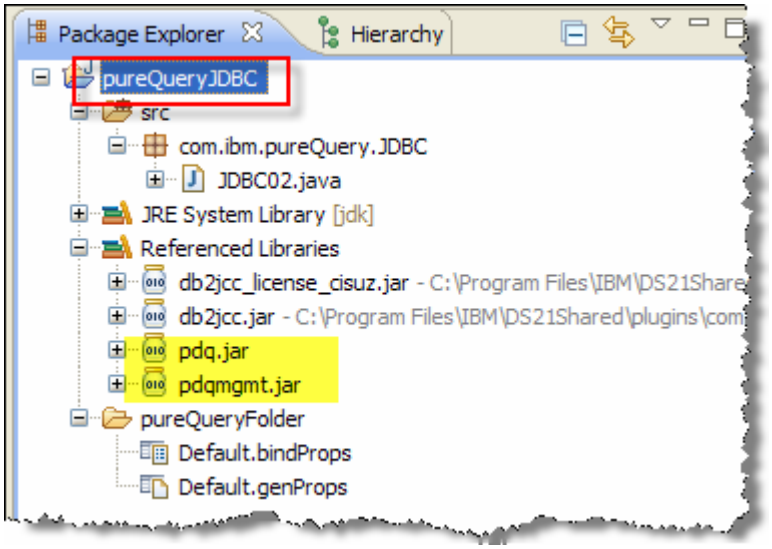


__7. Right click on `pureQueryJDBC` project, select `pureQuery` and `Add pureQuery Support …`

__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.

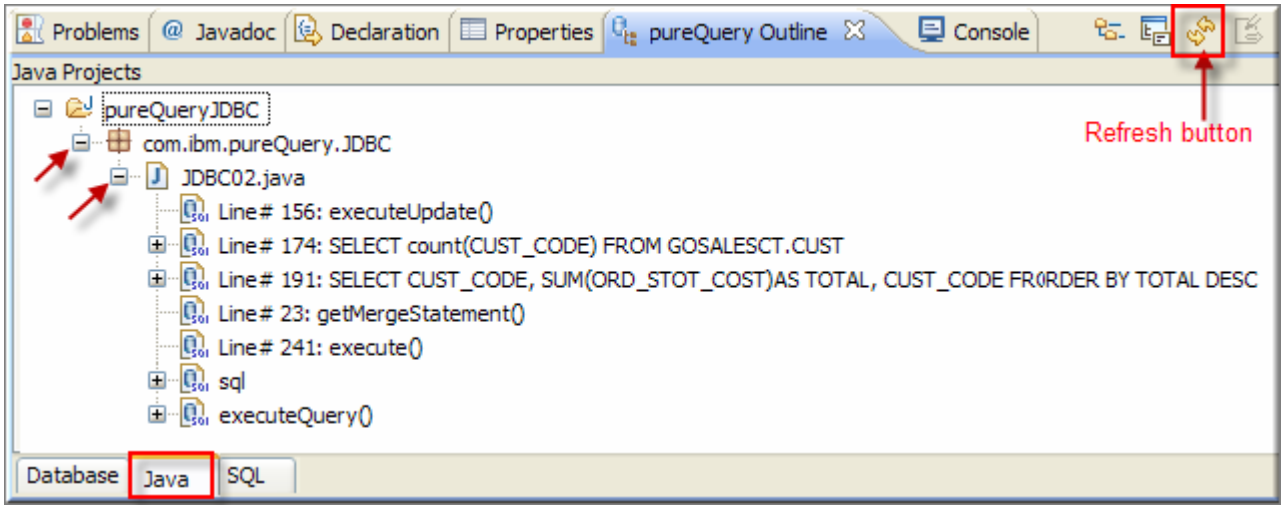


7.2 SQL Profiling when source is available

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

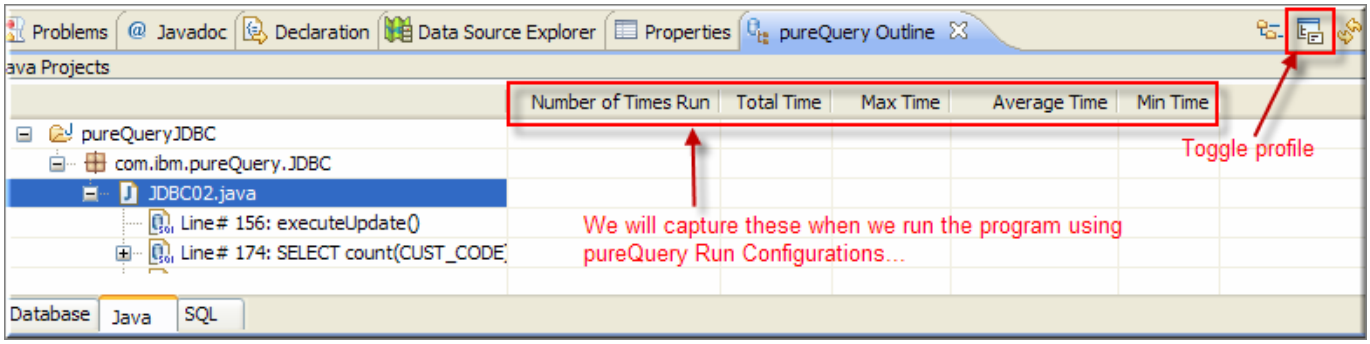
__10. Double click on `JDBC02.java` to open it in the editor.

__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.

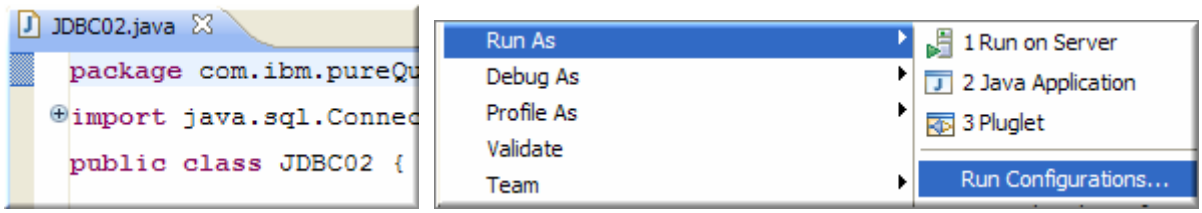


__12. You will notice the line numbers where SQL is getting executed. This information is captured even though you have not run the program.

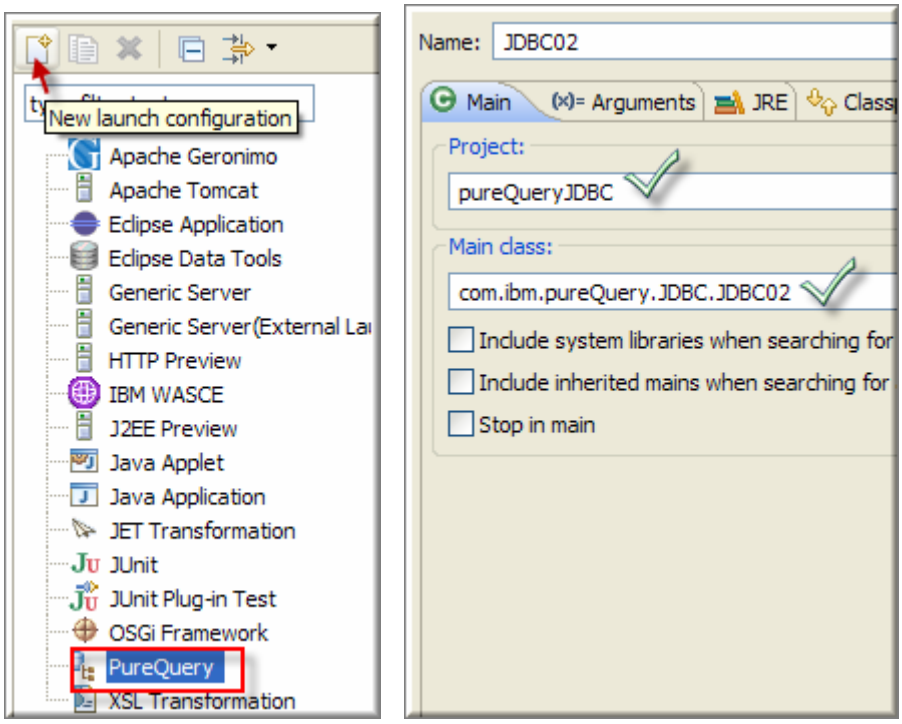
__13. Click on Toggle Profile button on the view to see the view for the SQL profiling. At this time, we will not see SQL profile data since we did not run the program.



