Developing SQL and PL/SQL with JDeveloper

Oracle JDeveloper 10*g* Preview

Technologies used: SQL, PL/SQL

An Oracle JDeveloper Tutorial
September 2003

Content

This tutorial walks through the process of using JDeveloper to develop, tune, and debug the SQL and PL/SQL portions of an application. The tasks in this tutorial are meant to be done in the order provided.

In this session, you will complete the following tasks:

Prerequisites

Software:

- Oracle JDeveloper 10*g* Preview
- Oracle9*i* Release 2 (9.2)

Other Requirements:

Access to database schemas

The common HR schemas provided with the Oracle database is used in this tutorial. If not already done, a DBA user needs to unlock the schema to

provide access. This can be done with the following commands:

```
ALTER USER hr UNLOCK ACCOUNT;

ALTER USER hr IDENTIFIED BY hr;
```

Database privileges

For the PL/SQL debugging portion of this tutorial, the HR user will need to have some more privileges:

```
GRANT debug any procedure, debug connect session TO hr;
```

Create a Connection and Browse the Database

JDeveloper allows you to store the information necessary to connect to a database in an object called a Connection. A connection is stored as part of the IDE settings, and can be exported and imported for easy sharing among groups of users. A connection serves several purposes from browsing the database, building applications, all the way through to deployment.

Create a Connection

In this lab, you will create a database connection to browse and modify database objects.

To create a new connection:

1. Go the Connection Navigator, either by clicking the Connection tab next to the Application tab, or by choosing **View > Connection Navigator** from the main menu.
2. In the Connection Navigator, select the Database node.
3. Right click on the Database node and choose **New Database Connection** from the context menu.

To provide the connection information:

1. In the first page of the Connection wizard after the Welcome page, enter `MyHRConn` as the name for your new connection.
2. The **Connection Type** drop-down list allows you to specify the type of database or database driver that you use to connect to your database. In this case, you will connect to an Oracle database, so make sure `Oracle (JDBC)` (Default) is selected.
3. Click **Next**.
4. In Step 2 of 4 of the wizard, enter the username and password for the database user you want to connect as.

- Username: `hr`
- Password: `hr`

Note: You can enter the username and password on one line using the <username>/<password> syntax, for example, `hr/hr`.

5. Click **Next**.
6. In Step 3 of the wizard, specify the details about the location of the database you want to connect to, as well as the type of driver you want to use.
   - Driver: `thin`
   - Host Name: `localhost` (replace with the appropriate host name in case of a remote database)
   - JDBC Port: `1521` (replace with the appropriate SQL*Net listener port)
   - SID: `ORCL` (replace with the appropriate database SID)
7. Click **Next**.
8. Step 4 of the wizard allows you to test your connection. Click **Test Connection**.

   If there are any errors reported with your connection, use the **Back** button to go to the appropriate page in the wizard where you can fix the problem.

9. Once you have verified that your connection tests successfully, click **Finish**.

Navigate the contents of the database

One of the uses of a connection in JDeveloper is for browsing and editing database objects. In this section of the lab, you will use the connection you just created to browse the contents of the database.

To explore the objects in the database:

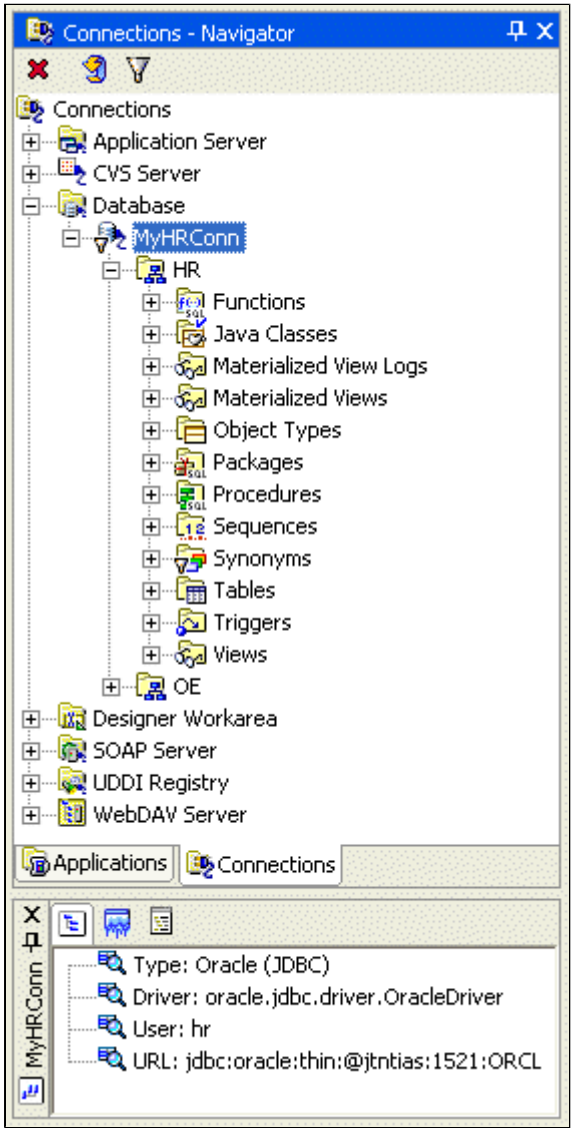
1. In the Connection Navigator, expand the MyHRConn node.

