

# *PL/SQL Developer 7.1*

## *User's Guide*

*March 2007*



**allround**automations



# Contents

<b>CONTENTS .....</b>	<b>3</b>
<b>1. INTRODUCTION.....</b>	<b>9</b>
<b>2. INSTALLATION.....</b>	<b>13</b>
2.1 SYSTEM REQUIREMENTS.....	13
2.2 WORKSTATION INSTALLATION.....	13
2.3 SERVER BASED INSTALLATION .....	13
2.4 SCRIPTED INSTALLATION.....	14
2.5 UNINSTALLING PL/SQL DEVELOPER.....	14
<b>3. WRITING PROGRAMS .....</b>	<b>15</b>
3.1 CREATING A PROGRAM .....	15
3.2 SAVING A PROGRAM .....	16
3.3 MODIFYING A PROGRAM.....	17
3.4 COMPILING A PROGRAM.....	17
3.5 SUBSTITUTION VARIABLES .....	18
<b>4. TESTING PROGRAMS .....</b>	<b>19</b>
4.1 CREATING A TEST SCRIPT.....	19
4.2 EXECUTING A TEST SCRIPT.....	20
4.3 VARIABLE TYPES.....	21
4.4 SAVING TEST SCRIPTS.....	22
4.5 TRACKING RUN-TIME ERRORS.....	23
4.6 PACKAGE STATES & JAVA SESSION STATES .....	23
4.7 VIEWING RESULT SETS .....	23
4.8 VIEWING DBMS_OUTPUT .....	23
4.9 VIEWING HTTP OUTPUT.....	24
4.10 DEBUGGING.....	24
4.11 TRACING EXECUTION.....	27
4.12 REGRESSION TESTING .....	28
<b>5. OPTIMIZING.....</b>	<b>29</b>
5.1 USING THE EXPLAIN PLAN WINDOW.....	29
5.2 AUTOMATIC STATISTICS.....	30
5.3 THE PL/SQL PROFILER.....	31
5.4 SQL TRACE .....	32
<b>6. AD HOC SQL.....</b>	<b>34</b>
6.1 USING THE SQL WINDOW .....	34
6.2 RESULT GRID MANIPULATION .....	35
6.3 QUERY BY EXAMPLE MODE.....	39
6.4 LINKED QUERIES.....	40
6.5 SUBSTITUTION VARIABLES .....	40
6.6 UPDATING THE DATABASE .....	41
6.7 VIEWING AND EDITING XMLTYPE COLUMNS .....	42
6.8 DIRECT QUERY EXPORT.....	42
6.9 SAVING SQL SCRIPTS.....	42
6.10 CREATING STANDARD QUERIES.....	43
<b>7. THE COMMAND WINDOW.....</b>	<b>44</b>
7.1 ENTERING SQL STATEMENTS AND COMMANDS.....	44
7.2 DEVELOPING COMMAND FILES.....	45
7.3 SUPPORTED COMMANDS.....	46

<b>8.</b>	<b>CREATING AND MODIFYING NON-PL/SQL OBJECTS .....</b>	<b>49</b>
8.1	THE TABLE DEFINITION EDITOR.....	49
8.2	THE SEQUENCE DEFINITION EDITOR.....	60
8.3	THE SYNONYM DEFINITION EDITOR .....	60
8.4	THE LIBRARY DEFINITION EDITOR.....	61
8.5	THE DIRECTORY DEFINITION EDITOR .....	61
8.6	THE JOB DEFINITION EDITOR.....	62
8.7	THE QUEUE DEFINITION EDITOR.....	62
8.8	THE QUEUE TABLE DEFINITION EDITOR.....	63
8.9	THE USER DEFINITION EDITOR.....	64
8.10	THE ROLE DEFINITION EDITOR.....	67
8.11	THE PROFILE DEFINITION EDITOR.....	68
8.12	THE DATABASE LINK DEFINITION EDITOR.....	68
<b>9.</b>	<b>DIAGRAMS .....</b>	<b>69</b>
9.1	CREATING A DIAGRAM.....	69
9.2	SAVING AND OPENING A DIAGRAM FILE .....	71
9.3	UPDATING A DIAGRAM .....	71
<b>10.</b>	<b>REPORTS .....</b>	<b>72</b>
10.1	STANDARD REPORTS.....	72
10.2	CUSTOM REPORTS .....	73
10.3	VARIABLES .....	74
10.4	REFINING THE LAYOUT .....	77
10.5	THE STYLE LIBRARY .....	84
10.6	OPTIONS.....	84
10.7	THE REPORTS MENU .....	86
<b>11.</b>	<b>GRAPHICS .....</b>	<b>87</b>
<b>12.</b>	<b>PROJECTS .....</b>	<b>89</b>
12.1	CREATING A NEW PROJECT .....	89
12.2	SAVING A PROJECT .....	89
12.3	ADDING FILES TO A PROJECT .....	90
12.4	ADDING DATABASE OBJECTS TO A PROJECT.....	90
12.5	WORKING WITH PROJECT ITEMS.....	90
12.6	COMPILING A PROJECT .....	91
<b>13.</b>	<b>TO-DO ITEMS .....</b>	<b>92</b>
13.1	CREATING A TO-DO ITEM.....	93
13.2	EDITING A TO-DO ITEM .....	93
13.3	CLOSING A TO-DO ITEM.....	94
13.4	DELETING A TO-DO ITEM.....	94
<b>14.</b>	<b>WINDOWS, DATABASE SESSIONS AND TRANSACTIONS .....</b>	<b>95</b>
14.1	SESSION MODE.....	95
14.2	EXECUTION IN MULTI SESSION OR DUAL SESSION MODE.....	95
<b>15.</b>	<b>BROWSING OBJECTS .....</b>	<b>96</b>
15.1	USING THE BROWSER .....	96
15.2	BROWSER FILTERS.....	101
15.3	BROWSER FOLDERS.....	102
<b>16.</b>	<b>PREFERENCES .....</b>	<b>104</b>
16.1	ORACLE – CONNECTION .....	105
16.2	ORACLE – OPTIONS.....	106
16.3	ORACLE – DEBUGGER.....	107

16.4	ORACLE – OUTPUT.....	108
16.5	ORACLE – TRACE.....	109
16.6	ORACLE – PROFILER.....	109
16.7	ORACLE – LOGON HISTORY.....	110
16.8	ORACLE – HINTS.....	111
16.9	USER INTERFACE – OPTIONS.....	113
16.10	USER INTERFACE – TOOLBAR.....	115
16.11	USER INTERFACE – BROWSER.....	115
16.12	USER INTERFACE – EDITOR.....	117
16.13	USER INTERFACE – FONTS.....	120
16.14	USER INTERFACE – CODE ASSISTANT.....	121
16.15	USER INTERFACE – KEY CONFIGURATION.....	122
16.16	USER INTERFACE – APPEARANCE.....	123
16.17	USER INTERFACE – DATE/TIME.....	124
16.18	WINDOW TYPES – PROGRAM WINDOW.....	125
16.19	WINDOW TYPES – SQL WINDOW.....	127
16.20	WINDOW TYPES – TEST WINDOW.....	129
16.21	WINDOW TYPES – PLAN WINDOW.....	129
16.22	TOOLS – DIFFERENCES.....	129
16.23	TOOLS – DATA GENERATOR.....	130
16.24	TOOLS – TO-DO LIST.....	131
16.25	TOOLS – RECALL STATEMENT.....	131
16.26	FILES – DIRECTORIES.....	132
16.27	FILES – EXTENSIONS.....	133
16.28	FILES – FORMAT.....	134
16.29	FILES – BACKUP.....	134
16.30	FILES – HTML/XML.....	135
16.31	OTHER – PRINTING.....	136
16.32	OTHER – UPDATES & NEWS.....	137
16.33	PREFERENCE SETS.....	138
<b>17.</b>	<b>TOOLS.....</b>	<b>140</b>
17.1	BROWSER.....	140
17.2	FIND DATABASE OBJECTS.....	140
17.3	COMPILE INVALID OBJECTS.....	141
17.4	EXPORT TABLES.....	142
17.5	IMPORT TABLES.....	145
17.6	EXPORT USER OBJECTS.....	147
17.7	TEXT IMPORTER.....	148
17.8	ODBC IMPORTER.....	151
17.9	DATA GENERATOR.....	153
17.10	COMPARE USER OBJECTS.....	156
17.11	COMPARE TABLE DATA.....	158
17.12	EVENT MONITOR.....	160
17.13	SESSIONS.....	161
17.14	USER DEFINED TOOLS.....	163
17.15	TEST MANAGER.....	167
<b>18.</b>	<b>THE EDITOR.....</b>	<b>170</b>
18.1	SELECTION FUNCTIONS.....	170
18.2	COLUMN SELECTION.....	170
18.3	CODE ASSISTANT.....	171
18.4	RECALLING STATEMENTS.....	172
18.5	SPECIAL COPY.....	172

18.6	CONTEXT SENSITIVE HELP.....	173
18.7	DATABASE OBJECT POPUP MENU.....	174
18.8	EXPLAIN PLAN.....	174
18.9	MACROS.....	174
18.10	BOOKMARKS.....	175
18.11	COLOR MARKS.....	176
18.12	CODE CONTENTS.....	177
18.13	CODE HIERARCHY.....	178
18.14	CODE FOLDING.....	179
18.15	SPLIT EDITING.....	179
18.16	HYPERLINK NAVIGATION.....	180
18.17	NAVIGATION BUTTONS.....	180
18.18	REFACTORING.....	181
18.19	SEARCH BAR.....	181
<b>19.</b>	<b>THE LARGE DATA EDITOR.....</b>	<b>183</b>
19.1	EDITING PLAIN TEXT.....	184
19.2	EDITING RTF.....	184
19.3	EDITING XML.....	185
19.4	EDITING IMAGES.....	186
19.5	EDITING HEXDATA.....	186
19.6	INVOKING AN EXTERNAL VIEWER OR EDITOR.....	187
<b>20.</b>	<b>THE QUERY BUILDER.....</b>	<b>189</b>
20.1	CREATING A NEW SELECT STATEMENT.....	189
20.2	MODIFYING AN EXISTING SELECT STATEMENT.....	192
20.3	MANIPULATING THE QUERY DEFINITION.....	192
20.4	QUERY BUILDER PREFERENCES.....	193
20.5	QUERY BUILDER PLUG-INS.....	194
<b>21.</b>	<b>THE PL/SQL BEAUTIFIER.....</b>	<b>195</b>
21.1	DEFINING THE OPTIONS.....	195
21.2	DEFINING THE RULES.....	196
21.3	USING THE BEAUTIFIER.....	197
<b>22.</b>	<b>TEMPLATES.....</b>	<b>198</b>
22.1	THE TEMPLATE WINDOW.....	198
22.2	USING A TEMPLATE.....	199
22.3	CREATING AND MODIFYING TEMPLATES.....	200
<b>23.</b>	<b>WINDOW LIST.....</b>	<b>205</b>
<b>24.</b>	<b>DOCKABLE AND FLOATING TOOLS.....</b>	<b>206</b>
<b>25.</b>	<b>AUTHORIZATION.....</b>	<b>207</b>
25.1	ENABLING AUTHORIZATION.....	207
25.2	DEFINING AUTHORIZATION.....	208
25.3	DISABLING AUTHORIZATION.....	209
<b>26.</b>	<b>ORACLE FILE SYSTEM (OFS).....</b>	<b>210</b>
26.1	OFS MANAGER.....	210
26.2	OFS USAGE.....	212
<b>27.</b>	<b>HELP SYSTEMS.....</b>	<b>214</b>
27.1	MS HELP FILES.....	214
27.2	HTML MANUALS.....	214

<b>28.</b>	<b>CUSTOMIZATION.....</b>	<b>218</b>
28.1	PREFERENCES .....	218
28.2	WINDOW LAYOUT.....	218
28.3	ON-LINE DOCUMENTATION.....	218
28.4	COMMAND-LINE PARAMETERS .....	220
28.5	SQL, PL/SQL, COMMAND, JAVA AND XML KEYWORDS.....	222
28.6	PLUG-INS .....	222





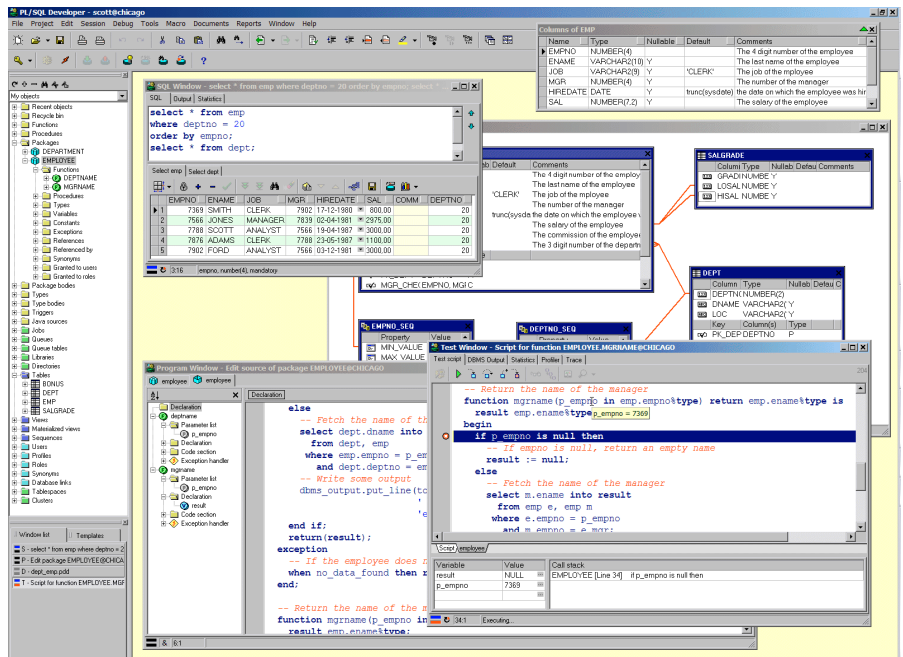
# 1. Introduction

PL/SQL Developer is an Integrated Development Environment (IDE) for developing stored program units in an Oracle Database. Using PL/SQL Developer you can conveniently create the server-part of your client/server applications.

As a worst-case scenario, up to now you might have been working like this:

- You use a text editor to write program units (procedures, triggers, etc.).
- You use Oracle SQL\*Plus to compile the source files.
- If there is a compilation error, you have to find out where it is located in the source file, correct it, switch back to SQL\*Plus to recompile it, only to find the next error.
- You use SQL\*Plus or the client-part of your application to test the program unit.
- In case of a runtime error, again you have a hard time locating the cause of the problem and correcting it.
- You use the Explain Plan utility or tkprof to optimize your SQL statements.
- To view or modify other objects and data in your database, you use SQL\*Plus or yet another tool.

These tasks - editing, compiling, correcting, testing, debugging, optimizing and querying - can all be performed without leaving PL/SQL Developer's IDE. Furthermore, PL/SQL Developer provides several other tools that can be helpful during everyday PL/SQL development.



## Editing

PL/SQL Developer, like any other serious development environment, assumes that you store your source files on disk. Other tools just let you edit sources in the database, but this does not allow for any version control or deployment scheme. The source files can be run through SQL\*Plus, so you can

deploy them on any platform without using PL/SQL Developer. You can edit many files at once through a standard multiple document interface.

The editor offers a wide range of assistance to the programmer. There is context sensitive help on SQL statements and PL/SQL statements. We've all been there: you start typing *substr*, but have forgotten the exact meaning of the parameters. Now you can simply hit *F1* and you're taken to the appropriate topic in the *SQL Reference Manual*. Tables, views and program units can be described for you in a roll-up window from within the editor in the same way. A Code Assistant is integrated into the editor that automatically displays information of database objects as you type their name, allowing you to browse and pick elements from this description. For large package or type bodies, the program editor provides a tree view with the code contents for easy navigation, highlights code structures and variable references, allows you to fold/unfold code sections, and provides hyperlink navigation. The Query Builder allows you to graphically create select statements. PL/SQL Developer's extensible templates make it easy to insert standard SQL and PL/SQL code into your programs. All editors use the appropriate SQL, PL/SQL and SQL\*Plus syntax highlighting to make your code more readable.

## Compiling & correcting

From within the editor you can compile a source file. In case of a compilation error you are automatically taken to the appropriate source line. All compilation errors are reported in a list at the bottom of the editor. This list can include hints for code that may indicate common programming errors or violations of user-defined naming conventions.

Because you can compile a source file without saving it, you can safely explore many alternatives of solving a problem. The editor keeps track of the fact that you have changed the file without saving or compiling it.

## Testing & debugging

To test your program unit, you can write a test script. The body of the test script contains a PL/SQL block in which you can program the test-code. Any variables that you want to use in the PL/SQL block can be declared, assigned a value for input, and viewed after execution.

When you execute a test script, a runtime error might occur. In this case, PL/SQL Developer allows you to view the sources of the error stack. Each source line of the error stack that was involved in the runtime error is highlighted, so you can easily backtrack to the cause of the problem.

If you are using Oracle 7.3.4 or later you can use PL/SQL Developer's integrated debugger. You can step through your code, set breakpoints, view/set variables, view the call stack, and so on.

On Oracle8i and later you can additionally use the *dbms\_trace* feature to log selected events of a program run. This can help you analyze the program flow and exceptions.

Output from calls to the *dbms\_output* and the *PL/SQL Web Toolkit* packages are automatically shown in a corresponding tab page of the Test Window.

For regression testing you can use the Test Manager to quickly run and verify a set of Test Scripts.

## Optimizing

To optimize the SQL statements in your program units, Oracle's Explain Plan utility can be a big help. Therefore it is integrated into PL/SQL Developer's IDE. By simply selecting the SQL statement in the source file and pressing F5, the query plan is visually presented to you in a separate Explain Plan window. You can then modify the statement to optimize its query plan outside the source file, and copy it back afterwards.

You can also view statistics about executed SQL statements and PL/SQL program units. These statistics can include elapsed time, CPU time, logical reads, physical reads, physical writes, and so on.

Oracle8i introduced a PL/SQL Profiler that allows you to profile your PL/SQL code. For each executed line of PL/SQL code you can determine the execution time, and how many times it was executed.

## Querying

To query the data in the database, you can use a SQL window to execute any SQL statement. All executed statements are kept in a history buffer, so you can easily re-execute them. Any query results are conveniently displayed in a separate grid that you can subsequently use to insert, update, or delete records. The result grid can additionally be used in a Query By Example mode, so that you can easily find the information you need.

To query database objects you can use the Object Browser. All relevant properties of database objects such as tables, views, sequences, functions, procedures, packages, types and triggers can be viewed, including any dependencies between the objects. The browser uses a tree view similar to the explorer in Windows for easy point-and-click browsing.

## Running SQL scripts

PL/SQL Developer includes a Command Window that can be used to run SQL scripts or execute SQL statements. You can additionally execute commands that are very similar to the SQL\*Plus commands that you may be familiar to.

## Creating and modifying table definitions

You can easily create and modify table definitions with using any SQL statement. Just fill in the definition in a dialog window to modify columns, constraints, indexes, privileges, storage information, comments, and so on. You can apply this definition in the database, and view, modify and save the resulting SQL.

## Diagrams

To visualize the database objects of your application or project and their relations, you can create diagrams. A diagram can be used for documentation purposes, but can also serve as a workspace to work with related objects.

## Reporting

PL/SQL Developer comes with a number of standard reports, which are HTML based. You can view these reports within PL/SQL Developer, print them, or save them as HTML files. You can also create your own custom reports. Reports can be made easily accessible from the reports menu.

## Graphics

The Graph Window can be launched from within a SQL Window or Report Window to obtain a graphical representation of the queried data.

## Projects

To organize your work you can use PL/SQL Developer's project concept. A project consists of a number of files and database objects. These objects are easily accessible through the Project Items Window, and can be compiled through a single mouse-click.

## Tools

PL/SQL Developer provides several tools that can be helpful during development. These tools include a Find Database Object tool, allowing you to search for text in database object sources, a Compile Invalid Objects tool, to quickly compile objects that have become invalid during development, Export and Import tools, a Data Generator tool to create test data, an Export User Objects tool to export the DDL

statements of a user's objects, a Compare User Objects tool to compare the object definitions of 2 users, a Session information tool, an Event monitor, and a Compare Table Data tool to compare and equalize table contents.

In addition to these standard tools, you can define your own tools and include them in PL/SQL Developer's tools menu.

## 2. Installation

There are basically two ways to install PL/SQL Developer:

- Workstation installation. In this case you install the software locally on each workstation that it will be used on.
- Server installation. In this case you install the software on a server at a location that can be accessed from each workstation that it will be used on.

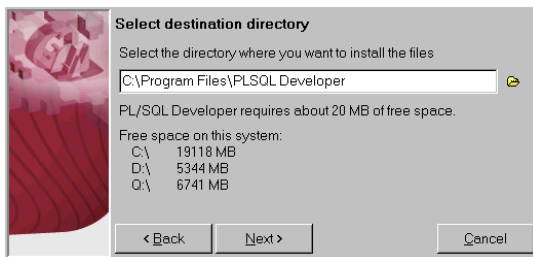
Both installation types will be explained here.

### 2.1 System requirements

PL/SQL Developer is a 32-bit application for Windows. Therefore you must have at least Windows 95 or Windows NT 4 installed on your workstation. To connect to an Oracle database, PL/SQL Developer requires a 32-bit version of SQL\*Net, Net 8, Net 9 or Net 10.

### 2.2 Workstation installation

To install PL/SQL Developer locally on a workstation, run the setup program from the installation medium. After doing this, the following dialog will appear:



You can select a destination directory for the program files, a folder in the start menu, a location for a shortcut to PL/SQL Developer on the desktop, and installation options. After pressing the *Finish* button on the final page, the program files are copied and the shortcuts are created.

### 2.3 Server based installation

For multi-user licenses you can create a single PL/SQL Developer installation on a file server and provide access to all licensed users. To do so, you can run the setup program on the file server in the same way as described in the previous section. You must choose a destination directory that is shared to the users that will use the software. When PL/SQL Developer is first run on a workstation, this type of installation is automatically detected, and local installation is then completed without the user noticing it.

PL/SQL Developer can be installed into a directory where the users have only read access. There are a few exceptions though:

- The Macros subdirectory must be writeable, as this is the place where all macros will be stored.
- The Preferences subdirectory must be writeable, as this is the place where personal preference sets will be stored. See chapter 16.33 for details.

## 2.4 Scripted installation

In addition to the installation procedure described above, you can also create a script for unattended installation. Read the included *install.txt* file for detailed information.

## 2.5 Uninstalling PL/SQL Developer

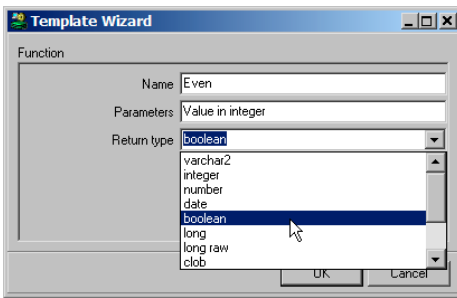
If for some reason you wish to uninstall PL/SQL Developer, you can remove it using Add/Remove Program in the Control Panel.

### 3. Writing programs

In an Oracle database, you can distinguish five different types of stored program units: functions, procedures, packages, types and triggers. The Program editor allows you to create and modify these five types of program units in one uniform way.

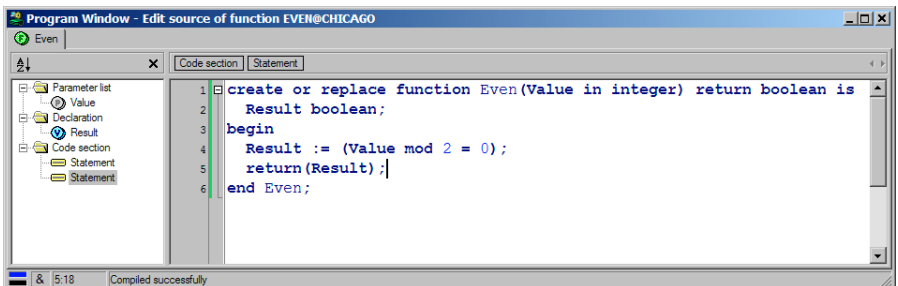
#### 3.1 Creating a program

To create a new program, press the *New* button on the toolbar and select the *Program Window* item and, for example, the *Function* sub item. You are now prompted for various variables that are relevant to this program type. In this case, where we are creating a function, the name of the function, the parameter list, and its return type:



As the title of this dialog suggests, this information comes from a template. PL/SQL Developer provides several standard templates, which you can modify as needed. You can also define new templates. Information about defining templates is provided in chapter 22.

After you have entered the variables and pressed the *OK* button, a Program Editor Window appears with a template function in it. Each program you create in the Program Editor unit must start with the familiar 'create or replace' SQL syntax. In this case we are creating a function 'even', and the source file could look like this:



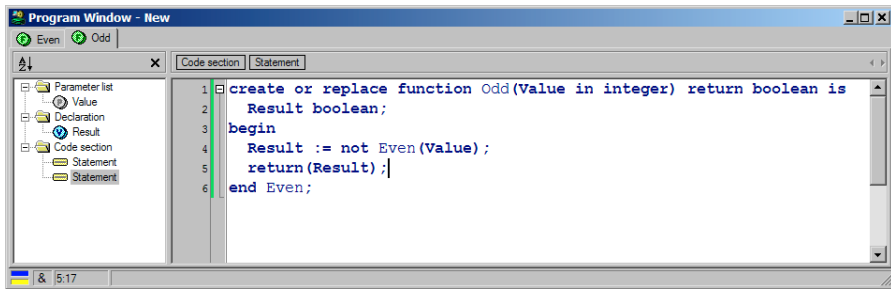
At the left side of the editor you see the Code Contents, which displays the structure of the program unit. This is useful when navigating large program units such as package bodies and type bodies. This feature is described in detail in chapter 18.12.

Above the editor you see the Code Hierarchy, which displays the hierarchy from the cursor position. In this case it shows that you are located at a *Statement* (the return statement) within a *Sequence* of statements within a *Function*. For complex programs, this can be helpful to see where exactly you are

located, and can help you to select certain portions of the PL/SQL code. This is described in detail in chapter 18.13.

A program file can contain more than one program unit. By right clicking in the Program Editor, a popup menu appears that allows you to add or delete a program unit. You can switch between the program units by selecting the appropriate tab at the top of the window. This way you can conveniently keep related program units together in one source file. A package specification and body are a good example of this feature.

A program unit should be positioned after any other program unit in the program file it might reference. If you create a function 'odd' that references the previously created function 'even', the program editor should look like this:



## 3.2 Saving a program

You can save a program file by pressing the *Save* button on the toolbar. A save dialog appears that shows some standard extensions for program files:

Program type	Extension
Function	.fnc
Procedure	.prc
Package specification & body	.pck
Package specification	.spc
Package body	.bdy
Type specification & body	.typ
Type specification	.tps
Type body	.tpb
Trigger	.trg
Java source	.jsp

You can change these extensions through a preference as described in chapter 16.27. If one single function, procedure, package, type, trigger or java source is in the program file, the filename is automatically derived from the name and type of this program unit. After saving the file, the filename is shown in the window title.



The saved program file has a format that is compatible with Oracle SQL\*Plus. As an example, the source file containing the 'odd' and 'even' function looks like this:

```
create or replace function Even(Value in integer) return boolean is
    Result boolean;
begin
    Result := (Value mod 2 = 0);
    return(Result);
end Even;
/
create or replace function Odd(Value in integer) return boolean is
    Result boolean;
begin
    Result := not Even(Value);
    return(Result);
end Odd;
/
```

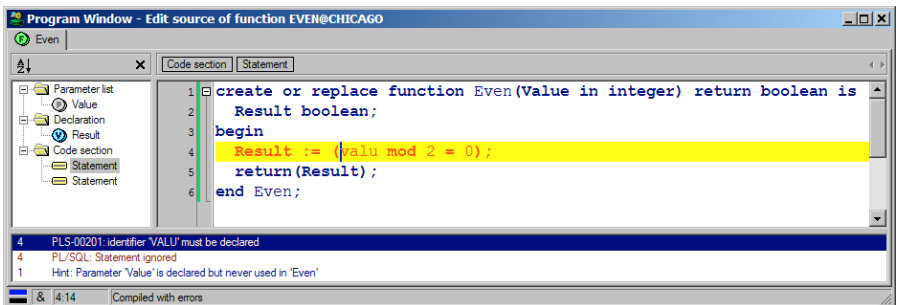
When this file is executed through SQL\*Plus, both functions will be created.

### 3.3 Modifying a program

You can open a previously saved program file by pressing the *Open* button on the toolbar and selecting the *Program file* item. If the file was recently used, you can also open it by selecting *Reopen* from the *File* menu or by clicking on the arrow next to the *Open* button on the toolbar. When you start editing, a blue colored indicator at the bottom of the window lights up. This indicates that the file is modified, but not yet saved. There is also a yellow colored indicator, which indicates that the file is modified, but not yet compiled.

### 3.4 Compiling a program

You can compile a program file by pressing the *Execute* button on the toolbar. All program units in the program file are compiled, starting with the first one, regardless of the program unit that is currently selected. When an error occurs, compilation is terminated and the editor is positioned at the source line that caused the error:



If there is more than one error and you wish to correct them, you can click on the next error message at the bottom of the Program Editor to go to the error location. If you have configured PL/SQL Developer to use the HTML manuals as described in chapter 27.2, you can double-click on a compilation error to display the corresponding paragraph in the *Oracle Error Messages* manual.

The compilation error list can also contain hints. Hints do not cause the compilation to fail, but they may indicate an issue that could cause a run-time error (such as a comparison with null, a function without a return value, an unused parameter, and so on). For more information about hints, see chapter 16.18.

If an error message is displayed in a message box after compilation, this means that the create statement failed without actually compiling the source. The error message should explain the cause of the error. If for example you get the 'create or replace' syntax wrong, it will say "ORA-00900: Invalid SQL statement" in a message box.

Note: If you are using Oracle Server 7.2 or earlier, a trigger compilation error is always reported in a message box. Only since Oracle Server 7.3, trigger compilation errors are reported in the same way as procedures, functions, packages and types.

## 3.5 Substitution variables

If your Program File includes substitution variables as used in SQL\*Plus (prefixed with an ampersand), you can press the & button at the lower left of the window. The sources in the program file will be scanned for substitution variables, for which you can subsequently enter a value. If the *Save window state* preference is enabled (see chapter 16.18), these substitution variable values will be saved and restored if a program file or database object is viewed or edited later.

## 4. Testing programs

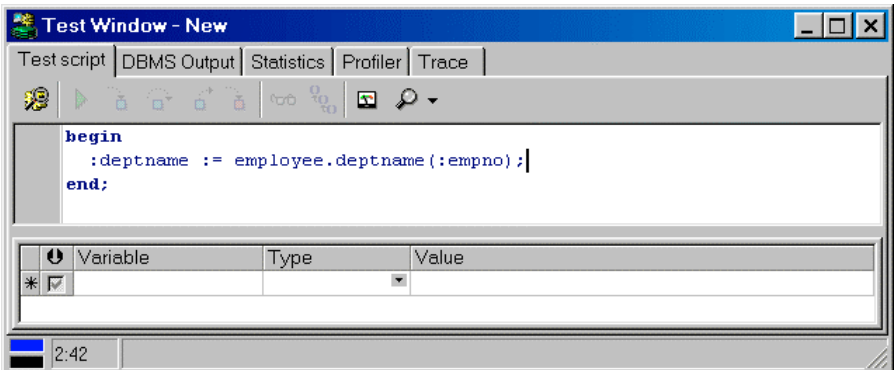
After successfully compiling a program, you'll need to test it. To achieve this, you can use PL/SQL Developer's Test Scripts. A Test Script allows you to execute one or more program units, define input, output and input/output variables and view and assign values to variables. If a run-time error occurs during execution of the Test Script, you can view the sources of the program units that were involved in the error.

If you are using Oracle 7.3.4 or later you can use PL/SQL Developer's integrated debugger. You can step through your code, set breakpoints, view and set variables, view the call stack, and so on.

### 4.1 Creating a Test Script

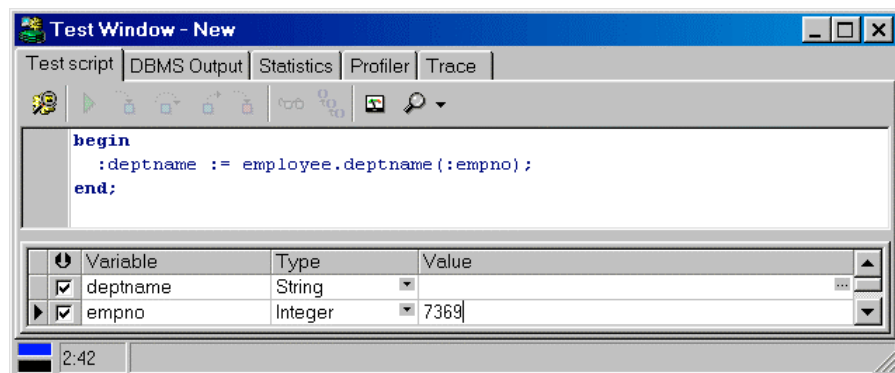
There are two ways to create a Test Script. You can simply right-click on a program unit (a standalone function or procedure, a packaged function or procedure, or an object type method) in the Browser and select the *Test* item. This will create a new Test Script with variables for the parameters and result. For the purpose of this manual, we are going to create a Test Script manually from scratch though.

To create a new empty Test Script, press the *New* button on the toolbar and select the *Test Window* item. You can now type a PL/SQL block with the familiar declare...begin...end syntax in the body of the Test Script. As an example we're going to test a package *employee*, which implements some employee functions of the employee/department demo tables. Function *deptname* returns the name of the department of an employee, and can be tested using the following PL/SQL block:



The PL/SQL block contains a simple call to the function that we want to test, and uses variables *deptname* and *empno*. We'll have to assign a value to *empno*, and after execution, check the value of *deptname* to determine if the function returned the correct value.

By prefixing these variables with a colon, they can be declared at the bottom of the Test Script:

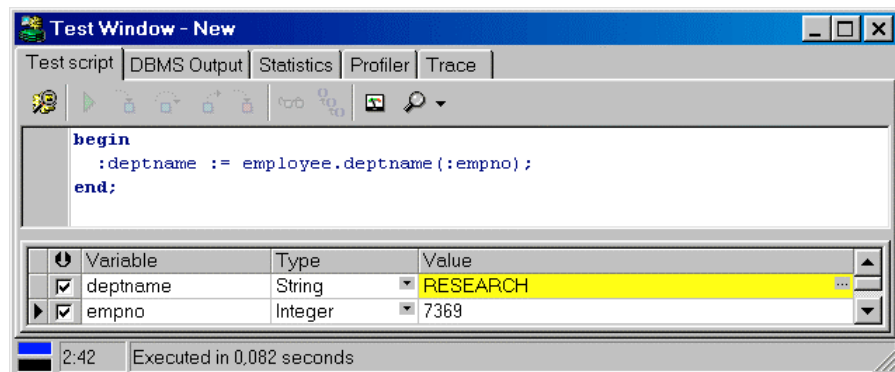


The *Scan source for variables* button (🔍) at the upper-left of the variables list can be used to quickly copy the variables from the source. After this, the integer data type and value 7369 (Mr. Smith from the research department) is assigned to *empno*. Now we're ready to execute the script.

You can also declare variables locally in a declare part of the PL/SQL block, but you will not be able to modify or view the values of these variables. An advantage of these local variables is the fact that you can use record types, PL/SQL tables, types and so on.

## 4.2 Executing a Test Script

To execute the Test Script, press the *Execute* button on the toolbar. After execution, the values of the variables are updated. Any variable whose value has changed is displayed with a yellow background:



The result of the function is 'RESEARCH', so it obviously functions correctly. At the bottom of the window, the execution time is displayed. This information can be used to optimize your code for performance. For optimization purposes you can also view the statistics of the execution of the PL/SQL block by selecting the Statistics tab. You can also create a profile report for each executed line of PL/SQL code by pressing the *Create Profiler report* button before executing the script. After execution you can switch to the *Profiler* page to view the report. Statistics and Profiler reports are explained in chapter 5.2 and 5.3 respectively.

You can abort a running Test Script by pressing the *Break* button, which is particularly useful when a program is stuck in an endless loop, or execution is taking longer than expected. Note that pressing the *Break* button will not always be successful. If for example the program is waiting for a lock, it will not respond to a break signal.

After executing a script, a transaction may have been started by the program units that were executed. The *Commit* and *Rollback* button on the toolbar will be enabled if this is the case. For more information about transactions, see chapter 14.

### 4.3 Variable types

The type list in the variables section of the Test Window contains all types that can be used. The characteristics of these types are explained in this chapter.

Type	Description
Integer	Can be used for numeric values in the range of $-2^{32}$ to $2^{32} - 1$ .
Float	Can be used for other numeric values.
String	Equivalent of a varchar2 data type in a table. Can be up to 2000 (Oracle7) or 4000 (Oracle8) characters. The value cell of this data type has a cell button that invokes a text editor, so that you can view and modify multi line strings easily.
Date	The date and time data type.
Long	The Long data type does not display the value in the grid, but just displays <Long>. Press the cell button to invoke the text editor.
Long Raw	Pressing the cell button of a Long Raw value will invoke an import/export file dialog, allowing you to load or save the contents of this variable.
Cursor	This data type can be used where you can use a cursor variable in PL/SQL. After executing the Test Script, you can press the cell button of the value to display the result set in a SQL Window.
CLOB, BLOB & BFile	These LOB Locator variables must be initialized on the server before you can view the LOB data. The CLOB will invoke a text editor, the BLOB and BFile will invoke an import/export file dialog.
PL/SQL String	Use this data type for PL/SQL varchar2 values of up to 32000 characters.
Char	A fixed length (space padded) string.
Substitution	Substitution variables can be used without the restriction of bind variables. The variable name in the PL/SQL block is substituted by its value in the text before it is sent to the server.
Temporary CLOB & BLOB	These LOB Locators can already hold data before they are passed to the server through the Test Script.

## Boolean variables

When you select the variable type list-box, you will notice that the Boolean data type is missing. This is because SQL\*Net does not support this data type. To use a Boolean variable, you can declare it as an integer and use the *sys.dbutil.bool\_to\_int* and *sys.dbutil.int\_to\_bool* functions provided by Oracle. These functions convert between null/true/false and null/0/1. If you use the *Test* function in the Browser, this conversion is automatically generated for you.

## 4.4 Saving Test Scripts

To save the Test Script, press the *Save* button on the toolbar. The save dialog uses *.tst* as the default extension for Test Scripts, though you can change the default extension with a preference described in chapter 16.27. The PL/SQL block, all variables (name, type and value), and the debugger watches (see chapter 4.10) are saved. By saving a Test Script you can easily re-test a program unit if you modify it later.