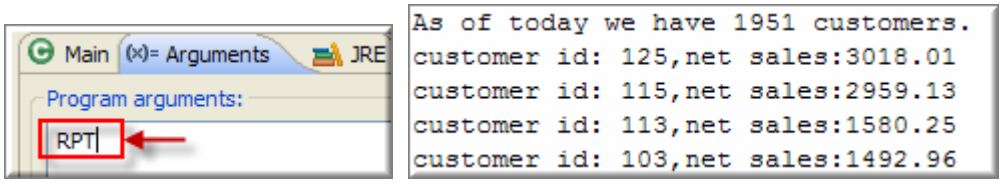
__14. Right click within `JDBC02.java` program and select `Run As ⇨ Run Configurations…`



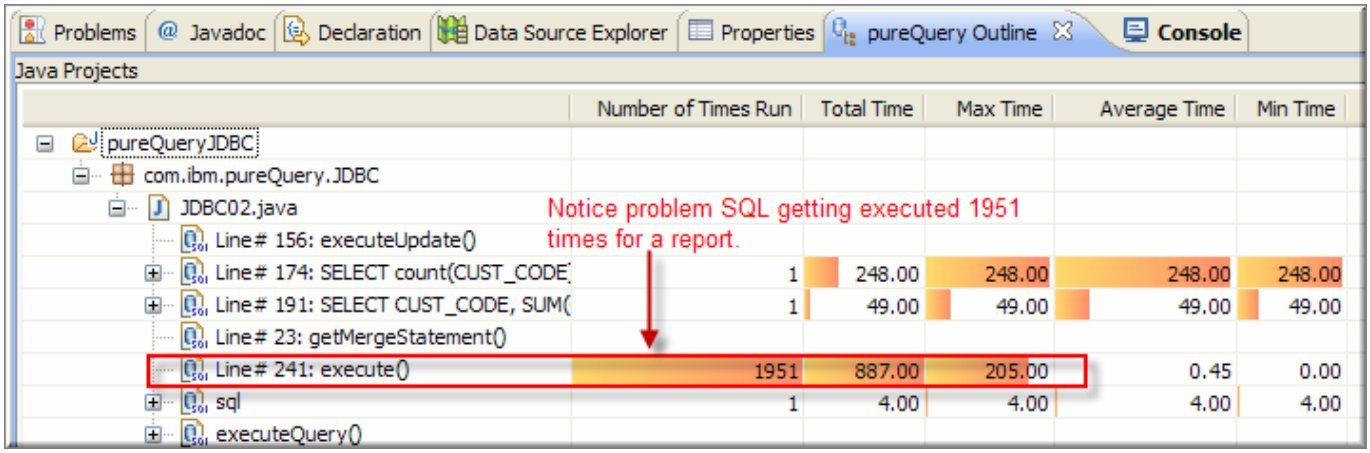
__15. Click on pureQuery to select it first. Click on `New launch configuration`. You will see right hand side window populated with `JDBC02.java` information.



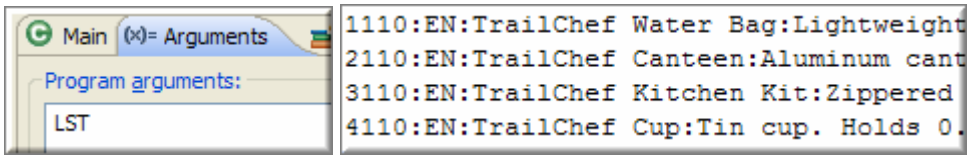
__16. Click on the `Arguments` tab and specify `RPT` and click `Run` to run the program. You will see following console.



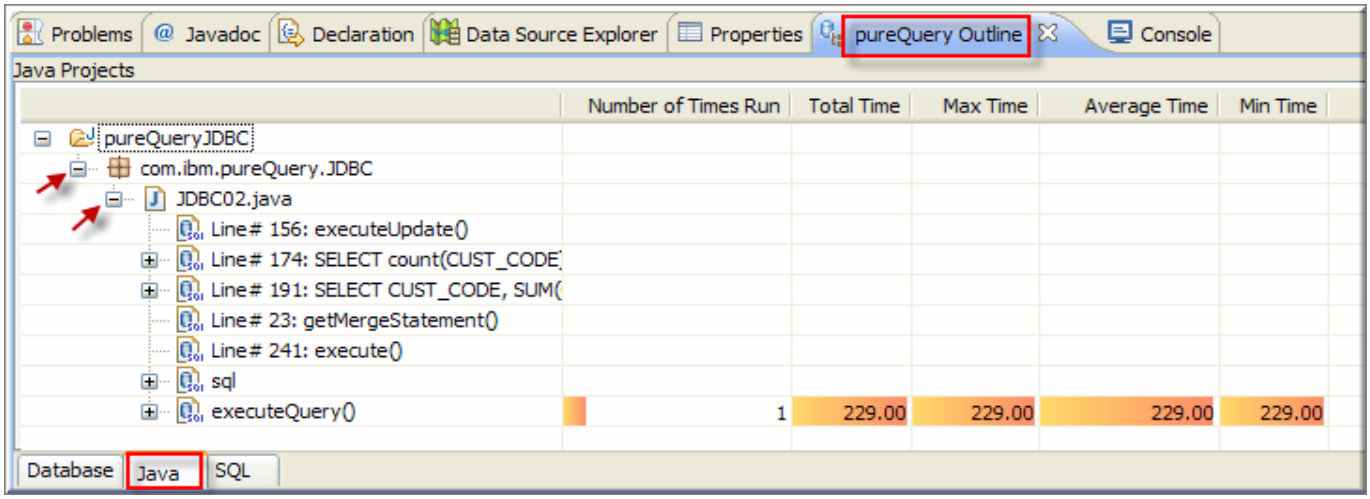
__17. Go to the *pureQuery Outline* view and hit the refresh button. You will see a view similar as shown.



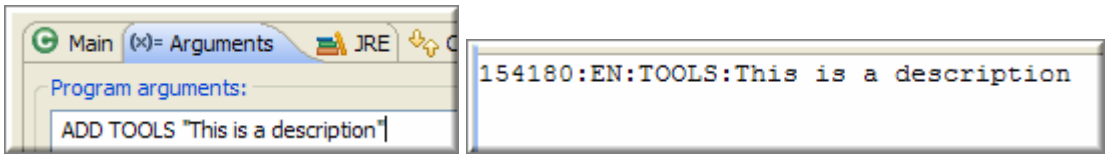
__18. Click on menu `Run ⇨ Run Configurations….` Click on `Arguments` tab and specify `LST` option. Click Run to run the program.



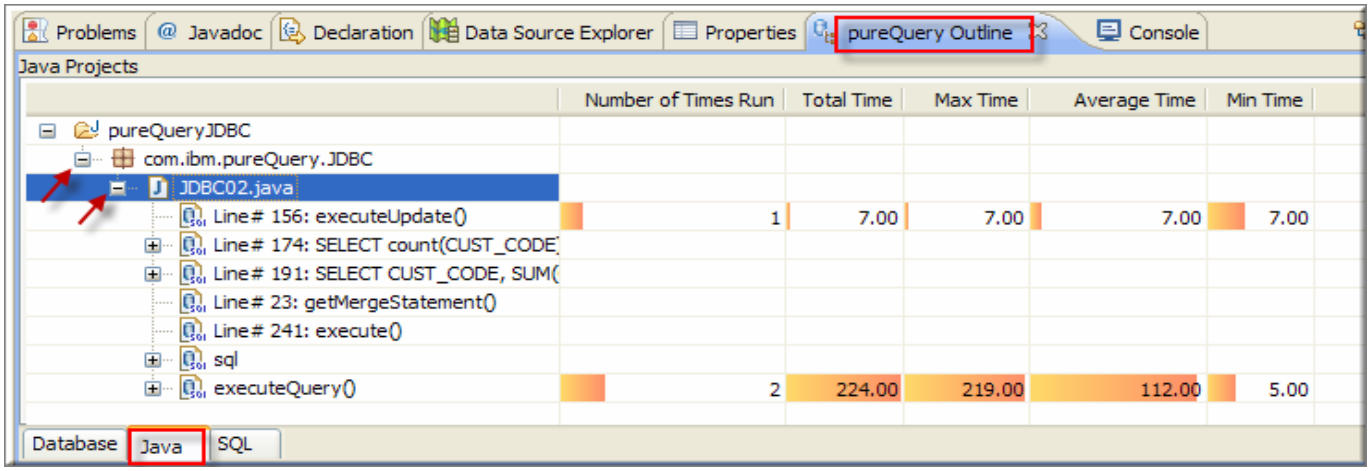
__19. Go to the *pureQuery Outline* view and hit the refresh button. You will see a view similar as shown.



__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.



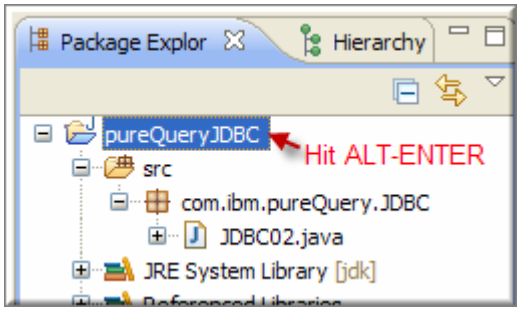
__21. Go to the *pureQuery Outline* view and hit the refresh button. You will see a view similar as shown.



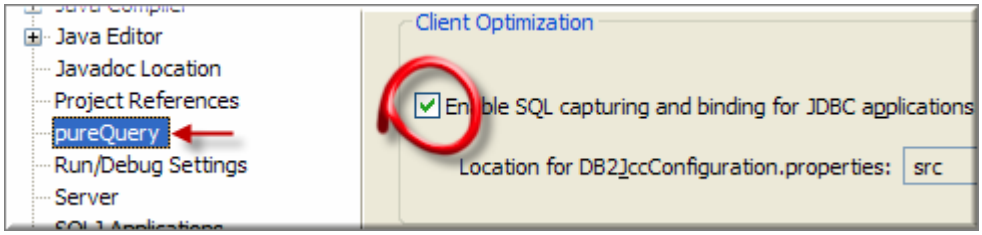
7.3 Optimization when source is available

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

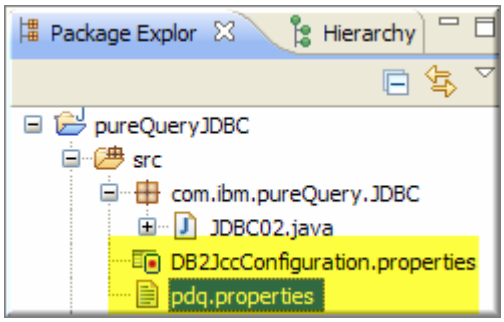
__22. Click `pureQueryJDBC` project and hit `ALT-ENTER` to open properties.