   By default, this displays only the database objects owned by the current user.

To access other database schemas:

1. Select the MyHRConn node.
2. Right click and choose **Apply Filter** from the context menu.
3. Select OE from the **Available Schemas** list and move it to the **Selected Schemas** list.
4. Click **OK**.

You should now see both the HR and OE schemas.



To filter the list of database objects:

When working with many objects in the navigator, it may be difficult to find the node you are looking for. JDeveloper provides filters at every level of a database connection to limit the number of objects you need to look through. You've already seen one example of a filter when you selected which schemas you wanted to view.

1. Expand the HR node in the Connection Navigator (all of your work in this lab from this point on will be in the HR schema).
2. Expand the Synonyms node.

   There is a filter set up on the Synonyms node to only show your private synonyms by default.

3. Right click on the Synonyms node and choose **Apply Filter** from the context menu.
4. Enter USER% in the **Filter Text** field.
5. Check the **Show Public Synonyms** checkbox.
6. Click **OK**.

   You should now be able to see all the public synonyms whose names begin with USER.

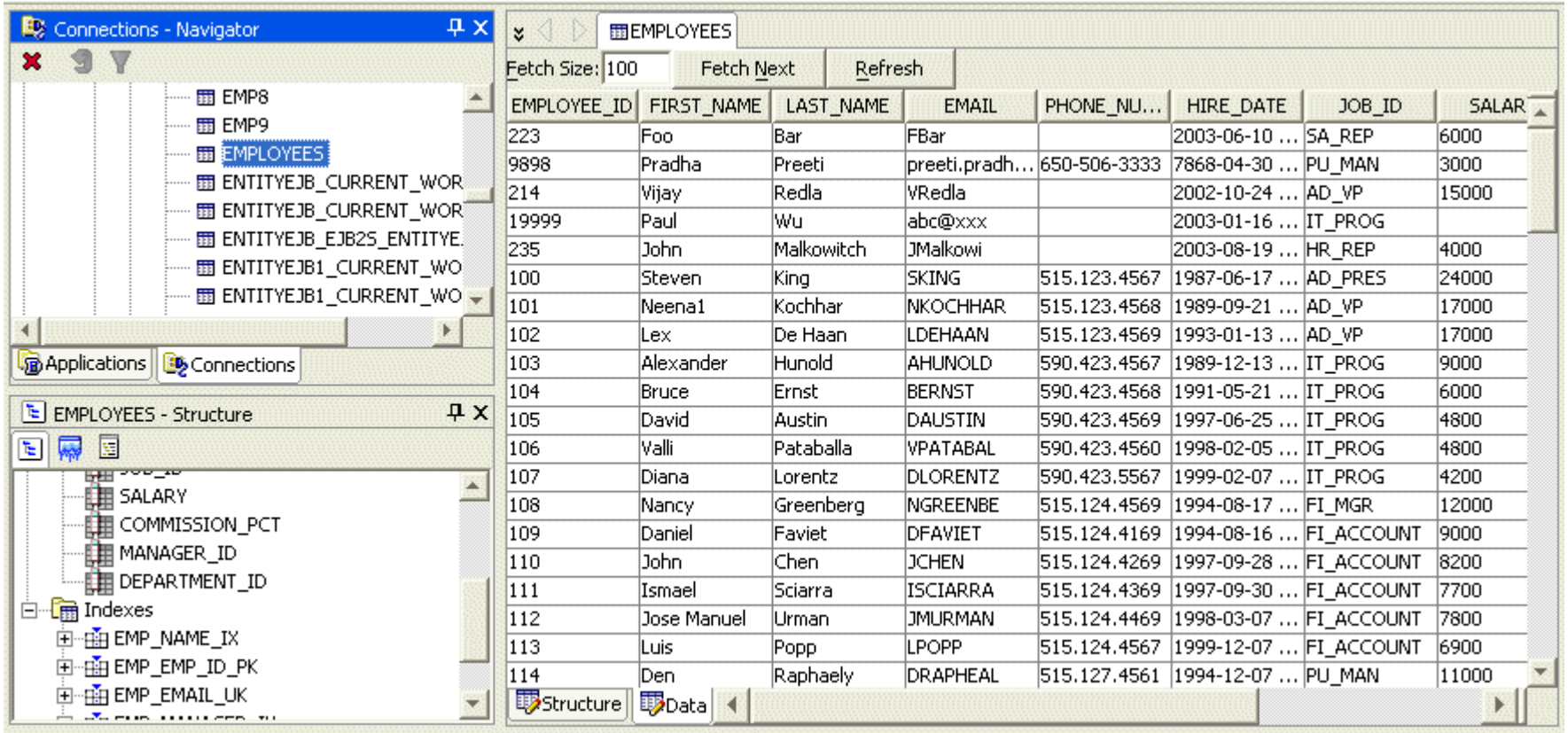
Note: There's another way to find what you are looking for in the Navigator. With the Navigator in focus, you can simply begin typing the name of the node you are looking for. JDeveloper will automatically navigate you to the first node it finds starting with the text you've typed in.