You can open a previously saved Test Script by pressing the *Open* button on the toolbar and selecting *Test Script*, which will create a new Test Window. You can alternatively right-click on a previously created Test Window and select the *Load* item, which will open a Test Script in the existing Test Window.

### Saving as SQL\*Plus script

You can alternatively save a Test Script in SQL\*Plus compatible format. This allows you to run the script in an environment where PL/SQL Developer is not available (e.g. on a Unix server). To do so, select *SQL\*Plus script (\*.sql)* as file type in the file selector dialog. The example script from the previous chapter would be saved like this:

```
rem PL/SQL Developer Test Script

set feedback off
set autoprint off

rem Declare variables
variable result varchar2(2000)
variable p_empno number

rem Set variables
begin
    :result := null;
    :p_empno := 7369;
end;
/

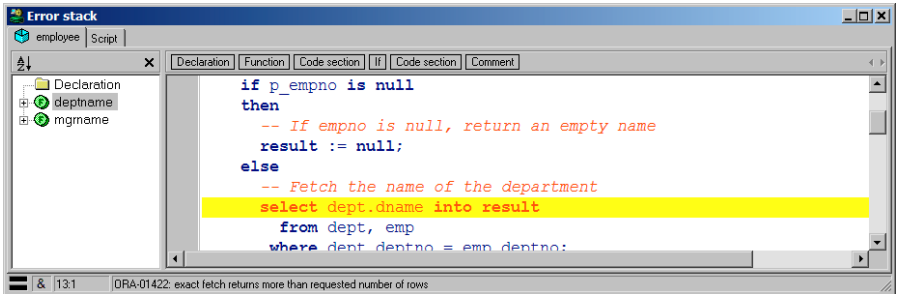
rem Execute PL/SQL Block
begin
    :result := employee.deptname(:p_empno);
end;
/

rem Print variables
print result
print p_empno
```

**Note:** You cannot open a SQL\*Plus script again as a Test Script. If you want to reuse it in a Test Window, make sure you save it as a Test Script as well!

## 4.5 Tracking run-time errors

Whenever your program unit causes a run-time error, PL/SQL Developer will ask you if you wish to view the sources of the error stack. This way you can quickly find the cause of the error. If there were an error in our *deptname* function, the error stack window would look like this:



At the top you see a tab for each program unit involved in the run-time error. The program units have been called in right to left order. You can flip through the tabs to easily find the program flow that lead to the error. In this case it shows that the Test Script has called function *employee.deptname*. This obviously can get more complicated and even go across triggers.

Note: If you are using Oracle Server 7.2 or earlier, trigger source will not be shown in the error stack window.

The error stack highlights each line that was involved in the error in red. For the last program unit in the stack this is the line that caused the error. For all other program units this is the line where the call to the next program unit was made.

## 4.6 Package states & Java session states

When you are editing, compiling and testing packages, the package states will be discarded by Oracle after each compilation. All global variables in a package will be reset, and the initialization block will be executed again. This can lead to unexpected results. Whenever PL/SQL Developer detects this situation, a warning is issued in the status line of the Test Window: "Warning: ORA-04068: Existing state of packages has been discarded".

Similarly, if you edit and compile a Java source any existing Java session state will be cleared. PLSQL Developer will also handle this situation, and will display "Warning: ORA-29549: class <class name> has changed, Java session state cleared" in the status line.

## 4.7 Viewing result sets

In a Test Script you are not limited to PL/SQL blocks. You can also execute single SQL statements, including select statements. Executing a select statement will display an additional *Results* tab page, which displays all selected rows.

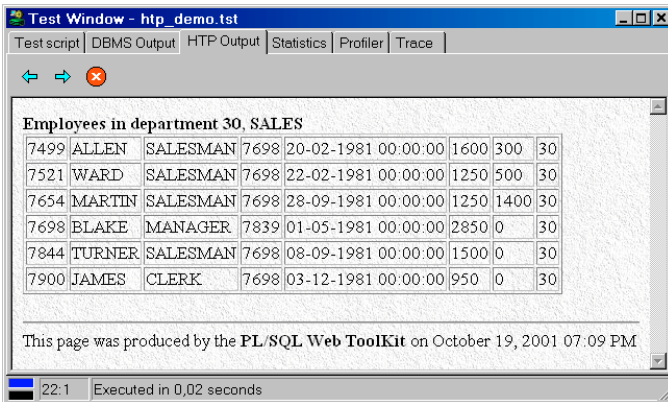
## 4.8 Viewing dbms\_output

For debugging purposes it can sometimes be necessary to "print" some information to the screen from within a program unit. For this purpose Oracle created the *dbms\_output* package. By calling *dbms\_output.put\_line*, information is placed in the output buffer. After execution of a Test Script, you can select the *Output* tab at the top of the Test Window to view the contents of the output buffer. On

this page you can additionally set the size of the output buffer or enable/disable buffering. The default settings on the output page are controlled by the output preferences as described in chapter 16.4.

## 4.9 Viewing HTP output

If you want to test program units that make use of the PL/SQL Web Toolkit, the *HTP Output* tab page will be displayed if any HTP output was generated:



If there is no HTP output, this tab page will not be visible.

## 4.10 Debugging

For those programming errors that are really hard to track, the Test Window provides an integrated debugger. At the top of the window you find a toolbar with functions related to the debugger. To start a debug session, just press the *Start* button at the left of the debug toolbar instead of the *Execute* button in the main toolbar. The other buttons are now enabled and you are ready to debug.

### Controlling execution

After starting the debugger, execution will pause before the first statement in the Test Script. After this, you can control execution with the buttons in the debug toolbar:



Run the script until completion.



Step into a procedure, function or method call on the next line. If the next line contains an update, insert or delete statement that will cause a trigger to be fired, you will step into that trigger.



Step over the next line. It will be executed, but you will not step into the source.



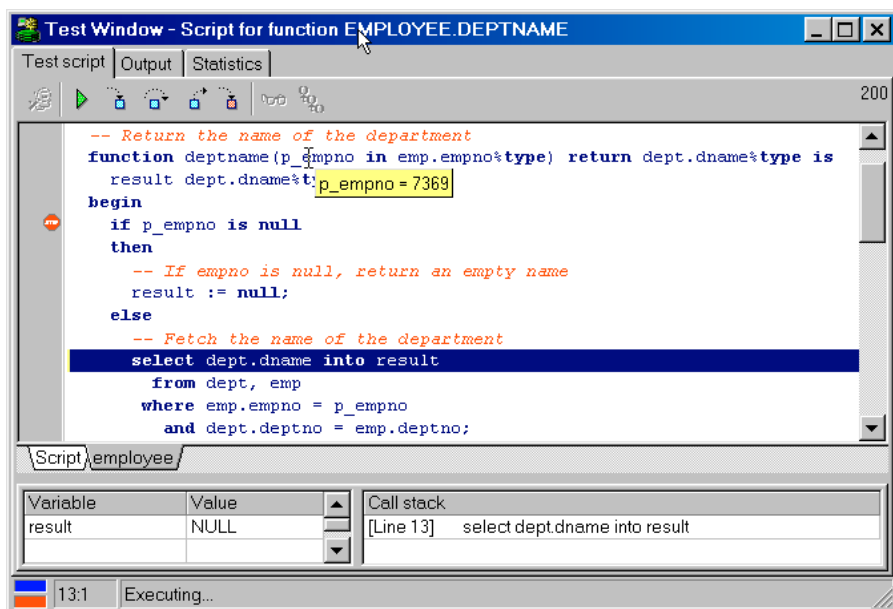
Step out of the current program unit.



Run until an exception occurs. Execution will be paused on the line that causes the exception. After the next step the exception will actually be raised.

Whenever you step into a program unit, its source will automatically be loaded into the Test Window. The bottom of the editor panel will now show tabs for each program unit, so that you can easily switch between them to view the source, set/remove breakpoints, and so on. By right-clicking on the editor panel you can remove a program unit from the Test Window if you are no longer interested in it:





## Viewing and setting variable values

To view the value of a variable during a debug session, you can simply move the mouse cursor over the variable in the source. After ½ a second, its value will be displayed in a popup. Variables in the PL/SQL block of Test Script can never be displayed. Complex variables (e.g. record and object types) cannot be displayed either, though you can view individual fields. Future versions of Oracle may remove these limitations.

To set the value of a variable, right-click in its name in the program source and select the *Set Variable* item from the popup menu. An input field appears with the current value of the variable. You can enter a new value and press *Enter* to apply it, or *Esc* to cancel the operation:



From the same popup menu you can also select to add the variable to the watch list, which means that after each debug step the variable value will automatically be displayed and updated in the watch list at the bottom-left of the Test Window.

If a variable is a collection (a PL/SQL table, varray or nested table) of a scalar data type you can view the entire collection by right clicking on it and selecting *View collection variable* from the popup menu.

Note that variable values can only be viewed and set if the program units are compiled with debug information. A preference exists that will cause each compilation to include debug information, and you can manually add debug information by right-clicking on a program unit in the Browser and selecting the *Add debug information* item from the popup menu.

## Using Breakpoints

Breakpoints can be used to halt program execution on a certain line in your PL/SQL code. When execution halts, you can view and set variables, step through the code, and so on. You can define a

condition for a breakpoint, in which case execution will only be halted when this condition is met. For each breakpoint you can define a message that will be placed in the output page when the breakpoint is reached.

### **Setting breakpoints**

There are two ways to set breakpoints: in a Program Window or in a Test Window. In both cases you simply need to click on the appropriate line in the left margin of the editor. A breakpoint mark will appear, indicating that a breakpoint is present on that line. When you execute a Test Window in debug mode, execution will stop if one of the breakpoints is encountered.

If you set a breakpoint in a Program Window, it can be that this particular program unit has not yet been compiled into the database. Therefore, such a breakpoint cannot be applied to the database yet either. In this case the breakpoint mark will have a different appearance. When you subsequently compile the program unit, the breakpoint will be applied and the corresponding mark will change to reflect this. As long as the program unit is not compiled, any previously set breakpoints in this program unit will be effective.

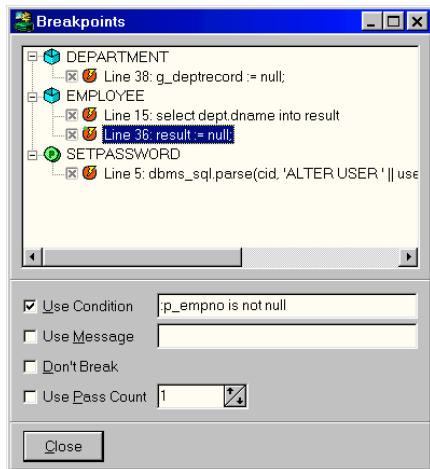
You cannot set breakpoints in the PL/SQL block of the Test Script.

To delete a breakpoint, simply click on the breakpoint mark again.

### **Breakpoint conditions**

Sometimes you define a breakpoint on a line that is executed very often, even though you are only interested in the program status under certain circumstances. In that case you can define a condition for the breakpoint. Execution will only halt when the condition is met.

To define a condition for a breakpoint, right-click on the mark and select the *Modify Breakpoints* item from the popup menu. The following dialog will appear:



In this dialog you see all program units that have breakpoints, with the breakpoints listed below them. Each breakpoint has a checkbox that can be used to enable or disable it. The bottom section displays the following fields:

- **Use Condition** – The checkbox enables or disables the condition. The condition itself should be a boolean expression. When the breakpoint line is reached, execution will only be halted if the condition evaluates to True. You can use any SQL expression in the condition, and you can use

any variable that is known at the location of the breakpoint. These are the same variables that you can view or set during interactive debugging. Variables should be preceded with a colon. For example `upper ( :ename ) = ' SMITH '` is a valid condition if *ename* is a valid variable at the breakpoint location.

- **Use Message** – The checkbox enables or disables the message. When the breakpoint line is reached, and if the conditions are met, the message will be placed on the output page.
- **Don't Break** – This checkbox is only useful if you also define a message. When it is checked, execution is never halted on this breakpoint line. This way you can define breakpoints that only generate messages on the output page.
- **Use Pass Count** – This checkbox enables or disables the pass count, which defines how many times the breakpoint line must be passed before execution is halted. If, for example, you define a pass count of 10, execution will halt every 10th time the breakpoint line is reached.

## Viewing the call stack

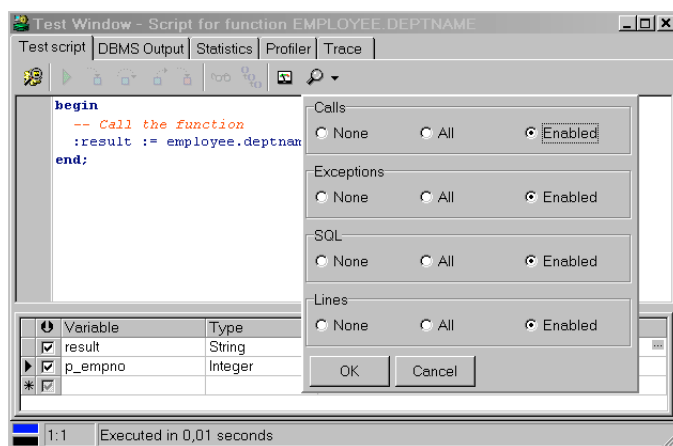
The call stack can be viewed at the bottom-right of the Test Window. It is automatically updated after each debug step.

## Debugger preferences

In the *Preferences* item in the *Tools* menu you can find a section of debugger related preferences. These preferences are described in detail in chapter 16.3.

## 4.11 Tracing execution

If you are using Oracle8i or later, you can use the Test Window's Trace facility to trace the execution of your PL/SQL code. You can configure which events you want to trace: Calls, Exceptions, SQL, or even every executed line of PL/SQL code. Press the *Select Trace Levels* button next to the *Create Trace report* button to bring up the following configuration screen:



You can control if you want to trace specific events in each program unit (*All*), only in those program units that are compiled with debug information (*Enabled*), or never (*None*).

To create a Trace report, simply press the *Create Trace report* on the toolbar of the Test Window and execute your Test Script. After execution you can switch to the *Trace* tab page to view the report, and to view reports of previous runs:

Comment	Event time	Unit name	Unit line	Unit line text	Exception
PL/SQL Virtual Machine started	19-10-2001 19:29:41				
Procedure Call	19-10-2001 19:29:41				
New line executed	19-10-2001 19:29:41	EMPLOYEE.DEPTNAME			
New line executed	19-10-2001 19:29:41	EMPLOYEE.DEPTNAME			
New line executed	19-10-2001 19:29:41	EMPLOYEE.DEPTNAME	9	if p_empno is null	
New line executed	19-10-2001 19:29:41	EMPLOYEE.DEPTNAME	15	select dept.dname into result	
SELECT DEPT.DNAME FROM DEPT	19-10-2001 19:29:41	EMPLOYEE.DEPTNAME	15	select dept.dname into result	
Exception raised	19-10-2001 19:29:41	EMPLOYEE.DEPTNAME	15	select dept.dname into result	1403
Exception handled	19-10-2001 19:29:41	EMPLOYEE.DEPTNAME	27	when no_data_found then return(null);	1403
New line executed	19-10-2001 19:29:41	EMPLOYEE.DEPTNAME	27	when no_data_found then return(null);	
New line executed	19-10-2001 19:29:41	EMPLOYEE.DEPTNAME	28	end;	
Return from procedure call	19-10-2001 19:29:41	EMPLOYEE.DEPTNAME	28	end;	
PL/SQL Virtual Machine started	19-10-2001 19:29:41				

1:1 Executed in 0,01 seconds

In this (simple) report you can see that a select statement on line 15 of function *employee.deptname* raised exception 1403 (No data found), which was handled on line 27 in the same program unit.

For each traced event, you can include information in the report like the event description, time, program unit, line number, exception, and so on. To configure this, press the *Preferences* button at the upper left of the Trace toolbar. This will bring up the Trace preferences page as described in chapter 16.5.

To view old trace reports, select a previous trace run from the *Run* selection list. Press the *Delete Run* button to delete the currently selected run.

For more information on the Oracle Trace facility, see the *DBMS\_TRACE* chapter in the *Oracle Supplied Packages Reference* manual.

## 4.12 Regression testing

To use Test Scripts for regression testing, you can use the Test Manager tool (see chapter 17.15) to define and run Test Sets based on Test Scripts.

## 5. Optimizing

To optimize the SQL statements in your program units, Oracle's Explain Plan utility can be a big help by showing the execution path of a statement. Therefore it is integrated into PL/SQL Developer's IDE.

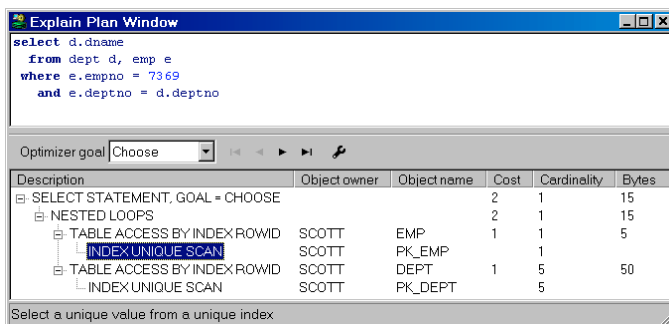
To view the actual resource use of a SQL statement or PL/SQL program unit, PL/SQL Developer can display statistics about its execution. You can configure which statistics you wish to display, and can include elapsed time, CPU time, logical reads, physical reads, physical writes, and so on.

To determine the execution time of each individual line of PL/SQL code, you can use the PL/SQL Profiler. This feature is not available on Oracle 8.0 or earlier.

Finally, you can use Oracle's *tkprof* utility to get resource use information about all executed SQL statements in a program unit by enabling SQL Trace.

### 5.1 Using the Explain Plan Window

To use Oracle's Explain Plan utility, press the *New* button on the toolbar and *select Explain Plan Window*. An empty Explain Plan Window appears. In the top half of the window, you can type the SQL statement you wish to analyze. After pressing the *Execute* button on the toolbar the execution plan is displayed in the bottom half of the window:



You can now change the SQL statement and press the *Execute* button again to see the impact of the changes. For more information about execution plans, you can read Oracle's *Server Tuning* manual.

To see the effect of different optimizer goals on the query plan, select the corresponding entry from the *Optimizer goal* list. The plan will immediately be updated to reflect these changes.

Use the *First*, *Previous*, *Next*, and *Last operation* buttons to navigate through the query plan in order of operation. After the plan has been determined, the first operation will be highlighted.

The Explain Plan utility uses a so-called plan table to store the execution plan. If such a table is not available to the current user, PL/SQL Developer will ask you if it should create this table in the schema of the current user. To define which columns from the plan table you want to see, and in which order, press the *Preferences* button. This will bring up the corresponding preference page (see chapter 16.21).

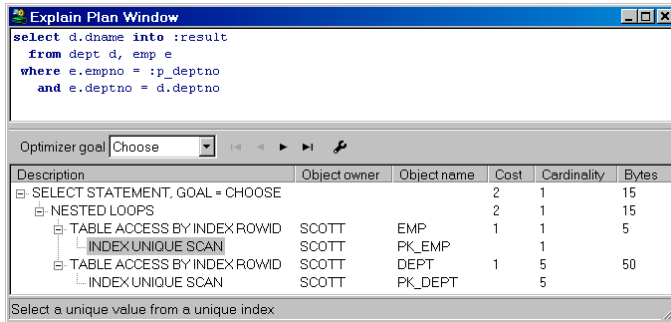
**Note:** If you are using Oracle Server 7.2 or earlier, the cost and cardinality are not available in the plan table. If you are using Oracle Server 7.3 or higher and cost and cardinality are not displayed, you probably need to upgrade the plan table.

If you right-click on the *Object name* column, this will bring up the popup menu for the selected object.

## Analyzing SQL in a program unit

Most of the time, the SQL statements you wish to analyze are contained in the source of a program unit. To do this, you can select the SQL statement in the Program Editor and select *Explain Plan* from the *Help* menu (or press F5). A new Explain Plan Window is created for the selected SQL statement, and all PL/SQL variables are replaced by bind variables.

If you explicitly copy & paste a SQL statement that contains PL/SQL variables from the program unit and you execute the Explain Plan window, you will receive an "ORA-00904: Invalid column name" error. The Explain Plan utility mistakenly assumes all these variables to be columns. Each variable must therefore be prefixed with a colon. If we wanted to analyze the SQL statement in the *employee.deptname* function, the *result* and *p\_empno* variables would have to be altered in the following way:



Now the Explain Plan utility knows which identifier is a column, and which identifier is a PL/SQL variable.

## 5.2 Automatic statistics

When you execute a statement in a SQL Window or in a Test Window, PL/SQL Developer will automatically generate a statistic report of this execution. One condition is that you need to have select privileges on the dynamic performance tables *v\$session*, *v\$statname* and *v\$sesstat* (provided through the standard *plustrace* role).

You can view the statistic report by changing to the Statistics tab at the top of the SQL Window or Test Window. The statistics for the execution of the *employee.deptname* function might look as follows:

The screenshot shows the 'Test Window - New' with the 'Statistics' tab selected. The table displays the following statistics:

Name	Last	Total
CPU used by this session	0	0
physical reads	5	12
physical writes	0	0
session logical reads	17	45
sorts (disk)	0	0
sorts (memory)	2	2
sorts (rows)	5	5
table fetch by rowid	1	6
table scan blocks gotten	3	5
table scan rows gotten	19	21
table scans (long tables)	0	0
table scans (short tables)	3	5

At the bottom, a status bar indicates: 'Executed in 0.084 seconds'.

For each statistic you see the value of the last execute, and the total for the current session. You can configure if and which statistics are displayed by setting a preference, as described in chapter 16.2. The default set of statistics is relevant to the tuning of your SQL and PL/SQL, and is described here:

Statistic	Meaning
CPU used by this session	The CPU usage in hundredths of a second
Physical reads	The number of blocks read from disk
Physical writes	The number of blocks written to disk
session logical reads	The number of blocks read from the block buffer or from disk
sorts (disk)	The number of sorts performed in a temporary segment
sorts (memory)	The number of sorts performed in memory
sorts (rows)	The number of rows that were sorted
table fetch by rowid	The number of rows fetched by rowid, usually as a result of index accesses
table scan blocks gotten	The number of blocks read for full table scans
table scan rows gotten	The number of rows read for full table scans
table scans (long tables)	The number of full table scans on long tables
table scans (short tables)	The number of full table scans on short tables

Which other statistics you can include depends on the version of the Oracle Server and are not described in this manual. If you wish to get information about them, there are many Oracle tuning books available that address this topic. The *Oracle Server Reference* also briefly describes these statistics.

The statistics can be exported to a CSV file (Comma Separated Values) that can be opened in a spreadsheet application later. Just right-click on the statistics, select the *Export* item and choose the *CSV file* item. You can alternatively select a TSV, XML or HTML format, or copy it to the clipboard.

## 5.3 The PL/SQL Profiler

The PL/SQL Profiler is a very powerful tool to help you optimize your PL/SQL code. For each executed line of code, the total time, maximum time, minimum time, average time, and the number of occurrences will be determined.

The Profiler is easily accessible in the Test Window. Before executing a Test Script, simply press the *Create Profiler report* button on the toolbar of the Test Window. If you subsequently execute the script, you can switch to the *Profiler* page to view the report.

The following example report shows that in the *employee.deptname* function, the select statement took 149 milliseconds, and the 3 *dbms\_output* calls took 57 milliseconds:

Unit	Line	Total time	Occurrences	Text
ANONYMOUS BLOCK	4	27	2	:result := employee.deptname(p_empno => :p_empno);
EMPLOYEE	9	1	1	1 if p_empno is null
EMPLOYEE	12	0	0	0 result := null;
EMPLOYEE	15	149	1	1 select dept.dname into result
EMPLOYEE	20	57	3	3 dbms_output.put_line(to_char(sysdate, 'dd-mm-yyyy hh24:mi:ss'))
EMPLOYEE	24	5	1	1 return(result);
EMPLOYEE	27	0	0	0 when no_data_found then return(null);
EMPLOYEE	28	4	1	1 end;
EMPLOYEE	34	0	0	0 if p_empno is null then
EMPLOYEE	36	0	0	0 result := null;
EMPLOYEE	39	0	0	0 select m.ename into result
EMPLOYEE	44	0	0	0 dbms_output.put_line(to_char(sysdate, 'dd-mm-yyyy hh24:mi:ss'))
EMPLOYEE	48	0	0	0 return(result);
EMPLOYEE	50	0	0	0 when no_data_found then return(null);
EMPLOYEE	51	0	0	0 end;

1:1 Executed in 0 seconds

By default the *Profiler* page will display the report of the last run. You can also select previous runs from the *Run* list. The *Unit* list allows you to zoom in on a specific program unit of a run.

The *Total time* column shows a graphical representation of the relative time of the line, compared to the line with the highest time. This allows you to quickly identify the lines that are most expensive. The report can be sorted by pressing on the sort-buttons in the heading of the columns.

If a source line is displayed in red, this means that the program unit has been changed since the profile report was created. Therefore the displayed line of code can now be different than when the profile was created.

You can change various layout aspects of the Profiler report by pressing the *Preferences* button. This will bring up the corresponding preference page, as described in chapter 16.6.

For more information about the PL/SQL Profiler, see the *dbms\_profiler* chapter in the “*Oracle8i Supplied Packages Reference*” manual.

Note: not all platforms provide equally accurate timing information.

## 5.4 SQL Trace

You can enable SQL Trace by pressing the *SQL Trace* button on the toolbar. After this, all server processing caused by a SQL Window or Test Window will be logged in a trace file on the database server. One condition is that the timed statistics parameter of the database instance you are using must be set to true.

You can disable SQL Trace by pressing the *SQL Trace* button again.

You can view the information in the trace file by using Oracle's *tkprof* utility on the database server. It will generate a report about elapsed time, CPU time, I/O, and so on for each SQL statement. Therefore, it can give you some insight into which statements are the most costly in a program unit. To learn more about Oracle's *tkprof* utility, you might read the “*Oracle 7 Server Tuning*” manual.

The *tkprof* report for the execution of the *employee.deptname* function might look as follows:



TKPROF: Release 7.2.2.3.1 - Production on Fri Sep 26 14:59:08 1997

Copyright (c) Oracle Corporation 1979, 1994. All rights reserved.

Trace file: ora07087.trc

Sort options: default

```
*****
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
*****
```

```
begin
  :deptname := employee.deptname(:empno);
end;
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.01	0.01	0	0	0	1
Fetch	0	0.00	0.00	0	0	0	0
total	2	0.01	0.01	0	0	0	1

Misses in library cache during parse: 0

Optimizer hint: CHOOSE

Parsing user id: 16

```
*****
SELECT DEPT.DNAME
FROM
  DEPT,EMP WHERE EMP.EMPNO = :b1 AND DEPT.DEPTNO = EMP.DEPTNO
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	5
Fetch	1	0.06	0.06	4	4	0	1
total	3	0.06	0.06	4	4	0	6

Misses in library cache during parse: 0

Optimizer hint: CHOOSE

Parsing user id: 16 (recursive depth: 1)

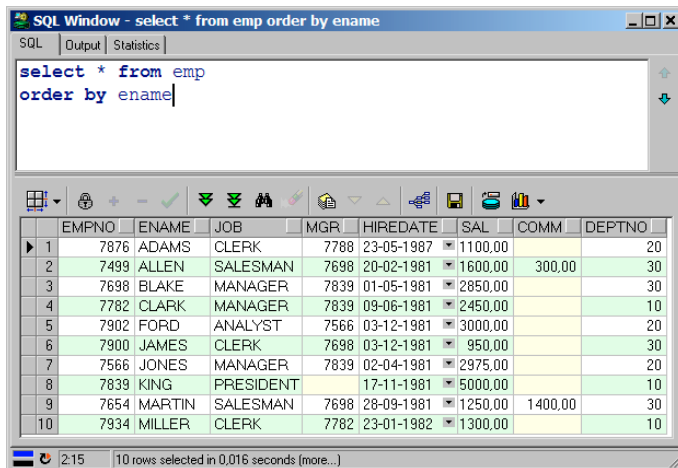
```
*****
```

## 6. *Ad hoc SQL*

Often during program development you need to execute some SQL statement. Either to test the SQL statement, to view data in a table, or to update data. You can do this from within PL/SQL Developer by using the SQL Window.

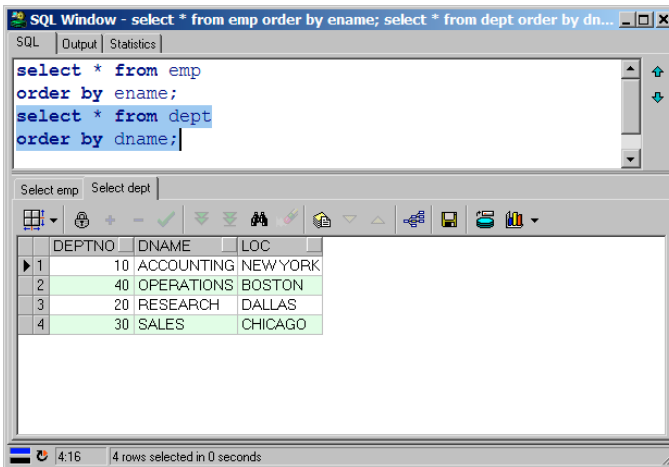
### 6.1 Using the SQL Window

To create a new SQL Window, press the *New* button on the toolbar and select *SQL Window*. An empty SQL Window appears, in which you can type a SQL statement. If it is a select-statement, the window splits into two parts and shows the results in a grid in the bottom half. If we wanted to view all the employees in the employee table, the SQL Window would look like this:



The SQL statement is executed and 10 rows have been retrieved. The result set is larger than 10 rows, which is indicated by the fact that both the *Next Page* and *Last Page* buttons on the result set toolbar are enabled, and the fact that *(more...)* is displayed on the status line. Only 10 are initially retrieved because this is the number of records that can be displayed on the grid. Pressing the *Next Page* will retrieve the next 10 rows, and so on. Pressing the *Last Page* button will retrieve all rows.

If you have multiple SQL statements in the SQL Editor and press the *Execute* button, all statements will be executed, and the results will be displayed in separate tab pages:



SQL statements must be terminated with a semi-colon or a slash, and PL/SQL blocks must be terminated with a slash, just like in SQL\*Plus. If you select a result tab, the corresponding text in the SQL Editor will be highlighted.

If you select a part of the text in the SQL Editor, only the selected text will be executed. This way you can have more than one statement in the editor, and still execute them one by one.

If you execute a SQL statement that takes a long time to finish you can press the *Break* button to abort it. Note that pressing the *Break* button will not always be successful. If for example the statement is waiting for a lock, it will not respond to a break signal. If you have pressed the *Break* button when the SQL Window is retrieving rows, it will simply stop and display the rows that have already been retrieved. You can now continue retrieving records with the *Next Page* and *Last Page* buttons.

Note that the SQL Window preferences allow you to determine how many rows are initially retrieved for a select statement. This is described in chapter 16.19.

At the right side of the window you see two buttons that allow you to navigate through all SQL statements that you have entered in the SQL Window. This way you can quickly re-execute statements entered previously.

For optimization purposes you can view the statistics of the execution of the SQL statement by selecting the *Statistics* tab. Statistics are explained in chapter 5.2.

When you print a SQL Window, the SQL statement and the result grid will be printed. By selecting rows in the result grid, you can limit the amount of rows that will be printed.

## 6.2 Result grid manipulation

The result grid of the SQL Window can be manipulated in various ways. Some cell types have special behavior, rows, columns, and cell ranges can be selected and printed, columns can be moved, rows can be sorted, you can switch to a single record view, and so on.

## Recognizing null values

Null values are displayed as a cell with a light yellow background, so that you can quickly distinct them from a value with all blanks. It is also helpful to recognize null values for cells that do not display the value directly in the grid, such as longs and LOB's. You can change the color of the null value cells through a preference (see chapter 16.19).

## Viewing large data columns

The values of Long, Long Raw, CLOB, BLOB and BFILE columns are not displayed in the result grid. Instead, they are simply displayed as <Long>, <Long Raw>, <CLOB>, <BLOB> and <BFILE>:

BCC_ADDRESS	S
<CLOB>	F
<CLOB>	C
<CLOB>	C

When you click on the cell button of such a column, the Large Data Editor is invoked, which allows you to view or edit the data in various formats. See chapter 19 for more details.

You can also click on the cell button of character columns to invoke the Large Data Editor. The cell button is only present if the column size is larger than 20 characters.

## Viewing date columns

A date column has a cell button that displays a calendar with the current date highlighted. If the date also has a time fraction, you can view its value on the calendar as well:



## Viewing timestamp columns

Timestamp columns are displayed in the format that is specified in the NLS\_TIMESTAMP\_FORMAT and NLS\_TIMESTAMP\_TZ\_FORMAT registry settings of your Oracle Home.

## Viewing XML data

There are various ways to store XML data in the database: as CLOB's (Oracle8i), as XMLTYPE (Oracle9i), or even as Varchar2 or Long columns. In any case the Large Data Editor will recognize the XML format if it starts with the standard XML header, and will switch to the XML format. See chapter 19 for more details.

## Viewing nested cursor columns

If you include a nested cursor in the field list of a select statement, the column value will initially be displayed as <Cursor>. Pressing the cell button will bring up a new SQL Window with a result grid with the cursor result set. This can be used to view simple nested queries. Note that each nested cursor value will implicitly result in an open cursor, so for large result sets you can easily run into the OPEN\_CURSORS limit and get "ORA-01000: maximum open cursors exceeded" errors.

## Selecting columns, rows and cells

To select rows or columns in the result grid, just click on the row heading or column heading and drag the mouse pointer to highlight the selection:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
1	7369	SMITH	CLERK	7902	17-12-1980	800.00
2	7499	ALLEN	SALESMAN	7698	20-02-1981	1600.00
3	7521	WARD	SALESMAN	7698	22-02-1981	1250.00
4	7566	JONES	MANAGER	7839	02-04-1981	2975.00
5	7654	MARTIN	SALESMAN	7698	28-09-1981	1250.00
6	7698	BLAKE	MANAGER	7839	01-05-1981	2850.00
7	7782	CLARK	MANAGER	7839	09-06-1981	2450.00
8	7788	SCOTT	ANALYST	7566	09-12-1982	3000.00
9	7839	KING	PRESIDENT	17-11-1981	5000.00	
10	7844	TURNER	SALESMAN	7698	08-09-1981	1500.00

A column selection can now be moved by releasing the mouse button, clicking on one of the selected column headings again, and dragging the selection to the new location.

To select a specific range of cells, move the mouse pointer over the left edge of a cell until its shape changes, press the mouse button, and drag the mouse to highlight the selection:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
1	7369	SMITH	CLERK	7902	17-12-1980	800.00
2	7499	ALLEN	SALESMAN	7698	20-02-1981	1600.00
3	7521	WARD	SALESMAN	7698	22-02-1981	1250.00
4	7566	JONES	MANAGER	7839	02-04-1981	2975.00
5	7654	MARTIN	SALESMAN	7698	28-09-1981	1250.00
6	7698	BLAKE	MANAGER	7839	01-05-1981	2850.00
7	7782	CLARK	MANAGER	7839	09-06-1981	2450.00
8	7788	SCOTT	ANALYST	7566	09-12-1982	3000.00
9	7839	KING	PRESIDENT	17-11-1981	5000.00	
10	7844	TURNER	SALESMAN	7698	08-09-1981	1500.00

To select all rows and columns, press the upper left cell, or right-click on the grid and select the *Select All* item from the popup menu.

The highlighted selection can be copied or printed as usual.

## Exporting data

There are several ways to export the data in the result set grid. After executing a select statement, you can select a range of cells as described above, right-click on it, and select the *Export Results* item from the popup menu. This will display a submenu where you can choose to export the data in CSV format (Comma Separated Values), TSV format (Tab Separated Values), HTML format, XML format, or SQL format. After selecting the format, you can specify the export file. The SQL format will generate a SQL file with insert statements for the queried table or view.

You can alternatively copy the selection to the clipboard by pressing *Ctrl-C*, or by right-clicking on the selection and selecting the *Copy* or *Copy with Header* item from the popup menu. You can subsequently paste this data in another application like a spreadsheet, word processor, and so on.

To quickly manipulate the result set information in Microsoft Excel, select the *Copy to Excel* item. This will open a new Excel instance, and all selected data will be copied.

## Sorting rows

To sort the rows in a result grid, press the heading button of the column on which you want the rows to be sorted:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	
8	7788	SCOTT	ANALYST	7566	09-12-1982	3000.00	
1	7369	SMITH	CLERK	7902	17-12-1980	800.00	
11	7876	ADAMS	CLERK	7788	12-01-1983	1100.00	
4	7566	JONES	MANAGER	7839	02-04-1981	2975.00	
6	7698	BLAKE	MANAGER	7839	01-05-1981	2850.00	
7	7782	CLARK	MANAGER	7839	09-06-1981	2450.00	
9	7839	KING	PRESIDENT		17-11-1981	5000.00	
2	7499	ALLEN	SALESMAN	7698	20-02-1981	1600.00	
3	7521	WARD	SALESMAN	7698	22-02-1981	1250.00	
10	7844	TURNER	SALESMAN	7698	08-09-1981	1500.00	

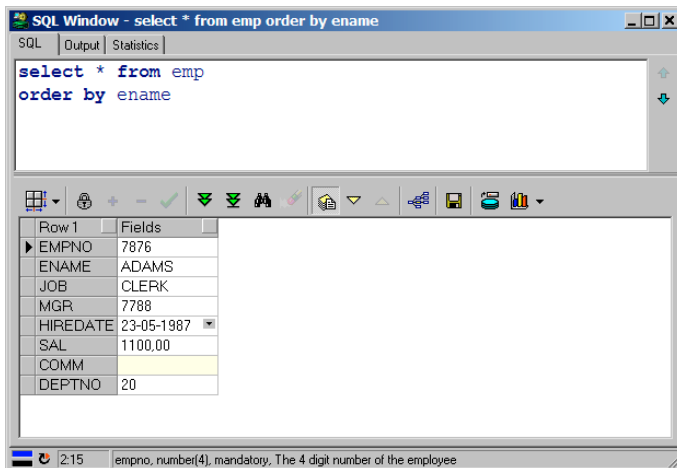
The rows will be sorted in ascending order, as indicated by the heading button. Pressing the heading button again will sort the rows in descending order. Pressing it a 3rd time will undo the sorting. Pressing the heading button in another column will sort the rows on this column, but will also use the previous sort column as the secondary sort column (indicated by the dot in the heading button). In the example above, the *job* column is the primary sort column, and the *hiredate* column is the secondary sort column.