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



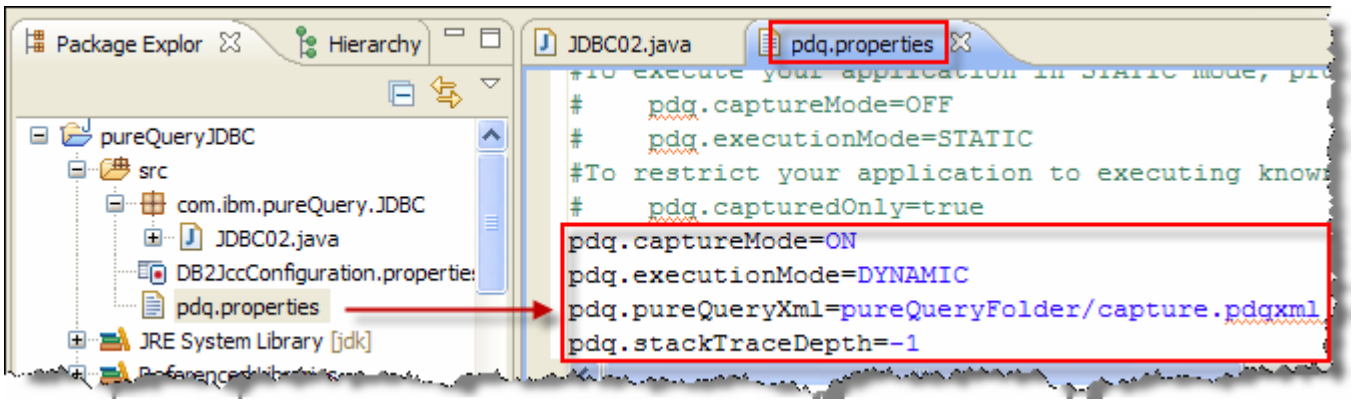
__24. After we enable SQL capture, you will notice addition of 2 files in the java project as shown.



__25. We will be running this program using scripts given in `C:\POT_PDQ\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.

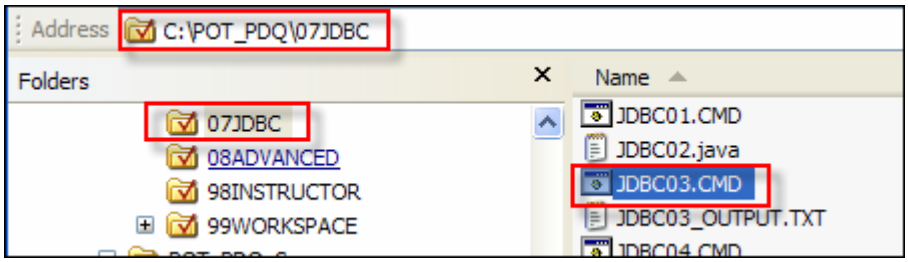
7.3.1 Capture metadata

__26. Double click on `pdq.properties` to open it in an editor. It has `pdq` properties set to capture the SQL metadata.



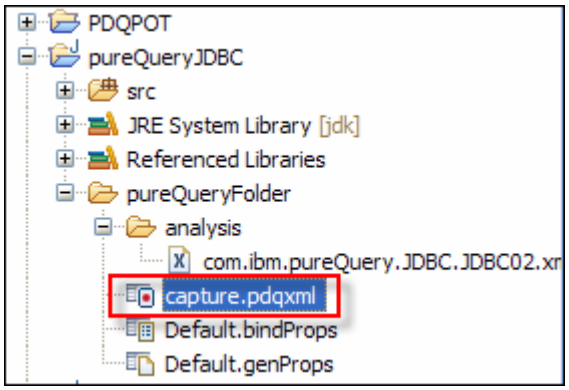
__27. Go to *Windows Explorer* and navigate to `C:\POT_PDQ\07JDBC` and right click on `JDBC03.CMD` and click `Edit`. Review the contents of the file and close it.

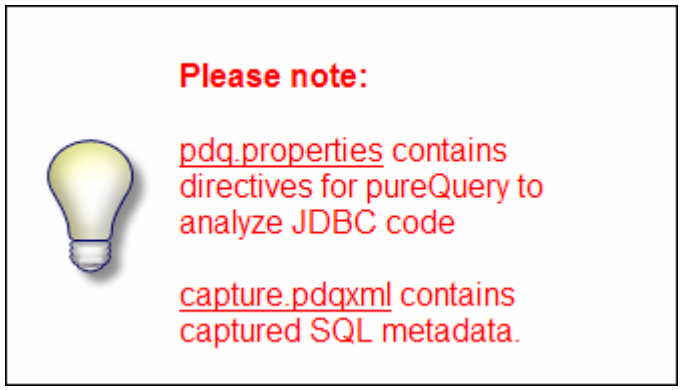
__28. Double click on `JDBC03.CMD` to run the JDBC application with different test cases and to capture SQL metadata.



__29. Review `JDBC03_OUTPUT.TXT` file. This contains output from our custom JDBC application.

__30. 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.)



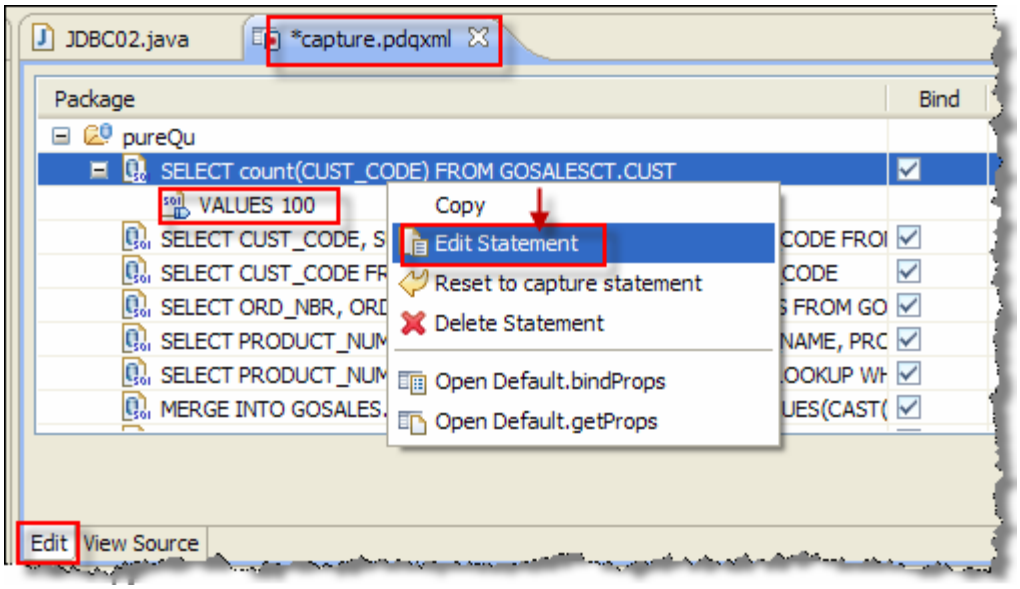


7.3.2 Browsing the captured metadata

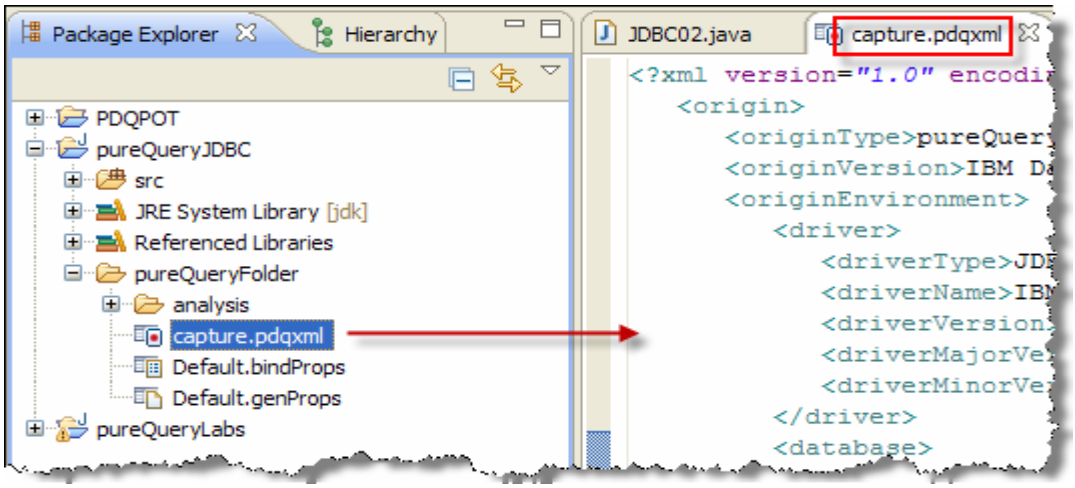
__31. In the *Package Explorer* and hit F5 to refresh it. Expand folder `pureQueryFolder` and double click `capture.pdqxml`. You can view this file in two modes 1. In Edit mode and 2. In View Source mode.

- 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.