View a Table and its data

To browse a table:

1. Collapse the Synonyms node.
2. Expand the Tables node.
3. Double-click the **EMPLOYEES** table to open the table in the Table Viewer.

   The Table Viewer shows information about the table, including column names and datatypes, primary keys, and not null constraints. Notice as well that the Structure window shows information about indexes on the selected table.



To browse the table data:

1. Click the **Data** tab in the Table Viewer.

   By default, the Table Viewer fetches 100 rows at a time.

2. Change the **Fetch Size** to `10`.
3. Click **Refresh** to re-execute the query. This time you will only see the first ten rows.
4. Click **Fetch Next** to retrieve the next 10 rows.

Create and Compile PL/SQL

In this section, you will create, edit, compile, and test a PL/SQL procedure. Later you will tune a SQL statement embedded in the PL/SQL code debug the procedure.

Create and Compile a PL/SQL Procedure

To create a PL/SQL procedure:

1. Right-click on the Procedures node in the System Navigator and choose **New PL/SQL Procedure**.
2. Enter `emp_list` as the package name.
3. Click **OK**.

   This will create a skeleton procedure.

To edit the PL/SQL procedure:

1. There are many features in the PL/SQL editor, but this tutorial does not try to explore all of them. Instead, simply copy and paste the following code into the PL/SQL editor:

```
PROCEDURE EMP_LIST
  (pMaxRows NUMBER)
AS
  CURSOR emp_cursor IS
    SELECT l.state_province, l.country_id, d.department_name, e.last_name,
      j.job_title, e.salary, e.commission_pct
    FROM locations l, departments d, employees e, jobs j
    WHERE l.location_id = d.location_id
    AND d.department_id = e.department_id
    AND e.job_id = j.job_id;
  emp_record emp_cursor%ROWTYPE;
  TYPE emp_tab_type IS TABLE OF emp_cursor%ROWTYPE INDEX BY BINARY_INTEGER;
  emp_tab emp_tab_type;
  i NUMBER := 1;
  JavaSPReturn VARCHAR2(50);
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO emp_record;
  emp_tab(i) := emp_record;
  -- add Java stored procedure call here --
```

```
  WHILE ((emp_cursor%FOUND) AND (i <= pMaxRows) LOOP
    i := i + 1;
    FETCH emp_cursor INTO emp_record;
    emp_tab(i) := emp_record;
  END LOOP;
  CLOSE emp_cursor;
  FOR j IN REVERSE 1..i LOOP
    DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE(JavaSPReturn);
END;
```

Note: This actually contains a syntax error that you will discover in the next section.

To find the syntax error:

There are several ways to detect the syntax error in your sample code:

1. Expand the Errors folder in the Structure window. You can then navigate to the detected error by simply double-clicking on the error.
2. Place your cursor next to one of the parentheses in the WHILE statement. JDeveloper will highlight the matching symbol for the parenthesis at the cursor. If you place your cursor next to the first parenthesis in the statement, you will notice that it is highlighted in red which indicates that it does not have a matching symbol.
3. Compile the PL/SQL subprogram by clicking the **Save** button in the toolbar. Compilation errors are shown in the log window. You can navigate to the line reported in the error by simply double-clicking on the error. Note that when an invalid PL/SQL subprogram is detected by JDeveloper, the status is indicated with a red `X` over the icon for the subprogram in the System Navigator.

To fix the syntax error:

1. Add the missing `)` at the end of the WHILE statement just after pMaxRows and before the LOOP keyword
2. Click the **Save** button in the toolbar.

   You should see a message in the status bar indicating a `Successful compilation`.

Run a PL/SQL Procedure

To test the PL/SQL procedure:

1. With the procedure selected, choose **Run > Run EMP_LIST** from the main menu.

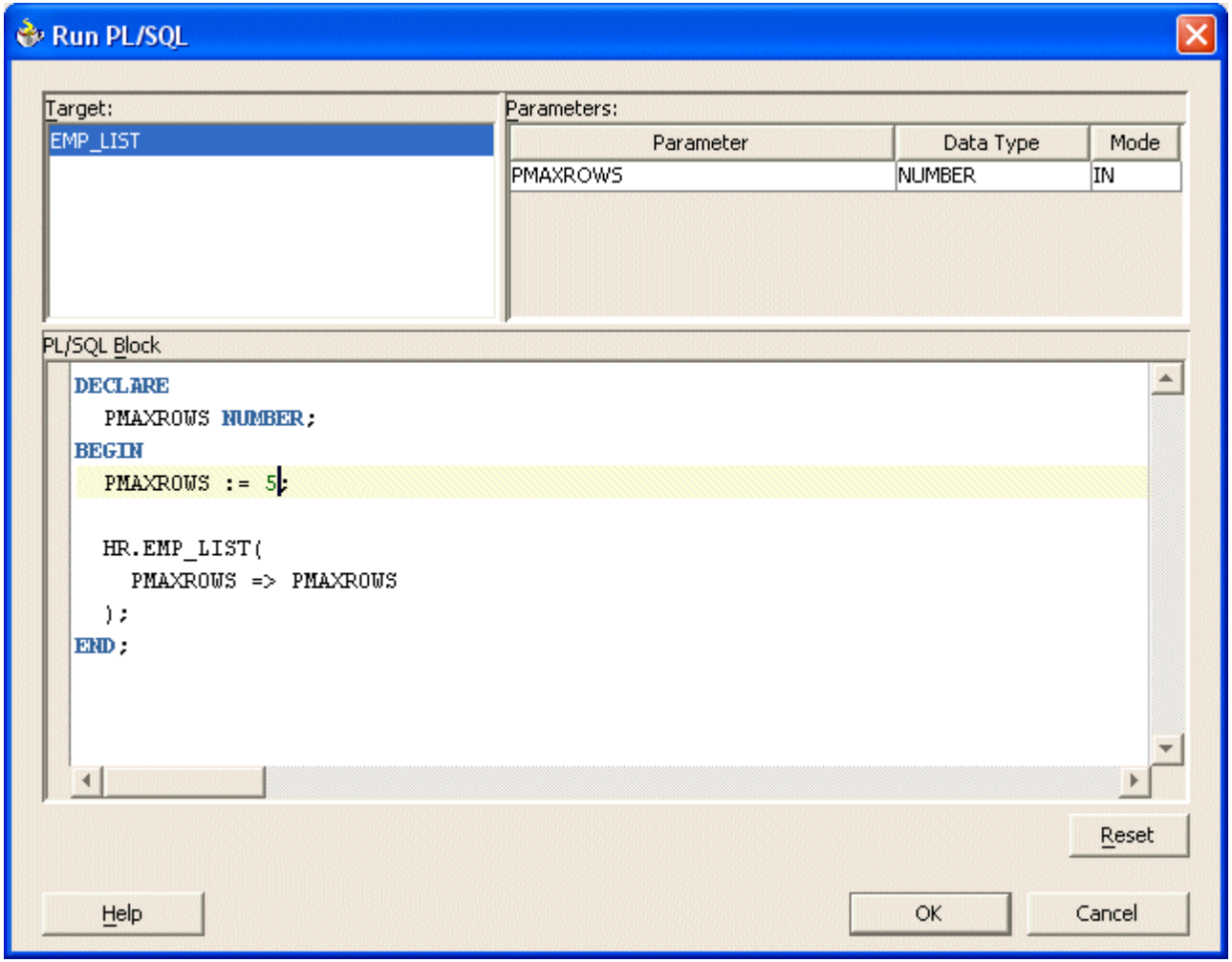
   This invokes the Run PL/SQL dialog. The Run PL/SQL dialog allows you to select the target procedure or function to run (useful for packages) and displays a list of parameters for the selected target. In the PL/SQL block text area is some generated code that JDeveloper will use to call the selected program. Use this area to populate parameters to be passed to the program unit and to handle complex return types.

2. In the PL/SQL Block replace

   ```
   PMAXROWS := NULL;
   ```

   with

   ```
   PMAXROWS := 5;
   ```



3. Click **OK**.
4. You should see the results of the 5 rows returned in the Log window.

Execute and Tune SQL Statements

So far in this tutorial, you have explored some of the capabilities in JDeveloper for creating and browsing database objects. In this section, you will use JDeveloper's SQL Worksheet to enter, execute, and tune ad-hoc SQL statements.

Execute a Statement

To execute a SQL statement:

1. Right-click on the MyHRConn node in the System Navigator and choose **SQL Worksheet**.
2. Enter a simple statement, for example:

```
SELECT *
FROM employees;
```

3. Click the **Execute Statement** button.

Get the Explain Plan for a Statement

To get the explain plan for a statement:

1. Delete all of the contents of the SQL Worksheet.
2. If the EMP_LIST source is no longer visible, double-click the EMP_LIST node to open the procedure in the code editor.
3. Highlight the SELECT statement from the cursor declaration and copy it to the clipboard (Ctrl+C).

```
SELECT l.state_province, l.country_id, d.department_name, e.last_name,
j.job_title, e.salary, e.commission_pct
FROM locations l, departments d, employees e, jobs j
WHERE l.location_id = d.location_id
AND d.department_id = e.department_id
AND e.job_id = j.job_id
```