Note that sorting is performed locally, and only for the rows that are already retrieved. If you retrieve additional rows after sorting the results, these new rows will be added at the end of the result grid, without any sorting. For large result sets, local sorting can take a long time. In this case it might be better to use an *order by* clause in the select statement and let the Oracle Server do the sorting.

## Single Record View

If the result set contains many columns, it may be inconvenient that each record is displayed on a single line. You need to scroll back and forth to view related columns (though you can move the columns), and cannot view all the columns of the record at once.

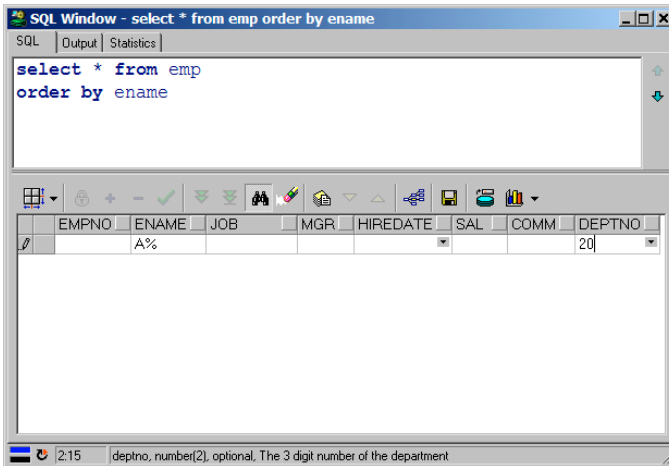
By pressing the *Single Record View* on the grid toolbar, you can view a single record at a time:



Now each row displays a single column name and value. The *Next Record* and *Previous Record* on the grid toolbar can be used to navigate through the result set. To switch back to the Multi Record View, press the *Single Record View* button again.

## 6.3 Query By Example mode

After executing a select statement, you can press the *Query By Example* button above the result grid if you want to search for specific records. Pressing this button will clear the grid, and leave one empty record where you can enter your query values. If, for example, you want to find all employees that start with the letter A in department 20, enter A% in the *ename* column and 20 in the *deptno* column:



Pressing the *Query By Example* button again, or pressing the *Execute* button, will execute the modified query and display the restricted results. Press the *Query By Example* button again to continue with the previous query values. Press the *Clear record* button to clear the query values.

Query values are not restricted to single values with wildcard characters. You can use the following expressions (alternative expressions between square brackets):

- value [= value]
- value with wildcards [like value with wildcards]
- > value
- < value
- != value [<> value]
- in (value1, value2, ...)
- between value1 and value2
- null [is null]
- not null [is not null]

Note that if you use a literal value, you can omit quotes for character values (e.g. SMITH will automatically be converted to 'SMITH' in the SQL text). For all other expressions you must provide a literal value that the Oracle Server understands (e.g. != 'SMITH'). This is also the case for number and date values. If you use it in an expression, use a format that the Oracle Server understands.

Several preferences exist that allow you to control the default behavior of the Query By Example functionality (e.g. case sensitivity). Chapter 16.19 describes these preferences.

## 6.4 Linked Queries

When viewing a result set, you often want to query a related table. For example, when viewing the *dept* table, you may want to query all employees of a department. Or vice versa: when viewing the *emp* table, you may want to query an employee's department.

Most of the time you will have foreign key constraints between these parent and child tables, in which case the SQL Window can automatically generate and execute these queries for you. If you are located on a specific record in the result set, press the *Linked Query* button on the grid toolbar. This will display a popup menu with all parent and child tables of the table of the current select statement. If, for example, you were querying the *emp* table, the following popup menu might appear:

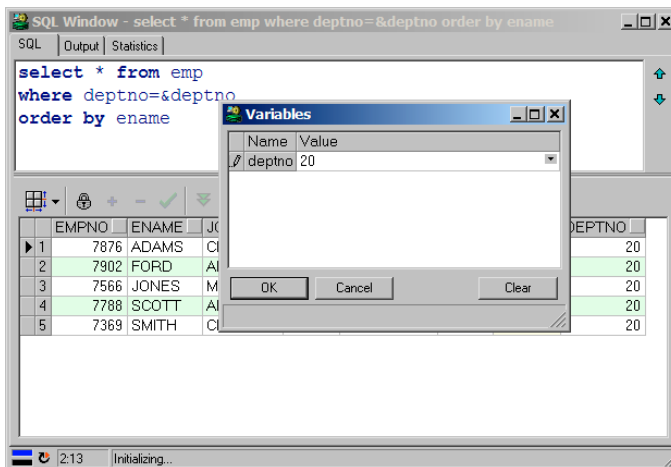
MGR	HIRE	dept (emp_dept_fk)	DEPTNO
7788	12-01	emp (emp_mgr_fk)	20
7698	20-02	emp (emp_mgr_fk)	30
7839	01-05	emp (emp_mgr_fk)	30
7839	09-06-2081		10

The top section displays the parent tables (and the foreign key names), and the bottom section displays the child tables. The *dept* item will generate a query for this employee's department (30). The first (parent) *emp* item will generate a query for this employee's manager (7839). The second (child) *emp* item, will generate a query for all employees that are managed by this employee.

The linked query will be executed in the same SQL Window, unless you enable the SQL Window preference *Linked Query in New Window* (see chapter 16.19). If you hold down the *Ctrl* key while pressing the *Linked Query* button, the opposite of this preference will occur.

## 6.5 Substitution variables

You can use substitution variables in your SQL text to allow user input when the query is executed. The most simple form of an substitution variable is similar to the way you may be used to from SQL\*Plus:



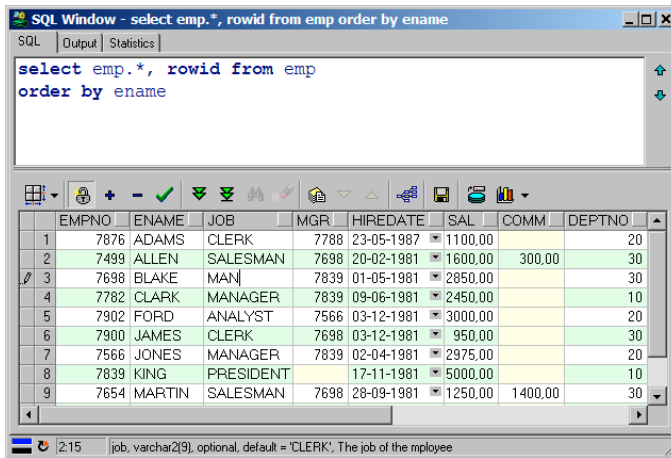
By specifying *&deptno* in the SQL text, you will be prompted for a value for this variable. The variable reference in the SQL text will be substituted by this value before execution. You can specify more than one substitution variable, and you can specify a single substitution variable more than once.



Furthermore you can define the data type, default value, selection lists, list queries, checkboxes, and other advanced options. These features are also used in the Report Window, and are described in detail in chapter 10.

## 6.6 Updating the database

To update, insert or delete records in the database, you can execute the appropriate DML statements in a SQL Window. It is probably much more convenient to make a result grid updateable by including the rowid in the select list or use a *select ... for update* statement:



You should be aware that a *select ... for update* statement will lock all selected records, so including the rowid might generally be the best way to make a result grid updateable. If the select statement is a join, columns of the first table can be updated. All other columns will be read-only. Columns that are given an alias cannot be updated either.

If the result grid is updateable, you can press the *Edit data* button at the right side of the result grid and edit the records. You can insert or delete records on the grid by pressing the *Insert record* or *Delete record* button. If you have selected multiple records, all selected records will be deleted.

Modifying the result grid does not actually change anything in the database. To post the updated, inserted or deleted records to the database press the *Post changes* button. If you have the *AutoCommit SQL Window* option disabled, the *Commit* and *Rollback* button on the toolbar will be enabled if a transaction has been started. For more information about transactions, see chapter 14.

### Editing large data columns

As explained in the previous chapter, you can click on the cell button of a Long, Long Raw, CLOB, BLOB and BFILE columns to view the value in various formats. When the result grid is updateable, you can use the Large Data Editor to change a column value.

## 6.7 Viewing and editing XMLTYPE columns

The SYS.XMLTYPE type is not supported by Oracle Net 9.0 and earlier, so you will not be able to directly access the XML data that is stored inside such a column. If, for example, *xml\_text* is a SYS.XMLTYPE column, then the following query will not provide the expected result:

```
select id, xml_text from xml_table
```

Only the *id* column will show up in the result set. To view the XML data, use the *getclobval()* member function:

```
select id, t.xml_text.getclobval() from xml_table t
```

This way the CLOB can be viewed in the Text Editor, and XML syntax highlighting will automatically be applied. To edit the XML data, make the result set updateable as usual by including the rowid:

```
select id, t.xml_text.getclobval(), rowid from xml_table t
```

Now you can edit the CLOB and post the modified data to the database. Note that the *View Data* and *Edit Data* functions for tables and views will automatically apply these rules, so the easiest way to view or edit XMLTYPE columns is to right-click on the table or view, and selecting the corresponding items from the popup menu.

## 6.8 Direct Query Export

For queries with large result sets (ten thousands of records or more) it may be inconvenient to first query the data into the result grid, to subsequently export it to a file. This may take a long time and a lot of memory resources. In this case it is much more efficient to directly write the result set into an export file. To do so, you can press the *Export Query Results...* button on the result set toolbar, instead of the *Execute* button on the main toolbar. This will display a popup menu where you can choose to export the data in CSV format (Comma Separated Values), TSV format (Tab Separated Values), HTML format, or XML format. After selecting the format, you can specify the export file, after which the query will be executed. The result set will not be displayed in this situation, but will only be written to the export file.

## 6.9 Saving SQL Scripts

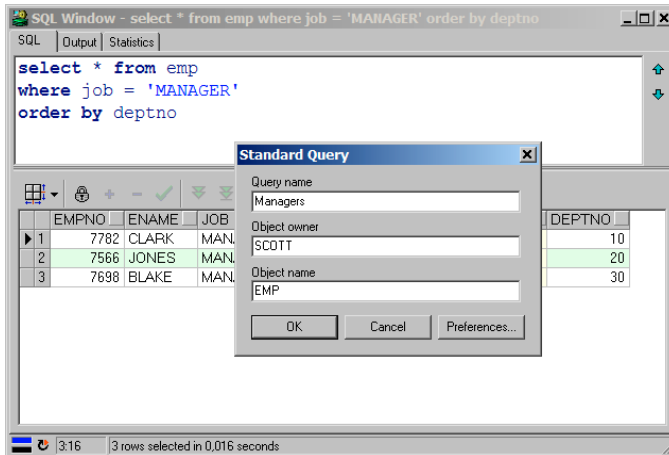
If you have created a SQL statement that you wish to re-execute later, you can save it to a SQL Script. To do so, press the *Save* button on the toolbar and enter an appropriate filename in the save dialog. The default extension for a SQL Script is *.sql*, though you can change the default extension with a preference described in chapter 16.27. Note that only the current SQL statement is saved to the file, not the entire history of statements. To include the history in the saved file, right-click on the SQL Window and select the *Save with history* item. All statements will be separated with a slash in that case.

You can open a previously saved SQL Script by pressing the *Open* button on the toolbar and selecting *SQL Script*, which will create a new SQL Window. You can alternatively right-click on a previously created SQL Window and select the *Load* item, which will open a SQL Script in the existing SQL Window.

## 6.10 Creating Standard Queries

For most tables and views you will have various standard queries that you frequently need to run. For example: “show all employees that are managers”, or “show the sum of the salary of all employees, ordered by department number”. You can save such a query as a so called “Standard Query”, which can easily be invoked later from the popup menu for the corresponding table or view. This popup menu will appear when right-clicking on the table or view in the Object Browser or in a PL/SQL or SQL source.

After creating a standard query in the SQL Window, you can press the *Save as Standard Query* button on the toolbar of the result grid. The following dialog will appear:



You can enter the name of the query, which will be displayed in the popup menu. You can also change the owner and name of the table or view for which the standard query will be displayed. Press the *Preferences* button to change the directory where the standard queries will be stored.

The following information will be saved with the standard query:

- The SQL text
- The size of the SQL Window
- The grid mode: multi record or single record view

Note that you can use substitution variables (see chapter 6.5) to make the standard queries more flexible.

## 7. The Command Window

The Command Window allows you to execute SQL scripts in a way that is very much similar to Oracle's SQL\*Plus. To create a Command Window press the *New* button on the toolbar or select the *New* item in the *File* menu. A Command Window is created and you can type SQL and SQL\*Plus commands like you are used to, without leaving PL/SQL Developer's IDE:

The screenshot shows the 'Command Window - New' interface with tabs for 'Dialog' and 'Editor'. The 'Editor' tab is active, displaying the following SQL commands and their results:

```
SQL> select * from emp
2  where deptno = 20
3  order by ename;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	12-01-1983	1100,00		20
7902	FORD	ANALYST	7566	03-12-1981	3000,00		20
7566	JONES	MANAGER	7839	02-04-1981	2975,00		20
7788	SCOTT	ANALYST	7566	09-12-1982	3000,00		20
7369	SMITH	CLERK	7902	17-12-1980	800,00		20

```
SQL> exec dbms_output.put_line(employee.deptname(7876))
RESEARCH

PL/SQL procedure successfully completed

SQL> desc dep
Object dep does not exist.

SQL> desc dept
Name      Type          Nullable Default Comments
-----
DEPTNO    NUMBER(2)      Yes       The 2-digit identification number of the department
DNAME     VARCHAR2(14)   Yes       The name of the department
LOC       VARCHAR2(13)   Yes       The location of the department