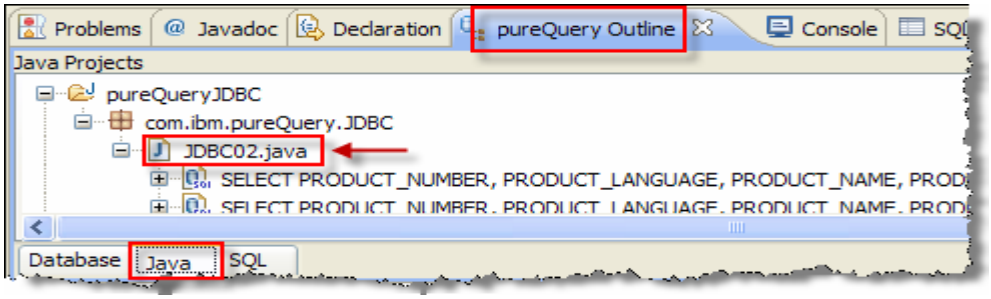
- Go ahead and modify `SELECT count(CUST_CODE)` statement to `VALUES 100` where you replaced original statement with a fixed return value of 100.

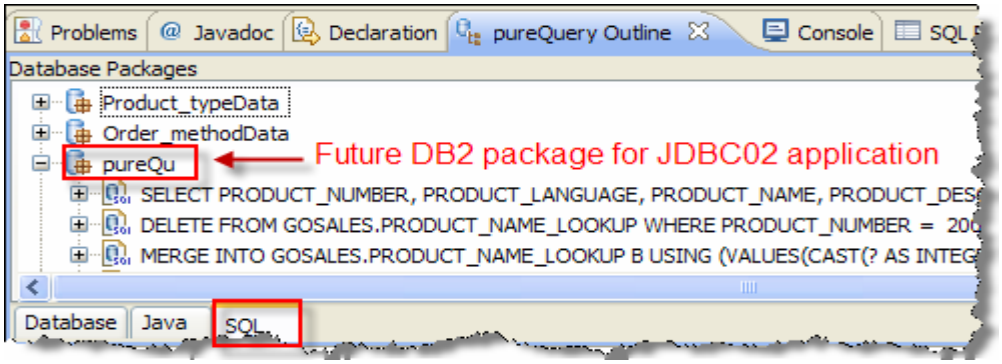


- In `View Source` mode

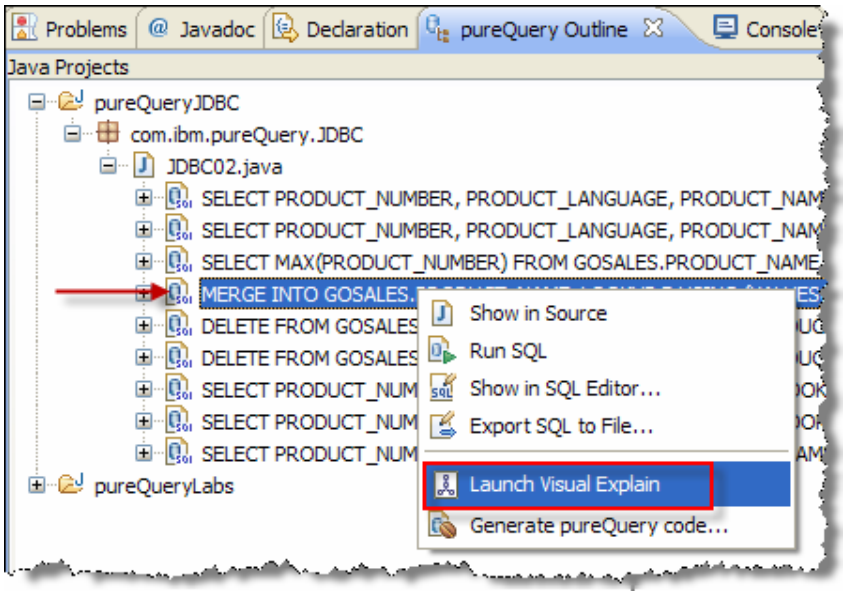


__32. Navigate to the *pureQuery Outline* view. Explore the contents of the `Java` and `SQL` tabs to browse same information in different views. You will notice actual SQL statements now.

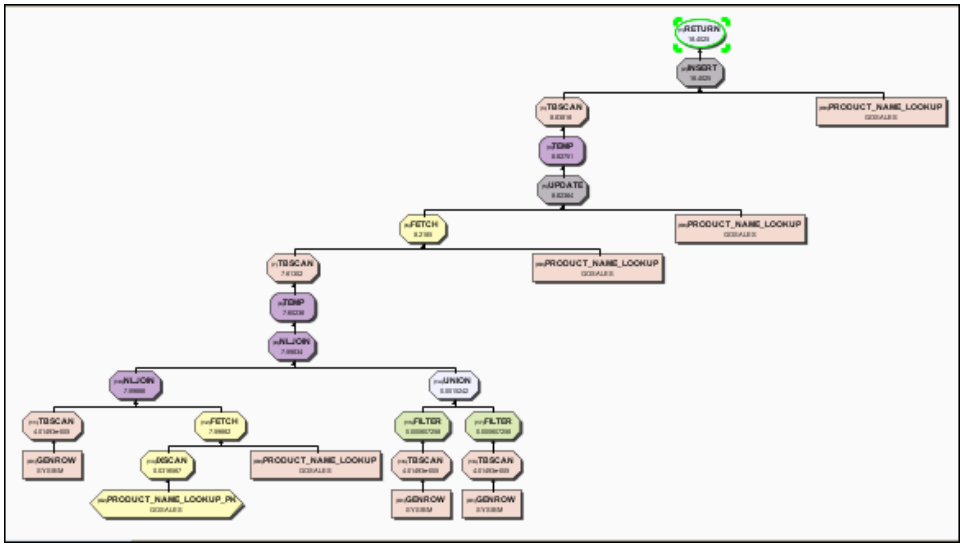




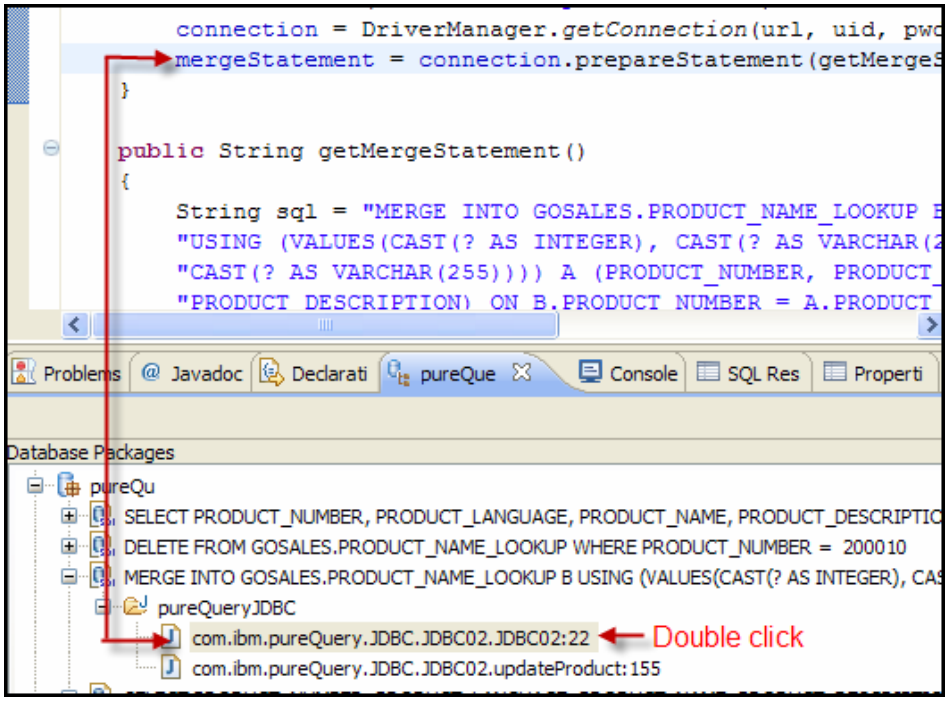
__33. You can do a number of activities on the SQL shown in the `pureQuery Outline` view. Right click on any SQL in any view to explore different actions. Try seeing explain plan for the MERGE statement used in the JDBC application.



__34. The visual explain plan of the MERGE statement looks like following.



__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.



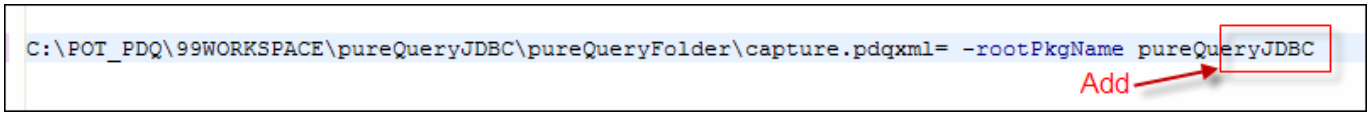
7.3.3 Configuring captured metadata

__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:

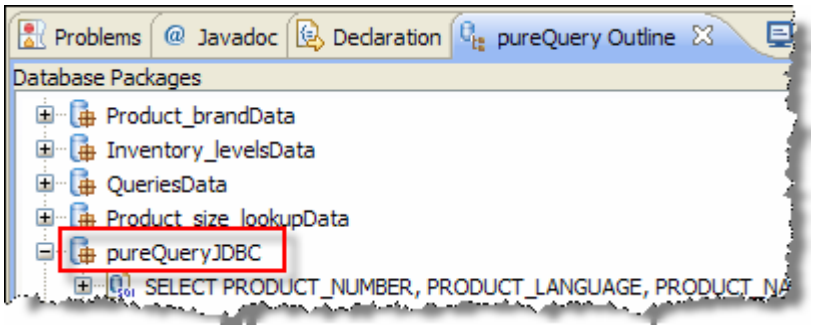
- Package name prefix
- Database collection id (or schema name containing package)
- If packages are versioned
- Maximum number of SQL statements that are to be included within a single package before a new one is created.