4. Paste (Ctrl+V) the SELECT statement into the SQL Worksheet.
5. Click the **Execute Statement** button.

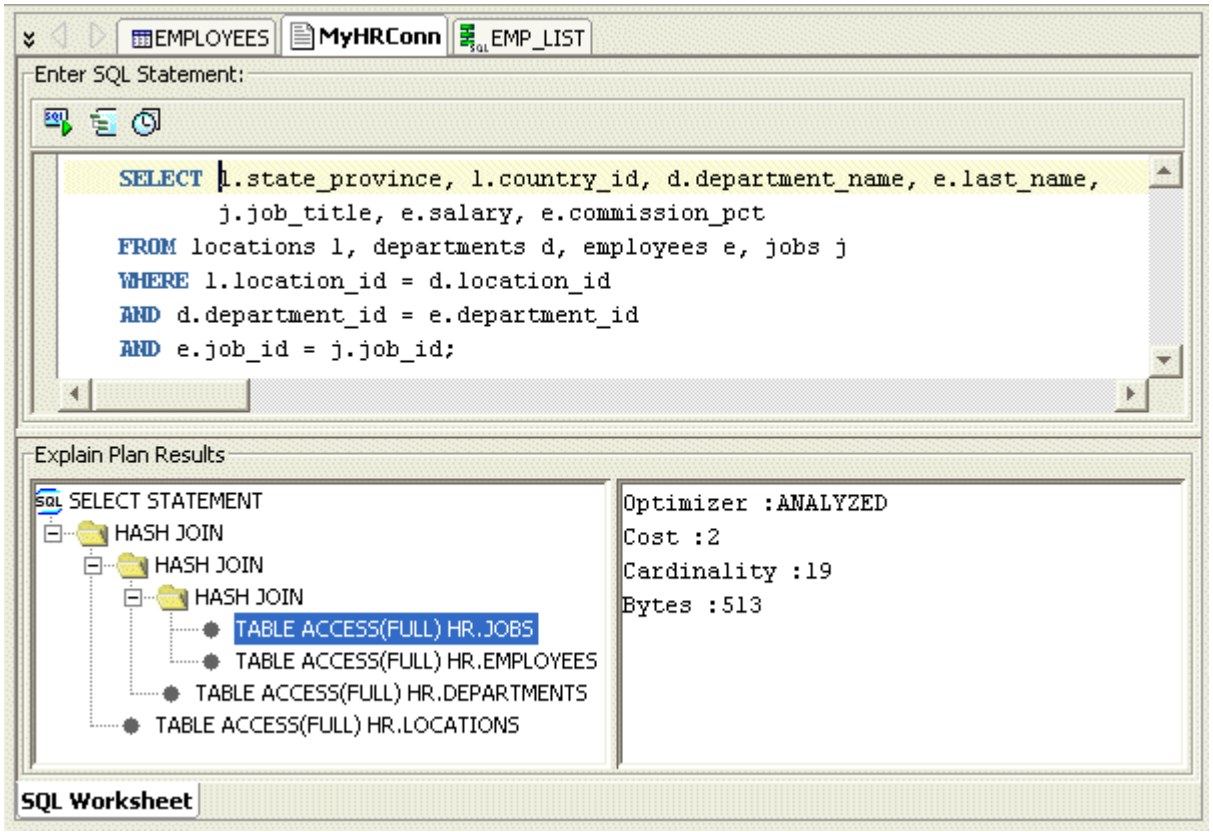
   You should see the results of the statement.

6. Click the **Explain Plan** button.

   Note: If you do not have a PLAN_TABLE in the current schema, you will be prompted to create it. In this case, simply click **OK** to create the PLAN_TABLE.

You should see the results of the SQL explain plan.

By default, the system will likely be performing a full table scan on all the tables as shown in the Explain Plan Results section in the following illustration.



Note: If your results are significantly different from those shown below, it may be because the schema has not been analyzed. To analyze the schema, enter and execute the following in the SQL Worksheet, then try again:

```
BEGIN
  DBMS_UTILITY.ANALYZE_SCHEMA('HR', 'COMPUTE');
END;
```

Tune the Statement

To tune the statement:

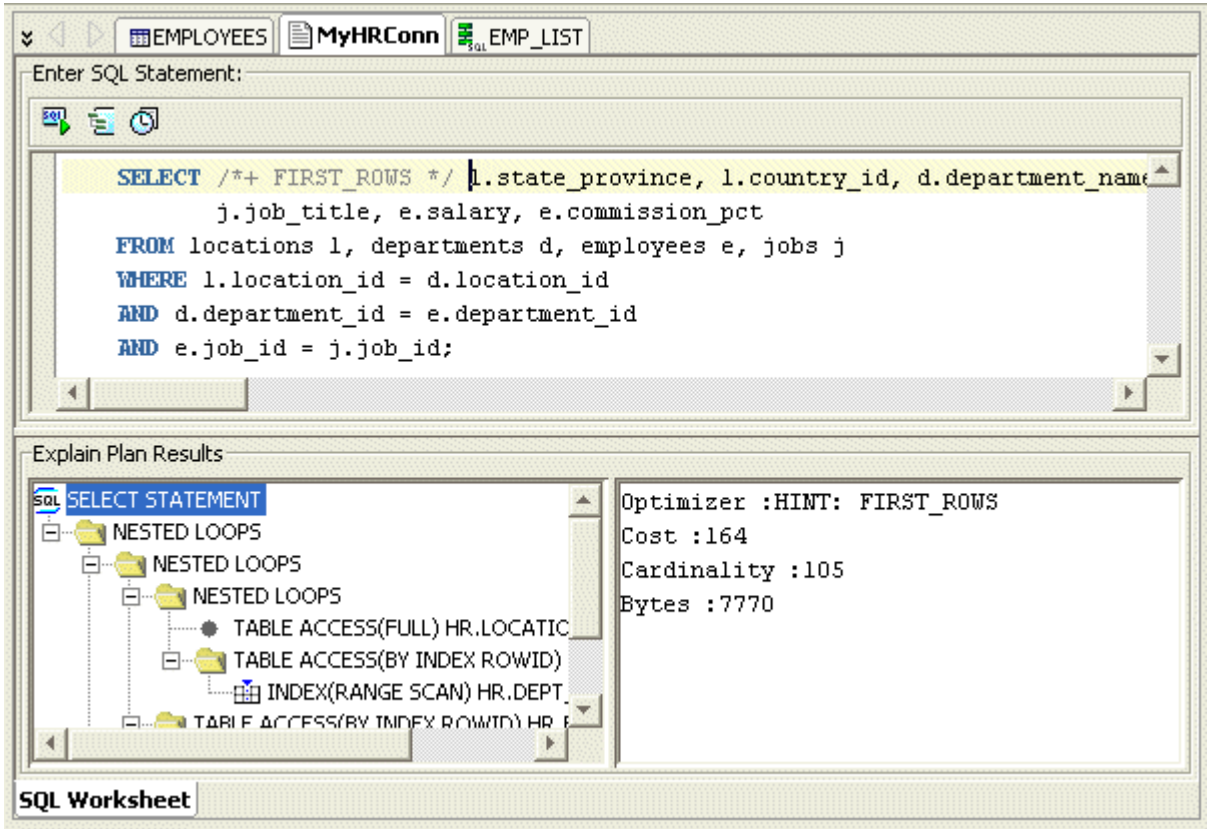
1. Edit the SQL statement to include the optimizer hint FIRST_ROWS. In this case, you should see a visible difference in the explain plan by requesting the server to tune to retrieve the first set of rows as quickly as possible.