__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.

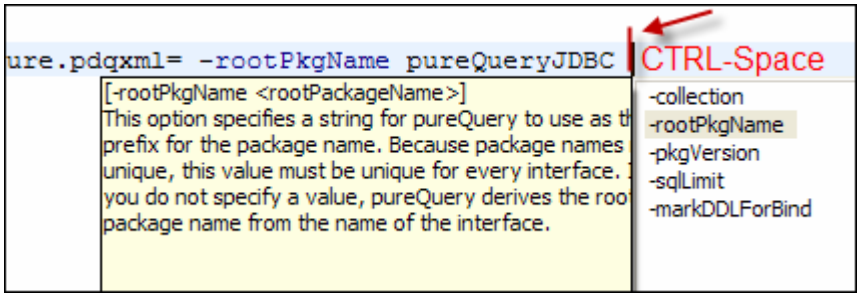
__38. Go and change this name to `pureQueryJDBC` in `Default.genProps` and save the configuration file. A warning will be displayed indicating that the configuration properties have been changed and a rebuild may be necessary. Click <`Yes`>.



__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.

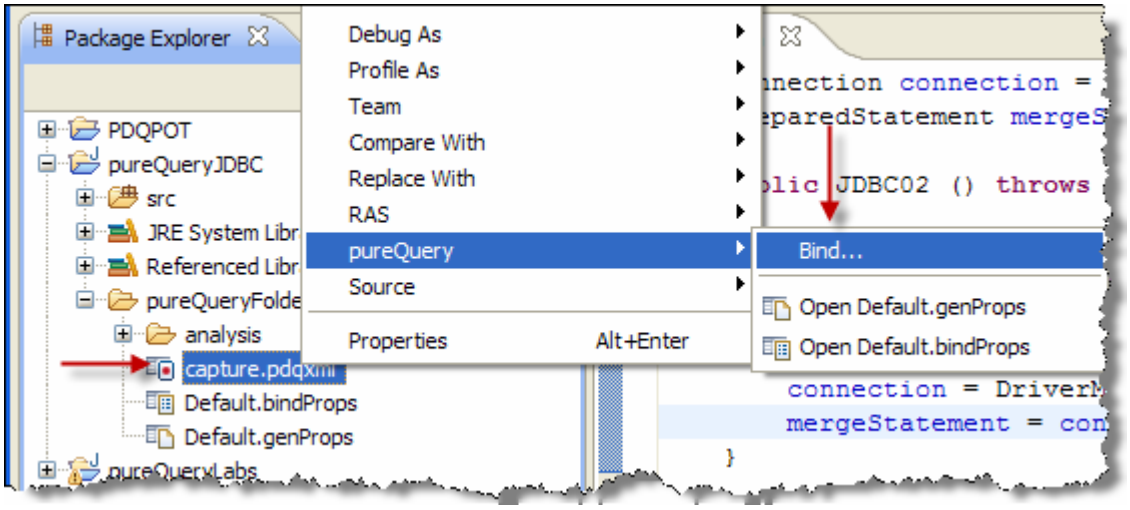
```
defaultOptions= -isolationLevel CS
```

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

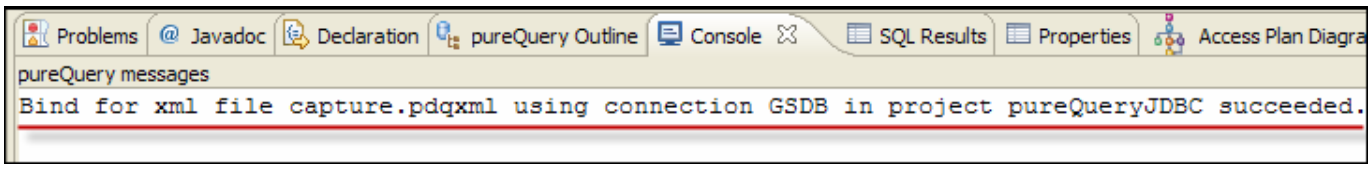
7.3.4 Binding captured SQL statements

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

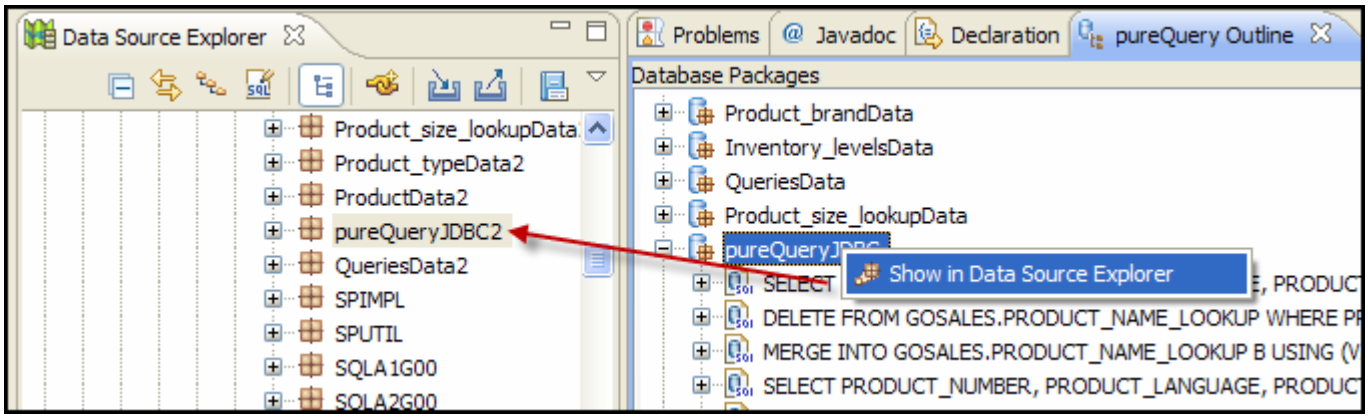
__45. Right click on it and select `pureQuery` ⇨ `Bind` … The bind wizard is displayed prompting for a database connection. Select `GSDB` database to bind the captured SQL statements from `capture.pdqxml` file to DB2 database.



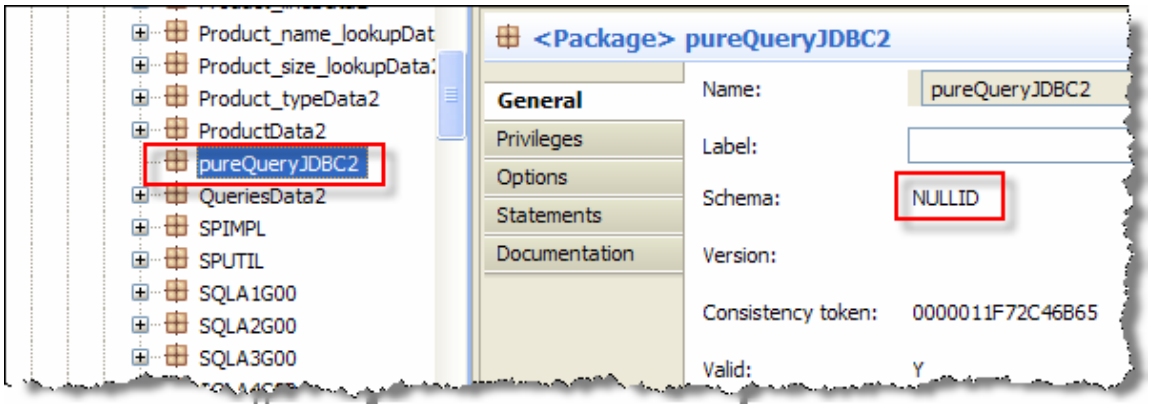
__46. Check console view for the message.



__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`.

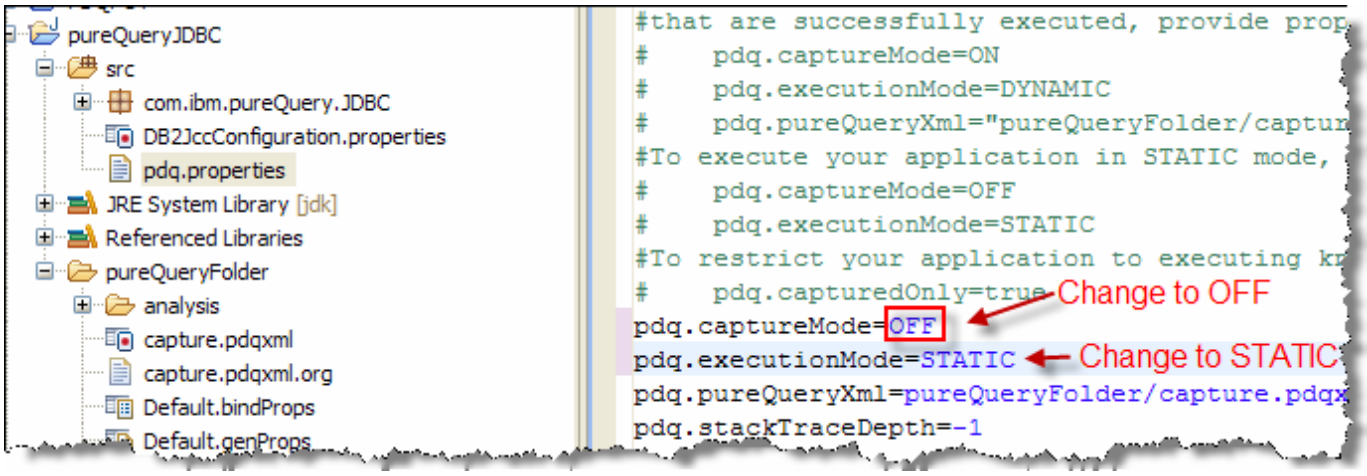
7.3.5 Run Application using static SQL

Let us recap what we have done so far:

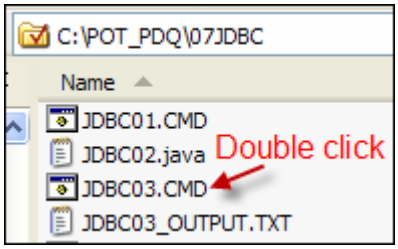
- Enabled the java project for pureQuery JDBC.
- Captured the SQL statements by setting properties in `pdq.properties`
- Browsed the SQL statements and their associated metadata. Configured `Default.genProps` to set the `rootPkgName`.
- Bound the package

__49. Now, you will modify `pdq.properties` to set properties so that the application runs in the static SQL mode. In the *Package Explorer* double click the `pdq.properties` file to open it in an editor.

__50. 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.



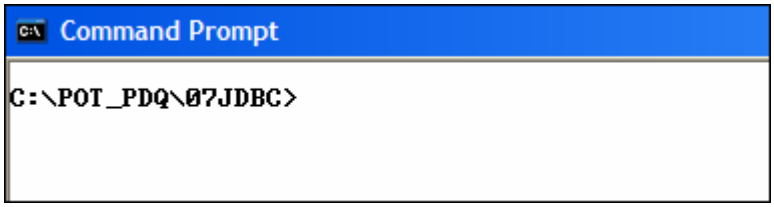
__51. Save the changes and go to the *Windows Explorer* and navigate to `C:\POT_PDQ\07JDBC`. Double click `JDBC03.CMD` to run the application and view the `JDBC03_OUTPUT.TXT` to view the output.



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.

Launch Command Prompt and change directory to `C:\POT_PDQ\07JDBC`



__52. For this lab, 5 administration scripts have been created for you.

JDBC04.JAR	This is our custom application JAR file
JDBC05.CMD	This script runs the custom application as it is
JDBC06.CMD	This script runs the application for different test cases and captures SQL statements and puts them in capture.pdqxml
JDBC07.CMD	This script configures `capture.pdqxml` file for binding purposes.
JDBC08.CMD	This script binds the SQL statements from capture.pdqxml to the database
JDBC09.CMD	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

__53. At your command prompt, run JDBC05 to execute custom JDBC program as shown below.

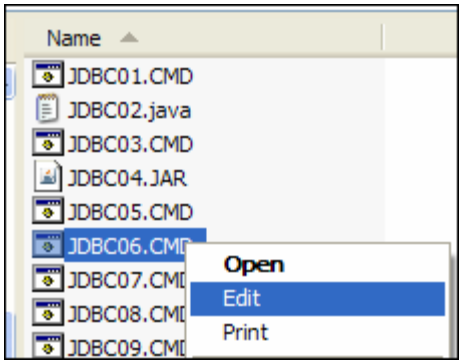
```
C:\POT_PDQ\07JDBC\>JDBC05
```

7.4.2 Capture SQL metadata

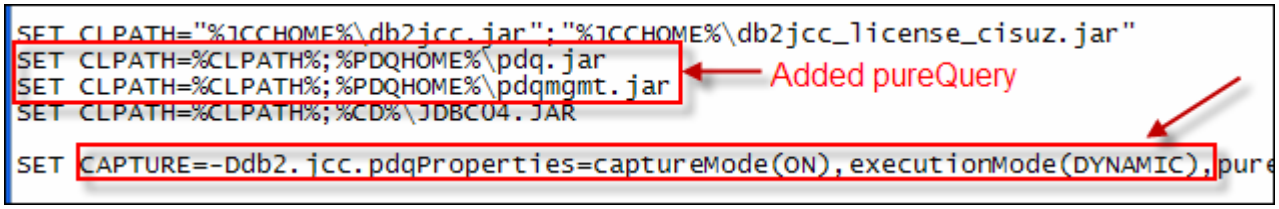
__54. To capture the SQL statements that are being issued by our custom application, we will modify a few runtime environment settings for the application.

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

__55. In your *Windows Explorer*, right click on file `JDBC06.CMD` and click on `Edit` to open this file to review settings changes.



__56. Notice following highlighted changes that were added to capture the SQL statements from the custom JDBC application program.



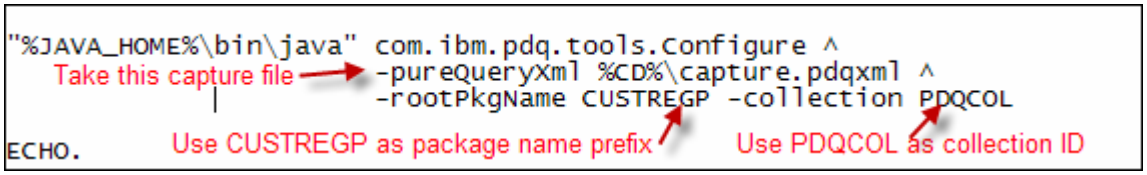
The `JDBC06.CMD` script is very similar to `JDBC05.CMD` that was used in the previous section. The Java classpath was updated to include the required pureQuery runtime JAR files and a db2.jcc.pdqProperties property was passed to the JVM. Through this property, we signaled to the DB2 JCC driver to start capturing the SQL and create a `capture.pdqxml` file to store the metadata. Now, run the script to capture the SQL.

```
C:\POT_PDQ\07JDBC\>JDBC06
```

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

__57. 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.CMD` file and this script invokes `Configure` utility and it assigns a package prefix and a collection ID or schema name for the package.

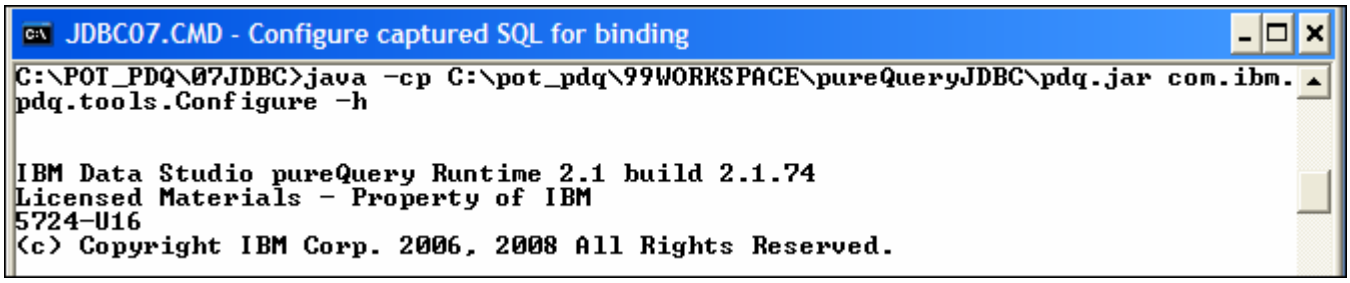


__58. Go ahead and run `JDBC07.CMD` to make changes in the `capture.pdqxml` file.

```
C:\POT_PDQ\07JDBC\>JDBC07
```

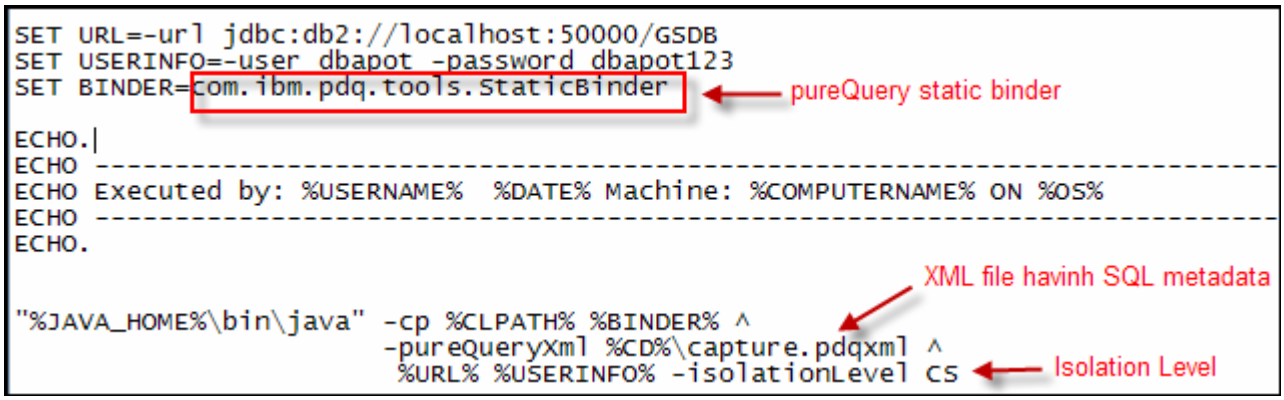
```
IBM Data Studio pureQuery Runtime 2.1 build 2.1.74
Licensed Materials - Property of IBM
5724-U16
(c) Copyright IBM Corp. 2006, 2008 All Rights Reserved.
================================================================================
The Configure utility successfully processed 'C:\POT_PDQ\07JDBC\capture.pdqxml'.

================================================================================
Results of the Configure utility's activity:

    Number of pureQueryXml files for which the configure operation SUCCEEDED: 1
    Number of pureQueryXml files for which the configure operation FAILED: 0
```