   After inserting the FIRST_ROWS optimizer hint, the query should appear as follows:

   ```
   SELECT /*+ FIRST_ROWS */ l.state_province, l.country_id, d.department_name, e.last_name,
   j.job_title, e.salary, e.commission_pct
   FROM locations l, departments d, employees e, jobs j
   WHERE l.location_id = d.location_id
   AND d.department_id = e.department_id
   AND e.job_id = j.job_id
   ```

2. Click the **Explain Plan** button.

   You should now see a different explain plan that uses one or more indexes for data retrieval.



Create and Deploy a Java Stored Procedure

JDeveloper facilitates working with Java stored procedures by simplifying deployment of Java stored procedures and by allowing debugging of Java stored procedures. In this portion of the lab, you will create a new Java class as the basis for the Java stored procedure, then deploy it to the database. In the next section you will debug it.

Create a Java Stored Procedure

To create a new Application Workspace:

1. In the Application Navigator, right click on the Applications node and choose **New Application Workspace** from the context menu.
2. Enter `DBApplication` as the Application name.

3. Select **Custom Application [All Technologies]** as the Application template.
4. Click **OK**.

To create a new Java Class for the Java stored procedure:

1. Select the new **Project** node in the Application Navigator.
2. Choose **File > New** from the main menu.
3. Select the **General** category on the left side, and **Java Class** from the list of items on the right.
4. Click **OK**.
5. Enter `JavaStoredProc` as the name of the new Java class.
6. Click **OK**.

To write the code for the Java stored procedure:

1. You could write pretty much any Java code with a public static method for your Java stored procedure. In this case, you may just want to copy and paste the following:

```
public class JavaStoredProc
{

  public JavaStoredProc()
  {
  }

  public static String getHelloFromJava ()
  {
    String _string = new String();
    for (int i = 0; i < 3 ; i++)
    {
      _string = _string + "Hello World ";
    }
  return _string;
  }
}
```

2. Choose **File > Save All** from the main menu to save your work.

Deploy a Java Stored Procedure

To create a new deployment profile:

1. Select the Project node in the Application Navigator.
2. Choose **Project > Make Project.jpr** from the main menu.
3. Choose **File > New** from the main menu.
4. Select the Deployment Profiles category.
5. Select the **Loadjava and Java Stored Procedures** profile and click OK.
6. Enter `MyJavaSPProfile.deploy` as the profile name and click Save.
7. In the Source Files list, make sure only JavaStoredProc.java is selected.
8. Click **OK**.

To create a new PL/SQL wrapper definition:

1. Select MyJavaSPProfile.deploy in the Application Navigator.
2. Right click and choose **Add Stored Procedure** from the context menu.
3. Select the getHelloFromJava method.
4. Click **OK**.

To deploy the Java stored procedure:

1. Choose **File > Save All** from the main menu.
2. Select MyJavaSPProfile.deploy in the Application Navigator.
3. Right click and choose **Deploy to > MyHRConn** from the context menu.

   Verify that the deployment completed successfully. You should see the following in the Log window:

```
Invoking loadjava on connection 'MyHRConn' with arguments:
-order -resolve -thin
Loadjava finished.
Executing SQL Statement:
CREATE OR REPLACE FUNCTION getHelloFromJava RETURN VARCHAR2 AUTHID CURRENT_USER AS LANGUAGE JAVA NAME
'JavaStoredProc.getHelloFromJava() return java.lang.String';
Success.
Publishing finished.
---- Stored procedure deployment finished. ----
```

To test the Java stored procedure:

1. Expand the Functions node in the Application Navigator.
2. Right click on the GETHELLOFROMJAVA node and choose **Run GETHELLOFROMJAVA** from the context menu.
3. Click **OK**.
4. Confirm that the output `Hello World Hello World Hello World` appears in the Log window.