__59. The `Configure` command provides many other options and you can see the help by running the following command in your command shell window. (See `JDBC10.CMD` for command)

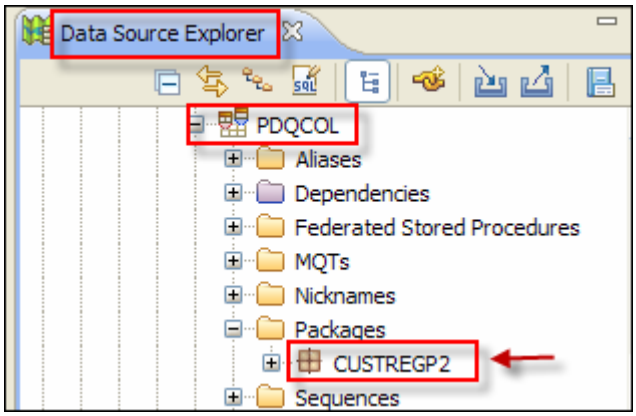


7.4.4 Bind SQL metadata

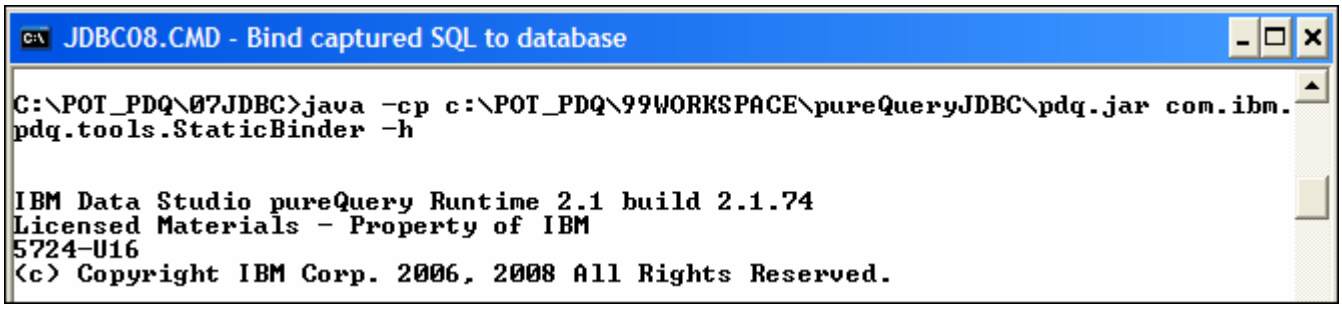
__60. Open `JDBC08.CMD` to view it. The bind utility processes the previously captured and configured metadata and creates one or more packages in the database. You are invoking the `StaticBinder` utility to bind SQL and its metadata from `capture.pdqxml` file.



__61. Go ahead and run `JDBC08.CMD` from the command prompt and you can see the package created through Data Studio.



__62. There are many options available with the `StaticBinder` and you can run following command to see them. (See `JDBC10.CMD` for command)



7.4.5 Run Packaged Application in STATIC SQL mode

__63. Open `JDBC09.CMD` in an editor and review the options to run this custom JDBC application in STATIC mode.

```
SET CAPTURE=-Ddb2.jcc.pdqProperties=captureMode(OFF),
SET CAPTURE=%CAPTURE%executionMode(STATIC),
SET CAPTURE=%CAPTURE%pureQueryXml(%CD%\capture.pdqxml),
SET CAPTURE=%CAPTURE%allowDynamicSQL(true)
```

__64. You will notice that we have specified `captureMode OFF` and `executionMode` has been specified as `STATIC` and `dynamicSQL` are still allowed.

__65. Go ahead and run `JDBC09`.

```
C:\POT_PDQ\07JDBC\>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.

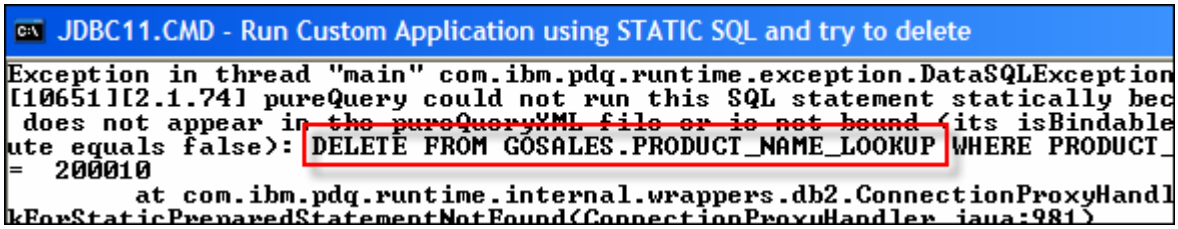
__66. Open JDBC11.CMD in an editor and review the options. Notice, we modified the `allowDynamicSQL` from `true` to `false` and will try to delete one of the product that we registered before.

Remember: We did not capture the DELETE statement.

```
SET CAPTURE=-Ddb2.jcc.pdqProperties=captureMode(OFF),
SET CAPTURE=%CAPTURE%executionMode(STATIC),
SET CAPTURE=%CAPTURE%pureQueryXml(%CD%\capture.pdqxml),
SET CAPTURE=%CAPTURE%allowDynamicSQL(false)
```

__67. Go ahead and run `JDBC11.CMD`.

`C:\POT_PDQ\07JDBC\>``JDBC11`



__68. 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.

__69. 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:

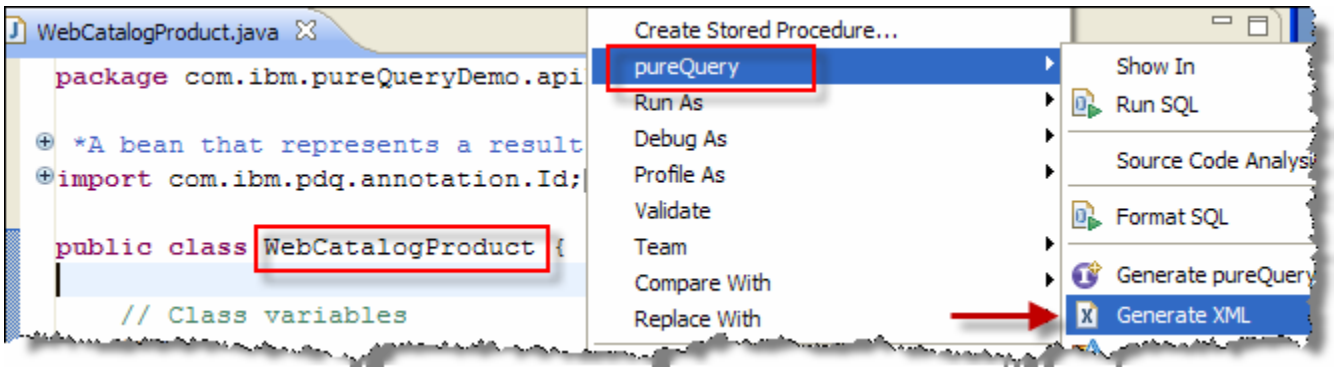
- Generate JPA compliant XML for annotated method SQL statements.
- Custom `ResultHandler` to return a XML data structure.
- Custom `ResultHandler` to map `ResultSet` into HTML output
- Custom `ResultHandler` to populate nested beans.
- Use of `Hook` callback as a built in performance monitor.

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

__2. To generate XML for the `WebCatalogProduct` bean, right click anywhere within `WebCatalogProduct.java` and select `pureQuery` ⇨ `Generate XML`



__3. 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:id name="product_number">
            <orm:column name="PRODUCT_NUMBER"/>
            <orm:generated-value/>
        </orm:id>
        <basic name="product_number">
            <orm:column name="PRODUCT_NUMBER"/>
        </basic>
```

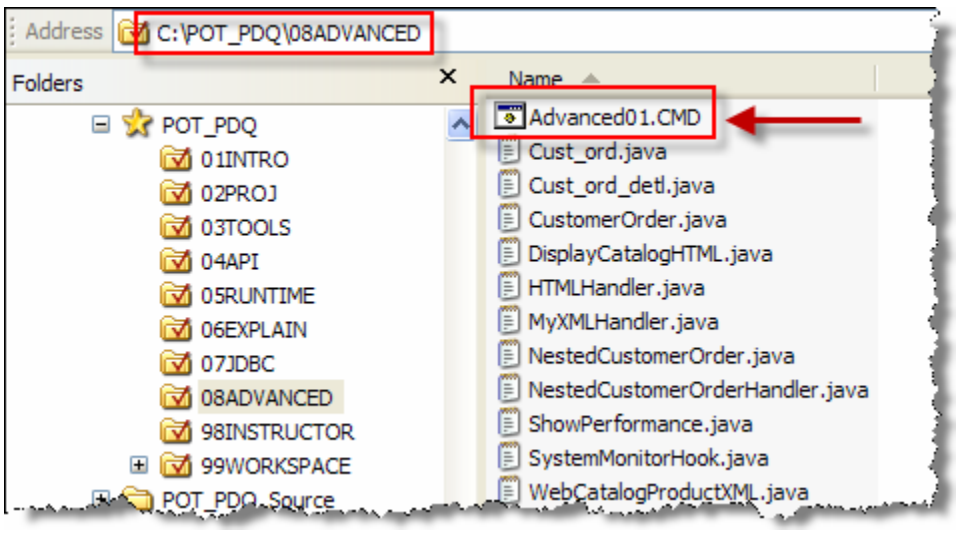
Note: The orm.xml file is created in the `pureQueryFolder` under the `pureQueryLabs` project. Did you know that why we exported attributes of the bean first before we go to the next step?

__4. 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.

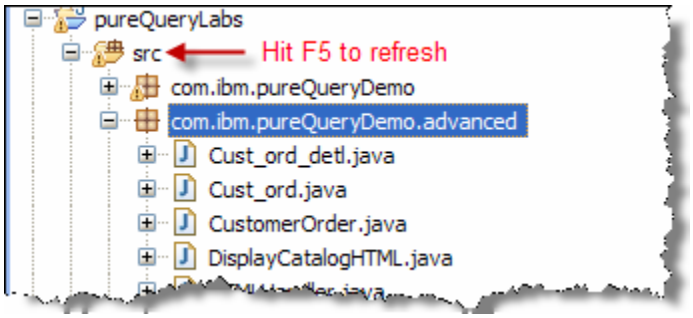
```
    <orm:query><![CDATA[SELECT P.PRODUCT_NUMBER,
</orm:named-native-query><orm:named-native-query
    <orm:query><![CDATA[SELECT P.PRODUCT_NUMBER,
```

8.2 Examples of the `ResultHandler`

__5. Navigate to the `C:\POT_PDQ\08ADVANCED` directory in *Windows Explorer*. Double click on file `ADVANCED01.CMD` to copy java source files from this directory in the Java project `pureQuerylabs`.



__6. Select `src` folder in `pureQueryLabs` project in your package explorer and hit `F5` to refresh the view.

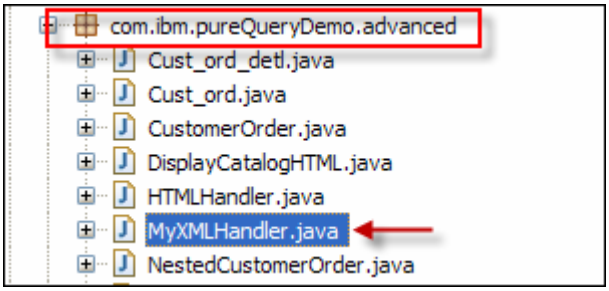


8.2.1 XML handler

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.

__7. Open the `MyXMLHandler.java` class. We will not edit this file.



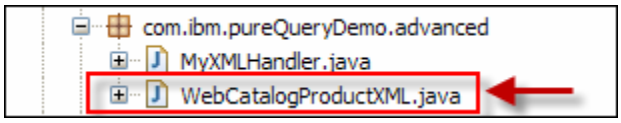
Notice that the `MyXMLHandler` class implements `ResultHandler` of generic type `String`:

```
public class MyXMLHandler implements ResultHandler
```

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:

```
result.append("<" + m.getColumnName(col) + ">");
result.append(rs.getObject(col));
result.append("</" + m.getColumnName(col) + ">");
result.append("\n");
```

__8. 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);
```

__9. 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/NT

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

8.2.2 Nested bean handler

__10. 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.

```
public class CustomerOrder extends Cust_ord {

    protected List<Cust_ord_detl> details;

    public CustomerOrder()
    {
        super();
        details = new ArrayList<Cust_ord_detl>();
    }

    public List<Cust_ord_detl> getDetails() {
        return details;
    }

    public void setDetails(List<Cust_ord_detl> details) {
        this.details = details;
    }

}
```

__11. 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.

__12. 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);

SampleUtil.printClass((CustomerOrder)order.get(0));
```

__13. Right click anywhere in `NestedCustomerOrder.java` and select `Run As ⇨ Java Application`.

__14. You should see results similar to one shown below.

```
Successfully Connected to DB2/NT

CustomerOrder[getDetails=
Cust_ord_detl[getOrd_detl_code=1003, getOrd_nbr=100002, getOrd_ship_date=200
Cust_ord_detl[getOrd_detl_code=1004, getOrd_nbr=100002, getOrd_ship_date=200
Cust_ord_detl[getOrd_detl_code=1005, getOrd_nbr=100002, getOrd_ship_date=200
Cust_ord_detl[getOrd_detl_code=1006, getOrd_nbr=100002, getOrd_ship_date=200
Cust_ord_detl[getOrd_detl_code=1007, getOrd_nbr=100002, getOrd_ship_date=200
```

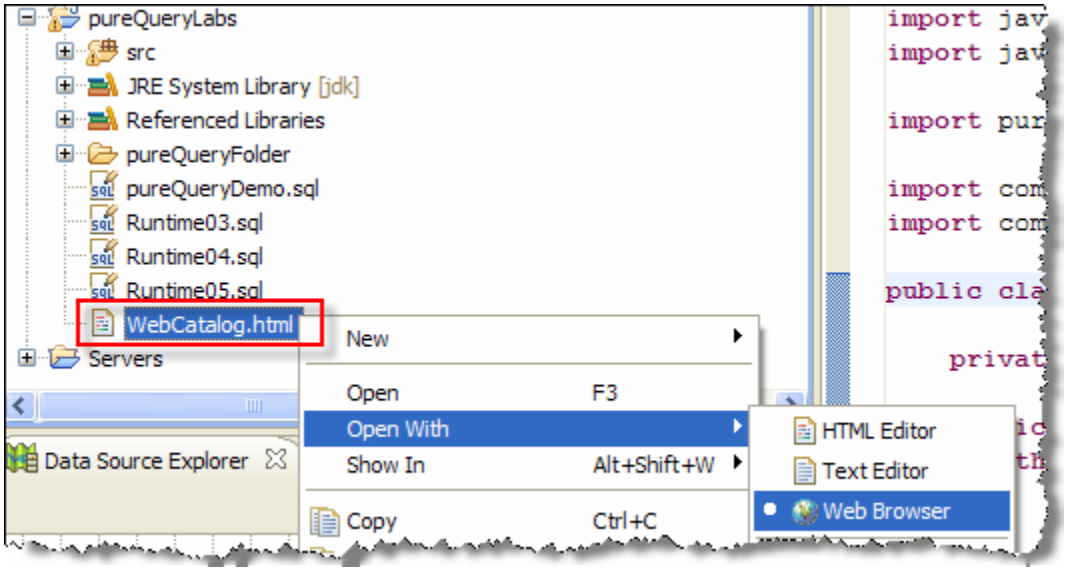
8.2.3 HTML table handler

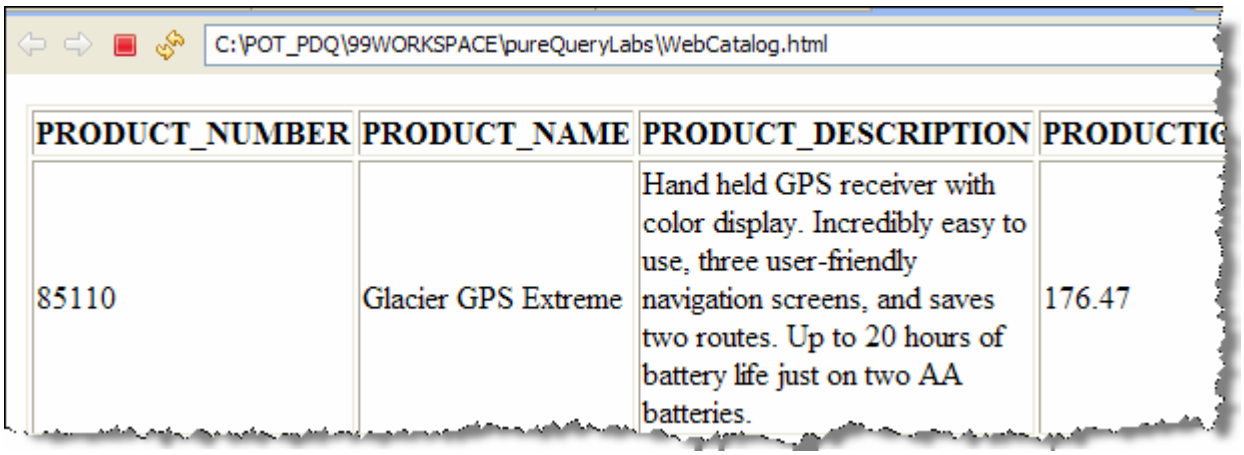
__15. Open the `HTMLHandler.java` class in the editor. It demonstrates the use of custom `ResultHandler` to format the output of a `ResultSet` in an HTML. The handler can be used with nearly any database query to format it into a displayable HTML representation of the query results.

__16. Open `DisplayCatalogHTML.java`. Right click anywhere in `DisplayCatalogHTML.java` and select `Run As ⇨ Java Application`.

```
Successfully Connected to DB2/NT

WebCatalog.html created.
```

__17. 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.





PRODUCT_NUMBER	PRODUCT_NAME	PRODUCT_DESCRIPTION	PRODUCTIC
85110	Glacier GPS Extreme	Hand held GPS receiver with color display. Incredibly easy to use, three user-friendly navigation screens, and saves two routes. Up to 20 hours of battery life just on two AA batteries.	176.47

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.

__18. 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.

__19. 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);
}
```

__20. Right click anywhere ShowPerformance.java and click using the Run As ⇨ Java Application.

__21. You will see an output similar to the one shown below:

```
Successfully Connected to DB2/NT

Performance of method: query(java.lang.String,com.ibm.pdq.runtime.har
Application Time: 212 milliseconds
Core Driver Time: 79669 microseconds
Network Time: 14112 microseconds
server Time: 13572 microseconds
CustomerOrder[getDetails=
Cust_ord_detl[getOrd_detl_code=1003, getOrd_nbr=100002, getOrd_ship_
```

**** 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.