Debug a PL/SQL Subprogram and Java Stored Procedure

JDeveloper also supports PL/SQL debugging with Oracle8*i* and Oracle9*i* databases. (Debugging Java stored procedures is available only with Oracle9*i* Release 2 and later.)

Debug a PL/SQL Procedure

In this case, we're going to debug the PL/SQL code calling the Java stored procedure. To accomplish this, the first step is to modify the PL/SQL code to actually call the Java stored procedure.

To modify the PL/SQL to call the Java stored procedure:

1. Look for the line of code in the EMP_LIST procedure:

   ```
   -- add Java stored procedure call here --
   ```

   Just below this line, add the following:

   ```
   JavaSPReturn := getHelloFromJava;
   ```

2. Click the **Save** button in the toolbar.

To debug a PL/SQL procedure:

1. Set a breakpoint in the EMP_LIST procedure by clicking in the margin at the line with the `OPEN emp_cursor;` statement:

```
PROCEDURE EMP_LIST
  (pMaxRows  NUMBER)
AS
  CURSOR emp_cursor IS
    SELECT l.state_province, l.country_id, d.department_name, e.last_name,
           j.job_title, e.salary, e.commission_pct
    FROM locations l, departments d, employees e, jobs j
    WHERE l.location_id = d.location_id
    AND d.department_id = e.department_id
    AND e.job_id = j.job_id;
  emp_record emp_cursor%ROWTYPE;
  TYPE emp_tab_type IS TABLE OF emp_cursor%ROWTYPE INDEX BY BINARY_INTEGER;
  emp_tab emp_tab_type;
  i NUMBER := 1;
  JavaSPReturn VARCHAR2(50);
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO emp_record;
  emp_tab(i) := emp_record;
  -- add Java stored procedure call here --
  JavaSPReturn := getHelloFromJava;
  WHILE ((emp_cursor%FOUND) AND (i <= pMaxRows)) LOOP
    i := i + 1;
```

2. Right-click on the EMP_LIST node in the Connection Navigator and choose **Debug EMP_LIST**.

   The values you passed as parameters before should reappear in the generated PL/SQL Block.

3. Click **OK**.

   The debugger should halt at the line where you placed the breakpoint. You can now control the flow of execution, modify values of variables and perform other debugging functions.

```
  JavaSPReturn VARCHAR2(50);
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO emp_record;
  emp_tab(i) := emp_record;
```

To step through code:

1. Click **Step Into** .

   You should now be in the emp_cursor cursor declaration.

```
AS
  CURSOR emp_cursor IS
    SELECT l.state_province, l.country_id, d.department_name,
           j.job_title, e.salary, e.commission_pct
   FROM locations l, departments d, employees e, jobs j
   WHERE l.location_id = d.location_id
   AND d.department_id = e.department_id
```

2. Click **Step Into** .

The Smart Data window shows a limited list of variables, namely those used in the line of code that is about to be executed, and in the previously executed line.

1. Click the Data tab to see all the variables that are in scope.

Name	Value	Type
PMAXROWS	5	NUMBER
EMP_RECORD		Rowtype
EMP_TAB	indexed table	EMP_TAB_TYPE
I	1	NUMBER
JAVASPRETURN	NULL	VARCHAR2(50)

Smart Data | Data | Watches

2. Click **Step Into** twice more.

   You should now be about to execute the statement `JavaSPResult := getHelloFromJava;`

```
emp_tab(i) := emp_record;
-- add Java stored procedure call here --
JavaSPReturn := getHelloFromJava;
WHILE ((emp_cursor%FOUND) AND (i <= pMaxRows)) LOOP
  i := i + 1;
```

   Note: getHelloFromJava is the PL/SQL wrapper for the Java stored procedure you created earlier.

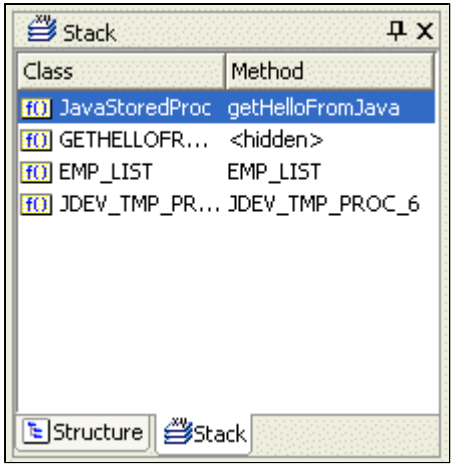
3. Click **Step Into**.

   You should now be debugging the Java code for the Java stored procedure.

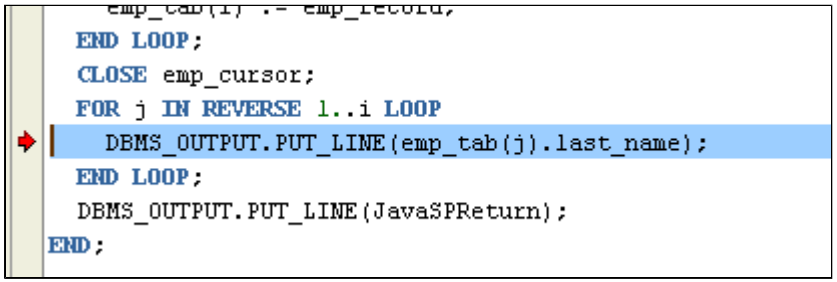
   Note: If you get a message about not being able to locate the source of the Java stored procedure, select the option to look in a project, and select Project from the list.

```
public static String getHelloFromJava ()
{
  String _string = new String();
  for (int i = 0; i < 3 ; i++)
  {
    _string = _string + "Hello World ";
```

   The Stack window shows the PL/SQL call stack and the Java call stack together.



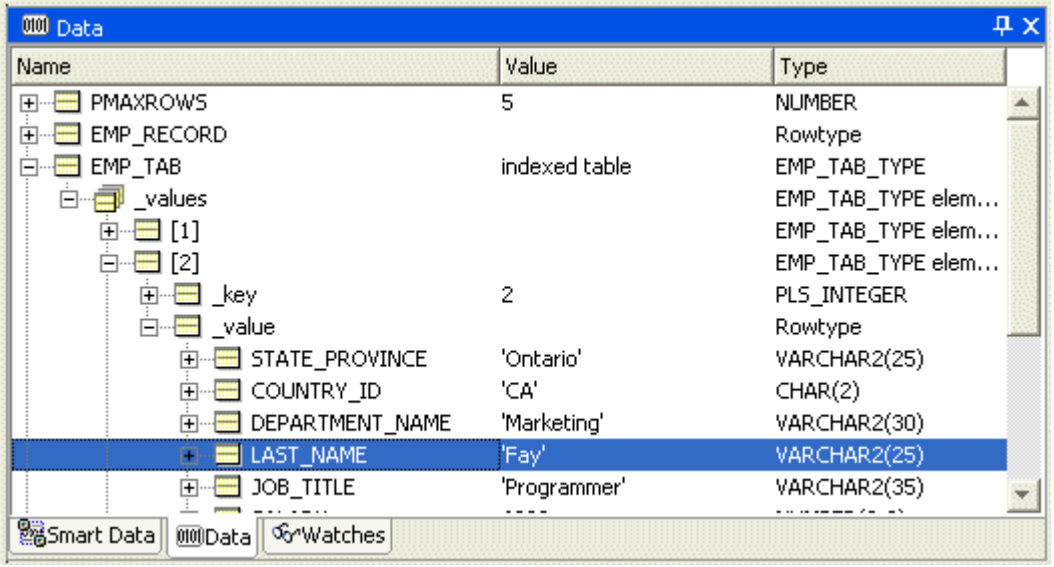
4. Click **Step Out** to complete the Java stored procedure and return to the NEW_EMP procedure.
5. Right click on the line that reads `DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);` and choose **Run to Cursor** from the context menu.



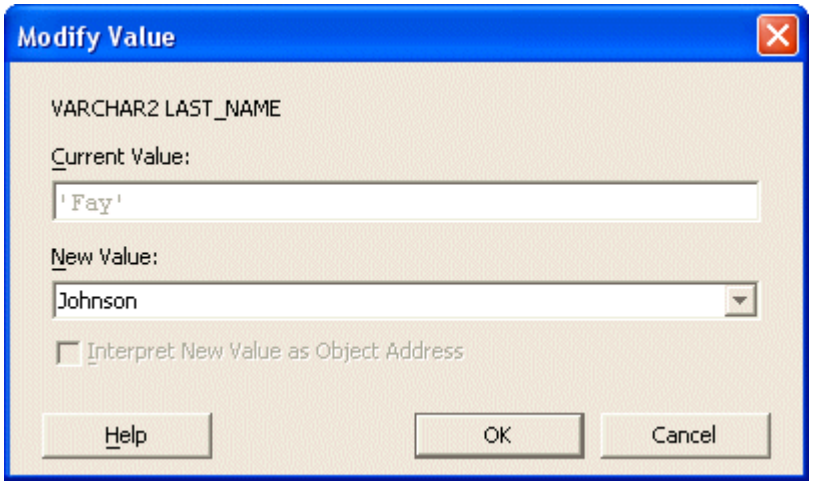
6. Use the data window to drill into the PL/SQL table of records called emp_tab.

   In the data window you will find that you can access the entire structure of composite datatypes.

7. Keep drilling down until you see the values of the fields in a given record of the table.



8. Right click on the LAST_NAME field of the record and choose **Modify Value** from the context menu.
9. Modify the last name to a name of your choice.



10. Click **OK**.
11. You've now modified the value in the PL/SQL table of records on the fly.

12. Click **Resume** to allow the PL/SQL to run to completion.
13. Check to see that your modified value is displayed in the Log window.

```
Executing PL/SQL: CALL HR."JDEV_TMP_PROC_6"()
Tobias
Himuro
Colmenares
Hartstein
Johnson
Whalen
Hello World Hello World Hello World
Debugger disconnected from database.
Process exited.
```

End of Tutorial

This concludes the tutorial for SQL and PL/SQL development in Oracle JDeveloper.

For more information, refer to the JDeveloper product page on OTN:
/products/jdev