SQL> |
```

The status bar at the bottom shows: `ECH TRM FDB VER APR HDG TMG` and `PL/SQL procedure successfully completed in 0 seconds`.

### 7.1 Entering SQL statements and commands

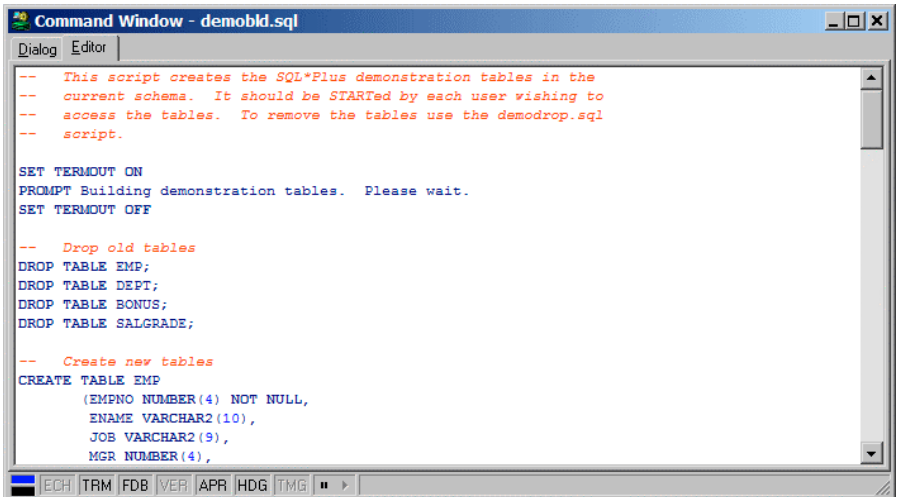
Just like in SQL\*Plus, you can type SQL statements across multiple lines, and end a statement with a semi-colon or slash. You can use the cursor left and right keys to edit the command line, and use the cursor up and down keys to recall previously entered lines

By entering the *edit* command, you can use a simple text editor to edit the entire input buffer. After editing the buffer, you can execute it by entering a slash on the command line. The editor has a history buffer with all previously executed commands, so that you can quickly execute a modified command.

The status line of the Command Window shows the status of the Echo, Termout, Feedback, Verify, Autoprint, Heading and Timing options. You can additionally double-click on such an option to change it between on and off.

## 7.2 Developing command files

To develop a command file with multiple SQL statements and commands, you often need to edit the file, run it, edit it again, run it again, and so on. To make this a comfortable process, the Command Window has a built-in editor with SQL, PL/SQL and SQL\*Plus syntax highlighting:



### Opening, Executing, and Saving a command file

To create a command file, switch to the *Editor* page and enter the commands. To execute the commands in the editor, simply press the *Execute* button on the toolbar, or press *F8*. The Command Window will switch back to the *Dialog* page and will execute all commands.

To edit an existing command file, press the *Open* button on the toolbar and select the *Command File* item. This will load the command file into the editor of a new Command Window. You can alternatively right-click in the editor and select the *Load* item from the popup menu. To save a modified Command File, press the *Save* button on the toolbar.

### Stepping through a command file

If you press the *Pause Execution* button on the status line before or during execution, you can single-step through the script in a controlled manner. The next command will be highlighted in the Editor, and will be executed when you press the *Execute Next Command* button on the status line. If you press the *Pause Execution* button again, execution will continue normally after the next step. You can force such a pause from within a script by using the *SET EXECPAUSE* command.

## 7.3 Supported commands

Besides all SQL statements, the command window supports the following standard SQL\*Plus commands in this release:

Command	Meaning
/	Executes the SQL buffer
? [Keyword]	Provides SQL help on the keyword
@ [ @ ] [Filename] [Parameter list]	Runs the specified command file, passing the specified parameters
ACC[EPT] Variable [DEF[AULT] Value] [PROMPT Text] [NOPR[OMPT]]	Allows the user to enter the value of a substitution variable
CL[EAR] [SCR[EEN]]	Clears the screen
CL[EAR] SQL	Clears the SQL buffer
COL[UMN] [Column] [Format] [NEW_VALUE Variable]	Defines the format of a column, displays the format of a column, or displays all column formats
CONNECT [username/password@database]	Connects to the database with the specified user
DEF[INE] [Variable] [= Text]	Defines a substitution variable, displays a variable, or displays all substitution variables.
DESC[RIBE] Object	Gives a description of the specified object
DISC[ONNECT]	Disconnects from the database
EDIT	Displays a text editor to edit the SQL buffer
EXEC[UTE] Procedure	Executes the specified procedure
EXIT [APPLICATION]	Quits a running script or closes the Command Window. Adding the APPLICATION parameter will also close PL/SQL Developer.
GET [Filename]	Loads a command file into the editor
HOST [Command]	Executes the host command
HELP [Keyword]	Provides SQL help on the keyword
PAUSE [Message]	Displays the message and pauses until the user presses Okay or Cancel
PRI[NT] [Variable]	Displays the value of the bind variable, or all bind variables
PROMPT [Text]	Displays the specified text
QUIT [APPLICATION]	Quits a running script or closes the Command Window. Adding the APPLICATION parameter will also close PL/SQL Developer.
R[UN]	Executes the SQL buffer
REM[ARK] [Text]	A comment line
SET AUTOP[RINT] [ON   OFF]	Determines if bind variables are automatically displayed after executing a SQL statement or PL/SQL block.
SET COLSEP [Separator] [OFF]	Determines the column separator (default = " ").
SET CON[CAT] [Character] [ON   OFF]	Determines the character that terminates a substitution variable reference (default = .)
SET DEF[INE] [Character] [ON   OFF]	Determines the character that starts a substitution variable reference (default = &)
SET ECHO [ON   OFF]	Determines if executed commands in a script are displayed
SET ESC[APE] [Character] [ON   OFF]	Determines the character that escapes the character that starts a substitution variable reference (default = \)
SET FEED[BACK] [ON   OFF]	Determines if the number of affected rows of a SQL statement is displayed
SET HEA[DING] [ON   OFF]	Determines if headings are displayed above the columns of a result set
SET LONG [Width]	Determines the maximum display width of a long column
SET NUM[WIDTH] [Width]	Determines the maximum display width of a number column without precision
SET PAGES[IZE] [Size]	Determines the number of lines that are displayed for a result set, before the headings are repeated
SET PROMPT [IPrompt]	Replace the standard SQL> prompt. Instead of a literal text

Command	Meaning
	you can also use the variables [user], [db], or [connection]. Furthermore you can include a bind variable (set prompt :bind_var_name).
SET SCAN [ON   OFF]	Determines if substitution variables should be scanned
SET SERVEROUT[PUT] [ON   OFF] [SIZE n]	Determines if output of calls to dbms_output.put_line is displayed, and what the size of the output buffer is
SET SPOOL*DIRECTORY [Directory]	Determines in which directory spool files are stored if the SPOOL command does not specify an absolute path.
SET TERM[OUT] [ON   OFF]	Determines if output of executed SQL statements is displayed
SET TIMI[NG] [ON   OFF]	Determines if timing information about executed SQL statements is displayed
SET VER[IFY] [ON   OFF]	Determines if substitution variables are displayed when used in a SQL statement or PL/SQL block
SHO[W] ERR[ORS] [Type Name]	Displays errors for the previous compilation, or for the specified object
SHO[W] REL[EASE]	Displays Oracle release information for the current connection
SHO[W] SQLCODE	Displays the result code of the executed SQL statement
SHO[W] USER	Displays the username of the current connection
SPO[OL] [Filename] [OFF]	Starts or stops spooling
STA[RT] [Filename] [Parameter list]	Runs the specified command file, passing the specified parameters
STORE SET [Filename]	Stores the values of all options in the filename. You can execute this file later to restore these options.
UNDEF[INE] Variable	Undefines the given substitution variable
VAR[iable] [Variable] [Datatype]	Defines a bind variable, displays a bind variable, or displays all bind variables.
WHENEVER [OSERROR   SQLERROR] [Action]	Specify an action whenever an OS error or SQL error occurs. The action can either be EXIT or CONTINUE, optionally followed by COMMIT or ROLLBACK.

All of these commands function the same as in SQL\*Plus. The following commands are specific to PL/SQL Developer:

Command	Meaning
BEAUT[IFY] File   Object	Beautifies the specified file or database object, using the current rules or the rules specified with the SET BEAUTIFIERRULES command.
BROWSE Object	Select the Object in the Object Browser
EDIT Object	Opens an editable window with the object's definition
EDITD[ATA] Table   View	Opens a SQL Window for the table or view with an editable result set
EXPORT[DATA] Table	Opens the Export Tool for the specified table
INFO	Displays information about the connection
PROPERTIES Object	Displays a Property Window for the specified object
QUERY[DATA] Table   View	Opens a SQL Window for the table or view with a read-only result set
REC[OMPILE] Object	Recompiles the object
SET BEAUT[IFIERRULES] [File]	Temporarily use the beautifier rules from the specified file. This can be used with the BEAUTIFY command.
SET COL[WIDTH] [Width]	Determines the maximum column width in a result set. If Width = 0, the width is unlimited. The default is 80.
SET EXEC[PAUSE] [ON   OFF]	Pauses execution at the next command (ON), or continues normally with the next command (OFF).
SQLPLUS	Invokes SQL*Plus with the current file.
TEST ProgramUnit	Opens a Test Window with a standard Test Script for the

	specified program unit
VIEW Object	Opens a read-only window with the object's definition



## 8. Creating and modifying non-PL/SQL objects

During PL/SQL development you may find that you often need to create a table, modify a constraint or index, reset a sequence, and so on. PL/SQL Developer has several functions that allow you to create and modify tables (and related elements), sequences, synonyms, libraries, directories, users, and roles. This chapter does not explain the functionality of these objects, but merely explains how you can create, modify and view their properties. For more information about the each object's functionality, see the Oracle documentation like the "*SQL Reference Guide*".

To create an object, you can either press the *New* button on the toolbar and select the corresponding object type, or select the *New* item in the *File* menu. You can also right-click on the root folder of the object type in the Object Browser and select the *New* item from the popup menu.

To modify an object, select it in the Object Browser, right-click on it, and select the *Edit* item from the popup menu. You cannot change the name of an object like this, but must explicitly use the *Rename* function from the Object Browser to accomplish this (if the object type supports it). If you just want to view the object, select the *View* item from the popup menu.

You can additionally duplicate an object by selecting the *Duplicate* item in the popup menu of the Object Browser. The object's editor window will appear with all properties filled in, except for the owner (if applicable) and name. This way you can quickly create a similar object to explore an alternative, perform some risky or destructive test, or whatever.

### 8.1 The table definition editor

The table definition editor has 7 tab pages for various aspects of a table:

**Edit table EMP**

General | Columns | Keys | Checks | Indexes | Privileges | Partitions

Owner: SCOTT  
Name: EMP

☒ Recreate table  
Warning: this will delete all data, triggers, and foreign key references

**Storage**

Tablespace: SYSTEM  
Initial Extent: 12 KB  
Next Extent: 12 KB  
%Free: 10  
%Used: 40  
%Increase: 50  
Ini Trans: 1  
Min Extents: 1  
Max Trans: 255  
Max Extents: 249 ☐ Unlimited

**Cluster**

Name:   
Columns: ...

**Duration**

☐ Temporary ☐ Preserve rows on commit

**Organization**

☒ Heap ☐ Index

Comments:

Apply Refresh Close Help Query... View SQL

At the bottom of the editor, you see the following 6 buttons:

- **Apply** – Will apply all modifications you have made in the editor to the database.

- **Refresh** – Retrieves the definition from the database again, discarding any changes you may have made.
- **Close** – Closes the editor window.
- **Help** – Shows the online help.
- **Query** – Invokes a SQL Window with a query that allows you to view and edit the table data.
- **View SQL** – Displays a text editor with the SQL statements that have resulted from the changes made in the editor. If you have not yet made any changes, it will show the complete creation SQL.

The following chapters will describe the different pages of the table definition editor.

## General page

The *General* page is displayed in the previous chapter, and contains the table owner and name, storage information, cluster information, and comments. For a new table you can leave all properties empty, except for the name. All other properties will get a default value:

- Owner – the current user that is logged on.
- Tablespace – the default tablespace for the current user.
- %Free – 10
- %Used – 40
- Initial transaction entries – 1
- Maximum transaction entries – 255

The default values of the segment properties (initial extent, next extent, %increase, minimum extents and maximum extents) depend on the corresponding defaults of the tablespace.

The cluster information, storage information, and duration properties are mutually exclusive, because the cluster implicitly defines the storage characteristics of the table, and temporary tables cannot be clustered and also have implicit storage characteristics.

If you enter or select a cluster name, the storage properties and duration properties will become read-only, and the cluster columns can be entered. Clearing the cluster name has the opposite effect.

If you define the table as temporary, the cluster and storage information will become read-only, and you can define if rows are preserved after a commit.

If you define an index organization table, you must define a primary key. For a heap organized table (the default) this is not a requirement.

If you are modifying an existing table, not all properties can be changed. The tablespace, initial extent, minimum extents, cluster information, duration properties and organization properties cannot be changed for a table that already exists in the database. If you want to change any of these properties, you need to check the *Recreate table* option at the top of the window. As a result, the table will first be dropped and recreated with the new definition. All data, triggers, and foreign key references will be lost, so you should be careful when using this option!



## Keys page

On the *Keys* page you can view, add, delete, and modify the table's primary, unique and foreign key constraints:

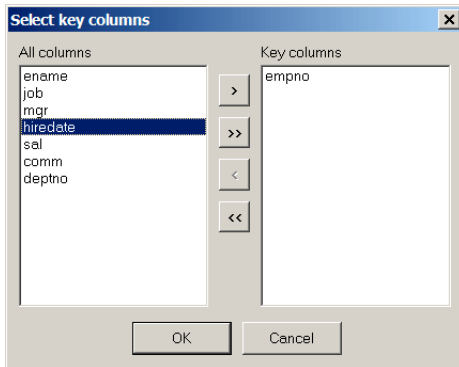
Name	Type	Columns	Enabled	Referencing table	Referencing columns	On Delete	Deferrable	Deferred
EMP_FK	Primary	EMPNO	<input checked="" type="checkbox"/>				<input type="checkbox"/>	<input type="checkbox"/>
EMP_DEPT_FK	Foreign	DEPTNO	<input checked="" type="checkbox"/>	DEPT	DEPTNO	No action	<input type="checkbox"/>	<input type="checkbox"/>

This grid can be manipulated in the same way as the column grid in the previous chapter. You can add, delete and modify the key constraints. Moving the constraints will not have any effect, as they are not really ordered.

Note that the storage information of the underlying index that is used to enforce a primary or unique key constraint is not defined on this page. If the constraint is enabled, there will automatically be a corresponding index on the *Indexes* page with the same name. For this index you can define the storage definition. Disabling or deleting a primary or unique key constraint will implicitly delete the underlying index.

The *Type* column has a list that allows you to select just the Primary, Unique or Foreign key type. You can quickly select the correct type by typing the first character (P, U or F). Note that a table can only have one primary key constraint, and the default value for this column (Primary or Unique) depends on this. Changing the type will affect the *Referencing table*, *Referencing columns* and *Cascade columns*, as they are only valid for foreign key constraints. Changing the type to Foreign will also implicitly delete the index.

To define the *Columns* to which the constraint applies, you can simply type the column names in the appropriate cell (separated by commas), or press the cell button. This will invoke a column selection screen where you can easily add, remove or move (reorder) the key columns:



For foreign key constraints you can type or select a *Referencing table*. Doing so will automatically select default *Referencing columns*, based on the primary and unique key constraints of this table. The selection list for this column shows all column sets that make up primary or unique key constraints for the referencing table, and are therefore the only candidates here. In the *On Delete* listbox you can define what action should be performed for the records in the child table when the parent record is deleted. The *Set null* action is supported in Oracle8i and later.

If you are using Oracle8, you can additionally define if the constraint is deferrable, and if it is initially deferred. These 2 options are obviously related: you cannot have a non-deferrable constraint that is initially deferred.

## Checks page

On the *Checks* page you can view, add, delete, and modify the table's check constraints:

Name	Condition	Enabled	Deferrable	Deferred	Last change
▶ EMP_COMM_CK1	comm >= 0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	5-6-2004 17:05:03
EMP_MGR_CK1	mgr <> empno	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	5-6-2004 17:05:03
EMP_SAL_CK1	sal >= 0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	5-6-2004 17:05:03
*		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Buttons: Apply, Refresh, Close, Help, Query..., View SQL

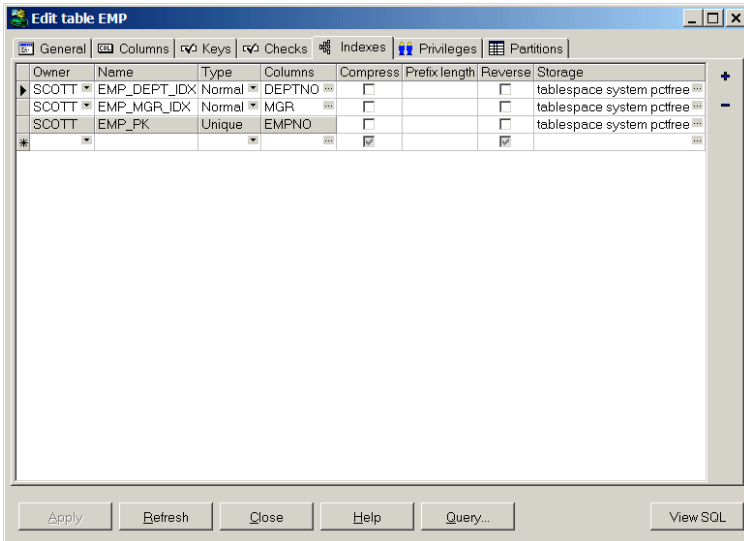
This grid can be manipulated in the same way as the column grid. You can add, delete and modify the check constraints.

The *Condition* column should contain the boolean expression that must be true (or null) for all rows in the table. You cannot apply a check constraint if one or more rows violate the condition, unless you also uncheck the *Enable* option. The check constraint can be enabled after correcting the violating rows.

If you are using Oracle8 or later, you can additionally define if the constraint is deferrable, and if it is initially deferred. These 2 options are obviously related: you cannot have a non-deferrable constraint that is initially deferred.

## Indexes page

On the *Indexes* page you can view, add, delete, and modify the table's indexes:



This grid can be manipulated in the same way as the column grid. You can add, delete and modify the indexes.

The *Owner* of the index can be left empty, the default owner is the user that is currently logged on. If you want to use a different owner for the index, just select it from the list.

The *Type* of the index can be Normal, Unique, or Bitmap. The last index type is only available on Oracle7.3 Servers and later. Note that you can only create a unique index if the index columns of each row in the database are indeed unique. As long as this is not the case, you cannot apply this index to the database.

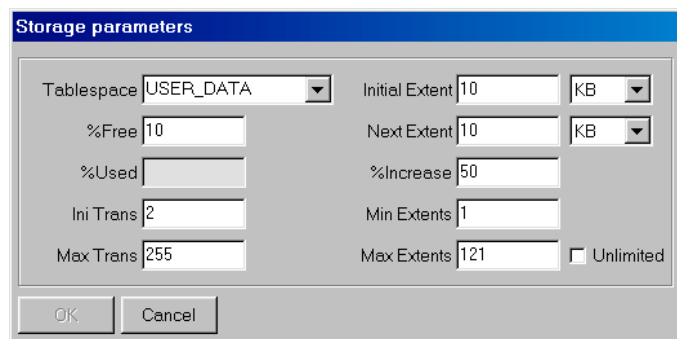
To define the *Columns* that you want the table to be indexed by, you can simply type the column names in the appropriate cell (separated by commas), or press the cell button. This will invoke a column selection screen where you can easily add, remove or order the index columns.

For function based indexes you can specify the functions, separated by commas. For descending indexes, you can follow the column name by *DESC*.

You can additionally enable key compression for the index, including the number of prefix columns to compress, and define the index as a reverse index.

If the index is for a primary or unique key then the owner, name, type, and columns cannot be modified. These properties are all derived from the key constraint. Changing the name of the key constraint will automatically change the name of the corresponding index, changing the constraint columns will change the corresponding columns for the index. The owner will always be the same as the owner of the table, and the type of the index will always be unique.

The *Storage* parameters can be defined or viewed by pressing the cell button. This will invoke a storage editor:

The image shows a 'Storage parameters' dialog box with a blue title bar. It contains two columns of input fields. The first column has: 'Tablespace' (a dropdown menu showing 'USER\_DATA'), '%Free' (a text box with '10'), '%Used' (an empty text box), 'Ini Trans' (a text box with '2'), and 'Max Trans' (a text box with '255'). The second column has: 'Initial Extent' (a text box with '10' and a 'KB' dropdown), 'Next Extent' (a text box with '10' and a 'KB' dropdown), '%Increase' (a text box with '50'), 'Min Extents' (a text box with '1'), and 'Max Extents' (a text box with '121' and an 'Unlimited' checkbox). At the bottom are 'OK' and 'Cancel' buttons.

Storage parameters	
Tablespace	USER_DATA
%Free	10
%Used	
Ini Trans	2
Max Trans	255
Initial Extent	10 KB
Next Extent	10 KB
%Increase	50
Min Extents	1
Max Extents	121 <input type="checkbox"/> Unlimited

For new indexes you do not need to enter any of this information, as each property has a default value:

- Tablespace – the default tablespace for the current user.
- %Free – 10
- Initial transaction entries – 2
- Maximum transaction entries – 255

The default values of the segment properties (initial extent, next extent, %increase, minimum extents and maximum extents) depend on the corresponding defaults of the tablespace. The %Used property does not apply to indexes.

Note that if you modify the %free, initial extent, and minimum extents properties of an existing index, this will implicate that the index (and constraint) will be dropped and recreated to apply this change. For large indexes, this may be time consuming.



## Privileges page

On the *Privileges* page you can grant/revoke privileges on the table to/from users and roles:

Grantee	Select	Insert	Update	Delete	References	Alter	Index
DEMO_SELECT	Yes						
DEMO_UPDATE	Yes	Yes	Yes	Yes			
KING	Grantable	Yes	Yes	Yes	Yes	Yes	Yes
PUBLIC	Yes						
*							

This grid can be manipulated in the same way as the column grid. You can add, delete and modify a grantee. Deleting a grantee is equivalent to revoking all individual privileges.

The *Grantee* is the user or role to which you want to grant certain privileges. Note that privileges granted to a role will often not suffice for the purpose of development of stored program units. If a user is granted select privilege on a table through a role, and references this table in a stored program unit, compilation will fail because the table is unknown. The user must personally be granted select privilege if the object is to be used in a program unit that he or she owns.

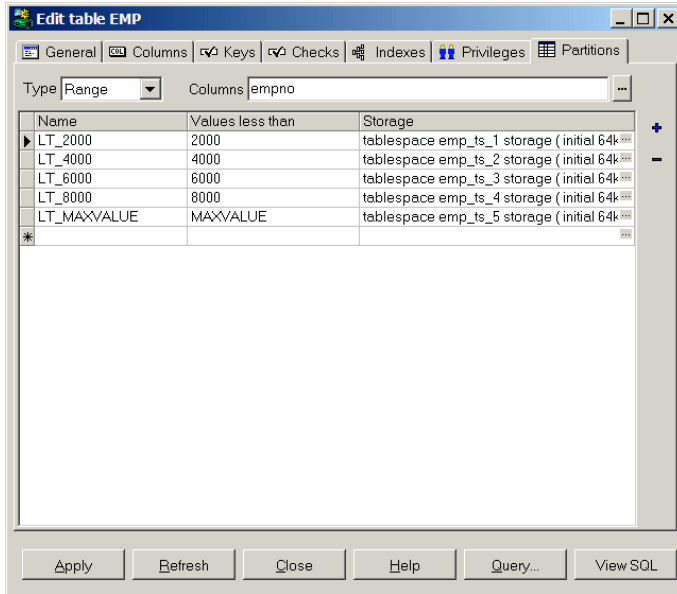
The individual privilege columns can have 3 values:

- (Blank) – The grantee does not have the privilege
- Yes – The grantee has the privilege
- Grantable – The grantee has the privilege, and can grant it to other users or roles

The *Select*, *Insert*, *Update* and *Delete* privileges allow the grantee to perform this action on the records of the table. The *References* privilege allows the grantee to create a foreign key to this table. The *Alter* privilege allows the grantee to alter the table definition with the alter table command. The *Index* privilege allows the grantee to create an index on the table.

## Partitions page

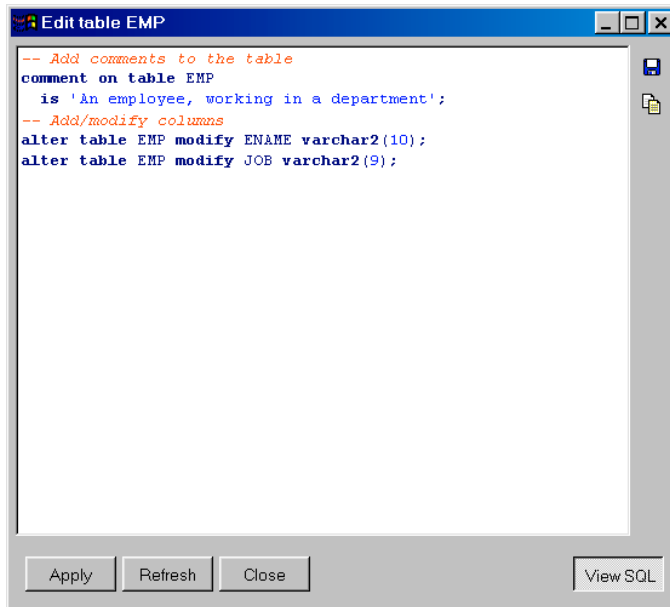
The *Partitions* page is only visible when you (re)create a table, or when you edit an existing partitioned table:



At the top of the page you can select *Range*, *Hash*, or *List* partitioning, as well as the partition columns. In the grid you can define the name, column values (if applicable), and storage of each partition. If you have defined more than one partition column, you must separate the values by a comma.

## Viewing and modifying the SQL

After creating a table, or after modifying an existing table, you can view the resulting SQL by pressing the *View SQL* button. After adding a comment to the table and modifying the *ename* and *job* columns to a *varchar2* data type, the resulting SQL would look like this:



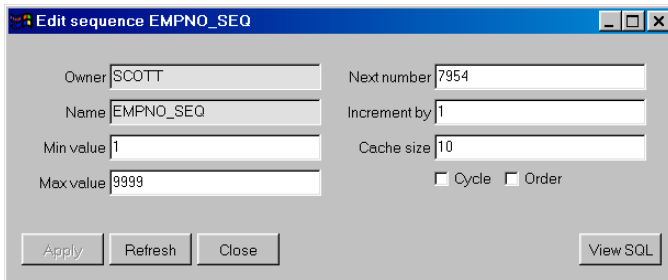
You can save the SQL to a file, or copy it to the clipboard with the corresponding buttons at the upper right corner of the window. The *Print* button on the toolbar will print the SQL. You can additionally make changes to the SQL and apply the changed SQL. After applying the changed SQL, the window will return to form mode and will refresh the table definition. Changes made to the SQL will immediately be reflected in the forms.

If you press the *View SQL* button again, the window will return to form mode, discarding any unapplied changes you may have made to the SQL.

If you selected the *View* option in the Object Browser to view the table definition, the *View SQL* button will display the creation SQL for the table, without preceding it with a drop command. This way you can save a table definition in a file, or copy its definition to another database or user.

## 8.2 The sequence definition editor

The sequence editor allows you to easily create or modify a sequence:



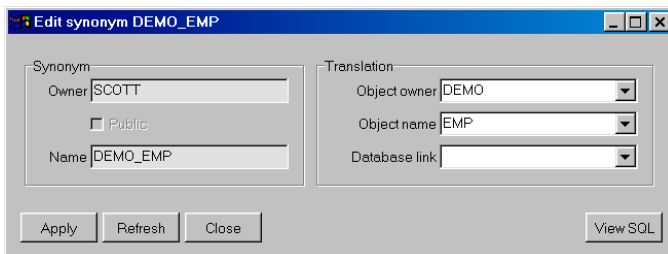
Owner: SCOTT      Next number: 7954  
Name: EMPNO\_SEQ      Increment by: 1  
Min value: 1      Cache size: 10  
Max value: 9999      ☐ Cycle    ☐ Order  
Apply   Refresh   Close   View SQL

Most properties have a default value, and you only need to supply a name when creating a new sequence:

- Owner – The user that is currently logged on
- Minimum value – 1
- Maximum value – Unlimited
- Initial value – 1
- Increment by – 1
- Cache size – 20
- Cycle – Off
- Order – Off

## 8.3 The synonym definition editor

The synonym editor allows you to easily create or modify a synonym:

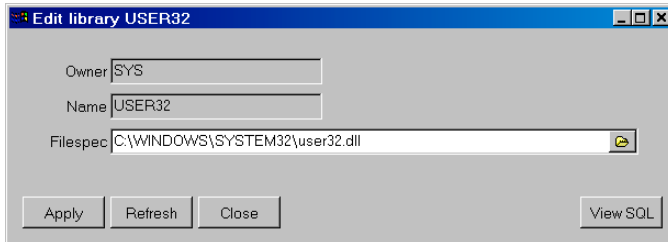


Synonym      Translation  
Owner: SCOTT      Object owner: DEMO  
☐ Public      Object name: EMP  
Name: DEMO\_EMP      Database link:  
Apply   Refresh   Close   View SQL

The default value for the *Owner* will be the user that is currently logged on. Checking the *Public* option will create a public synonym, and will make the *Owner* field read-only. The *Object owner* and *Object name* fields have a suggestion list to easily select a translation object for the synonym. If the *Object owner* field is empty, the *Object name* list will display all objects.

## 8.4 The library definition editor

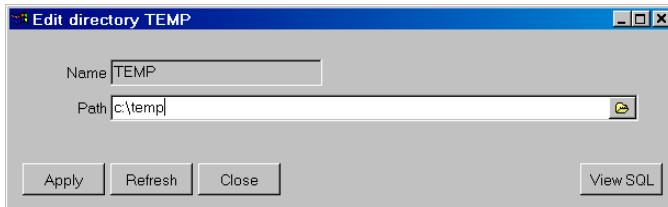
The library editor allows you to easily create or modify a library:



The default value for the *Owner* will be the user that is currently logged on, and you can use the suggestion list to create a library with a different owner. The *Filespec* is the full file specification of the 3GL library that you want to use. If you are developing on the database server, then you can use the *Select DLL file* button to select the dynamic link library. Otherwise you have to type the name of the DLL (Windows) or shared object (UNIX). The validity of the *Filespec* is not checked.

## 8.5 The directory definition editor

The directory editor allows you to easily create or modify a directory, which can subsequently be used for BFILE's:



If you are developing on the database server, then you can use the *Select directory* button to select the directory path. The validity of the directory path is not checked.

## 8.6 The job definition editor

The job editor allows you to create or modify a job:

**Edit job 21**

Submitter	SCOTT	Priv user	SCOTT
Job	21	Schema user	SCOTT
What	begin clear_transaction_log; end;	Last date	
Next date	28-9-2002	This date	
Interval	sysdate + 2	Total time (sec)	0
<input type="checkbox"/> Broken		Failures	0
		NLS environment	NLS_LANGUAGE='DUTCH' NL

Buttons: Apply, Refresh, Close, Help, View SQL

Creating a new job is equivalent to calling `dbms_job.submit`. Modifying an existing job is equivalent to calling `dbms_job.change`. For more information about jobs, see the *DBMS\_JOB* chapter in the *Oracle Supplied Packages Reference* manual.

The *Open PL/SQL Editor* button next to the *What* field invokes a more convenient PL/SQL Editor with syntax highlighting, Code Assistant, and so on.

## 8.7 The queue definition editor

The queue editor allows you to create or modify an advanced queue:

**Create queue**

Owner	SCOTT	Max retries	5
Name	DEPT_QUEUE	Retry delay (sec)	0
Table name	DEPT_QUEUE_TABLE	Retention time (sec)	0
Queue type	Normal queue		
Comment			

Buttons: Apply, Refresh, Close, Help, View SQL

For the queue table name you can select an existing queue table, or you can press the *Create* button to create a new queue table (see chapter 8.8).

For more information about Oracle's Advanced Queuing, see the following Oracle documentation:

- *Application Developer's Guide - Advanced Queuing*
- *Supplied PL/SQL Packages and Types Reference*

## 8.8 The queue table definition editor

The queue table editor allows you to create or modify an advanced queue table:

The screenshot shows the 'Edit table DEPT\_QUEUE\_TABLE' dialog box with the 'Queue' tab selected. The 'Payload type' dropdown is set to 'SCOTT.TDEPT'. The 'Sort list' text box contains 'ENQ\_TIME'. The 'Multiple consumers' checkbox is unchecked. The 'Message grouping' dropdown is set to 'Transactional'. The 'Competible' text box contains '8.1.3'. The 'Primary instance' and 'Secondary instance' text boxes both contain '0'. At the bottom, there are buttons for 'Apply', 'Refresh', 'Close', 'Help', 'Query...', and 'View SQL'.

On the *General* tab page you can define the name, storage parameters, and comment for the table. On the *Privileges* tab page you can grant privileges on the table to users and roles. See also chapter 8.1.

On the *Queue* page you can supply the queue table specific information. For the *payload type* you can select an existing object type, or *RAW* to create a raw queue.

For more information about Oracle's Advanced Queuing, see the following Oracle documentation:

- *Application Developer's Guide - Advanced Queuing*
- *Supplied PL/SQL Packages and Types Reference*

## 8.9 The user definition editor

The user editor allows you to create or modify a user, its privileges, and tablespace quotas:

The screenshot shows the 'Edit user SCOTT' dialog box. The 'General' tab is selected. The fields are as follows:

- Name: SCOTT
- Password: (empty)
- Default tablespace: USERS
- Temporary tablespace: TEMP
- Profile: DEFAULT
- Identified externally: ☐
- Password expire: ☐
- Account locked: ☐

Buttons at the bottom: Apply, Refresh, Close, View SQL.

Note that you need specific system privileges (such as provided by the *DBA* role) to view and modify user information.

### General page

On the general page you basically only need to enter the name and password. All other properties have a default value:

- Default tablespace – SYSTEM
- Temporary tablespace – SYSTEM
- Profile – DEFAULT
- Password expire – Off
- Account Locked – Off

If you modify an existing user, the current password will not be displayed. Enter a new password to change it, or leave it empty to leave the password unchanged. If the user is identified externally, you cannot enter a password.



## Object privileges page

Object	Select	Insert	Update	Delete	References	Alter	Index	Execute	Read
dbms_alert								Yes	
dbms_pipe								Yes	
plan_table	Yes	Yes	Yes	Yes					
temp									Yes

Buttons: Apply, Refresh, Close, View SQL

This page allows you to view and modify the object privileges granted to the user. This does not include the privileges that are indirectly granted through a role.

In the first column you can see the object on which privileges are granted to the user. All other columns represent a specific privilege. Each privilege can have the following value:

- (Blank) – The user does not have the privilege
- Yes – The user has the privilege
- Grantable – The user has the privilege, and can grant it to other users or roles

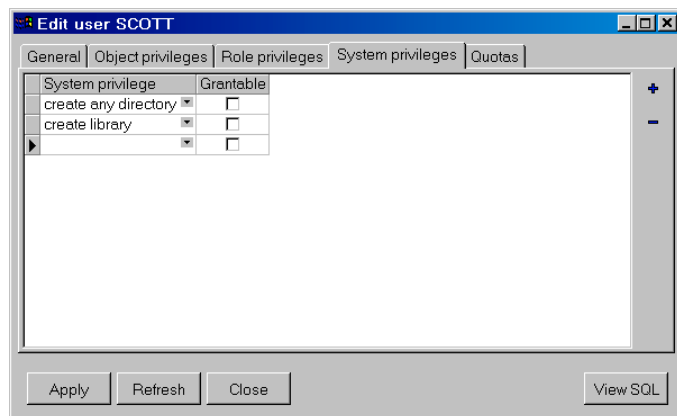
## Role privileges page

Role	Grantable	Default
connect	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
demo_role	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
resource	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Buttons: Apply, Refresh, Close, View SQL

This page displays the roles that are granted to the user. Each role privilege can be *grantable*, so that the user can grant it to other users and roles. If a role privilege is *default*, the role will be enabled when the user logs on. If it is not default, the role privilege must explicitly be enabled for the session by executing a *set role* command after the user has logged on.

## System privileges page



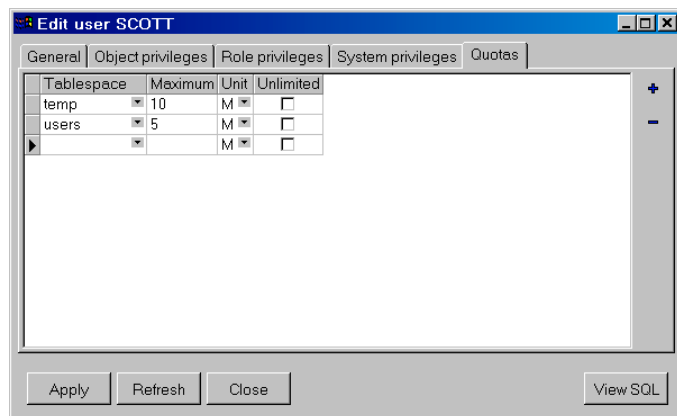
The screenshot shows the 'Edit user SCOTT' dialog box with the 'System privileges' tab selected. The dialog has tabs for General, Object privileges, Role privileges, System privileges, and Quotas. The System privileges tab contains a table with the following data:

System privilege	Grantable
create any directory	<input type="checkbox"/>
create library	<input type="checkbox"/>
	<input type="checkbox"/>

At the bottom of the dialog are buttons for Apply, Refresh, Close, and View SQL.

This page displays the system privileges that are granted to the user. Each system privilege can be *grantable*, so that the user can grant it to other users and roles.

## Quotas page



The screenshot shows the 'Edit user SCOTT' dialog box with the 'Quotas' tab selected. The dialog has tabs for General, Object privileges, Role privileges, System privileges, and Quotas. The Quotas tab contains a table with the following data:

Tablespace	Maximum	Unit	Unlimited
temp	10	M	<input type="checkbox"/>
users	5	M	<input type="checkbox"/>
		M	<input type="checkbox"/>

At the bottom of the dialog are buttons for Apply, Refresh, Close, and View SQL.

This page displays the tablespace quotas of the user. You can enter a *maximum* number of bytes, kilobytes, or megabytes that the user can allocate in a tablespace, or you can specify that the quota is *unlimited*.

## 8.10 The role definition editor

The role editor allows you to create or modify a role and its privileges:

The screenshot shows a window titled "Edit role DEMO\_ROLE". It contains four tabs: "General", "Object privileges", "Role privileges", and "System privileges". The "General" tab is selected. Inside the "General" tab, there is a "Name" field containing "DEMO\_ROLE". Below it is an "Identification" section with three radio buttons: "Not identified" (which is selected), "Identified by password", and "Identified externally". Below the radio buttons is a "Password" field. At the bottom of the window, there are four buttons: "Apply", "Refresh", "Close", and "View SQL".

By default a role does not need to be identified. You can also define password identification or external identification for a role. Password identification requires that you enter a password, and is useful if the role is not granted by default to users. Such a role needs to be explicitly set after a user logs on, which requires the password.

The other 3 tab pages (Object privileges, Role privileges and System privileges) work in the same way as the corresponding pages of the User definition function.

## 8.11 The profile definition editor

The profile editor allows you to edit the resource limits, password limits, and users of a profile:

**Edit profile APP\_DEFAULT**

General Users

Name: APP\_DEFAULT

**Resource limits**

- Sessions / user: 2
- CPU / session (0.01 sec): Default
- CPU / call (0.01 sec): Default
- Connect time (min): Default
- Idle time (min): 15
- Logical reads / session: Default
- Logical reads / call: Default
- Composite limit: Default
- Private SGA (Bytes): Default

**Password limits**

- Failed login attempts: 3
- Password life time (days): 30
- Password reuse time (days): Default
- Password reuse max: Default
- Password lock time (days): Default
- Password grace time (days): Default
- Password verify function: Default

Apply Refresh Close View SQL

Each limit can be set to *Default*, in which case the value will be inherited from the standard profile *DEFAULT*, to *Unlimited*, or to a concrete value.

On the Users tab page you can define to which users this profile applies. Note that if you remove a user from the profile, it will implicitly be changed to the *DEFAULT* profile.

## 8.12 The database link definition editor

The database link editor allows you to create and edit a database link:

**Create database link**

**Database link**

Owner: PUBLIC

Name: CHICAGO

☒ Public

☐ Shared

**Connect To**

Username: SCOTT ☐ Current

Password: (masked)

Database: CHICAGO

**Authenticated By**

Username:

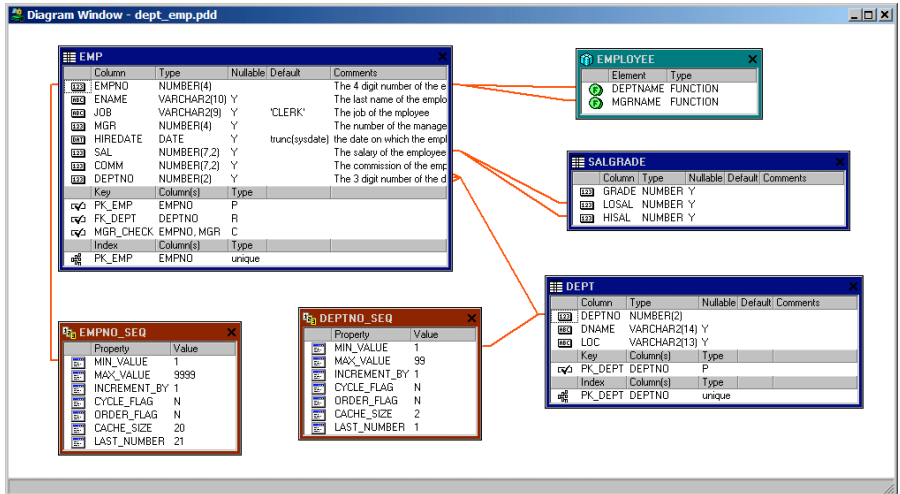
Password:

Apply Refresh Close View SQL

Note that the *Password* and the *Authenticated By* properties will not be retrieved when you edit an existing database link. You will have to enter this information every time you edit the database link.

## 9. Diagrams

The Diagram Window allows you to create a graphical representation of a selection of objects. This way you can easily visualize (a part of) the database objects of your application or project and their relations:



You can add all relevant object types to a diagram, and include the relations between them. Foreign key relations between tables are automatically included. For large diagrams you can add additional structure by including object groups.

A diagram can be used for documentation purposes, but can also serve as a workspace. Right-clicking on an object provides access to all object functions, and double-clicking on an object will invoke the default action for the object type.

### 9.1 Creating a diagram

To create a new diagram, select the *Diagram Window* from the *New* submenu in the *File* menu. An empty Diagram Window will appear. To add a database object to the diagram, drag it from the Object Browser into the Diagram Window. You can drag & drop multiple objects at once.

To move a diagram object, click on the title bar and drag it to the new location. To resize a diagram object you can drag an edge or a corner. To delete an object from the diagram, press on the X at the upper-right of the object. This will only affect the diagram, and the database object will not be dropped.

#### Showing and hiding items

To show or hide a specific item of a diagram object, right-click on it and go to the *Items* submenu. Here you will see a submenu with all items that are applicable for the selected object. For a table you can select which items of a column, key or index you want to show. You can also hide the complete key or index section.

By default all items will be displayed, but if you right-click on an object and select *Set as default*, the visible items will apply to all objects added to a diagram in the future. You can do this for all object types.

## Adding relations

To add a relation between 2 objects, click on the icon of an element of the source object, and drag it to the element of the target object. A line will be drawn between the 2 elements of the objects. You can move the line to another element by selecting the line and moving the selection point up or down. To remove a line, select it and press the *Delete* key.

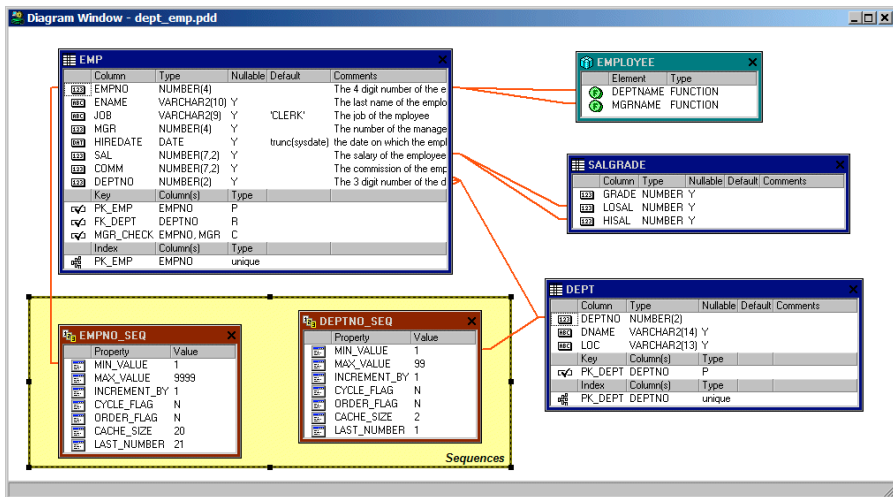
If you add a table to a diagram, foreign key relations with other tables on the diagram will automatically be included. These relations can be deleted if necessary.

By default relations will be drawn as straight lines. You can add additional bends by right-clicking on a line and selecting *Insert bend* from the popup menu. You can also hold down the *Ctrl* key while clicking on a line to insert a bend. A new selection point is added to the line as a result.

To create a one-to-many relation, right-click on the end of the line and select *Change line end* from the popup menu.

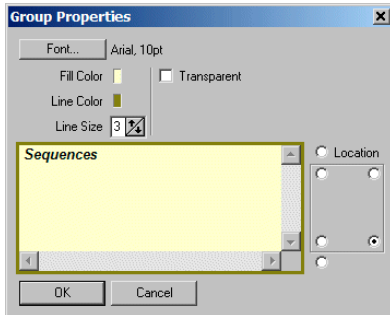
## Adding groups

To add additional structure to a large diagram, you can create object groups. Groups can have distinctive colors, and can have a title at the upper or lower left or right:



To add a group, right click at an empty position on the diagram and select *Add group* from the popup menu. A new group will appear at this location, which you can resize to appropriate dimensions. All objects that are completely covered by the group will implicitly be part of that group, and moving the group will also move these enclosed objects. If you hold down the *Ctrl* key while moving a group, the enclosed objects will not be moved.

To change the appearance of a group, right-click on it and select the *Properties* item from the popup menu. You can also double-click on the group. The following screen will appear:



You can change the font, fill color, line color and line size of the group, and make the group transparent. You can optionally enter the title of the group, and specify the location of the title.

To create a default group appearance, right-click on it and select *Set as default*. This will apply to all groups added to a diagram in the future.

To delete a group from the diagram, select it and press the *Delete* key. This will only delete the group, and not the enclosed objects.

## 9.2 Saving and opening a diagram file

To save a diagram to a file, select the *Save* item from the *File* menu. A diagram will be saved in a file with a *.pdd* extension, which can be reopened later. To define a standard location for diagram files, go to the *Directories* preference page and set the corresponding option.

## 9.3 Updating a diagram

A diagram is a static representation of the database. If database objects change over time, you will need to update the diagram to reflect these changes. To do so, right-click on a diagram object and select *Update from DB* from the popup menu. To update all objects on the diagram, right-click on it and select *Update All from DB*.

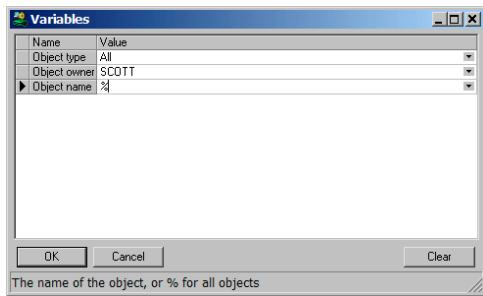
If an object is dropped from the database, you will also need to delete it from the diagram. If an object is renamed in the database, you will need to delete the diagram object and add it with the new name.

## 10. Reports

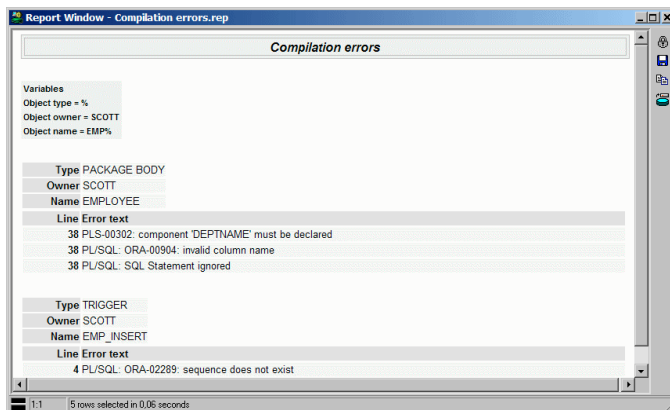
To run reports against your application data or against the Oracle dictionary, you can use PL/SQL Developer's built-in Report functionality. There are a number of standard reports, and you can easily create your own custom reports. These custom reports can be saved in a report file, which can in turn be included in the reports menu. This makes it very easy to run your own frequently used custom reports.

### 10.1 Standard reports

The standard reports are accessible through the *Reports* main menu. If, for example, you select the *Compilation errors* report, you will first be prompted for the object type, owner, and name for which you want to display the current compilation errors:



After entering the appropriate values and pressing the *OK* button, the report will be displayed:



The result is an HTML document that is displayed by PL/SQL Developer's internal HTML viewer or by the Internet Explorer ActiveX control, depending on the preferences (see chapter 16.9).

You can subsequently print the report by pressing the *Print* button on the toolbar, or you can save the report in HTML format by pressing the *Save results* button on the right side of the Report Window. The *Copy as HTML* button copies the report results to the clipboard. The *Export Results* button allows you to export the results in CSV, TSV or XML format, or to export the results directly to Excel.

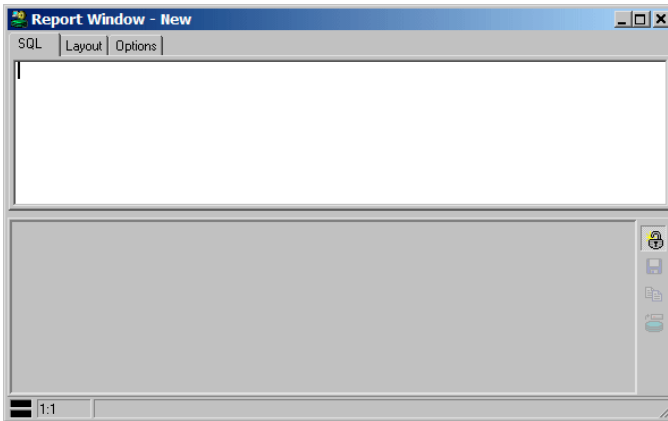
The *Edit report* button provides access to the report definition, if the report is not locked.

You can right-click on the results for additional options.

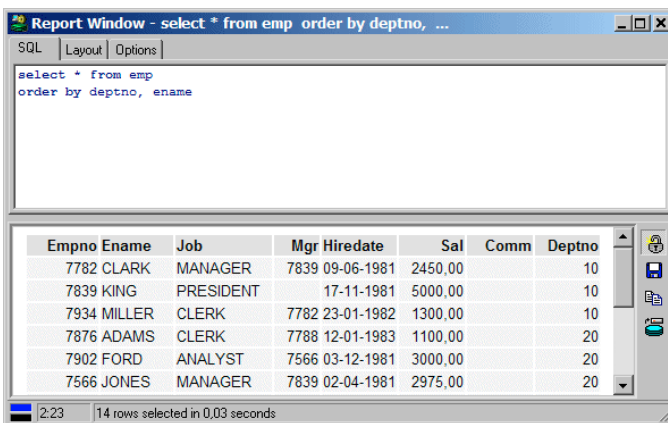


## 10.2 Custom reports

Custom reports are very easy to create. First you create a new, empty report by pressing the *New* button on the toolbar, and selecting the *Report Window* item from the popup menu. An empty Report Window will appear:



In the *SQL* editor you can type a single SQL select statement for your report. For master/detail reports you can use a join select statement (this will be described later). Let's start simple with the well-known *emp* table by typing the select statement and pressing the *Execute* button on the toolbar:



This is a very basic report, with only the default style properties applied. The following chapters describe how you can refine the report's functionality.

## 10.3 Variables

Very often your reports will require some additional input before they can run. Let's assume that you want to restrict the report from the previous chapter to the employees of just one department. In that case you need supply a substitution variable in the SQL text:

```
select * from emp
where deptno = &department
order by ename
```

When this report is run, the user is prompted for a *department* before the select statement is executed. This is of course not very foolproof. What if the user enters nothing? The statement would lead to a *ORA-00936: missing expression* exception. What if the user enters a value that is not a number? Or a number that does not exist in the *dept* table? To prevent these situations, you can use the parameter options described in the following chapters. These options must be specified between brackets, just like attributes in an HTML or XML document.

### Name option

The name of a variable is used as a prompt in the variable input form. Therefore you should make sure that it describes the meaning of the variable. The name is the only option that does not have to be specified between brackets, if it is the only option. If the name contains spaces or other special characters, enclose it in double quotes. The following 2 examples are equivalent:

```
select * from emp
where deptno = &"Department number"
order by ename
```

and

```
select * from emp
where deptno = &<name="Department number">
order by ename
```

### Hint option

Besides the name option, you can additionally specify a hint text. This hint will be displayed on the status line of the variable input form when this variable has the focus:

```
select * from emp
where deptno = &<name="Department number"
             hint="Only employees from this department will be listed">
order by ename
```

### Type option

The type option restricts the user to the information that can be entered, and also controls how the value should be inserted into the resulting SQL text:

```
select * from emp
where deptno = &<name="Department number"
             type="integer">
order by ename
```

In this situation the user can only enter values that are valid integer values. The entered value will be replaced in the SQL text as is. For strings however, you would want the value to be enclosed in quotes. Consider the following select statement:

```
select * from emp
where ename = &<name="Employee name"
             type="string">
```

If you enter *SCOTT* (without quotes) in the variable input form, the resulting SQL would be:

```
select * from emp
where ename = 'SCOTT'
```

Valid types are: *none*, *string*, *integer*, *float*, and *date*. No validation will occur for date values, so that the user can additionally supply date expressions like *sysdate*, *trunc(sysdate, 'MM')* and so on.

### Required option

If you require a value for a certain variable, set its *required* option to “yes” or “true”:

```
select * from emp
where deptno = &<name="Department number"
               required="yes">
order by ename
```

Now the report can only be run if a value is specified for the department number.

### Default option

To present a default value for the variable, specify the *default* option:

```
select * from emp
where deptno = &<name="Department number"
               default="10">
order by ename
```

The default value can also be a select statement. This select statement should return just one row with one column. For example:

```
select * from emp
where deptno = &<name="Department number"
               default="select min(deptno) from dept">
order by ename
```

In this case the default value will be the smallest department number.

### Ifempty option

As an alternative to making a variable required or providing a default value, you can specify the *ifempty* value in case the user does not specify one:

```
select * from emp
where ename like &<name="Employee name"
                 type="string"
                 ifempty="%">
```

If the user does not specify a value, the percent sign will be used, and all employees will be retrieved. This option cannot be specified together with the *required* option.

### Uppercase option

If you have a string variable and want to implicitly convert its value to uppercase, set its *uppercase* option to “yes” or “true”:

```
select * from emp
where ename = &<name="Employee name"
              type="string"
              uppercase="yes">
```

The value is displayed as typed, but is converted to uppercase in the resulting SQL text.

### Readonly option

A variable can be defined as read-only by specifying *readonly*="yes". The user can see the variable and its value, but cannot change it.

### Hidden option

You can hide variable by specifying *hidden*="yes".

## List option

The *list* option provides the user with a list of possible values. You can specify a comma-separated list of items:

```
select * from emp
where deptno = &<name="Department number"
           list="10, 20, 30, 40">
order by ename
```

This allows the user to select one of the four items, or to type a different value. In this case it makes more sense to use a select statement though:

```
select * from emp
where deptno = &<name="Department number"
           list="select deptno from dept order by deptno">
order by ename
```

You can additionally provide a description for each item:

```
select * from emp
where deptno = &<name="Department number"
           list="10, ACCOUNTING,
                20, RESEARCH,
                30, SALES,
                40, OPERATIONS"
           description="yes">
order by ename
```

Each item is now followed by a description, and the *description* option is added. The list will only display the descriptions, but the actual value will be used in the resulting SQL text.

Again you can do the same with a select statement with 2 fields:

```
select * from emp
where deptno = &<name="Department number"
           list="select deptno, dname from dept order by dname"
           description="yes">
order by ename
```

To restrict the user to just the items in the list, specify the *restricted* option:

```
select * from emp
where deptno = &<name="Department number"
           list="select deptno, dname from dept order by dname"
           description="yes"
           restricted="yes">
order by ename
```

Now the user can only select values from the list, and cannot enter any other value by hand.

The items in a list can sometimes depend on the value of another variable. Imagine that you want to create a report that shows all columns of a specific table. A table is identified by the owner and the name. For the owner you can use a list of all users in the database. For the table name you can query the *all\_tables* view for the selected owner:

```
select * from all_tab_columns
where owner = &<name="Owner"
           type="string"
           list="select username from all_users order by username">
and table_name = &<name="Table"
           type="string"
           list="select table_name from all_tables
                where owner = :owner
                order by table_name">
order by column_id
```

As you can see, the select statement for the second (table) list refers to the first variable through the *:owner* bind variable. Whenever the value of the owner is changed, the table list will be populated.

Note that the name of the bind variable cannot be longer than 30 characters, and cannot contain spaces or other special characters. Spaces in the variable name will be converted to underscores. If the variable name had been *Owner of the table*, then the bind variable would have been `:owner_of_the_table`.

Also note that if the value of a variable is empty, the dependant list will also be empty. No query will be performed in this situation.

### Checkbox option

If the user can select one of two distinct possibilities, you can use the *checkbox* option. This option needs to be followed by the checked and unchecked values:

```
select * from emp
order by hiredate &<name="Descending sort order"
               checkbox="desc, asc">
```

Even though you must always specify 2 values, one of the values can be empty:

```
select * from emp
order by hiredate &<name="Descending sort order"
               checkbox="desc,">
```

Since the default sort order is ascending, these 2 examples are equivalent.

### Prefix and suffix options

If a variable value is empty, this may imply that certain fixed portions of the SQL text must also be omitted. Consider the example that you want to provide an optional sort column. If no column is specified, the *order by* clause should be completely omitted. In this case you can specify this text as *prefix* or *suffix* of the variable. If the variable value is empty, the prefix and suffix will be omitted as well:

```
select * from emp
&<name="Descending sort column"
list="empno, ename, hiredate"
prefix="order by "
suffix=" desc">
```

If no value is specified for this variable, the result will be empty as well:

```
select * from emp
```

If a column is specified, the result will be:

```
select * from emp
order by empno desc
```

Note the spaces after the prefix and before the suffix. These are important, because the prefix, value and suffix are concatenated exactly as specified!

### Escape character

If you wish to use an ampersand in the SQL text that should not be interpreted as a substitution variable, use a double ampersand instead. The following example will retrieve all employees from the 'R&D' department:

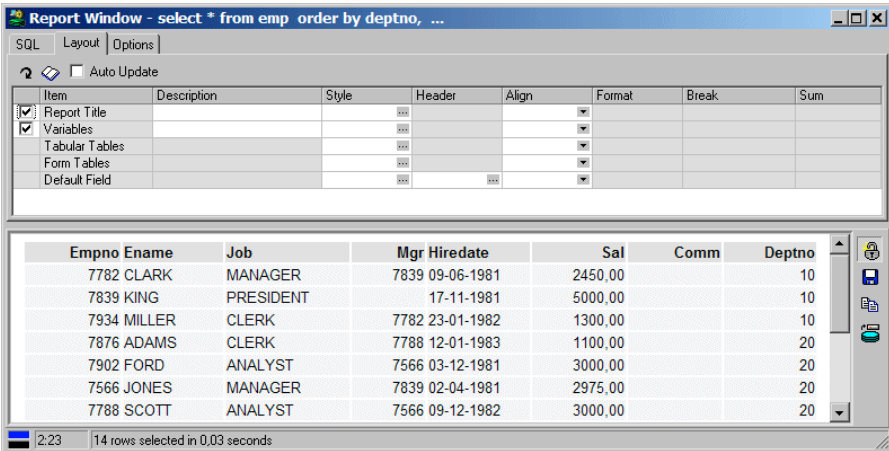
```
select * from emp
where emp.deptno in (select dept.deptno from dept
                    where dname = 'R&&D')
order by empno desc
```

If the text 'R&D' had been used instead, you would have been prompted for the D variable.

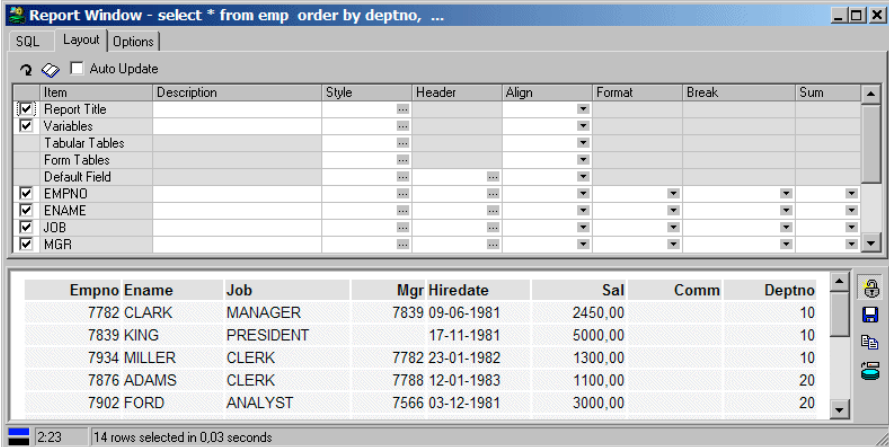
## 10.4 Refining the layout

After opening a new Report Window, typing a select statement, and executing the report, the layout will use the default style properties. It will have a simple tabular layout, with standard font properties, table

style, colors, and so on. To refine the standard layout, switch to the *Layout* tab page. For a new report, this page will look as follows:



As you can see, you can define various layout properties for all layout items (the report title, variables, tables, and the fields). If you want to define properties for an individual field, you first need to press the *Refresh Fieldlist* button on the toolbar:



Now all fields of the result set of the select statement will be included in the layout grid, and you can set the layout properties of each individual field. If you leave the *Style*, *Header* or *Align* property of a field empty, the corresponding property from the *Default Field* will be used.

After changing a layout property, you can execute the report again to view the effect this has on the results. You can alternatively enable the *Auto Update* option on the toolbar, in which case each change will immediately be reflected in the results.

The following chapters describe the various layout properties in detail.

## Displayed

The checkbox in the leftmost column of the layout grid indicates if the layout item should be displayed or not. This allows you to suppress the report title or variables, and allows you to control whether a specific field should be displayed or not.

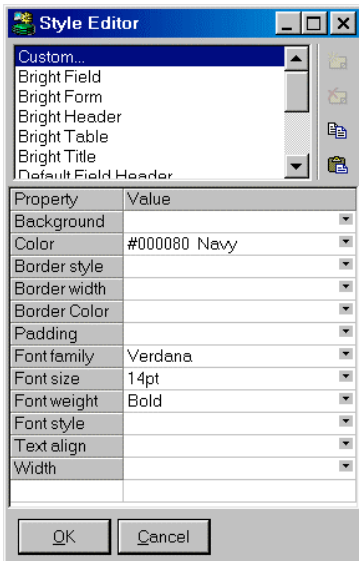
## Description

The description of the *Report Title* will be displayed at the top of the report and after each page break. The description of the *Variables* will be displayed above the variables.

For the individual fields you can define a description to override the standard field name. This description will be used for the header of the column.

## Style

The style controls the appearance of the layout items. Press the cell button (...) to bring up the style editor:



In the listbox at the top of the style editor you can see all standard styles, and a *Custom* style. You can select a standard style from the Style Library (see chapter 10.5), or you can select the custom style so that you can change the various style properties for the current layout item.

Note that this style system is based on the Cascading Style Sheets (CSS) standard. The properties that you see in the style editor are the ones that you will most frequently use, but you can also add other CSS properties and values below the standard properties. You can, for example, add a *Height* style property with a value of *20pt* to set the cell height to 20 points. See <http://www.w3.org/TR/REC-CSS2> for more information about Cascading Style Sheets.

The *Copy* and *Paste* buttons allow you to copy the properties of a style, so that you quickly derive a new style from an existing one.

If you define nothing for the field styles, the *Default Field* style from the Style Library will be used. You can override this by defining the style of the *Default Field* layout item. You can override this default style at the field level by specifying the corresponding style. Note that each field header inherits its style properties from the *Default Header*

style, so you only need to override the properties that you want to change. If, for example, you define the *Default Field* style as Verdana, 14 points, blue, and you want to display the *EMPNO* column in Verdana, 14 points, red, you only need to define the color at the *EMPNO* level. The font family and font size will automatically be inherited from the *Default Field* level.

The default styles of the various layout item types are described in chapter 10.5.

## Header

Just like you can set the style for the data of the individual fields, you can also set the style for their headers. If you define nothing, the *Default Field Header* style from the Style Library will be used. You can override this by defining the style of the *Default Header* layout item. You can override this default style at the field level by specifying the corresponding header style. Note that each field header inherits

its style properties from the *Default Header* style, so you only need to override the properties that you want to change.

**Align**

You can align the layout items in the following ways:

- Left – The item will be left-aligned within its cell
- Right – The item will be right-aligned within its cell
- Center – The item will be centered within its cell
- Default – Numbers will be right-aligned, all other items will be left-aligned
- None – No alignment will be specified

Note that you can also set the *Text Alignment* style property for a layout item. This will take precedence over the alignment option you can specify in the layout grid. Also note that the effect of the alignment may depend on the *Width* style property of the layout item.

**Format**

By default the format of a field is controlled by the data type, scale, and precision. Very often you will use calculated fields or aggregated fields, in which case the scale and precision are unknown. In these situations you can specify a format.

The Format property only has effect for date and number fields. For number fields you can use a 9 for a zero-suppressed digit, 0 for a normal digit, G for a thousand separator, D for a decimal separator, and E for scientific notation. You can additionally specify literal text in double or single quotes. Some examples:

Format	Value = 1234	Value = 5	Value = 0.1
999G990D00	1,234.00	5.00	0.10
9G990	1,234	5	0
0000	1234	0005	0000
0"%"	1234%	5%	0%
0D000e+00	1.234e+03	5.000e+00	1.000e-01
0D000e-0	1.234e3	5.000e0	1.000e-1

For date fields you can use the standard Windows date and time formats.



## Break

The break layout property allows you to structure your report results. Assume that you want to display the departments and their employees. You can specify the following query for this:

```
select d.*, e.*
from   emp e, dept d
where  e.deptno = d.deptno
order by d.deptno, e.empno
```

This would of course lead to a simple tabular report:

Report Window - select d.\*, e.\* from emp e, dept ...

SQL | Layout | Options

```
select d.*, e.*
from   emp e, dept d
where  e.deptno = d.deptno
order by d.deptno, e.empno
```

Deptno	Dname	Loc	Empno	Ename	Job	Mgr	Hiredate	Sal	Comm
10	ACCOUNTING	NEW YORK	7782	CLARK	MANAGER		09-06-1981	2450,00	
10	ACCOUNTING	NEW YORK	7839	KING	PRESIDENT		17-11-1981	5000,00	
10	ACCOUNTING	NEW YORK	7934	MILLER	CLERK		23-01-1982	1300,00	
20	RESEARCH	DALLAS	7369	SMITH	CLERK		17-12-1980	800,00	
20	RESEARCH	DALLAS	7566	JONES	MANAGER		02-04-1981	2975,00	
20	RESEARCH	DALLAS	7788	SCOTT	ANALYST		09-12-1982	3000,00	
20	RESEARCH	DALLAS	7876	ADAMS	CLERK		12-01-1983	1100,00	
20	RESEARCH	DALLAS	7902	FORD	ANALYST		03-12-1981	3000,00	
30	SALES	CHICAGO	7499	ALLEN	SALESMAN		20-02-1981	1600,00	300,00
30	SALES	CHICAGO	7521	WARD	SALESMAN		22-02-1981	1250,00	500,00
30	SALES	CHICAGO	7654	MARTIN	SALESMAN		28-09-1981	1250,00	1400,00

4:27 14 rows selected in 0,03 seconds

By specifying a *Break* on the *LOC* column, you will get some more structure into your report:

Report Window - select d.\*, e.\* from emp e, dept ...

SQL | Layout | Options

Auto Update

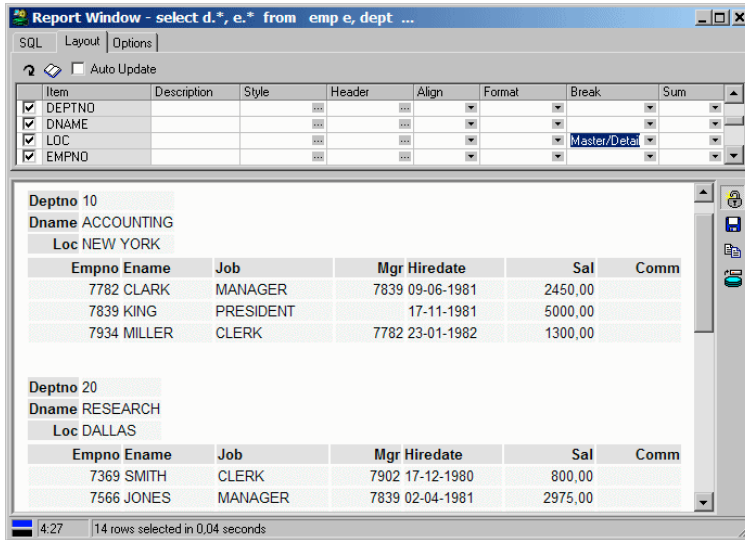
Item	Description	Style	Header	Align	Format	Break	Sum
<input checked="" type="checkbox"/> DEPTNO							
<input checked="" type="checkbox"/> DNAME							
<input checked="" type="checkbox"/> LOC						Break	
<input checked="" type="checkbox"/> EMPNO							

Deptno	Dname	Loc	Empno	Ename	Job	Mgr	Hiredate	Sal	Comm
10	ACCOUNTING	NEW YORK	7782	CLARK	MANAGER		09-06-1981	2450,00	
			7839	KING	PRESIDENT		17-11-1981	5000,00	
			7934	MILLER	CLERK		23-01-1982	1300,00	
20	RESEARCH	DALLAS	7369	SMITH	CLERK		17-12-1980	800,00	
			7566	JONES	MANAGER		02-04-1981	2975,00	
			7788	SCOTT	ANALYST		09-12-1982	3000,00	
			7876	ADAMS	CLERK		12-01-1983	1100,00	
			7902	FORD	ANALYST		03-12-1981	3000,00	
30	SALES	CHICAGO	7499	ALLEN	SALESMAN		20-02-1981	1600,00	300,00
			7521	WARD	SALESMAN		22-02-1981	1250,00	500,00

4:27 14 rows selected in 0,03 seconds

Repeating values for the fields up to (and including) *LOC* will be suppressed. In short: each dept record will only be displayed once. In this case it is necessary that the records are sorted by department (the *DEPTNO* column). If you want to repeat the header after each break, use the *Break + Header* option.

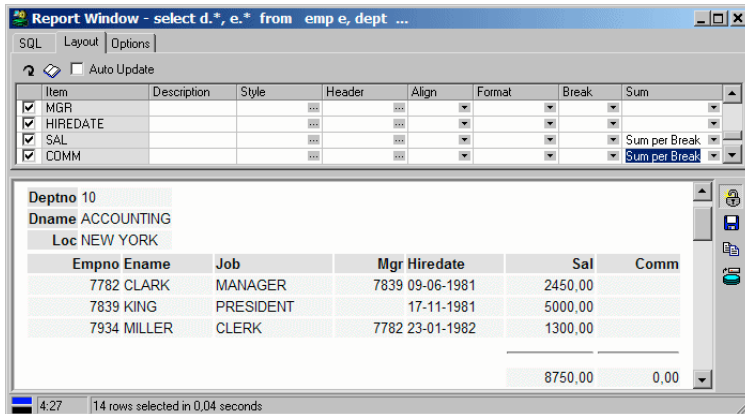
You can alternatively select a Master/Detail layout:



You can have multiple breaks for a report. The order by clause of the query must assure that the result set is sorted on all break columns.

## Sum

The sum property allows you include a sum for a field:



You can specify that the sum is to be calculated at the break level, at the report level, or both.

## Page headers

To define a page header, you first need to set the page size (see chapter 10.6). The Report Title will be repeated after each page break, and can include the following variables:

- OSUser – The Operating System user.
- DBUser – The user that is connected to the database.
- Database – The database that the user is connected to.
- Date – The current date.
- Time – The current time.
- Page – The current page.

To include such a variable in the page header, precede it with an ampersand. For example:

```
All employees (&dbuser@&database)
```

If you run the report connected as user *scott* to database *chicago*, the header will look as follows:

```
All employees (scott@chicago)
```

Since the report is formatted as HTML, you can additionally include HTML code. To include an image, you could include the `<img>` tag:

```
All employees <img src=logo.gif>
```

To include left aligned, centered, and right-aligned information, you can use a single-row table with 3 cells, and include the information in the corresponding cell:

```
<table width="100%"> <tr> <td>&dbuser</td> <td align="center">All  
Employees</td> <td align="right">&page</td> </tr> </table>
```

This will be displayed as:

Scott	All Employees	1
-------	---------------	---

Note that you can use these variables and HTML code in the description of all layout items, but they are most useful for the Report Title.

## 10.5 The Style Library

Instead of defining and redefining the same styles over and over again, you can press the *Style Library* button on the toolbar to create and modify your standard styles. If, for example, you would like all fields that contain SQL expressions to be displayed in Courier New, 12 points, blue, simply define a *SQL* style with these properties in the library and apply it where appropriate.

There are a number of style names that have a special meaning:

Style name	Description
Default Report Title	Will be applied to the report title
Default Variables	Will be applied to the variables
Default Tabular Tables	Will be applied to all tabular tables (e.g. detail records)
Default Form Tables	Will be applied to all form tables (e.g. master records)
Default Field Style	Will be applied to all fields that do not have a defined style
Default Field Header	Will be applied to all field headers that do not have a defined style

These are default styles, so you can override them in an individual report by defining the style for the corresponding layout item.

Note that all used styles (standard and custom) are stored in the report file. When a report is run and a style needs to be applied, PL/SQL Developer will try to locate it in the in the style library. When it is not available, the style from the report file will be used. This will ensure that your report will always display correctly but that styles can still be overruled.

## 10.6 Options

You can specify various options for your report:

**Report Window - select d.\*, e.\* from emp e, dept d ...**

SQL | Layout | Options

**Connect Parameters**

Username:

Password:

Database:

Connect as:

☒ Use this connection

Background Image:

Background Color:

**Security**

☐ Unlocked

☐ Locked for other users

☒ Locked for all users

Password:

**Layout**

☐ Form layout

Left margin:  Pixels

Top margin:  Pixels

Page break:  Lines

☐ Only NBSP

☐ Include SQL

**Preview:**

Deptno	Dname	Loc	Empno	Ename	Job	Mgr	Hiredate	Sal	Comm
10	ACCOUNTING	NEW YORK	7782	CLARK	MANAGER	7839	09-06-1981	2450.00	

11:16 | 14 rows selected in 0.04 seconds

### Connect Parameters

The Connect Parameters determine how the report query will be run. If, for example, it is necessary that your report is executed under the *SYS* account, you can enter this name in the *Username* field. You can additionally enter the *Password*, or you can leave it empty. If you leave it empty, you will be prompted for the password when the report is executed. You can also specify the *Database*, if the report should

always be executed in the same database instance. You can alternatively enter \* in the database field, which indicates that the report should be executed in the current database.

The *Use this connection* option allows you to disable or enable the Connect Parameters for a report. If it is disabled, the current PL/SQL Developer connection will be used.

## Security

The *Security* options allow you to lock the report definition (the SQL, Layout, and Options), so that a user cannot view or change it. This may be useful if the Connect Parameters specify a privileged account (such as SYS) with password. If a user can change the SQL, he or she could execute any statement under the SYS account if the report is not locked.

The Security can be set to one of the following values:

- Unlocked – Anybody can change the report definition.
- Locked for other users – You can only change the report definition if you know the report's *Password*, or if you are connected in PL/SQL Developer with the same account as specified in the Connect Parameters.
- Locked for all users – You are always prompted for the report's *Password* before you can change the report definition, regardless how you are currently connected in PL/SQL Developer.

If the report is locked, the report file (.rep) is encrypted so that it cannot be viewed or changed with a text editor.

## Layout options

- Form layout – By default all records are displayed in tabular layout, except for master records, which are displayed in form layout. By enabling this option, all records will be displayed in form layout. This may be useful if each record has many and/or wide fields, that cannot be displayed on a single line.
- Left margin – The number of pixels for the left margin of the report. This option only works if you are using the Internet Explorer as HTML viewer.
- Top margin – The number of pixels for the top margin of the report. This option only works if you are using the Internet Explorer as HTML viewer.
- Page break – The number of lines after which a page break should occur. The report header will be repeated after each page break.
- Only NBSP – When enabled, all spaces in the data will be converted to NBSP's (non-breaking spaces) in the HTML output. When disabled, normal spaces will be used. This can affect the layout, because HTML viewers will display consecutive spaces as one space.
- Include SQL – When enabled, the SQL text of the report is included in the HTML output.

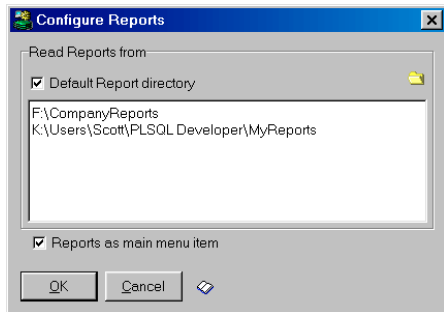
## Other options

- Background image – The image (.jpg or .gif) that should be displayed in the background of this report. You can use the *Select file* button to find your image file.
- Background color – The color that should be displayed in the background of this report.

## 10.7 The Reports menu

By default the *Reports* menu item is included in the main menu, and contains only the standard reports. These standard reports are located in the Reports subdirectory of the PL/SQL Developer installation directory. The menu is simply a representation of this directory structure. Selecting an item from the reports menu will execute that report.

You can add or remove reports in this directory, but you can also configure the reports menu through the *Configure Reports* item in the *Tools* menu. This will bring up the following dialog:



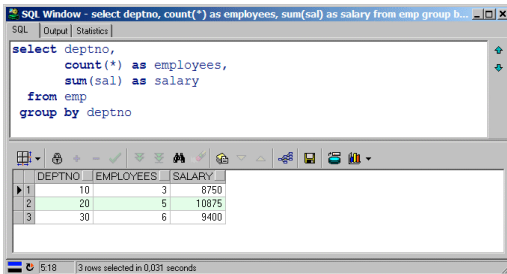
The *Default Report directory* option indicates whether the Reports menu should include the Reports subdirectory from PL/SQL Developer installation directory. This directory contains the standard reports. The edit field below this option contains a list of directories that should be included. You can type directories manually, or you can select a directory by pressing the *Add directory to list* button on the right.

Note that the report files and subdirectories will be merged in the Reports menu. You should try to prevent duplicate subdirectory and report filenames within this directory list, otherwise you will get ambiguous menu items in the Report menu.

The Style Library is also accessible from within this report configuration dialog (see chapter 10.5).

## 11. Graphics

To quickly display a graphical representation of column data queried in a SQL Window or Report Window, you can use the Graph Window. Consider the following query in a SQL Window:

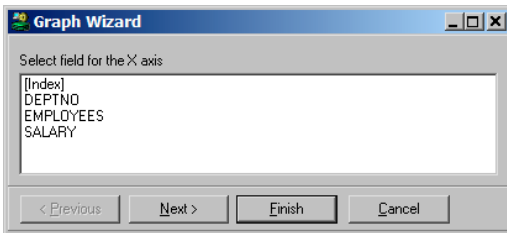


To see a graph of the salaries of the department you can press the *Graph* button:

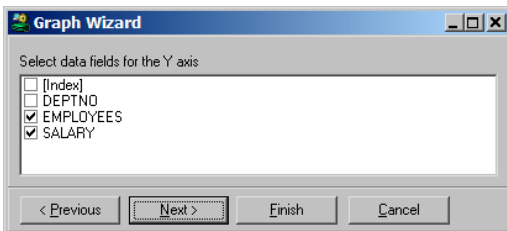


The same button is available on the Report Window. It will either start the *Graph Wizard* as described below, or you can select a pre-defined graph style or template, and appropriate columns will be selected for the X and Y axis.

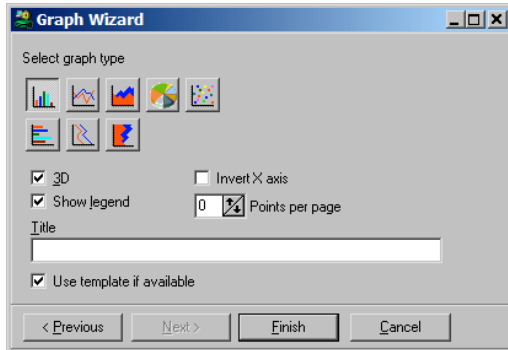
The *Graph Wizard* will first allow you to select the data for the X-axis:



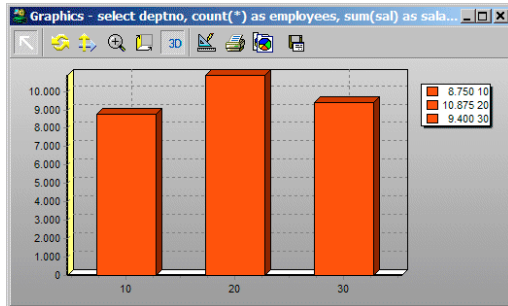
This shows all columns and an *[Index]* item, which is useful if you don't have an X-axis value but want to display data on the Y-axis in order of occurrence. In this case we want to display the salary by department number, so we select *DEPTNO* and press the *Next* button to select the data for the Y-axis:



After selecting *SALARY* for the Y-axis, you can press *Next* to select various graph style properties:



After pressing *Finish*, the resulting Graph Window will be displayed:



The toolbar of the Graph Window provides the following functions:

- Rotate – Press the left mouse button and move the mouse to rotate the graph around the X and Y axis.
- Move – Press the left mouse button and move the mouse to move the graph across the window. This function is always available by pressing the right mouse button, regardless of the activated toolbar function.
- Zoom – Press the left mouse button and move the mouse to zoom in to and out of the graph.
- Depth – Press the left mouse button and move the mouse to increase or decrease the depth of the graph.
- 3D – Switch the graph between a 2D and 3D view.
- Edit – Edit the graph data and layout.
- Save as template – Saves the layout of the graph as a template for future graphs based on the same column names. If you create a graph later and select *From template* from the Graph button popup menu, then the layout information from this template will be used.

You can additionally print a graph or copy it to the clipboard by pressing the corresponding buttons on the main toolbar.



## 12. Projects

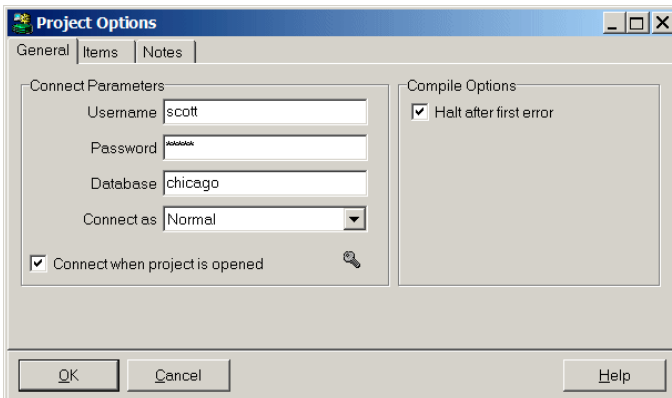
To organize your work you can use PL/SQL Developer's built-in project concept. A project consists of a collection of source files, database objects, notes, and options. It allows you to work within the scope of a specific set of items, instead of a complete database or schema. This makes it easy to find the project items you need, to compile all project items, or to move a project from one location or database to another.

A project keeps track of your desktop configuration. If you reopen a project, all items that were opened when the project was previously closed will be opened at the same position. The *AutoSave Desktop* preference must be enabled for this. This information is not stored in the project definition file (*Project.prj*), but in the project desktop file (*Project.dsk*)

**Note:** to work with projects, the *Use Projects* preference must be enabled. If this is not the case, the main menu will not contain the *Project* item. For all project menu items there are also corresponding toolbar buttons, which can be added to the toolbar through the *Toolbar* preference page (see chapter 16.10).

### 12.1 Creating a new project

To create a new project, select the *New* item from the *Project* menu. This will bring up the Project Options dialog:



On the *General* page you can define how you want to connect when this project is opened. You can also define if you want to continue project compilation after a compilation error has occurred.

The *Items* page allows you to view the project items. This can be source files of PL/SQL program units, Test Scripts, SQL Scripts, Reports, and so on. A project item can also be an object that is stored in the database, and that does not have a representation in a source file. Project items can also include other files such as MS Word documents, HTML documents, and so on. When the project is first created, the item list will be empty.

The *Notes* page allows you to maintain project notes. These notes can contain anything, such as a to-do list, project decisions, design, and so on.

### 12.2 Saving a project

After setting the options, you can save the project by selecting the *Save* item from the *Project* menu. This will create a project file with the extension *.prj*. The project file contains the item specifications,

the options, and the notes. Therefore you need to save a project whenever you add or remove an item, when you change an option, or when you change the notes.

Saving a project will not implicitly save the source files that are included in the project.

## 12.3 Adding files to a project

There are 2 methods to add a file to your project:

If the file is already opened in the IDE, you can simply select the *Add to Project* item from the *Project* menu. For program files (which contain the source of PL/SQL program units), the item is implicitly enabled for compilation. This means that when you make or build the project, this file will be included in the compilation process. Other files, such as Test Scripts, SQL Scripts, Report files, and so on, will not be enabled for compilation.

If the file is not opened in the IDE, you can press the *Add file(s)* button on the Project items toolbar and select the files to add.

Note that the path of the file is stored relative to the location of the project file, if possible. For example, if the file is located in the same directory as the project file, no path will be saved. If the file is located in a subdirectory of the project file, only the subdirectory path will be saved. If the file is located elsewhere, the full path will be saved. This makes the project easily 'portable' from one directory to another.

## 12.4 Adding database objects to a project

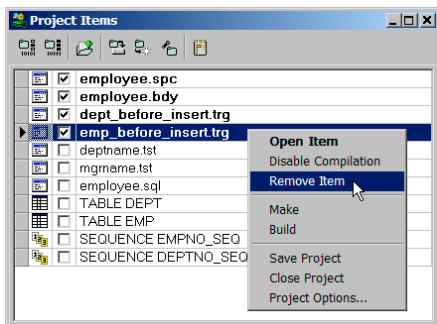
To add a database object to the project, right-click on it in the Object Browser and select the *Add to Project* item from the popup menu. Just like with Program Files, a PL/SQL program unit will implicitly be enabled for compilation. Other objects, such as tables and sequences, cannot be compiled.

You can alternatively drag & drop a database object from the Object Browser to the Project Items Window (see below).

If the current user owns the database object, the owner will not be saved in the project file. If another user owns the object, the owner will be saved as part of the object specification. This makes the project portable from one user to another.

## 12.5 Working with project items

After adding items to the project, you can select *Project Items* from the *Project* menu to work with the project items. The Project Items Window may look something like this:



The first 4 Program Files are enabled for compilation. The 2 Test Scripts, the SQL script, and the tables and sequences are not enabled for compilation.

You can keep the Project Items Window open at all times to have quick and easy access to all project items. This window is part of the project desktop, and will implicitly be reopened when the project is reopened.

To edit an item, you can simply double-click on it. If the item is a source file, it will be opened in a corresponding window (Program Window, Test Window, and so on). If the item is a database object, the appropriate editor will opened with the object definition. If an item is an external file, the corresponding application will be launched (e.g. Acrobat Reader for a PDF file).

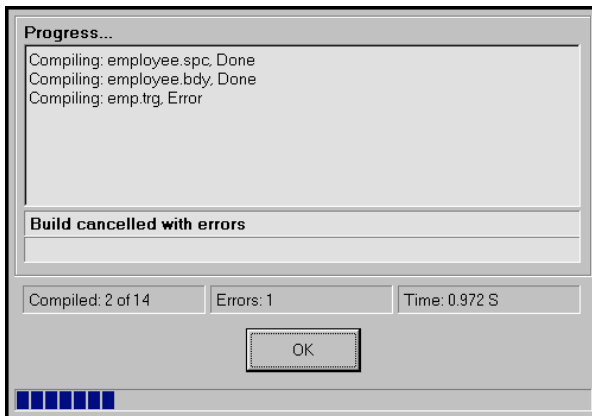
You can alternatively right-click on an item to bring up the popup menu as displayed above. Here you can disable/enable the item for compilation, or you can remove the item from the project. If the item is a database object, you will also have access to the object popup menu for that object as a submenu.

To change the order of the project items, you can click on the row header and drag it to the new location. The order is not only visually important, but also determines the order of compilation.

## 12.6 Compiling a project

To compile a project, you can select the *Build* or *Make* item from the *Project* menu.

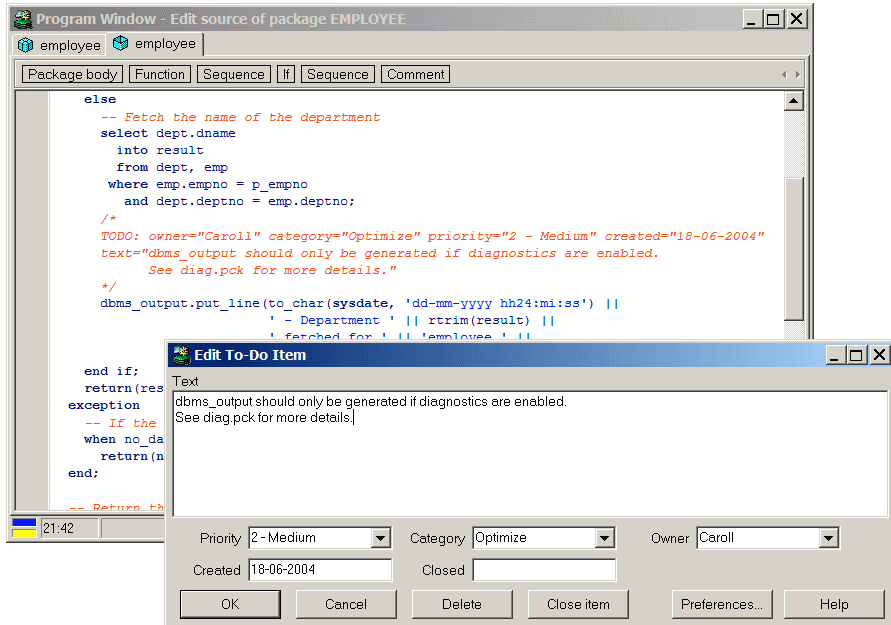
The *Build* function will simply compile all project items that are enabled for compilation. The items will be compiled in the order as they appear in the project item list. If the compilation of an item fails, the project option *Halt after first error* determines if the project compilation will continue or not:



The *Make* function will only compile those items that have been changed since the previous compilation. If a PL/SQL Program Unit is included as a database object, and it is invalid in the database, it will also be recompiled. The timestamp of the compilation is stored in the project's desktop file. Items that are enabled for compilation but are up-to-date, will be listed as *Skipped* in the compilation progress window.

## 13. To-Do Items

You can use To-Do Items in any source file to make a quick note that something needs to be done in this source file, and access this information later from the To-Do List. These To-Do Items are placed into the source file as a comment with a specific format, at an appropriate location:

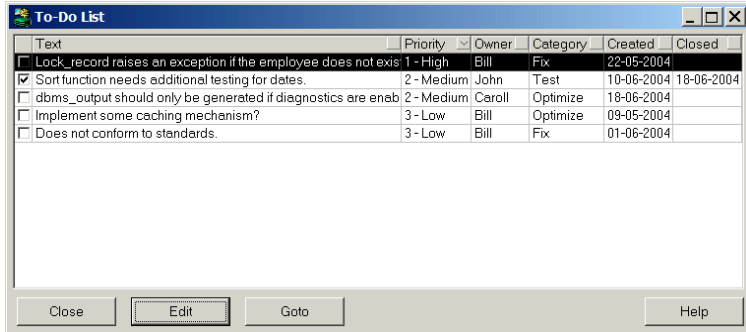


In this example you can see the comment in the source file (starting with the text *TODO:*) and the To-Do Item editor where you can enter the text, priority, category, owner, creation date, and close date. The contents of the selection lists for the priority, category and owner fields can be defined through a preference (see chapter 16.24).

To-Do Items can be accessed through the To-Do Item editor, which is a convenient way to create, edit, view or delete a To-Do Item. You can also edit the comment directly in the source file, as long as you do not change the format.

Because the To-Do Items are stored as comment in the source file, they will always correspond with the actual status of the source. Regardless whether the source comes from the database, from the file system, or is an old version from a version control system.

To access all To-Do Items for a specific source file you can invoke the To-Do List:



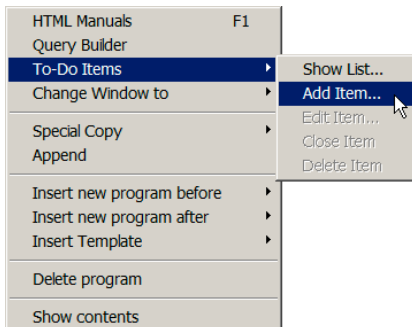
Here you can quickly find a To-Do Item and locate it in the source file by double-clicking on it or by pressing the *Goto* button.

The To-Do List can be sorted by clicking on the column header.

Right-click on the list to copy, print, or export the To-Do Items.

## 13.1 Creating a To-Do Item

To create a To-Do Item, you first need to place the cursor at an appropriate location in the source file. This is where the comment will be inserted, so it is important that this is near the location where the work needs to be done. Next you can right-click at this location and select *Add Item* from the *To-Do Items* submenu from the popup menu:



The To-Do Item editor mentioned above will appear, and you can enter the information for the new To-Do Item. After pressing the *OK* button the comment will be inserted at the exact cursor location.

## 13.2 Editing a To-Do Item

To edit an existing To-Do item, you can either right-click on the corresponding comment in the source file and select *Edit Item* from the *To-Do Items* submenu, or select the *Show List* item if you are not currently near the To-Do Item you want to edit. In that case you can select the To-Do Item from the list and press the *Edit* button.

Now you can change the various properties of the To-Do Item and press the *OK* button to update the comment in the source file.

### 13.3 Closing a To-Do Item

A To-Do Item can be closed by right-clicking on the comment in the source file and selecting *Close Item* from the *To-Do Items* submenu. Alternatively you can invoke the To-Do Item editor and press the *Close* button.

### 13.4 Deleting a To-Do Item

A To-Do Item can be deleted by right-clicking on the comment in the source file and selecting *Delete Item* from the *To-Do Items* submenu. Alternatively you can invoke the To-Do Item editor and press the *Delete* button.

You can also simply delete the comment from the source file.

## 14. Windows, database sessions and transactions

PL/SQL Developer provides a multi session / multi threading environment.

Multi session means that even though you logon only once in PL/SQL Developer, separate database sessions can be used for the Test Windows, SQL Windows and Command Windows. Moreover, program unit compilations will also occur in a separate database session. After all, a compilation is a DDL statement and implicitly commits the current transaction. This could lead to unwanted effects if it shared the same session with other windows.

Multi threading means that execution of Test Windows, SQL Windows and Command Windows can run simultaneously, without blocking each other. The advantage of this is that you can continue doing other work while long running SQL statements are executing, that you can break or kill a session, that you can test locking behavior of your program units, and so on.

### 14.1 Session mode

Multi threading implies that you need a multi session environment. After all, the Oracle Server can only execute one statement at a time for any given database session. Things can only run simultaneously if they run in different sessions. Not all development environments allow for an unlimited amount of sessions, so that a preference can be set (see chapter 16.2) that controls the amount of sessions that PL/SQL Developer will use. This preference is called the Session mode, and can have one of the following values:

- Multi session - Each Test Window, SQL Window and Command Window will have its own session, and another session will be used for compilations. This is the most flexible setting, and will obviously lead to the largest amount of database sessions. Another possible disadvantage is that updates in Window X will only be visible in Window Y after they are committed.
- Dual session - The Test Windows, SQL Windows and Command Windows will share one session, and another session will be used for compilations. Disadvantage of this mode is that only one window can execute at a time.
- Single session - All windows and all compilations use the same session, making transaction management difficult. The debugger is disabled in this mode. Only use this setting if you are restricted to using one database session.

### 14.2 Execution in Multi session or Dual session mode

When you execute a Test Window, SQL Window or Command Window in Multi session or Dual session mode, the *Break* button on the toolbar will be enabled so that you can break execution at any time. A break may not always be successful, for example in case of a locking situation. In that case you can resort to the *Kill* item in the Session menu, if you are privileged to do so.

If you move the mouse cursor of a currently executing window, its shape will change to indicate that it is executing and you cannot make any modifications to the contents of that window.

After execution is finished, the *Commit* and *Rollback* button will be enabled if a transaction has been started. In Multi session mode, each window has its own session, and therefore also has its own transaction. The *Commit* and *Rollback* button will reflect the transaction status of the topmost window.

## 15. Browsing objects

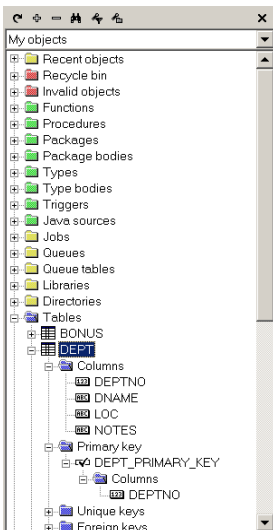
When you are developing a program unit in an Oracle database, it will always access other database objects. After all, if this would not be the case the program unit has no right to be in this database at all. To write such a program unit, you need to know exactly how these other database objects are defined. Therefore, questions like the ones listed below will be very familiar to an Oracle programmer:

- What is the data type of this column?
- What is the primary key of this table?
- What exactly was the name of that packaged function?
- What is the default value for this parameter?
- Which conditions cause this trigger to execute?
- Which objects call this function I'm working on?
- Et cetera...

All this information is available in the Oracle dictionary and accessible from within PL/SQL Developer by using the Browser.

### 15.1 Using the Browser

You can make the Browser available by selecting the *Browser* item in the *Tools* menu. The Browser shows all base object types in the database that are relevant to developing program units. You can navigate through this tree view in the same way that you use the Explorer in Windows to navigate through folders. Opening the *Tables* folder displays all tables that are accessible by the current session. Opening a specific table folder shows all relevant properties of this table. Opening the *Foreign keys* folder under a table displays all foreign keys for this table, and so on. The *Recent objects* folder contains all objects that you have recently used in PL/SQL Developer.



Whenever you open a folder, the information queried from the dictionary is read into memory to make subsequent accesses as quickly as possible. To force the Browser to re-query this information after something has changed, you need to select the modified object and press the *Refresh* button at the upper left side of the window. You can also refresh an object by selecting the corresponding item from the popup menu. To refresh the entire Browser, press the *Ctrl* key when pressing the *Refresh* button.

You can drag and drop objects from the Object Browser into the workspace. This will bring up an editor with the object definition. You can set a preference to control exactly what function should be performed (see chapter 16.11).

You can also drag and drop an object from the Object Browser into an editor. This will bring up a popup menu, whose contents depend on the object. You can select to copy the name (prefixed by the owner if necessary), properties, description, and other relevant text into the editor, at the location where you drop it. If you drag and drop a folder into an editor, all object names in the folder will be copied. These object names will be comma separated. You can set a preference to control if the object names are placed on one line or on multiple lines (see chapter 16.12).



By pressing the right mouse button on an object icon, a context sensitive popup menu appears that allows you to perform other functions on this object. The functions from this popup menu are described in the following paragraphs.

You can select multiple objects by using the *Control* or *Shift* key during selection. If multiple objects are selected, the object popup menu will show only those items that are applicable to all selected objects, and the drag and drop function will work on all objects simultaneously.

You can press *Ctrl* - and *Ctrl* + to quickly align the width of the browser with the contents.

The Browser preferences (see chapter 16.11) allow you to define which folders are visible, and in which order they are displayed. It also allows you to define the maximum number of objects that can be visible in the *Recent objects* folder. Furthermore you can define the double-click action for a browser object. By default this will expand the corresponding node, but you can also define that this will bring up the object definition editor, or the properties window.

## Creating a new object

To create a new program unit, table, view, sequence, or synonym, right click on such an object or on the root folder of this object type and select the *New* item from the popup menu.

Using the *New* function for a program unit will create a new Program Window. The initial contents are taken from a specific template, as described in chapter 22.

Views can be created in a SQL Window. The initial contents are taken from a specific template, just like with program units.

Tables, sequences and synonyms will be displayed in a specific editor window. The table, sequence and synonym editors are described in detail in chapter 8.

## Duplicate an object

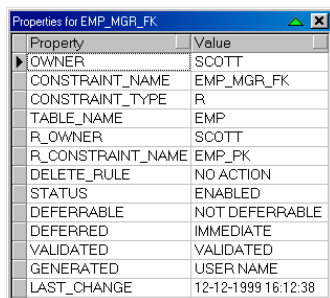
You can duplicate a table, sequence or synonym by selecting the *Duplicate* item from the popup menu. This will invoke an editor window with all properties of the selected object, except for the name. This can be useful if you want to create a similar table to perform some tests on.

## Copying an object or folder

To copy the name of an object or the names of the objects in a folder, select the *Copy comma separated* item. This will place the name(s) on the clipboard. You can set a preference to control if the object names are placed on one line or on multiple lines (see chapter 16.12).

## Viewing object properties

To view the properties of a specific object select the *Properties* item from the popup menu. The actual properties you see depend on the object type and version of your Oracle Server.



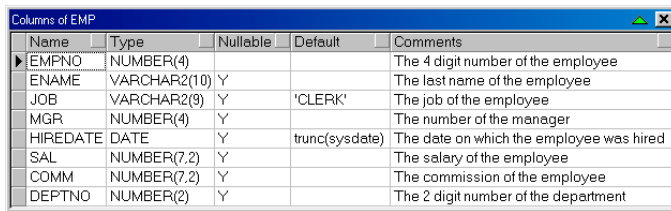
Property	Value
OWNER	SCOTT
CONSTRAINT_NAME	EMP_MGR_FK
CONSTRAINT_TYPE	R
TABLE_NAME	EMP
R_OWNER	SCOTT
R_CONSTRAINT_NAME	EMP_PK
DELETE_RULE	NO ACTION
STATUS	ENABLED
DEFERRABLE	NOT DEFERRABLE
DEFERRED	IMMEDIATE
VALIDATED	VALIDATED
GENERATED	USER NAME
LAST_CHANGE	12-12-1999 16:12:38

The properties are displayed in a modal window that you can place anywhere on the screen. You can keep a property window visible (it will stay on top of other windows) and continue editing in another window. You can additionally roll-up and roll-down the window by pressing the button on the upper-right of the window. This way you can easily make space for other things, and keep the properties available.

## Viewing object descriptions

Tables, views, functions, procedures, packages, types and methods can be 'described' in a way familiar from SQL\*Plus by selecting the *Describe* item from the popup menu. The description will be displayed in a modal window like the previously mentioned property window.

For tables and views all columns and their respective properties are displayed in a property window. For functions, procedures and methods, the parameters and their properties are displayed. For packages, the functions, procedures, variables, types, variables, constants and exceptions are displayed. For types, the methods and attributes are displayed.



Name	Type	Nullable	Default	Comments
EMPNO	NUMBER(4)			The 4 digit number of the employee
ENAME	VARCHAR2(10)	Y		The last name of the employee
JOB	VARCHAR2(9)	Y	'CLERK'	The job of the employee
MGR	NUMBER(4)	Y		The number of the manager
HIREDATE	DATE	Y	trunc(sysdate)	The date on which the employee was hired
SAL	NUMBER(7,2)	Y		The salary of the employee
COMM	NUMBER(7,2)	Y		The commission of the employee
DEPTNO	NUMBER(2)	Y		The 2 digit number of the department

You can select rows, columns or ranges of cells in a description window. To select rows or columns, just click on the row heading or column heading and drag the mouse pointer to highlight the selection. To select a specific range of cells, move the mouse pointer over the left edge of a cell until its shape changes, press the mouse button, and drag the mouse to highlight the selection. The selection can be copied or printed. To make a comma delimited copy of the selected cells, right-click on the window and select the corresponding item from the popup menu. To export the description to a CSV, TSV, HTML or XML-file, select the corresponding item from the *Export* submenu. To print the contents of a description window, select the *Print* item.

By double-clicking on a cell in a description window, the contents of the cell will be copied into the editor of the topmost window.

To sort the rows in a description window, press the heading button of the column on which you want the rows to be sorted.

## Viewing an object definition

All object types have some kind of definition. To view this definition, you can select the *View* item from the popup menu. This item is available for all program units, views, tables, sequences and synonyms. If you select the *View* item for a table element such as a column or index, it will display the table definition and navigate to the selected element.

Program units will be displayed in a Program Window, views will be displayed in SQL Window, and tables, sequences and synonyms will be displayed in a specific editor window. The table, sequence and synonym editors are described in detail in chapter 8.

## Editing an object definition

You can edit an object definition straight from the database by selecting the *Edit* item from the popup menu. You can disable this feature through a preference, as described in chapter 16.2.

## Renaming an object

Tables, views, sequences and synonyms can be renamed by selecting the *Rename* item from the popup menu. If you are using Oracle 9.2 or later you can also rename columns. A dialog will appear with the old name, which you can modify and apply.

## Dropping an object

Sometimes it is necessary to drop an object. It may have become obsolete, or must be changed from one type to another (e.g. from a function to a procedure). To accomplish this, just select the *Drop* item from the popup menu.

## Browsing a related object

Most objects have either referencing or referenced objects. By selecting the *Browse* item you can quickly locate a related object in the Browser.

## Recompiling an object

Objects often reference other objects in the database. Whenever these referenced objects change, the referencing object can become invalid. In this case a red mark is displayed in the icon of this object. To recompile an invalid object, select the *Recompile* item from the popup menu. If compilation succeeds, the mark disappears. If compilation fails, you can view the errors by selecting the *View* item from the popup menu.

## Adding an object source to an editor

The source file of a program unit can be added to a Program Window or Test Window by selecting the *Add source to editor* item.

For a Program Window, the source will be added on a new page after the current program unit. This way you can create a Program File that contains several program units that already exist in the database.

For a Test Window, the source will be added after the currently selected source during a debug session. This way you can include a source in the Test Window and subsequently set or remove breakpoints before actually stepping into the program unit.

## Adding debug information to an object

If you want to view or set variable values during a debug session, the object that contains these variables needs to contain debug information. You can manually add debug information by selecting the *Add debug information* item. This menu item has a checkmark that indicates whether the object already contains debug information. The debugger preference section contains an option to automatically add debug information when a program unit is compiled, in which case you only need to use this item for program units that you do not have in development.

## Recompiling referencing objects

When you alter an object that is referenced by other program units or views, these referencing objects will become invalid. Recompiling them will make them valid or will bring any incompatibilities to light, which can easily be accomplished by selecting the *Recompile referencing objects* item. This will bring up a Compile Invalid Objects Window, which will immediately be executed. More information about this tool can be found in chapter 17.3.

## Testing a program unit

To test a program unit (a standalone function or procedure, a packaged function or procedure, or an object type method) you can use Test Scripts, which are described in detail in chapter 4. A Test Script

will generally contain a simple function or procedure call. You need to set input parameters and view output parameters and function results after executing the Test Script. From the Browser you can select the *Test* item to create such a Test Script.

### Querying tables and views

To view the data in a table or view, select the *Query data* item from the popup menu. A SQL Window is created with a select statement for the table or view currently selected in the Browser. If you have the *AutoExecute SQL Window* preference enabled, the select statement will immediately be executed.

### Editing table data

If you want to edit the data in a table, you can select the *Edit data* item when the table is selected in the Browser. A SQL Window will be created with a select statement that includes the rowid. If you have the *AutoExecute SQL Window* preference enabled, the select statement will immediately be executed. The SQL Window is immediately in edit mode after the records are fetched.

### Exporting tables

To export the definition and data of a table, select the *Export data* item. This will bring up the *Export Tables* tool for the selected table. See chapter 17.4 for details.

### Enabling and disabling triggers and constraints

Triggers and constraints can be enabled or disabled from within the Object Browser by selecting the *Enable* or *Disable* item from the popup menu. Disabled triggers and constraints are displayed grayed out.

### Running a job

For jobs, submitted through the *dbms\_job* package, you can explicitly run it by selecting the *Run* item from the popup menu.

### Enabling and disabling enqueueing and dequeuing

For queues you can enable and disable enqueue and dequeue operations by selecting the corresponding item from the popup menu. A checkmark before the menu item indicates whether the operation is currently enabled.

### Adding an object to a user defined folder

To add an object to a user defined folder (see chapter 15.3), select the *Add to folder* item and subsequently select a folder from the submenu.

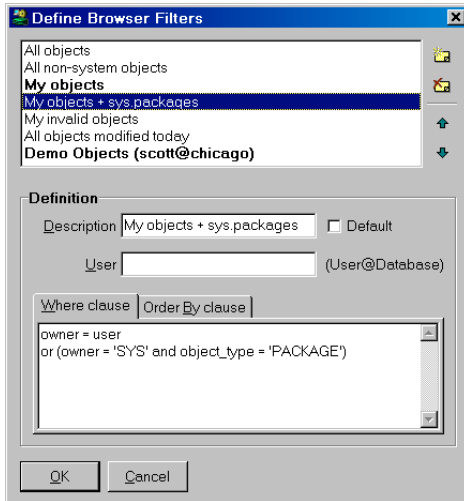
### Restoring or purging an object

If you are using Oracle10g or later, you can restore or purge tables that you have previously dropped, if this feature is enabled in the database. From an object in the *Recycle bin* folder you can select *Purge* to permanently drop the table and its data, or you can select *Restore* to restore it under the original name or under a new name,

## 15.2 Browser Filters

You can define filters that determine which objects are visible in the system folders of the Object Browser, and in which order these objects are displayed. A select list is located at the top of the Browser that allows you to quickly switch between the various filters.

Browser Filters can be defined by selecting the *Browser Filters* item in the Tools menu or by pressing the *Filters* button above the Browser. You can modify, create and delete filters in this dialog:



A filter is made up by the following properties:

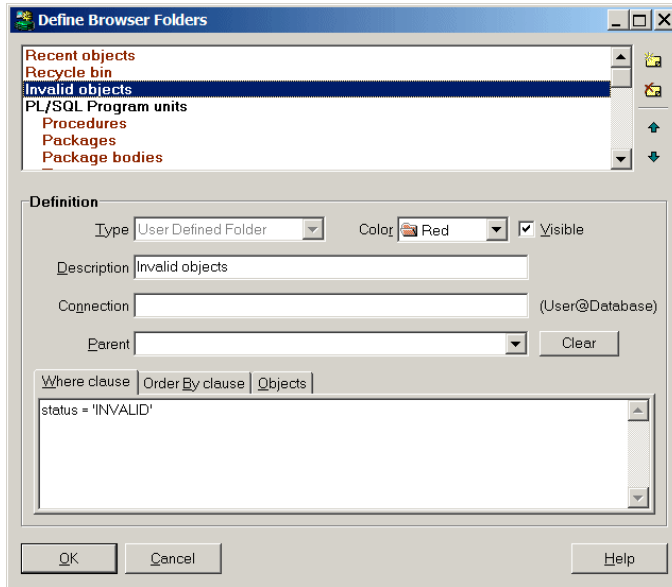
- **Description.**  
This description shows up in the select list at the top of the Browser.
- **User.**  
Defines for which Oracle user this filter is valid. If you leave this field empty, the filter will be valid for all Oracle users. If you specify a user and database (e.g. scott@chicago), the filter will only be visible if you are logged on as this user and database.
- **Default.**  
If you check this option, the selected filter will be the activated when you start PL/SQL Developer. You can define a default filter for each user. The default filters are displayed in bold.
- **Where clause.**  
A filter defines a select statement on the *all\_objects* (or *dba\_objects*) view. The where clause can be used to restrict the result set of this select statement. You can use the following columns:
  - owner - the owner of the object
  - object\_name - the name of the object
  - object\_type - the type of the object (TABLE, VIEW, PACKAGE, and so on)
  - status - the status of the object (VALID or INVALID)
  - created - the date/time that the object was created
  - last\_ddl\_time - the date/time that the object was most recently compiled
- **Order by clause.**  
Use the columns from the *all\_objects* view to order the objects in the Browser.

## 15.3 Browser Folders

The folders of the Object Browser can be customized in several ways. You can change the order of the folders in the browser, the color of each folder, a folder hierarchy, and you can hide specific folders based on your current connection.

You can also define your own folders and populate them with objects. You can either explicitly add specific objects to a folder, or define a query to populate it, or a combination of both. The query will be executed when the folder is opened in the Object Browser.

Browser Filters can be defined by selecting the *Browser Folders* item in the Tools menu or by pressing the *Folders* button above the Browser. You can modify, create and delete folders in this dialog:



At the top of this dialog you see all folders and their hierarchy. A red color indicates a system folder, a black color indicates a user defined folder. Some restrictions apply to system folders.

To the right of the folder list you have 4 buttons available to add a new folder at the position of the currently selected folder, to delete a folder, or to move it up or down in the list. The order in the Object Browser is the same as the order in this list.

At the bottom you can change the definition of the currently selected folder:

- **Type** – Shows the type of the folder: User Defined or System.
- **Color** – Select the color of the folder. Subfolders of objects in this folder will inherit the same color.
- **Visible** – Disable this option to make the folder invisible in the Object Browser. This can be used to hide specific system folders, and overrides the *Connection* option.
- **Description** – The description of the folder in the Object Browser. Cannot be changed for system folders.

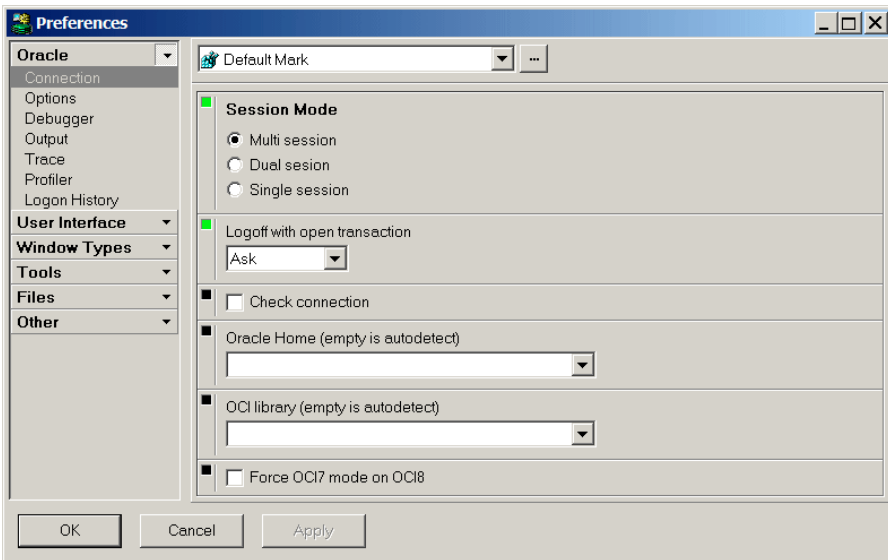
- **Connection** – Defines for which Oracle users this folder will be displayed. If you leave this option empty, the folder will be displayed for all Oracle users. If you specify a user and database (e.g. scott@chicago), the folder will only be visible if you are logged on as this user and database. You can use ? and \* wildcard characters to match multiple users. For example \*@prd for all users in the prd database.
- **Parent** – To hierarchically place a folder within another folder, you can select its parent from the list. To clear the parent, press the *Clear* button.
- **Where clause** – The where clause can be used for query based folders to restrict the selection of objects. In the example above, the where clause is set to status = 'INVALID' to populate the folder contents with all invalid objects.
- **Order By clause** – The order by clause can be used for query based folders to define the sort order of the selected objects.
- **Objects** – For manual folders you can define list of objects. You can place an object on each line in an *owner.name* format. For example: *SCOTT.EMP*. Note that it is usually more convenient to add an individual object from the Object Browser by right-clicking on it and selecting *Add to folder* from the popup menu.

The where clause and order by clause can include any SQL expression and any column from the *all\_objects* view. This works the same as for Browser Filters. See chapter 15.2 for more details.

Note that filters do not affect the object contents of user defined folders.

## 16. Preferences

The *Preferences* tool allows you to set various preferences for PL/SQL Developer. When you select this item, the following dialog appears:



At the left you can select a preference page from one of the 6 categories (Oracle, User Interface, Window Types, Tools, Files, and Other).

At the top you can select a preference set, which defines a set of preferences at a certain level. By default you will define personal preferences for your Windows user (in the screenshot above for the user *Mark*), but you can also define personal preferences for a specific Oracle connection. Whenever you use such a connection to logon to the database, the corresponding preference set will be used. Furthermore your system administrator can define default preferences at a global level and at Oracle connection levels. For detailed information about preference sets, see chapter 16.33.

Below the preference sets you see the preferences for the currently selected preference page (in the screenshot above for the *Oracle – Connection* page). For each preference you will see the current value. A green indicator to the left of the preference indicates that the value is set in the current preference set. If the indicator is black, the value is inherited from a different set (hold the mouse cursor over the indicator to view the preference set name). To revert a preference to its inherited value (in other words: remove it from the current preference set), you can click on the green indicator.

The following chapters will describe each preference page in detail.



## 16.1 Oracle – Connection

**Session Mode**

☒ Multi session  
☐ Dual session  
☐ Single session

**Logoff with open transaction**

Ask

☐ Check connection

**Oracle Home (empty is autodetect)**

**OCI library (empty is autodetect)**

☐ Force OCI7 mode on OCI8

- **Session mode.**  
Controls how many separate database sessions PL/SQL Developer will use. This feature is described in detail in chapter 14.
- **Logoff with open transaction.**  
When a session is logged off and this session has an open transaction, the default action is to commit that transaction. Through this preference you can define that such an open transaction is rolled back, or that you will be asked for confirmation.
- **Check Connection.**  
When this option is enabled, PL/SQL Developer checks every 60 seconds if your database connection is still alive. If your session is killed or the server is shutdown, PL/SQL Developer will automatically logoff.
- **Oracle Home.**  
By default PL/SQL Developer will use the Primary Oracle Home on your PC. You can define the Primary Oracle Home with the Oracle Home Selector (a standard Oracle utility). You can force PL/SQL Developer to use a specific Oracle Home by selecting one from this list.
- **OCI Library.**  
PL/SQL Developer will use the most recent version of SQL\*Net or Net8 that is installed. If this version of SQL\*Net or Net8 causes problems, you can force PL/SQL Developer to use another version. The DLL name will be something like ora72.dll (SQL\*Net 2.2), ora73.dll (SQL\*Net 2.3), oci.dll (Net8 8.1.x), and so on. These DLL's are located in Oracle's bin directory.
- **Force OCI7 mode on OCI8.**  
Some releases of Net8 (8.0.3 and 8.0.4) have some bugs that might cause problems in PL/SQL Developer. Setting this preference will cause the more solid SQL\*Net 2 functions of Net8 to be used instead. One example of a Net8 bug is that it reports incorrect parameter modes when describing procedures and functions. Disadvantage of this preference setting is that you can't query BLOB, CLOB and BFILE columns and that wrapped packages cannot be described.

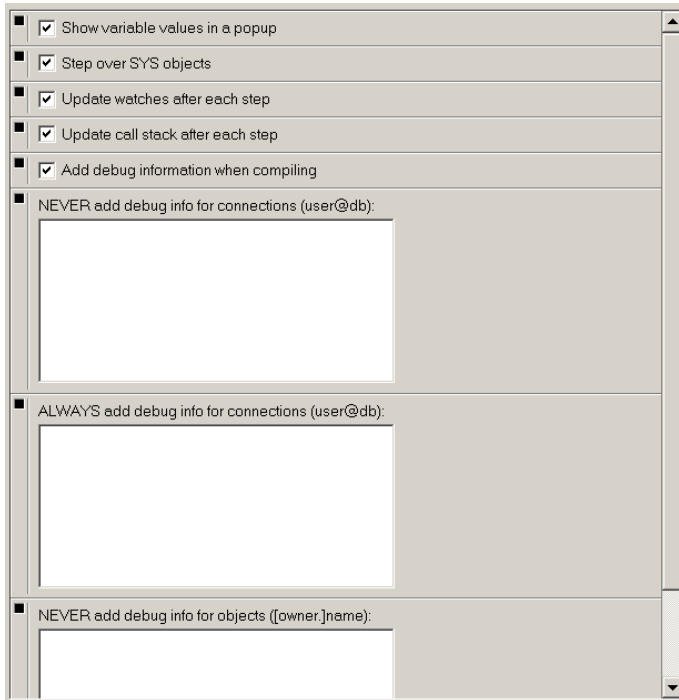
## 16.2 Oracle – Options

<input checked="" type="checkbox"/>	Allow editing of database source
<input checked="" type="checkbox"/>	Ask to save edited database source
<input checked="" type="checkbox"/>	Allow compilation of read-only source files
<input checked="" type="checkbox"/>	Confirm commit & rollback
<input type="checkbox"/>	Use DBA Views if available
<input checked="" type="checkbox"/>	Automatic statistics
<div> <div>Select...</div> <div> physical reads  physical writes  table scans (short tables)  table scans (long tables)  table scan rows gotten  table scan blocks gotten  table fetch by rowid </div> </div>	
<input checked="" type="checkbox"/>	Unicode enabled

- **Allow editing of database source.**  
This option controls if the source of a program unit or view can be edited directly from the Object Browser, or that it requires that it opened from the file system.
- **Ask to save edited database source.**  
If you edit a program unit from the source that is stored in the database dictionary (by right-clicking on the program unit and selecting *Edit* from the popup menu), you can prevent that you are asked to save these modifications to a file by disabling this option. If you have edited a program unit that was previously opened from the file system, you will always be asked to save these changes.
- **Allow compilation of read-only source files.**  
If a source file is read-only, it can not be compiled when loaded into a Program Unit Window. This preference is useful when using a Version Control System, which often leave a read-only copy of a source file in a working directory. Such a copy must not be compiled, as this might interfere with the work of other project members.
- **Confirm Commit & Rollback.**  
Whenever you commit or rollback a transaction by pressing the corresponding buttons, you will be asked for confirmation. Disabling this option will immediately commit or rollback without this confirmation.
- **Use DBA views if available.**  
PL/SQL Developer will try to use the DBA views (*dba\_objects*, *dba\_triggers*, and so on) instead of the ALL views (*all\_objects*, *all\_triggers*, and so on) if this preference is set. The DBA views might expose more information than the ALL views if a developer has certain system privileges.
- **Automatic statistics.**  
This option controls if a statistic report is to be generated for every executed SQL Statement and Test Script. If you disable this option, the statistic report will be empty. By pressing the *Select...* button, a list opens up with all statistics available in the database that you are currently connected to. You can select which statistic you wish to include in a statistic report. If you do not have access to the dynamic performance tables, the *Select...* button will be disabled.
- **Unicode enabled.**  
When selected, Unicode data will be fetched as such from the Oracle Server, and displayed as

Unicode text. When disabled, Unicode data from the server will be converted to the character set of the Oracle Client, in accordance with the NLS\_LANG key of the Oracle Home Registry.

## 16.3 Oracle – Debugger



- **Show variable values in a popup.**  
If you disable this preference, variable values will not automatically be displayed in a popup when you move the mouse cursor over its name in the Test Window. Instead, you must select the Set Variable item from the popup menu after right-clicking on the variable name.
- **Step over SYS objects.**  
If you have privileges to view sources of SYS objects, the debugger can step into these program units if you disable this option. Usually you do not want to do so, therefore this option is disabled by default.
- **Update watches after each step.**  
If you disable this option, you can manually refresh the watches by pressing the *Update watches* button on the debug-toolbar of the Test Window. Doing so may speed up debugging performance on slower configurations.
- **Update call stack after each step.**  
If you disable this option, you can manually refresh the call stack by pressing the *Update call stack* button on the debug-toolbar of the Test Window. Doing so may speed up debugging performance on slower configurations.

- **Add debug information when compiling.**  
When this option is enabled, each compilation in PL/SQL Developer will automatically add debug information, so that variable values can always be viewed and set during a debug session. If you disable this option, you can manually add debug information from the Browser.
- **NEVER add debug info for connections.**  
For this list of connection matches, debug information will never automatically be added, regardless of the “Add debug information when compiling” preference. This can be useful if you want to make sure that you never add debug information to production databases, or for certain users. You can use the familiar user@database syntax, where you can use the \* and ? wildcard characters (e.g. sys@\* for the sys user on any database).
- **ALWAYS add debug info for connections.**  
For this list of connection matches, debug information will always automatically be added, regardless of the “Add debug information when compiling” preference. This can be useful if you want to make sure that you always add debug information to development databases, or for certain users. You can use the familiar user@database syntax, where you can use the \* and ? wildcard characters (e.g. \*@devdb for the all users in the devdb database).
- **NEVER add debug info for objects.**  
For this list of objects debug information will never be added. Some Oracle Server versions can cause errors when adding debug information in specific situations. You will typically see ORA-00600, ORA-03113 or PLS-00801 errors during compilation. If this problem occurs, you can add the corresponding object to this list (owner.name) until Oracle provides a fix.

## 16.4 Oracle – Output



<input checked="" type="checkbox"/>	Enabled
<input checked="" type="checkbox"/>	Clear before execute
<input type="checkbox"/>	Buffer size
	10000 

- **Enabled.**  
In SQL Windows and Test Windows, calls to *dbms\_output.put\_line* will be buffered and displayed after execution of the script. This preference can be overruled on the output page of each individual SQL Window and Test Window.
- **Clear before execute.**  
Before executing a SQL Window or Test Window, the current output page will be cleared. This way, the output page will always contain just the output of the last execution.
- **Buffer size.**  
The size in bytes of the *dbms\_output* buffer. If more than this number of bytes is output during one execution, an exception will occur.

## 16.5 Oracle – Trace

Available Columns	Selected Columns
Event kind	Comment
Event sequence	Event time
Exception	Unit name
Proc line	Unit line text
Proc name	
Proc params	
Proc type	
Unit line	
Unit type	

These preferences control the *Trace* page of the Test Window (see chapter 4.11). You can control which columns from the trace table should be displayed, and in which order.

## 16.6 Oracle – Profiler

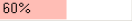

Available Columns	Selected Columns
Average time	Unit
Maximum time	Line
Minimum time	Total time
	Occurrences
	Text

**Time units**

☐ Seconds  
☒ Milliseconds  
☐ Microseconds

☐ Show 0 occurrences

**Graphical time display**

60%  

These preferences control the *Profiler* page of the Test Window (see chapter 5.3).

- **Columns.**  
Controls which columns from the profile table should be displayed, and in which order.
- **Time units.**  
Defines the unit of the *Total time*, *Maximum time*, *Minimum time* and *Average time* columns.
- **Show 0 Occurrences.**  
Determines if a reported source line with 0 execution occurrences will be displayed.
- **Graphical time display.**  
The *Total time* column will display a graphical representation of the relative time. You can control the brightness of the color of this bar, or make it invisible.

## 16.7 Oracle – Logon History

The screenshot shows the 'Definition' and 'History' sections of the PL/SQL Developer Preferences dialog. In the 'Definition' section, 'Store history' and 'Store with password' are checked. 'History size' is set to 8 and 'Display size' is set to 30. Under 'Sorting', 'Last used' is selected. The 'Fixed Users' list contains: >chicago, scott/tiger@chicago, sys@chicago as sysdba, >detroit, scott/tiger@detroit, sys@detroit as sysdba, and larry@detroit. The 'Keep fixed users grouped and unsorted' checkbox is checked, and 'Add fixed users to the history when used' is unchecked. The 'History' section shows a list with 'scott' and 'sys as SYSDBA'.

If you enable the *Store History* preference, PL/SQL Developer will store the username, password and database information in the Logon history. When you restart PL/SQL Developer later, the logon dialog allows you to quickly select a previously used account:

The screenshot shows the 'Oracle Logon' dialog box. The 'Username' field contains 'scott'. A dropdown menu is open, showing a list of logon history entries: 'scott', 'demo', 'chicago', 'detroit', 'scott@detroit', 'sys@detroit', and 'larry@detroit'. The 'detroit' entry is highlighted, and its sub-menu is also open, showing 'scott@detroit', 'sys@detroit', and 'larry@detroit'. The 'Database' field is empty, and 'Connect as' is set to 'Normal'. 'OK' and 'Cancel' buttons are at the bottom.

For security reasons you may not want to store passwords in the history, even though PL/SQL Developer encrypts them. This would imply that anyone could connect to the database with your accounts if they have access to PL/SQL Developer on your PC. In such a case you should disable the *Store with password* option. The *History size* and *Display size* preferences control how many connections will be stored (the oldest one will be removed from the history), and how many will be displayed on screen. The *Sorting* option controls how connections are sorted in the selection list.

Note that even if you have selected to store passwords in the logon history, you can still prevent this by holding down the *Control* key when pressing the *Return* key or the *OK* button during logon. This way you can prevent that passwords of privileged accounts (e.g. SYS or SYSTEM) can be recalled.

You can additionally define a number of fixed accounts in the *Fixed Users* section, with or without password. These fixed accounts will always be displayed and will never automatically be removed. You can additionally define submenu items for these fixed users by adding `>menuname`. In the example above, a submenu is created for *chicago* and *detroit*:

```
>chicago
scott/tiger@chicago
sys@chicago as sysdba
>detroit
scott/tiger@detroit
sys@detroit as sysdba
larry@detroit
```

In this case a submenu was created for a database, but you can create any kind of category you like. For example *Development*, *Test*, *Production*.

If the *Keep fixed users grouped and unsorted* option is enabled, the fixed users are displayed in a separate section from the history, and will not be sorted. If *Add fixed users to the history when used* option is enabled, selecting a fixed user during logon will also add it to the history.

In the *History* section you can see which users were added to the history as a result of a logon in PL/SQL Developer. To delete a specific item, select it and press the *Delete item* button next to the list.

## 16.8 Oracle – Hints

**Compiler Hints**

**Unused Declarations**

Variables	<input checked="" type="checkbox"/>
Constants	<input checked="" type="checkbox"/>
Types	<input checked="" type="checkbox"/>
Parameters	<input checked="" type="checkbox"/>
Exceptions	<input checked="" type="checkbox"/>

**Unused Assigned Values**

Enabled	<input checked="" type="checkbox"/>
Dummy Variable Prefixes	Dummy

**Errors**

Compare with null	<input checked="" type="checkbox"/>
Compare with empty string	<input checked="" type="checkbox"/>

**Naming Conventions**

Element	Prefix	1st char	Allowed chars	Suffix	Description
<input checked="" type="checkbox"/> Parameter	p_	Any	Any		must start with 'p_'
<input checked="" type="checkbox"/> Package	Self	None	None		must be 'Self'
<input checked="" type="checkbox"/> PLSQLType	T	Any	Any		must start with 'T'
<input type="checkbox"/>					

### Compiler Hints

You can enable and disable each individual hint, and control other properties of hints when appropriate. An explanation of each hint and property is displayed below the hint list.

## Naming Conventions

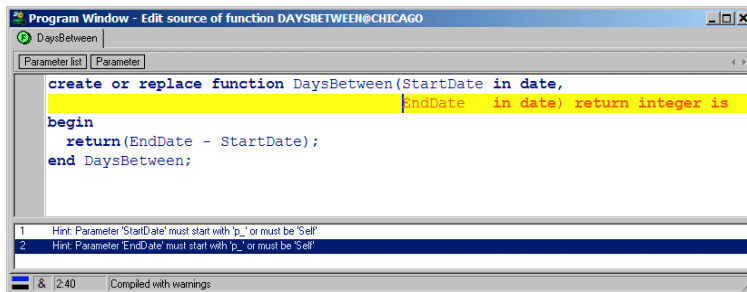
In this list you can define naming conventions that will be checked when a program unit is compiled, or when the *Show Compiler Hints* function is invoked. A hint will be displayed for each element in the program unit source that does not meet these naming conventions.

For each naming convention you can define the element type (parameter, variable, and so on), the required prefix for the element type, the possible values for the first and subsequent characters (after the prefix), the required suffix, and a description for the hint that should be displayed when the naming convention is not met.

The description should be in the form as displayed in the example above. The actual hint message will be <Element> <Name> <Description>. For example: Parameter 'StartDate' must start with 'p\_'.

If an element must meet one of multiple naming conventions, then you can simply add multiple lines for the same element type. The descriptions of all lines will be displayed, separated with 'or'. For the example above the parameter hint message will be:

Parameter 'StartDate' must start with 'p\_' or must be 'Self'





## 16.9 User Interface – Options

☒ Autosave username  
☐ Autosave desktop  
☐ Use internal HTML viewer  
☐ HTML help window : "stay on top"  
☐ Use multi-row tabs  
☐ Show complete file path in window titles  
☒ Use projects  
☒ Position messageboxes near mouse pointer  
☒ Show file dialogs with details view

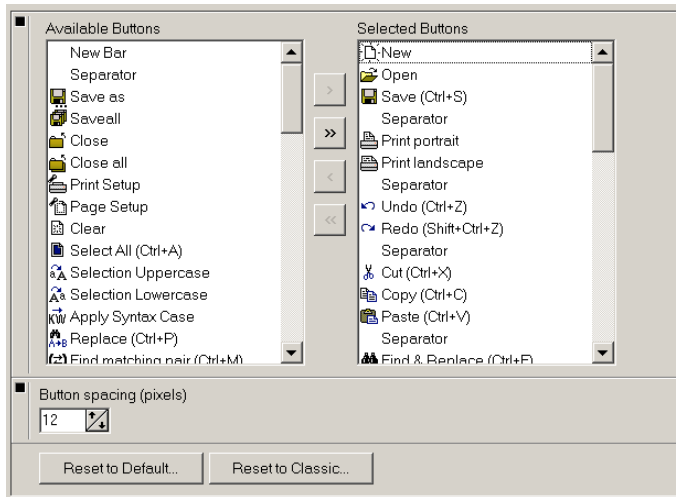
DSA Dialogs... Shows a list of dialogs that have the "Don't Show this message Again" option checked. You can delete lines for the messages you want to enable.

- Autosave username.  
 This option can be checked to automatically save the username of each PL/SQL Developer session. When you re-start PL/SQL Developer later, the last username will be default in the logon dialog.
- Autosave desktop.  
 When this option is enabled, all files that were opened when you stop PL/SQL Developer will automatically be reopened when it is started again. A currently opened Command Window or Explain Plan Window, which are not associated with a file, will also be restored. Furthermore, the dimensions of the main window and Object Browser will be restored.
- Use internal HTML viewer.  
 By default PL/SQL Developer will use an internal HTML viewer to display the HTML help documentation. If you have Microsoft Internet Explorer 4 or later, you can disable this option and use Microsoft's HTML components instead. This will give you some extra features in the popup menu, and will inherit preferences made in Internet Explorer.
- HTML help window: "stay on top".  
 If the option is enabled, the HTML Help window will stay on top until you close or minimize it. If it is disabled, the HTML Help window will appear as a separate task, with its own button on the Windows task bar. This way you can bring PL/SQL Developer main window or the Help window to the front whenever you need it.
- Use multi-row tabs.  
 If a window contains multiple tab pages, this preference controls whether these tab pages can be divided over multiple rows in case of an overflow. If this option is disabled, 2 scroll buttons will appear in case of a tab overflow situation. This currently only applies to the Program Window.
- Use complete path in window titles.  
 By default, only the filename is displayed in a window title. If this option is enabled, the complete path will be displayed.
- Use projects.  
 The *Project* main menu item will only be visible if this option is enabled. Disable it if you are not using projects, so that it does not take up any space in the main menu. For more details about projects, see chapter 12.
- DSA Dialogs.  
 Whenever you select the "Don't show this message again" option on a message box, an entry will be added to the DSA section in your preferences. If you subsequently want to re-enable this

message, press the *DSA Dialogs* button to edit this section. This will bring up a text editor that allows you to remove the corresponding line.

- Position message boxes near mouse pointer.  
When enabled, message boxes will be positioned near the mouse cursor when displayed, so that you can quickly reach a button. When disabled, message boxes will always be centered on the screen.
- Show file dialogs with details view.  
When enabled, file open and save dialogs will always start with the Details view. When disabled, the default view style will be used.

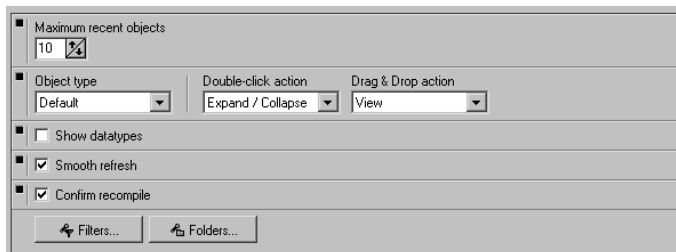
## 16.10 User Interface – Toolbar



You can control which buttons should be visible, and in which order. Use the *Separator* item to separate groups of related buttons. The *New Bar* item can be used to create a new toolbar below the previous toolbar.

The *Reset to Default* button resets the toolbar to its default state. The *Reset to Classic* button resets the toolbar to the PL/SQL Developer 3 layout. Use the *Button spacing* value to control the spacing between each button.

## 16.11 User Interface – Browser



These preferences control the appearance and behavior of the Object Browser (see chapter 15).

- **Maximum Recent objects.**  
Controls how many objects will be visible in the *Recent objects* folder. The least recently used object will be removed when more than this number of objects would be displayed.
- **Double-click and Drag & Drop actions.**  
Controls which action will be performed when you double-click on an object in the browser, or when you drag & drop it from the browser onto the workspace. You can specify a default action, and override or inherit this default action for each object type from the list.

- **Show data types.**  
When this option is enabled, the data types of columns (tables and views), attributes (objects) and parameters (functions and procedures) will be displayed in the browser.
- **Smooth Refresh.**  
On some systems this option may slow down the Browser's refresh operation (e.g. changing the Browser Filter, logging on as a different user, and so on). If this is the case you should disable it, though this may lead to some screen flickering when the Browser is refreshed.
- **Confirm Recompile.**  
When this preference is enabled, recompiling a program unit in the Object Browser will result in a success or failure message. When it is disabled, you will only see a message when compilation fails.

The *Filters* (see chapter 15.2) button and *Folders* (see chapter 15.3) button will invoke the corresponding dialog.

## 16.12 User Interface – Editor

Indent				
<input checked="" type="checkbox"/>	Automatic	Step (chars)	2	
Tabs & Wrapping				
<input checked="" type="checkbox"/>	Smart tab	Tab size	2	
<input type="checkbox"/>	Smart fill			
<input type="checkbox"/>	Use tab character			
<input type="checkbox"/>	Wrap lines			
<input type="checkbox"/>	Allow cursor after end of line	Visible margin	0	
Syntax Highlighting				
<input checked="" type="checkbox"/>	Enabled			
<input checked="" type="checkbox"/>	Keywords	Bold	Italic	Color
<input checked="" type="checkbox"/>	Comment	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Text
<input checked="" type="checkbox"/>	Strings	<input type="checkbox"/>	<input type="checkbox"/>	Bkg
<input checked="" type="checkbox"/>	Numbers	<input type="checkbox"/>	<input type="checkbox"/>	
<input checked="" type="checkbox"/>	Symbols	<input type="checkbox"/>	<input type="checkbox"/>	
<input checked="" type="checkbox"/>	Custom	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Custom Keywords...				
<input type="checkbox"/>	Keyword case	Unchanged		
AutoReplace				
<input checked="" type="checkbox"/>	Enabled			
	Definition file			
	Edit...			
Other				
<input checked="" type="checkbox"/>	Allow folding			
<input checked="" type="checkbox"/>	Display line numbers	Interval	All	
	Font	Size		
	Courier New	9		
<input checked="" type="checkbox"/>	Show line status			
<input type="checkbox"/>	Highlight edit Line			
<input type="checkbox"/>	Wrap to start / end of file when text is not found			
<input type="checkbox"/>	Allow variable width for fixed width fonts			
<input checked="" type="checkbox"/>	Highlight parentheses			
<input type="checkbox"/>	Highlight color			
<input type="checkbox"/>	Search hit color			
<input type="checkbox"/>	Paste comma separated values on one line			

### Indent

- Automatic.  
Determines if the editor will automatically indent after pressing the enter key.

- Step (chars).  
The number of characters that the editor will indent when indenting or unindenting a selection.

## Tabs & Wrapping

- Smart tab.  
Will cause the editor to tab to a position relative to the previous line.
- Tab size.  
Determines the number of characters that a tab will use.
- Smart fill.  
The editor will replace spaces by tabs if possible.
- Use tab character.  
When disabled, only spaces will be used for tabs and indents.
- Wrap lines.  
Lines that are wider than the window width will be continued on the next line.
- Allow cursor after end of line  
When enabled, you can place the cursor after the end of a line. When you insert text there, the line will be padded with spaces first.
- Visible margin.  
Will display a vertical line at the indicated position. 0 = disabled.

## Syntax Highlighting

This group of preferences control the syntax highlighting that is used in the editors. You can enable or disable syntax highlighting, and define the style and color of keywords, comment, strings, numbers and symbols (`:=`, `=>`, `||`, and so on). For keywords you can additionally control if they should be converted to uppercase, lowercase or capitalized. You can also define a custom syntax highlighting, and define the words that should be highlighted. Pressing the *Custom Keywords* button will bring up an editor that allows you to define these words.

## AutoReplace

When enabled, you can define words that are automatically replaced with a replacement text. For example, if you define the word *wo* with the replacement text “*when others then*”, you can simply type *wo* to insert that text.

Pressing the *Edit* button allows you to edit the file with words and replacement texts. On each line in this file you can enter a *word=replacement text* specification. For example:

```
wo=when others then
rs=rollback to savepoint
```

And so on. The *Definition file* field allows you to specify which file is used for the AutoReplace definitions.

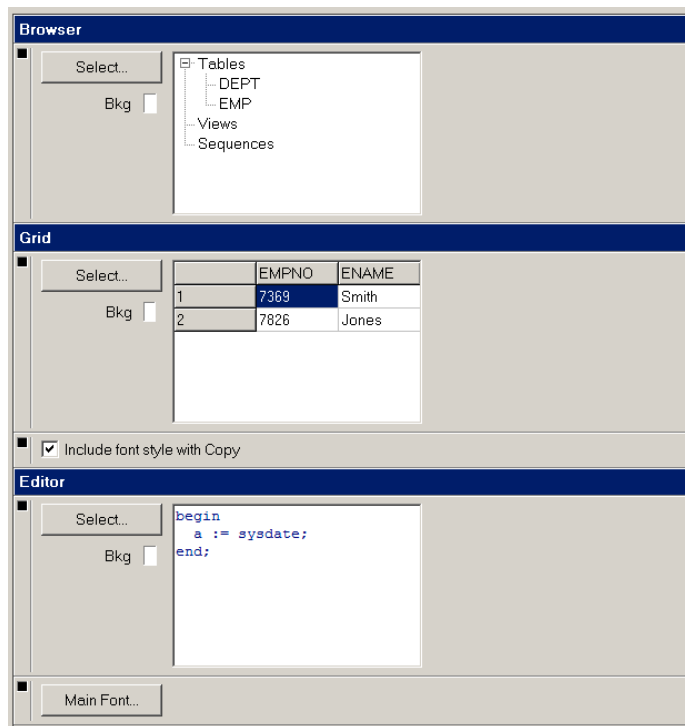
## Other

- Allow Folding.  
When enabled, the PL/SQL Editor of the Program Window will allow you to fold and unfold code sections, as described in chapter 18.14.
- Display line numbers.  
Check this option to make line numbers visible in the left margin. The *Interval* property controls

the interval between each displayed line number. Use the *Font* and *Size* to alter the appearance of the line numbers.

- **Show line status.**  
When enabled, the editor will show a yellow mark in the margin for changed lines, and a green mark for new lines.
- **Highlight edit line.**  
Controls if the line where the text cursor is located is highlighted.
- **Wrap to start / end of file when text is not found.**  
When using the find function, the search will continue at the start of the file (when searching forward) or the end of the file (when searching backward) if the search text is not found.
- **Allow variable width for fixed width fonts.**  
This option allows you to correctly display double-width characters (e.g. Japanese) with fixed width fonts. It can also be used for fixed width fonts that are wider in bold style than in regular style. If you are using such a font, you can use this option if you see truncated characters.
- **Highlight parentheses.**  
When enabled, the editor will highlight the parenthesis pair when the cursor is located at the opening or closing parenthesis.
- **Highlight color.**  
Determines the color of all highlights such as parentheses, variables, and so on.
- **Search hit color.**  
Determines the color of the search hits found by the search bar.
- **Paste comma separated items on one line.**  
Controls if multiple items selected in the Object Browser or in a Description Window are pasted into an editor on one line or on multiple lines.

## 16.13 User Interface – Fonts



The Fonts preferences control the browser font, editor font and the font used in the various grids like in the SQL window, the property windows, and so on. By pressing the *Select* button you can change the name, style and color of the font. You can also set the main font for all dialogs.



## 16.14 User Interface – Code Assistant

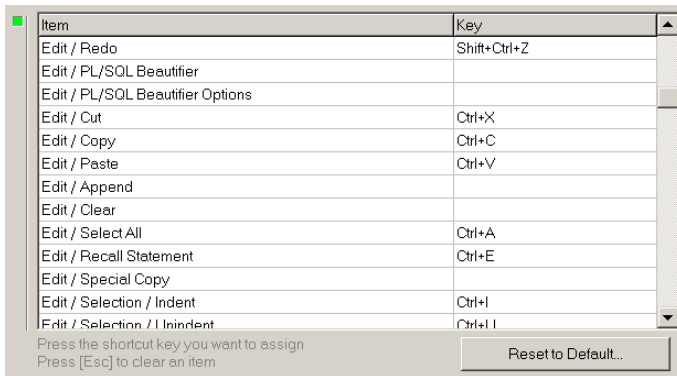
The screenshot shows the 'Code Assistant' preferences dialog box. It has a tree view on the left with the following items: 'Automatically activated', 'Delay (ms)', 'Coding Style', 'Describe Users', 'Describe Context', and 'Describe standard functions (e.g. to\_char)'. The 'Automatically activated' checkbox is checked. The 'Delay (ms)' field is set to 500. The 'Coding Style' dropdown is set to 'Smart'. The 'Use original case if possible' checkbox is checked. The 'Describe Users' section is expanded, showing a list of object types: 'Tables', 'Views', 'Sequences', 'Functions', 'Procedures', 'Packages', and 'Types', all of which are checked. The 'Describe Context' section is also expanded, showing 'Minimum characters' set to 3. The 'Describe standard functions (e.g. to\_char)' checkbox is checked.

The Code Assistant automatically displays information about database objects as you type their name (see chapter 18.3). This preference page allows you to define the behavior of this feature.

- **Automatically activated.**  
The Code Assistant can automatically be invoked after a certain delay (see below). You can also choose to manually activate the Code Assistant through a function key.
- **Delay.**  
The number of milliseconds that the editor will wait before it displays the Code Assistant List.
- **Coding Style.**  
Controls how selected items are inserted into the editor when you select them:
  - **Smart** – The Code Assistant will look at the described object to determine the style.
  - **Init Caps** – The first character of each word (separated by an underscore) in capitalized.
  - **Lowercase** – All characters are converted to lowercase.
  - **Uppercase** – All characters are converted to uppercase.
- **Use original case if possible.**  
When this option is enabled, the Code Assistant will determine the case of the identifier from the source that is stored in the Oracle Dictionary if possible. This applies to all program units and their elements (parameters, types, and so on) and to view columns, and overrides the Coding Style preference described above. If the original case cannot be determined, the Coding Style will be applied.  
You may want to disable this feature for performance reasons.
- **Describe users.**  
Determines if the objects owned by a user are listed when you type a username followed by a period. If this option is enabled, you can additionally define which object types you want to include in the list.

- Describe context.  
Determines if the Code Assistant should describe the context of the current user, editor, and program unit.
- Minimum characters.  
Determines how many characters of a word need to be typed before the context description can automatically be invoked. Note that you can always invoke the Code Assistant manually, even if the number of characters have not been typed.
- Describe standard functions.  
By default the Code Assistant will describe standard functions such as `to_char`, `add_months`, and so on. If you are very much familiar with these functions, you can disable this option.

## 16.15 User Interface – Key Configuration



On this page you can define your own key configuration for all functions in PL/SQL Developer. Just select the function in the list, and press the key combination that you want to use for this function. To clear a hotkey for a specific function, select it from the list and press the *Esc* key.

## 16.16 User Interface – Appearance

**Language**

[none] ▼

---

**Background Gradient**


☒ Enabled

☒ Vertical

☐ Default colors

Color 1 ■

Color 2 ■




---

**Connection Indicators**

Match (user@database)	Color
*@proddb	<span style="background-color: red; color: black;">...</span>
sys@*	<span style="background-color: yellow; color: black;">...</span>
*	<span style="background-color: white; color: black;">...</span>

☒ Set toolbar indicator

☒ Left aligned

☐ Right aligned

☐ Set editor background color

☐ Set application background color

---

☒ Faded disabled buttons  
(requires 64k colors or more)

---

☐ Display buffering  
(flicker free display, but slower performance)

---

☒ Autocomplete dropdown lists

---

**Disable XP Style**      Enable or disable visual XP style (only works on XP)  
Requires a restart, and works for all users

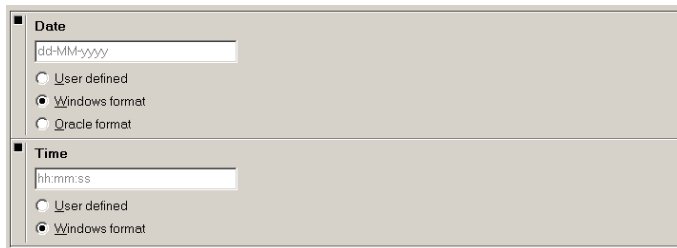
**Reset docking**

- **Language.**  
If you have a multi-language version of PL/SQL Developer, you can select a language here. All captions and messages will subsequently be displayed in that language.
- **Background Gradient.**  
Allows you to define the color and gradient of PL/SQL Developer's workspace. If you do not want to use a gradient, but just a single color, set Color 1 and Color 2 to the same value.
- **Connection Indicators.**  
These options allow you to provide some visual indication when PL/SQL Developer is using specific connections. You can define a number of connection matches in user@database format, and associate a color with it. This color can subsequently be associated with a left or right aligned toolbar indicator, the editor background, or with the application background (overriding the background gradient).  
A connection match can contain the \* and ? wildcard characters. In the example above, the editor background and right aligned toolbar indicator will be red when connected to the *PRODDB* database with any user. It will be yellow when connected as *SYS* to any database. The first item in

the list will take precedence, so the indicators will be red when connected as *SYS* to the *PRODDb* database.

- **Faded disabled buttons.**  
When enabled, disabled buttons are displayed with faded colors. When disabled, the standard grayed out appearance is used for disabled buttons. For 256 color displays, disabled buttons are always grayed out.
- **Display buffering.**  
The screen will show less flickering when moving or resizing windows, but the display performance may be slightly slower.
- **Autocomplete dropdown lists.**  
When enabled, edit fields with dropdown lists will automatically be completed with a matching item from the list.
- **Disable XP Style.**  
This button disables the XP Style as defined in the Display Properties of the Windows Operating System.
- **Reset docking.**  
Resets the docking windows to the default configuration.

## 16.17 User Interface – Date/Time



The screenshot shows a dialog box for configuring date and time formats. It is divided into two sections: 'Date' and 'Time'. The 'Date' section has a text input field containing 'dd-MM-yyyy' and three radio buttons below it: 'User defined' (unselected), 'Windows format' (selected), and 'Oracle format' (unselected). The 'Time' section has a text input field containing 'hh:mm:ss' and two radio buttons below it: 'User defined' (unselected) and 'Windows format' (selected).

On this page you can define the date and time formats that will be used in PL/SQL Developer.

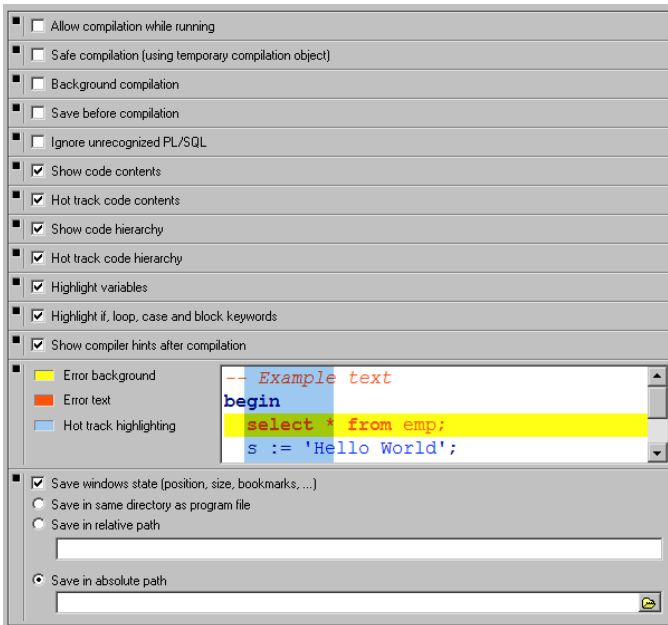
The date format can be defined in 3 ways:

- **User defined.**  
An explicitly defined date format, following the same rules as the Windows regional settings.
- **Windows format.**  
The format as defined in the Windows regional settings.
- **Oracle format.**  
The format as defined by your Oracle environment (NLS\_DATE\_FORMAT).

The time format can be defined in 2 ways:

- **User defined.**  
An explicitly defined time format, following the same rules as the Windows regional settings.
- **Windows format.**  
The format as defined in the Windows regional settings.

## 16.18 Window Types – Program Window



These preferences affect the behavior of the Program Window (see chapter 3).

- **Allow compilation while running.**  
In Multi or Dual session mode you are technically able to compile program units while you are running Test Scripts or SQL Scripts. This might lead to locks, which can be prevented with this preference.
- **Safe compilation (using temporary compilation object).**  
When enabled, each compilation is first tested by using a temporary compilation object. When this test compilation succeeds, the actual object is compiled. This can be useful if you are compiling objects in an environment where invalid objects can cause problems.
- **Background compilation.**  
When enabled, compilation is performed in a background thread, using a separate session. This way you can continue other work while the compilation occurs. It will lead to an extra database session for each Program Window though. Note that this preference is only effective if the Session Mode preference (see chapter 16.1) is set to Multi Session.
- **Save before compilation.**  
If you want to ensure that the database object and the source file on the file system stay in sync, you can enable this option.
- **Ignore unrecognized PL/SQL.**  
When enabled, the Program Window will ignore all SQL that does not start with “create or replace <program unit type>”. Other SQL or SQL\*Plus commands in the program file will be ignored.

- **Show code contents.**  
This preference controls whether the Code Contents pane will be displayed. See chapter 18.12 for more details.
- **Hot track code contents.**  
Controls whether the PL/SQL code in the editor will automatically be highlighted when the mouse moves over an item in the Code Contents pane.
- **Show code hierarchy.**  
This preference controls whether the Code Hierarchy pane will be displayed. See chapter 18.13 for more details.
- **Hot track code hierarchy.**  
Controls whether the PL/SQL code in the editor will automatically be highlighted when the mouse moves over an item in the Code Hierarchy pane.
- **Highlight variables.**  
When the cursor is located on a variable, all occurrences of this variable are highlighted. This makes it easy to see where a variable is used and what the impact of a change might be. If this option is disabled, you need to use the *Find matches* item from the *Edit* menu.
- **Highlight if, loop, case and block keywords.**  
When the cursor is located at an *if / then / else / elsif / end if* keyword, a *loop / end loop* keyword, a *for / loop / end loop* keyword, a *while / loop / end loop* keyword, a *case / when / then / else / end case*, and *declare / begin / end* keywords. If this option is disabled, you need to use the *Find matches* item from the *Edit* menu.
- **Show compiler hints after compilation.**  
Compiler hints provide information about unused declarations, unused assignments, potential errors, and warnings. When this preference is enabled, these hints will automatically be displayed after compilation, below the compilation errors (if present). If you disable this preference, you have to explicitly select the *Show Compiler Hints* item from the *Tools* menu or the toolbar.
- **Error background.**  
Select the color of the background of the text of a compilation error.
- **Error text.**  
Select the color of the text of a compilation error.
- **Hot track highlighting.**  
Select the background color of highlighted text for code contents hot tracking, code hierarchy hot tracking, and *if, loop, case and block* keywords.
- **Save window state.**  
When enabled, the position, sizes, bookmarks, color marks, and substitution variables of a Program Window are saved in a separate *.pfi* file and restored when the file or database object is subsequently viewed or edited again. The location of these files can also be specified.

## 16.19 Window Types – SQL Window

<input type="checkbox"/>	AutoCommit SQL
<input type="checkbox"/>	AutoCommit posted records
<input checked="" type="checkbox"/>	AutoExecute queries
<input type="checkbox"/>	AutoSelect statement
<input checked="" type="checkbox"/>	Ask to save new windows
<input type="checkbox"/>	Linked query in new window
<input type="checkbox"/>	Number fields to_char
<input type="checkbox"/>	Date fields to_char
<input checked="" type="checkbox"/>	Show dictionary info in result grid
<input type="checkbox"/>	Show gutter (line numbers)
<input type="checkbox"/>	Null value cell color
<input type="checkbox"/>	Alternate row color
<input checked="" type="checkbox"/>	Enabled
<input checked="" type="checkbox"/>	<b>Records per Page</b>
<input checked="" type="radio"/>	Automatically determined
<input type="radio"/>	Fixed
	100
<input type="radio"/>	All records
<input checked="" type="checkbox"/>	Maximum result set size (0 is unlimited)
	100  MB
<input checked="" type="checkbox"/>	<b>Query By Example</b>
<input type="checkbox"/>	Case insensitive
<input type="checkbox"/>	Automatic contains
<input type="checkbox"/>	Ignore time fraction
<input checked="" type="checkbox"/>	<b>Number Layout</b>
<input type="radio"/>	Left aligned
<input type="radio"/>	Right aligned
<input checked="" type="radio"/>	Formatted
<input type="radio"/>	Formatted with thousand separator

These preferences affect the behavior of the SQL Window (see chapter 6).

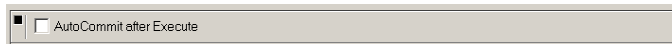
- **AutoCommit SQL.**  
This option controls if DML statements executed in a SQL Window are automatically committed.
- **AutoCommit posted records.**  
This options controls if records posted through the result grid are automatically committed.
- **AutoExecute queries.**  
If this option is enabled, a SQL Window that is created when selecting *Query Data* or *Edit Data* for a Table or View object will immediately be executed. If this option is disabled, you can modify the generated select statement before executing it.
- **AutoSelect statement.**  
If you have more than 1 SQL statement in the SQL Editor (separated by semi-colons), the SQL Window will automatically select the statement where the cursor is currently located before execution. This way you don't have to explicitly select the statement yourself.

- Ask to save new windows.  
When you close a SQL Window that was just created, you will normally be asked if you want to save it to a file. If you disable this option, this confirmation will be omitted. If you have modified a previously opened SQL file, you will always be asked to save these changes, regardless of this preference.
- Linked query in new window.  
When this preference is set, executing a linked query (see chapter 6.4) will create a new SQL Window. If it is not set, the linked query will be executed in the current SQL Window.
- Number fields to \_char.  
When this preference is enabled, number fields will be converted to character values on the server. The NLS parameters of the session will determine the format of the values. PL/SQL Developer will not perform any additional number formatting or validation.
- Date fields to \_char.  
When this preference is enabled, date fields will be converted to character values on the server. The NLS parameters of the session will determine the format of the values. PL/SQL Developer will not perform any additional date formatting or validation, and the date picker will not be available when editing date fields.
- Show dictionary info in result grid.  
When enabled, the following dictionary info will be displayed in the result grid:
  - The column data type, optionality, and comment will be displayed on the status line.
  - A lookup list will be displayed for columns with a check constraint that checks for specific values (e.g. col in (value1, value2, ...))
  - A Lookup list will be displayed for columns with a foreign key constraint to small tables of less than 1000 rows.
- Null value cell color.  
This color will be used to indicate null values. This makes null values easier to recognize for values with only blanks, and values that are not directly displayed in the grid (e.g. LOB's and Longs).
- Alternate row color.  
If enabled, this color will be used to visualize alternate rows in the result grid.
- Records per page.  
This preference determines how many records are initially retrieved for a select statement, and each time you press the *Next Page* button in the result grid of a SQL Window:
  - Automatically determined – The size of the result grid determines how many records are retrieved.
  - Fixed – The number of records specified are retrieved.
  - All records – All records are immediately retrieved when the query is executed.
- Number layout.  
This preference controls how number columns are displayed:
  - Left aligned – Values are displayed left aligned, without any formatting.
  - Right aligned – Values are displayed right aligned, without any formatting.
  - Formatted – Values are displayed right aligned, with a format that corresponds to the precision and scale of the field.



- Formatted with thousand separator – Similar to the previous option, but will also include a thousand separator (e.g. 1,277.65)
- Query By Example.  
These preferences control the default behavior of the SQL Window during QBE mode (see chapter 6.3):
  - Case insensitive – Query values of character fields are treated case insensitive.
  - Automatic Contains – Character fields only need to contain the query values.
  - Ignore Time fraction – The time fraction of date fields is ignored if the query value does not contain a time fraction.

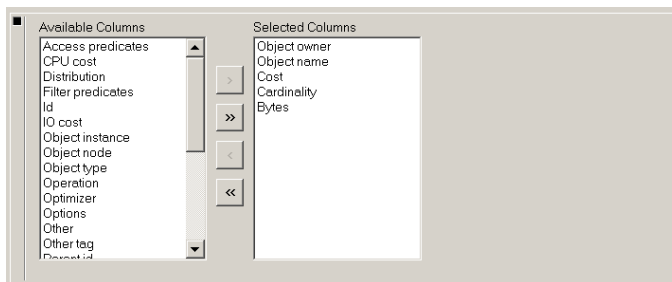
## 16.20 Window Types – Test Window



☐ AutoCommit after Execute

- AutoCommit after execute.  
This option controls if executed Test Scripts are automatically committed.

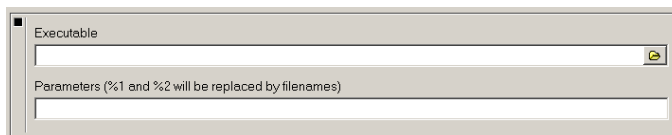
## 16.21 Window Types – Plan Window



Available Columns		Selected Columns
Access predicates	>	Object owner
CPU cost	>>	Object name
Distribution	<	Cost
Filter predicates	<<	Cardinality
Id		Bytes
IO cost		
Object instance		
Object node		
Object type		
Operation		
Optimizer		
Options		
Other		
Other tag		
Parent id		

These preferences control which columns of the plan table should be displayed in the Plan Window (see chapter 5.1), and in which order they should appear.

## 16.22 Tools – Differences



Executable

Parameters (%1 and %2 will be replaced by filenames)

On this preference page you can select the external difference tool for the Compare User Objects tool (see chapter 17.10).

# 16.23 Tools – Data Generator

General Preferences

■

Date Format

■

Commit every ..

0

■

Delay (mSec)

0

■

Number of items

10..20

User Definitions

■

Item	Value
*	

This preference page allows you to define a global set of defaults for the Data Generator tool. These defaults can be overruled for each data generator definition. For more details about the meaning of these preferences, see chapter 17.9.

## 16.24 Tools – To-Do List



The screenshot shows a preferences dialog box titled "Tools – To-Do List". It contains three sections, each with a list box and a scroll bar:

- Priorities:** A list box containing "1 - High", "2 - Medium", and "3 - Low".
- Categories:** A list box containing "Fix", "Optimize", "Finish", "Test", and "Review".
- Owners:** A list box containing "John", "Bill", and "Carroll".

These preferences control the values in the *Priority*, *Category* and *Owner* selection lists in the To-Do Item editor. For the priorities it can be useful to prefix each value with a number, so that the list can be sorted by priority. See chapter 13 for more details.

## 16.25 Tools – Recall Statement



The screenshot shows a preferences dialog box titled "Tools – Recall Statement". It contains two sections:

- Maximum size (statements):** A text box containing the value "200" and a spin button.
- Directory for recall file (PLSRecall.dat):** A text box containing the path "C:\WINDOWS\Application Data\PLSQL Developer\" and a browse button.

On this preference page you can define the maximum number of statements in the statement recall buffer, and the directory where it should be stored. See chapter 18.4 for more details.

## 16.26 Files – Directories

The screenshot shows a dialog box titled "Files – Directories". It contains a list of file types on the left and corresponding text input fields on the right. The file types are: Program files, Test scripts, SQL scripts, Report files, Command files, Diagram files, Macros, Templates, Plug-Ins, Projects, Standard queries, and OFS locations. Each input field has a browse button (a folder icon) to its right. At the bottom, there is a button labeled "OFS Manager...".

- **Program files**  
Defines the starting directory when opening or saving Program files. If you leave this directory blank, you will always start in the directory you opened or saved a Program file the last time.
- **Test scripts**  
Defines the starting directory when opening or saving Test scripts. If you leave this directory blank, you will always start in the directory you opened or saved a Test script the last time.
- **SQL Scripts**  
Defines the starting directory when opening or saving SQL scripts. If you leave this directory blank, you will always start in the directory you opened or saved a SQL script the last time.
- **Report files**  
Defines the starting directory when opening or saving Report files. If you leave this directory blank, you will always start in the directory you opened or saved a Report file the last time.

- **Command files**  
When open a command file from the menu or by typing *@file*, this directory will be used by default. If you leave this directory blank, the working directory will be used.
- **Macros**  
This directory will be used to store your macro library. If you leave this directory blank, the *Macro* subdirectory in the PL/SQL Directory will be used.
- **Templates**  
This directory will be used to store program file templates. If you leave this directory blank, the *Template* subdirectory in the PL/SQL Directory will be used.
- **Plug-Ins**  
This directory will be used to search for Plug-Ins. If you leave this directory blank, the *PlugIn* subdirectory in the PL/SQL Directory will be used.
- **Projects**  
Defines the starting directory when opening or saving Projects. If you leave this directory blank, you will always start in the directory you opened or saved a Project the last time.
- **Standard queries**  
Defines the directory where standard queries are stored. If you leave this directory blank, standard queries will be stored in the application data directory of your profile.
- **OFS Locations**  
Defines the OFS Location Directory File (ofs.ldf). The *OFS Manager* button allows you to launch the OFS Manager to view or edit this file. See also chapter 26.1.

## 16.27 Files – Extensions

The screenshot shows a dialog box titled "Files - Extensions" with a "Desktop" button in the top left. It contains four sections, each with a blue header and a checkbox:

- SQL Scripts**: Default , Others
- Report Files**: Default , Others
- Test Scripts**: Default , Others
- Command Files**: Default , Others

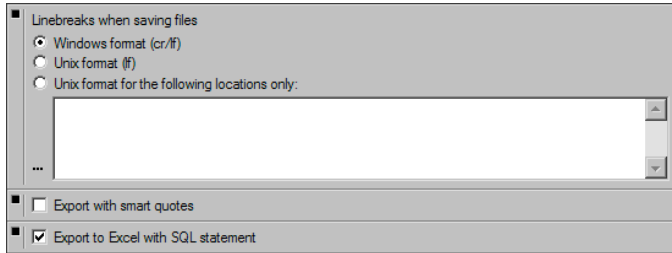
At the bottom, there is a button labeled "Register Filetypes...".

On this page you can define the default extensions of all file types. When saving or loading such a file, the default extension is assumed if you do not specify an extension in the file dialog. For each file type you can additionally specify 'other' extensions, which will be displayed in the file dialogs as well. Separate multiple extensions with a comma.

By pressing the *Register Filetypes* button you can associate these default extensions with PL/SQL Developer. Double-clicking a file with such an extension in the Windows Explorer will automatically

launch PL/SQL Developer. A dialog will appear that allows you to select which extensions you want to associate with PL/SQL Developer.

## 16.28 Files – Format

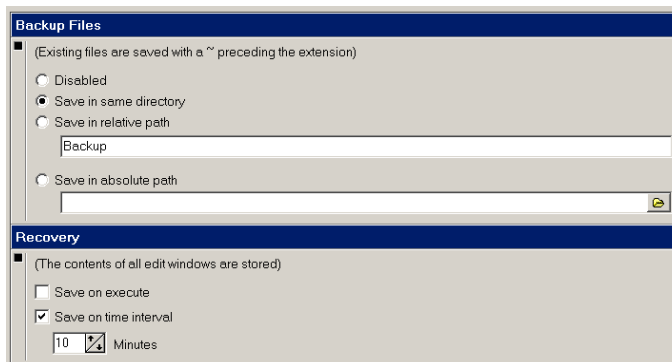


The *Linebreaks* option controls how lines are terminated in text files. The can either be terminated in Windows style by CR/LF character pairs, or in Unix/Linux style by a single LF character. You can additionally specify that files are in Unix format on specific locations, and in Windows format otherwise. You can specify multiple directories on separate lines, and select a directory by pressing the *Add location* button (...). Note that not only files saved in this directory are formatted in Unix style, but also all files saved in subdirectories below this directory.

The *Export with smart quotes* controls how values are exported in CSV format. When disabled, all values are enclosed in quotes. When enabled, only those values that require them are enclosed in quotes. This are values that contain comma's or quotes.

The *Export to Excel with SQL statement* option controls whether or not the SQL statement will be included on a separate Excel page when a query result is exported in Excel format.

## 16.29 Files – Backup



This preference page contains options for backup and recovery of your source files.

### Create backup files

Before PL/SQL Developer saves a file, the original file will be saved to a backup file before overwriting it. This backup file will have the same name, and the extension will be prefixed with a ~ character (e.g. employee.pck will be saved to employee.~pck). This way you will always have one generation of previous files.

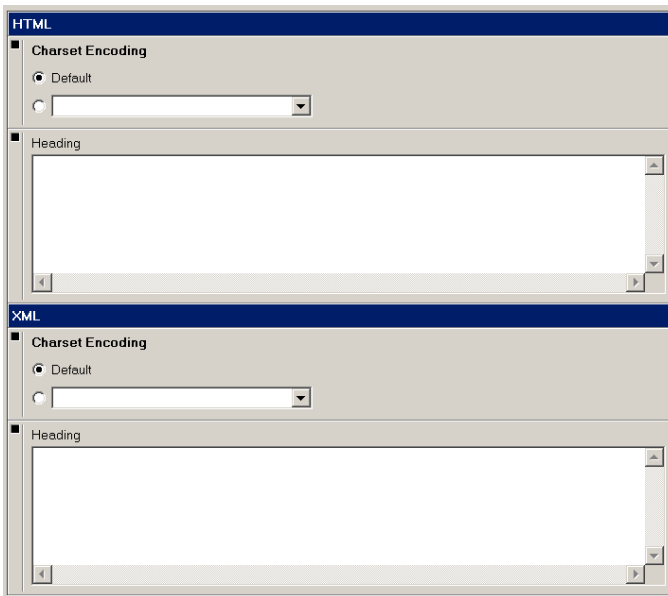
All file dialogs that allow you to open a file, will have a “Backup files (\*.~\*)” filter that allows you to easily open a backup file.

## Recovery

PL/SQL Developer can store all open files to a temporary location on certain events. In case of a severe problem (e.g. power failure, software crash, and so on), PL/SQL Developer allows you to recover these files when it is subsequently started. You can specify the following events:

- **Save on Execute.**  
All open files will be stored in a temporary directory before you execute a Program Window, Test Window, Command Window, SQL Window or Report Window.
- **Save on time interval.**  
All open files will be stored in a temporary directory at the specified time interval.

## 16.30 Files – HTML/XML



These preferences allow you to control the HTML and XML output that is generated by the Report Window and the Export functions:

- **Charset encoding.**  
Allows you to define the character set encoding of the HTML or XML document. This should correspond to the character set of your Oracle environment, so that the data is interpreted correctly.
- **Heading.**  
This text will be inserted into the <HEAD> section of an HTML file, or into the standard XML header of an XML file.

## 16.31 Other – Printing

<b>General</b>	
<input type="checkbox"/>	Show print dialog
<input type="checkbox"/>	Print line numbers
<input type="checkbox"/>	Wrap lines while printing
<input type="checkbox"/>	Print black/white
<input type="checkbox"/>	Print background
<input checked="" type="checkbox"/>	Print SQL text in SQL window
<b>Grid</b>	
<input checked="" type="checkbox"/>	Print lines
<input checked="" type="checkbox"/>	Inverted heading
<b>Header</b>	
<input type="checkbox"/>	Caption
<input type="checkbox"/>	Date
<b>Footer</b>	
<input type="checkbox"/>	Page numbers

### General

- Show print dialog.  
When pressing the *Print* button, the print dialog will be displayed first.
- Print line numbers.  
Will display a line number before each line when printing SQL or PL/SQL source code.
- Wrap lines while printing.  
Lines that are wider than the page width will be continued on the next line.
- Print black/white.  
All printed output will be black and white. Disabling this option will create output with colors or gray levels.
- Print background.  
When enabled, the background color will be printed.
- Print SQL text in SQL Window.  
Determines if the SQL text is printed before the query results when printing the contents of a SQL Window.

### Grid

- Print lines.  
Determines if the lines of a grid are printed.
- Inverted heading.  
Determines if the heading of a grid is printed with inverted color.



## Header

- **Caption.**  
When enabled, each page will have the file type and file name printed on the left side of the header.
- **Date.**  
When enabled, each page will have the current date and time printed on the right side of the header.

## Footer

- **Page numbers.**  
When enabled, the center of the footer will contain the page number.

## 16.32 Other – Updates & News

**Updates**

☐ Check Online Updates interval:

☐ Application Start

☒ Daily

☐ Weekly

☐ Monthly

☐ Manually

☐ Never (disabled)

Check Now... Latest Update: 22-09-2006

**News**

☐ Check Online News interval:

☐ Application Start

☒ Daily

☐ Weekly

☐ Monthly

☐ Manually

☐ Never (disabled)

Check Now... Latest News: 19-03-2007

**Proxy Settings**

☐ Enabled

Address Port

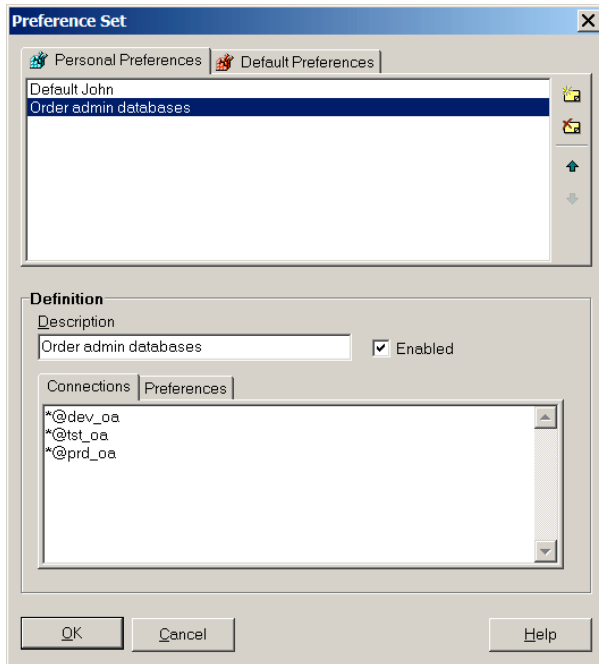
Username Password

☐ Ask for password

Allows you to control if and when PL/SQL Developer checks the Allround Automations web site for updates and news about PL/SQL Developer. If you disable this option, you can explicitly check for updates and news by pressing the corresponding option or though the *Check Online Updates* and *Check Online News* items in the *Help* menu.

## 16.33 Preference sets

As mentioned in chapter 16, you can define preference sets at multiple levels. To define a preference, press the *Configure preference sets* button next to the preference set selection list. The following dialog will appear:



The *Personal Preferences* are defined for the current Windows user (in this case *John*). The *Default Preferences* can be defined by the system administrator, and provide default preferences for all Windows users, either at a global level or at a connection level.

### Personal preferences

By default all users will have a *Default <user>* preference set. This is the preference set that you will access by default. To create an additional preference set for one or more Oracle connections, press the *New* button to the right of the preference list. Now you can enter a description and add one or more connections. For each connection you can use wildcards to match multiple users or databases. In the example above, a *Order admin databases* preference set has been defined by *John* for the *dev\_oa*, *tst\_oa*, and *prd\_oa* databases. Whenever John connects to one of these 3 databases, this preference set will be used. If a preference is not defined within this set, the corresponding preference will be used from the default user preference set.

### Default preferences

A system administrator can define global default preferences for all users and all connections, and can additionally define a default preference set for one or more connections. To do so, switch to the *Default Preferences* tab page, and press the *New* next to the preference list. If you do not enter any *Connections*,

the new preference set will define the default preferences for all users. If you do enter one or more connections, the preference set will define the default preferences for these connections.

### Preference set precedence

If a preference is defined at multiple levels, the most specific level will take precedence:

1. Personal connection preference set (highest precedence)
2. Personal windows user preference set
3. Default connection preference set
4. Default global preference set (lowest precedence)

### Preference set files and directories

A preference set is stored in a single file in the *Preferences* subdirectory in the PL/SQL Developer installation directory. Within this preference directory, the personal preferences are stored in a subdirectory with the name of the Windows user. For example:

```
C:\Program Files\PLSQL Developer\Preferences\John
```

The default preferences are stored in the *Default* subdirectory. For example:

```
C:\Program Files\PLSQL Developer\Preferences\Default
```

If you install PL/SQL Developer on a file server, you must provide read/write access to the *Preferences* directory, and read access to the *Preferences\Default* subdirectory. Alternatively, you can provide the *PREFPATH* command-line parameter to specify the directory for the user preferences:

```
plsqldev.exe prefpath=u:\userdata
```

If Windows user *John* uses PL/SQL Developer, his personal preference sets will be stored in *u:\userdata\john*. Default preferences are always stored in the *Preferences\Default* subdirectory within the PL/SQL Developer installation directory.

To ensure that all users use a similar preference path, you can set this parameter in the *params.ini* file (see chapter 28.4).

## 17. Tools

Several tools are available in the *Tools* menu. These tools are described in this chapter.

### 17.1 Browser

The *Browser* tool is described in detail in chapter 15. In the *Tools* menu, you can show or hide the browser by selecting the *Browser* item. You can also define *Browser Filters* from this menu, which are also described in chapter 15.

### 17.2 Find Database Objects

This tool allows you to find database objects based on specific criteria such as the object name, type, owner, or status, as well as additional full text search criteria for the DDL source of the objects:

Name	Type	Compiled
CURRENT_EMPLOYEES	VIEW	28-12-2005 15:55:42
DEPARTMENT	PACKAGE	26-11-2005 14:55:45
DEPARTMENT	PACKAGE BODY	01-11-2005 14:31:47
DEPT	TABLE	21-12-2005 16:32:58

You can enter search criteria in the 3 top sections of this form.

#### Full Text Search

The options in this dialog are more or less equivalent to the Find dialog in the text editor, and allows you to search case insensitive, search for whole words only, and search with regular expressions. If you omit the full text search then all objects that match the other criteria will be returned.

#### Object Criteria

Here you can enter the following criteria:

- **Browser Filter**– Limits the objects according to the criteria of the selected filter.

- **Owner** – The owner of the object. You can use a SQL wildcard characters (e.g. sys%). The default value is `<CURRENT USER>`, which will limit the search to objects for the currently connected user.
- **Name** – The name of the object. You can use a SQL wildcard characters (e.g. %common).
- **Status** – Limits the search to valid or invalid objects.
- **Created after/before** – Limits the search to objects created after and before the specified date. Use the checkbox to the left of the date field to enable or disable these criteria.
- **Modified after/before** – Limits the search to objects modified after and before the specified date. Use the checkbox to the left of the date field to enable or disable these criteria.

## Object Types

Here you can select the range of object types that you want to include in your search.

## Search results

After pressing the *Search* button the search operation is performed in the background and the search results list will be filled.

If you right-click on the search results you will see the popup menu of an object. The first item in this popup menu is *Locate text*, which opens the source in a Program Window, and will immediately take you to the first occurrence of the search text. You can also invoke the *Locate text* function by double-clicking on the item in the search results.

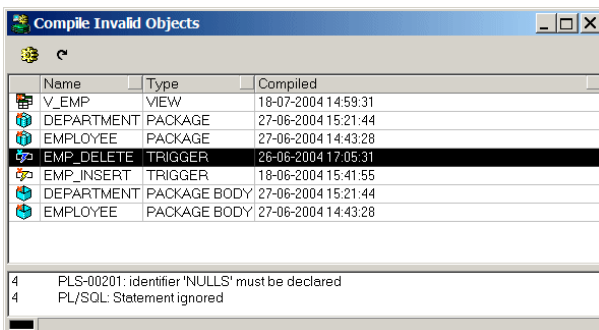
To save the search results you can right-click on them and select an appropriate format from the *Export* submenu item in the popup menu.

## Saving frequently used search criteria

To save frequently used search criteria, you can press the *Save* button and enter a directory and filename. You can recall these search criteria later you can press the *Load* button.

## 17.3 Compile Invalid Objects

With this tool you can easily recompile all invalid objects in the database. When you select it, a dialog is displayed that shows all invalid objects:



Note that the currently active Browser Filter controls the invalid objects that are included. The bottom section of the dialog will display any compilation errors that have previously occurred for the currently selected object.

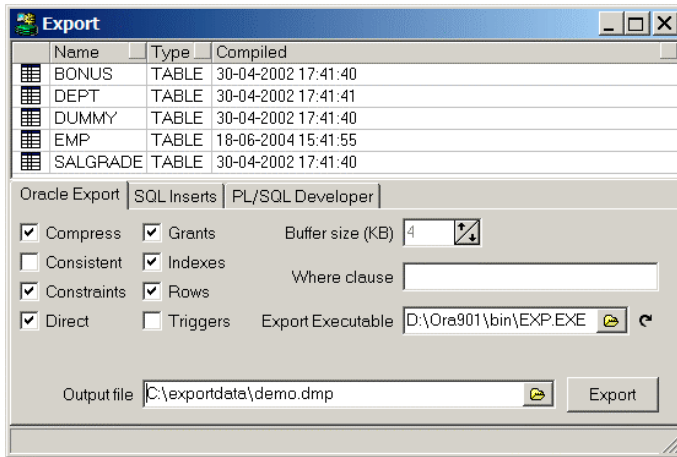
To recompile the objects, press the *Compile invalid objects* button on the toolbar or *Execute* on the main toolbar. The dependencies of the objects will be analyzed to determine in which order compilation should occur. The icons at the left of the object list will indicate which objects were successfully compiled. The compilation error section will be updated after compilation is completed.

To refresh the list of invalid objects, press the *Refresh object list* button on the toolbar. This will update the list with all currently invalid objects.

In the object list you can right-click on an object to access its popup menu.

## 17.4 Export Tables

The Export Tables tool allows you to export one or more table definitions and their data into a file, so that you can import the tables later. After starting the Export Tables tool, you can select the tables you wish to export, choose an export method (Oracle Export, SQL Inserts, or PL/SQL Developer), and set various options that apply to the export method:



The export function analyzes the foreign key dependencies that may exist between the tables, and places the tables in the correct order in the output file.

### Oracle Export method

The screenshot in the previous chapter shows the options of the Oracle Export method. This method lets you specify the options that you can supply on the command line of Oracle's export utility (see the "Oracle Server Utilities" guide), and will subsequently launch it to perform the export. The *Export Executable* field allows you to select a specific version of Oracle's export utility. By default the most recent version from the current Oracle Home will be used.

After the export is finished, a new *Log* tab page will be visible. This page contains the logging that was created by the export utility.

Advantages of this method are the speed and the portability of the output file. After all, you can import it anywhere with Oracle's import utility.

## SQL Inserts method

Oracle Export | **SQL Inserts** | PL/SQL Developer

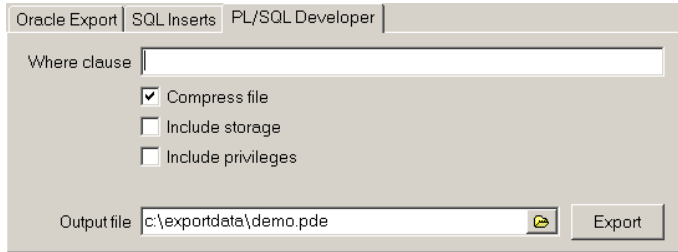
☐ Drop tables      ☒ Disable foreign key constraints  
☐ Create tables      ☐ Include storage      ☐ Include privileges  
☐ Truncate tables  
☒ Delete records      Commit every 100 records (0 = never)  
☒ Disable triggers      Where clause   
 Output file C:\exportdata\demo.sql

This export method creates a standard SQL script with insert statements and (optionally) DDL statements to recreate the table definition. The tab page contains the following options:

- **Drop tables** – The generated SQL script contains statements to drop the tables before they are recreated and loaded. The *Create tables*, *Truncate tables*, and *Delete records* options are disabled in this situation, and have an implicit value.
- **Create tables** – The generated SQL script contains statements to create the tables before they are loaded. This includes constraints, indexes and grants.
- **Truncate tables** – The generated SQL script uses truncate table statements to empty the tables before they are loaded. This option is faster than the *Delete tables*, but cannot be used if foreign key constraints exist.
- **Delete records** – The generated SQL script uses delete statements to empty the tables before they are loaded.
- **Disable triggers** – The generated SQL script disables all triggers of the tables before they are loaded, and enables them afterwards. This can improve performance, and can sometimes be necessary if the triggers contain checks that are not applicable to the import process.
- **Disable foreign key constraints** – The generated SQL script disables all foreign keys of the tables before they are loaded, and enables them afterwards. This can be necessary for self-referencing foreign keys. Foreign keys between different tables will not lead to conflicts, because the tables are exported in a correct order. Disabling foreign keys can also improve performance.
- **Include storage** – The generated SQL includes the original storage information such as tablespace names and initial sizes for table creation statements. These may differ across databases, so this may not always be appropriate.
- **Include privileges** – The generated SQL includes the grants of the object privileges to other users and roles. When the objects are recreated in a different database, these users and roles need to exist for obvious reasons.
- **Commit every** – Controls how many records will be inserted in the generated SQL script before a commit is executed. If this value is zero, all inserts will be committed at once at the end of the SQL script. For large export files and small rollback segments it will be necessary to supply an appropriate value here.
- **Where clause** – Only records that match the where clause will be exported. The where clause must be applicable to all selected tables!

Advantages of this method are the portability of the SQL script, and the possibility to edit the results with a text editor.


## PL/SQL Developer method



Oracle Export | SQL Inserts | **PL/SQL Developer**

Where clause

☒ Compress file  
☐ Include storage  
☐ Include privileges

Output file   **Export**

This export method creates a custom, compressed PL/SQL Developer export file with the table definitions and data. The tab page contains the following options:

- **Where clause** – Only records that match the where clause will be exported. The where clause must be applicable to all selected tables!
- **Compress file** – Disabling this option will disable the compression. This can lead to improved performance if the data cannot be very well compressed (e.g. for long raw or BLOB data that is already compressed in the database).
- **Include storage** – The generated SQL includes the original storage information such as tablespace names and initial sizes for table creation statements. These may differ across databases, so this may not always be appropriate.
- **Include privileges** – The generated SQL includes the grants of the object privileges to other users and roles. When the objects are recreated in a different database, these users and roles need to exist for obvious reasons.

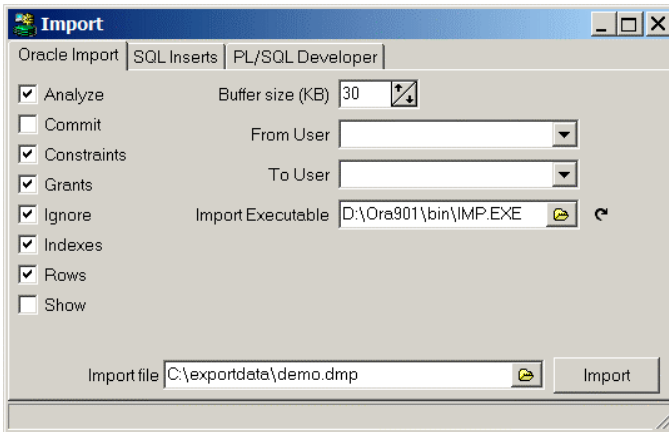
Advantages of this method are the greatly reduced file sizes of the export files, the fact that all options are deferred to the import phase, and that you can easily select tables during import.



## 17.5 Import Tables

The Import Tables tool allows you to import table definitions and data from a file that was previously exported with the Export Tables tool described in the previous chapter. Just like with the Export Table tool, there are 3 methods to import tables, each with its own file format.

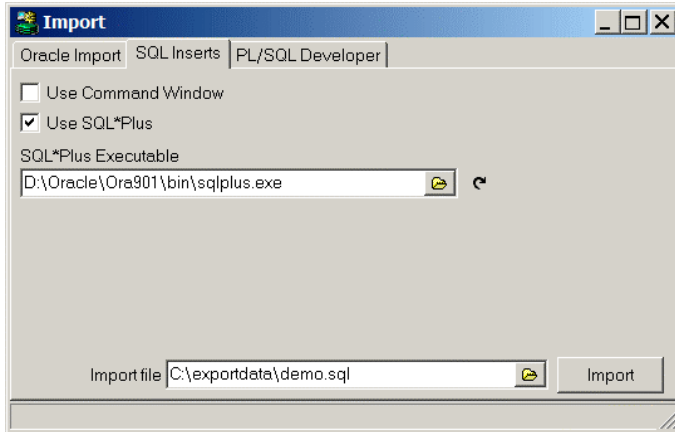
### Oracle Import method



This method uses a dump file created by the Oracle export utility, and launches Oracle's import utility to import the data. You can specify various options, each corresponding to a command line option of the import utility (see the "*Oracle Server Utilities*" guide). The *Import Executable* field allows you to select a specific version of Oracle's import utility. By default the most recent version from the current Oracle Home will be used.

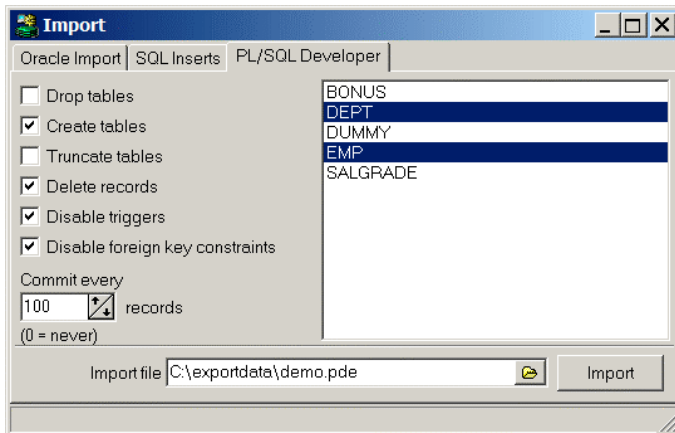
After the import is finished, a new *Log* tab page will be visible. This page contains the logging that was created by the import utility.

## SQL Inserts method



This method executes the SQL script that was generated by the Export Tables tool. You can choose to launch SQL\*Plus to run the script, or use an internal Command Window. For the SQL\*Plus alternative you can select a specific SQL\*Plus executable. By default the version from the current Oracle Home will be used.

## PL/SQL Developer method



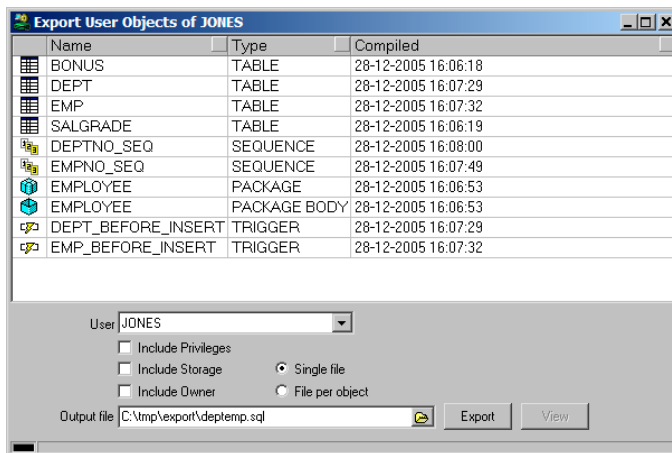
After selecting an import file that was previously exported using the PL/SQL Developer method, you can select to import one or more files from the list. The following options can be specified:

- **Drop tables** – Drop the tables before they are recreated and loaded. The *Create tables*, *Truncate tables*, and *Delete records* options are disabled in this situation, and have an implicit value.
- **Create tables** – Create the table before it is loaded. This includes constraints, indexes and grants.
- **Truncate tables** – Truncate the tables before they are loaded. This option is faster than the *Delete tables*, but cannot be used if foreign key constraints exist.

- Delete tables – Delete all records in the tables before they are loaded.
- Disable triggers – Disables all triggers of the tables before they are loaded, and enable them afterwards. This can improve performance, and can sometimes be necessary if the triggers contain checks that are not applicable to the import process.
- Disable foreign key constraints – Disables all foreign keys of the tables before they are loaded, and enable them afterwards. This can be necessary for self-referencing foreign keys. Foreign keys between different tables will not lead to conflicts, because the tables are exported in a correct order. Disabling foreign keys can also improve performance.
- Commit every – Controls how many records will be inserted before a commit is executed. If this value is zero, all inserts will be committed at once at the end of the SQL script. For large export files and small rollback segments it will be necessary to supply an appropriate value here.

## 17.6 Export User Objects

To export the DDL (Data Definition Language) statements of all objects of a user, you can use the *Export User Objects* tool. This way you can easily recreate the objects for another user, or in a different database. After selecting *Export User Objects* from the *Tools* menu, all objects of the current user will be presented in a grid:



After selecting an output file, you can press the *Export* button to export the objects. If no specific objects are selected in the grid, all objects will be exported. You can select objects by clicking on them, and using the *Control* and *Shift* keys to select multiple objects and ranges of objects.

The output file is a SQL script that is compatible with Oracle's SQL\*Plus, and with PL/SQL Developer's Command Window. After the export operation you can press the *View* button to open the generated file in a Command Window.

By selecting a different *User*, you can export objects owned by another user if the privileges of your current user allows it.

The *Include Privileges* option can be enabled if you want to include the grants of the object privileges to other users and roles. When the objects are recreated in a different database, these users and roles need to exist for obvious reasons.

Similarly you can enable the *Include Storage* option to include the storage information such as tablespace names and initial sizes. These may differ across databases, so this may not always be appropriate.

The *Include Owner* option controls whether objects are prefixed with the owner (e.g. SCOTT.EMP instead of EMP) in the resulting output file(s).

The *Single file* and *File per object* options control whether a single SQL script is created that contains the DDL for all objects, or if each object should be exported to a specific file (with the object name as filename, and .sql as extension). In that case the output file will contain a series of calls to these object specific files. All files will be written to the directory of the output file.

## 17.7 Text Importer

With the text importer you can import ASCII files into the database. Most line oriented formats like comma and tab separated fields are supported. The importer will try to determine the file format automatically, so most of the time you don't need to define anything, just select the file, select a table, and that's it.

When you open the text importer and load a text file, you will get something like the following:

**Text Importer - emp.csv**

Data from Textfile | Data to Oracle

**File Data**

```
" " "EMPNO","ENAME","JOB","MGR","HIREDATE","SAL","COMM","DEPTNO"
"1","7369","SMITH","CLERK","7902","17-12-1980","800,00","", "20"
"2","7499","ALLEN","SALESMAN","7698","20-02-1981","1.600,00","300,00","30"
"3","7521","WARD","SALESMAN","7698","22-02-1981","1.250,00","500,00","30"
"4","7566","JONES","MANAGER","7839","02-04-1981","2.975,00","", "20"
"5","7654","MARTIN","SALESMAN","7698","28-09-1981","1.250,00","1.400,00","30"
"6","7698","BLAKE","MANAGER","7839","01-05-1981","2.850,00","", "30"
```

**Configuration**

**General**

Fieldcount: 9

☒ Name in header

Quote character: "

Field1 (+0 .. ")

Field2 (+0 .. ") EMPNO

Field3 (+0 .. ") ENAME

Field4 (+0 .. ") JOB

Field5 (+0 .. ") MGR

Field6 (+0 .. ") HIREDATE

Field7 (+0 .. ") SAL

Field8 (+0 .. ") COMM

Field9 (+0 .. ") DEPTNO

**Field Start**

☒ Relative position

☐ Absolute position

☐ Character

**Field End**

☐ Length

☒ Character

**Result Preview**

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	17-12-1980	800,00		20
2	7499	ALLEN	SALESMAN	7698	20-02-1981	1.600,00	300,00	30
3	7521	WARD	SALESMAN	7698	22-02-1981	1.250,00	500,00	30
4	7566	JONES	MANAGER	7839	02-04-1981	2.975,00		20
5	7654	MARTIN	SALESMAN	7698	28-09-1981	1.250,00	1.400,00	30
6	7698	BLAKE	MANAGER	7839	01-05-1981	2.850,00		30

Import Import to Script Close emp.csv loaded, 1 KB

The importer loads the first 100 lines from the file, this raw data is shown in the top "File Data" section. The middle section allows you to make the definition of the text file. The bottom "Result Preview" has a grid with the data as it would be imported.

The toolbar has buttons to select a text file, or paste text from the clipboard. Files can be of virtually unlimited size, they are not read entirely into memory. The “New” button will clear everything in the text importer. The open and save definition buttons allow you to re-use the definition.

The configuration is auto determined as soon as you load a file, but if this is not correct, you can create or modify this yourself. Set the Fieldcount to the correct number, and select a field from the field list to define a field definition. In the file data section, the specified field data is highlighted, allowing you to check if the field definition is correct. You have the following options to configure the text import definition:

- General – Fieldcount  
The number of fields per record.
- General – Name in Header  
Indicates if the first record holds the fieldnames.
- General – Quote Character  
Indicates the string characters, usually double or single quotes.
- Field Start – Relative Position  
The field starts at a relative position to the end of the previous field. 0 indicates that the field starts where the previous field ended.
- Field Start – Absolute Position  
This indicates that the field starts at a fixed position.
- Field Start – Character  
This indicates that the field starts with a specific character. This is relative to the end of the previous field.
- Field End – Length  
Indicates that the field has a fixed length.
- Field End – Character  
Indicates that the field end with a specific character.

For standard comma separated files (csv) all fields will have a relative start position of 0, end the field end character should be a comma. The line end (cr/lf or lf or cr) will also indicate field end, so for the last field on a line, the end character doesn't matter.

When you have a correct file definition, you need to select an Oracle table and indicate which text fields should be imported in which Oracle fields. You can do this on the second tab page:

**Text Importer - emp.csv**

Data from Textfile | Data to Oracle

**General**

Owner: SCOTT Table: EMP

Commit every...: 100

☒ Overwrite duplicates  
☐ Ignore duplicates

**Fields**

Field1 ->  
Field2 EMPNO -> EMPNO (NUMBER)  
Field3 ENAME -> ENAME (VARCHAR2)  
Field4 JOB -> JOB (VARCHAR2)  
Field5 MGR -> MGR (NUMBER)  
**Field6 HIREDATE -> HIREDATE (DATE)**  
Field7 SAL -> SAL (NUMBER)  
Field8 COMM -> COMM (NUMBER)  
Field9 DEPTNO -> DEPTNO (NUMBER)

Field: HIREDATE (DATE)  
Fieldtype: Date  
SQL function: to\_date(#, 'DD-MM-YYYY')

additional Oracle processing, for example: substr(#, 1, 20)

**Result Preview**

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH&WESSON	CLERK	7902	17-12-1980	800,00		20
2	7499	ALLEN	SALESMAN	7698	20-02-1981	1.600,00	300,00	30
3	7521	WARD	SALESMAN	7698	22-02-1981	1.250,00	500,00	30
4	7566	JONES	MANAGER	7839	02-04-1981	2.975,00		20
5	7654	MARTIN	SALESMAN	7698	28-09-1981	1.250,00	1.400,00	30
6	7698	BLAKE	MANAGER	7839	01-05-1981	2.850,00		30

Import Import to Script Close emp.csv loaded, 1 KB

The top section has some general import parameters. The “Fields” section allows you to associate text file fields and Oracle fields. The bottom section is the result preview for your information.

You can set the following general preferences:

- Table  
The Oracle table (or view) you want to import the data to. After selecting a table, the importer will try to determine the field and field types automatically. This will only work if the text file has a header.
- Commit every...  
Indicates after how many records you want to do a commit. If you set this to 0, all data will be committed at the end of the entire import.
- Overwrite Duplicates  
A duplicate record will be updated in the database
- Ignore Duplicates  
Duplicate records will be ignored.

In the field definition you see a list with fields from the text file. For every field you can set the following:

- Field  
The Oracle field you want to associate with the text field. You can leave this empty if you don't want to import this field.

- **Fieldtype**  
The basic fieldtype: String, Number or Date.
- **SQL function**  
This option allows you to define additional SQL processing. For date fields a `to_date` function is added automatically. This is a very powerful option, allowing you to convert the imported data. Basically, you can enter anything that can be processed by Oracle. You can add a `#` to indicate the data. The “Create SQL” button will fill this field with a `to_date` function for date fields.

When the definition is complete, you can decide to save it. The toolbar has buttons to save and load definition files. The text importer will remember which definition file was used for which text file, and the definition file will be loaded automatically the next time you open the same text file.

You have two import buttons, “Import” and “Import to Script”. The first option will start importing the data into the selected table. The second option will create an SQL script with insert statements.

## 17.8 ODBC Importer

The ODBC Importer tool allows you to import data from any ODBC data source into an Oracle table:

**ODBC Importer**

Data from ODBC | Data to Oracle

**Connection**

User / System DSN: Excel-bestanden

User Name:

Password:

Connect Disconnect

**Table / Query**

Import Table Import Query Result

C:\temp\EMP "SQL Results\$"

View Data

**Result Preview**

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369.0	SMITH	CLERK	7902.0	1980-12-17 00:00:00	800.0		20.0
7499.0	ALLEN	SALESMAN	7698.0	1981-02-20 00:00:00	1600.0		30.0
7521.0	WARD	SALESMAN	7698.0	1981-02-22 00:00:00	1250.0		30.0
7566.0	JONES	MANAGER	7839.0	1981-04-02 00:00:00	2975.0		20.0

Import Import to Script Close Help

On the first page of the ODBC Importer you need to select the ODBC data source, and optionally provide a user name and password for this data source. You can subsequently connect, after which the table list will be populated with all tables from the data source. Selecting a table will populate the *Result Preview* pane. Instead of selecting a table, you can also provide a query by selecting the *Import Query Result* option and entering a query text. This can be useful if you want to limit the rows or columns, or if the results come from more than one table.

After entering the source specifications, you can switch to the *Data to Oracle* tab page to select the destination table and to specify the column mapping:

**ODBC Importer**

Data from ODBC | Data to Oracle

**General**

Owner: SCOTT Table: EMPLOYEES ☐ Clear Table

Commit every...: 100 ☐ Overwrite duplicates ☒ Ignore duplicates

**Fields**

EMPNO -> EMPNO  
ENAME -> ENAME  
JOB -> JOB  
MGR -> MGR  
HIREDATE -> HIREDATE  
SAL -> SAL  
COMM -> COMM  
DEPTNO -> DEPTNO

Field: JOB  
Fieldtype: (String)  
Create SQL

SQL function: additional Oracle processing, for example: substr(#, 1, 20)

**Result Preview**

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369.0	SMITH	CLERK	7902.0	1980-12-17 00:00:00	800.0		20.0
7499.0	ALLEN	SALESMAN	7698.0	1981-02-20 00:00:00	1600.0		30.0
7521.0	WARD	SALESMAN	7698.0	1981-02-22 00:00:00	1250.0		30.0
7566.0	JONES	MANAGER	7839.0	1981-04-02 00:00:00	2975.0		20.0

Import Import to Script Close Help

In the *General* section you can select an owner and a table into which the records should be imported. You can also select if you want to clear the table before the import, select a commit interval (0 = commit at end) and to ignore duplicate rows or to overwrite them.

In the *Fields* section you can map fields from the ODBC data source (left) to columns in the Oracle table (right). For each column you can additionally supply a SQL function. For example, if you want to convert a string field to uppercase, enter *upper(#)*. The hash sign will be replaced by the field data for each record, and the resulting expression will be supplied in the insert statement.

After completing the field mapping, you can press the *Import* button to import the data into the Oracle table. If you hold down the *Ctrl* key while pressing this button, an example insert statement will be displayed so that you can verify the field mapping and SQL functions you specified.

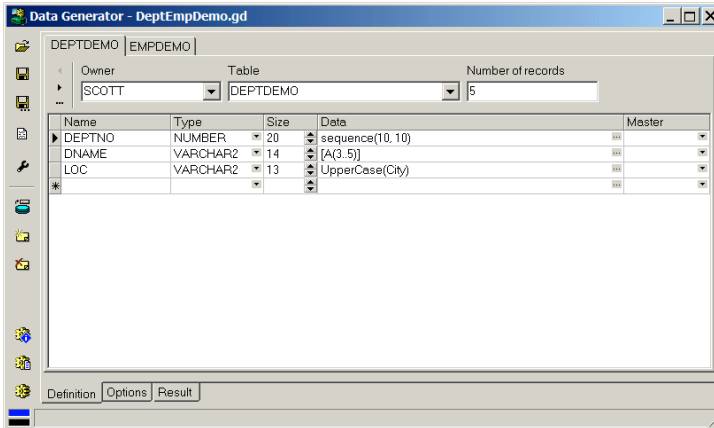
To reuse this import definition later, you can press the *Save Definition* button on the toolbar. This definition can be opened later by pressing the *Open Definition* button.



## 17.9 Data Generator

The data generator allows you to create demo and test data. This can be helpful to test applications and see how they perform with large amounts of data. You find the Data Generator under the Tools menu.

Basically, the definition consists of one or more tables, the number of records you want to generate, and the field data definitions. The buttons on the top left allow you to open and save a definition.



Above you see the supplied deptemp demo that creates data for deptdemo and empdemo tables, similar to the well-known dept and emp tables. You will find the following items on this page:

- Table – The name of the table.
- Number of records – The number of records you want to generate. This can be a number, but also a range like 10..100.
- Name – The field name.
- Type – The data type of the field.
- Size – The field size (when appropriate). For number fields this will be scale, precision.
- Data – The definition of the data for a field (see below).
- Master – If this table is the detail of another table, you can set the master. For every generated master record, the specified number of detail records are generated.

You can use the “Add existing table” button to add an existing table. You can also drag & drop one from the Object Browser.

### Data definition

The data definition determines the generated data. If you want to create simple characters, you can enter character definitions between square brackets: [data]

Data can be a mix of the following pre-defined sets:

- a: a..z (*lowercase characters*)
- A: A..Z (*uppercase characters*)
- @: A..Z and a..z (*all characters*)
- #: A..Z and a..z and 0..9 (*all characters and numbers*)
- \*: #33..#126 (*all ASCII characters*)
- 0: 0..9 (*all numbers*)
- 1: 1..9 (*all numbers except 0*)
- 9: 0..9 (*all numbers*)

For example:

[Aaa00] to generate strings like: Gxe21, Liy05, etc.

You can also add literal text between single quotes.

For example:

[AA '-' 1000] to generate strings like: CX-4903, SY-1044, etc.

Space characters in the definition are ignored, unless they are within quotes.

If you want a character repeated a number of times, you can add the number after the character between brackets (n). You can also add a random number (min..max).

For example:

[Aa(5..15) '' Aa(8..20)] for results like: "Masfae Qwwecdsadif"

Literal text in the definition does not have to be enclosed in brackets. In other words, ['hello'] and 'hello' are equivalent. Text entered without quotes is interpreted as a function.

The are several special functions available:

- **Signal(Min, Max, Delta, Noise)**  
Returns technical measurement data (like temperatures). Min and Max determine the range, Delta the maximum change, and you can also add some noise. Example: Signal(-10, 20, 0.1, 0.1)
- **Random([Min], Max)**  
Returns random numbers between Min and Max. If only Max is specified, Min is set to 0. For date fields you can enter dates for Min and Max.
- **Sequence(Start, [Inc], [WithinParent])**  
Returns sequence numbers, starting with Start, and incrementing with Inc (which is 1 by default). For a detail table you can additionally specify the WithinParent keyword to indicate that the sequence should be reset for each parent record.
- **List('item'(weight), 'item'(weight), ...)**  
Returns randomly one of the specified items. A weight can be added between brackets to allow specific items to occur more often than others.  
For example: List('CLERK'(50), 'SALESMAN'(30), 'MANAGER'(10))
- **List(select statement)**  
Like the previous List function, but the items are returned by the SQL select statement.
- **Text([Style], MaxCharacters, [WordsPerLine, LinesPerParagraph])**  
This function returns text. The optional Style parameter can be LoremIpsum (default), English, German or Japanese. It can also be a character set like [aA], in which case words are generated from the specified set.  
MaxCharacters determines the maximum size of the generated text, and the optional WordsPerLine, and LinesPerParagraph determine the size of a line and a paragraph. The specified size can be a number or a range (min..max).
- **File(path, path, ...)**  
Randomly selects a file from the given paths, and inserts the contents. Allows you to enter binary data (like images) into the database. The path can have wildcards like d:\images\\*.bmp.

You can change the text results of the previous functions by using the following functions:

- Uppercase()
- LowerCase()
- InitCaps()

For example: InitCaps( List(select ename from emp) )

There are also several predefined data sets available to generate more or less real data. You can use the following definitions:

- Firstname – General list of first names
- Lastname – General list of last names
- Company – Company names (random list of existing companies)
- Address1 – Address line 1
- Address2 – Address line 2
- ZIP – ZIP code
- City – City
- State – State
- Country – Country
- Email (related to Firstname, Lastname and Country)

There are also some example user-defined data sets available

- Components.Code – General article items: Article code
- Components.Description – Article description (Computer parts)
- Components.Price – Article price
- Elements.Name – Chemical elements (name)
- Elements.Symbol – Chemical elements (symbol)

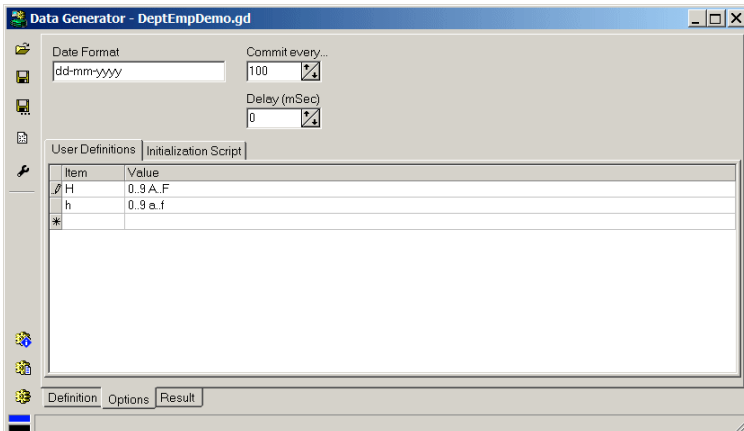
These data sets can be found in the DataGenerator\UserData directory as elements.txt and components.txt. You can add you own sets if you want. Simply add a comma-separated file where the first line holds the description between square brackets. You can use data from your file by specifying filename.description like the two examples.

All the functions and data mentioned above can be added together, for example: Random(10..99) + '+' + [A(4)]

The + is optional, but there should at least be a space as separator.

## Options

The options tab page allows you to set some preferences specific to the data generator definition. You can specify a date format (as used for example by the Random function). You can also specify after how many records you want a commit (set to 0 to commit once when finished). The delay preference is only useful in some real-time testing where you want data inserted at a specific speed.



The “User Definitions” section allows you to define or override character sets used in the [data] definition. The Item is always a single character, and the Value is one or more space separated ranges of characters like: A..Z a..z 0..9 #200..#220. For example: to define a hexadecimal character set, you could specify H as the item, and 0..9 A..F as the value.

The “Initialization Script” section allows you to specify SQL statements that should be executed before the data is inserted into the database. Typically, this would be a statement to create the tables, or to truncate the tables, to select a rollback segment, and so on. Multiple statements need to be separated by semicolons.

General options that will be used by default for all data generators can be defined by pressing the *Preferences* button on the left. See chapter 16.23 for more details. These preferences will be overruled if the corresponding preference in the data generator definition is also set.

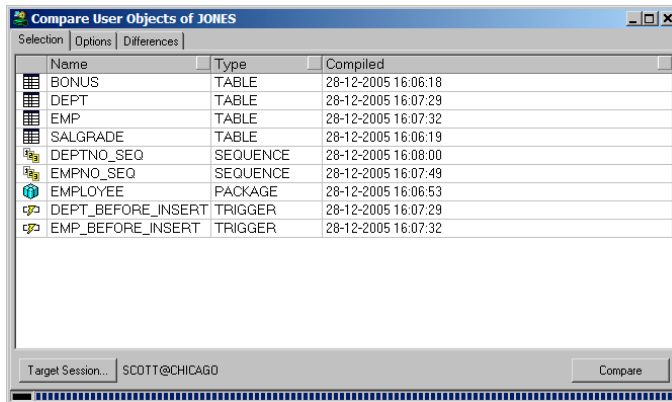
## Generating Data

There are three buttons available on the bottom left to generate the actual data:

- Start a test run – This will generate the data and display the results in a grid on the *Result* tab page. You can export these results in various formats by right-clicking on the grid.
- Create data as SQL – Generates the data as a SQL script. This will only work if you don't add data from files with the file function.
- Create data in DB – Generates the data into the Oracle database.

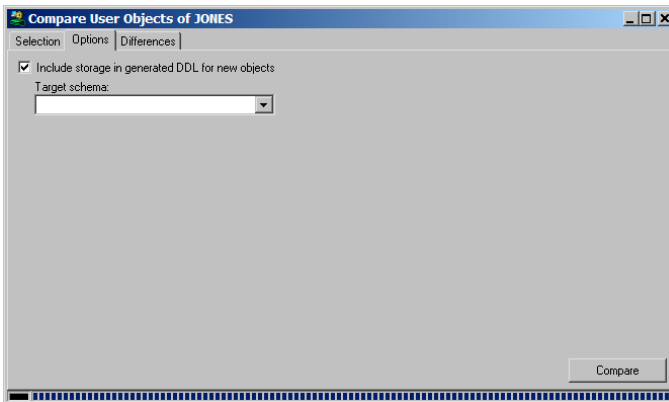
## 17.10 Compare User Objects

After making changes to table definitions, views, program units, and so on, it may be useful to propagate these changes to another database user. This maybe another development environment, or a testing environment. To compare the objects of your development user with another user, you can use the *Compare User Objects* function in the *Tools* menu. This will bring up the following dialog:



On the *Selection* tab page you can select the objects you wish to compare. After making this selection, you can press the *Target Session* button, to select the user and database that you want to compare. This will enable the *Compare* button, which you can press to start the compare operation.

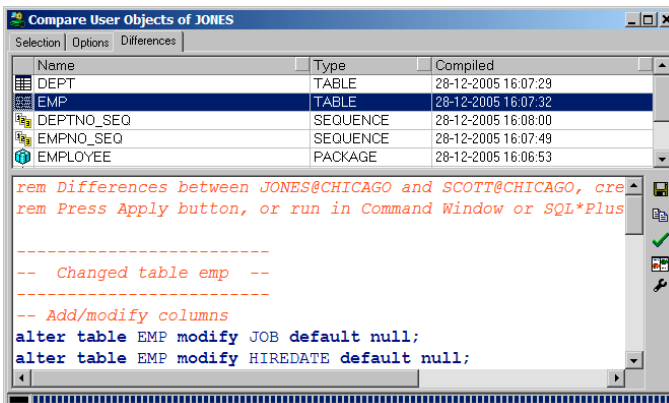
On the *Options* tab page, you can set the following options:



You can select the *Include storage...* option to include the storage information such as tablespace names and initial sizes for new objects. These may differ across databases, so this may not always be appropriate.

The *Target schema* can be set if the objects in the target session are owned by an other user than the target user that is connected.

When the compare operation is finished, the dialog will switch to the *Differences* tab page, which will show a list of all objects that are different:



This list is sorted in order of dependency. Below the list of different objects of the target user, you see the SQL that needs to be executed to make these objects equal to the corresponding objects of the current user. If no object is selected, the SQL of all objects is displayed. If one or more objects are selected, only the SQL for the selected object(s) is displayed. In the example above, the default values for the *JOB* and *HIREDATE* columns of the *EMP* table are cleared in the target schema.

The *Show Differences* button will show a visual line-by-line difference of the old and new source file of an object. This can be useful to view the changes made in Program Units, or can help you determine why a specific DDL statement was generated for other object types. The *Configure External Difference Tool* allows you to configure the difference tool should be used. By default the *ExamDiff* utility will be used, for which a Pro version is available (See the *About* item of *ExamDiff's Info* menu). See also chapter 16.22.

You can now press the *Apply SQL* button to execute this SQL in the target session. You can alternatively save the SQL in a file by pressing the *Save SQL* button, or you can copy it to the clipboard by pressing the *Copy SQL* button.

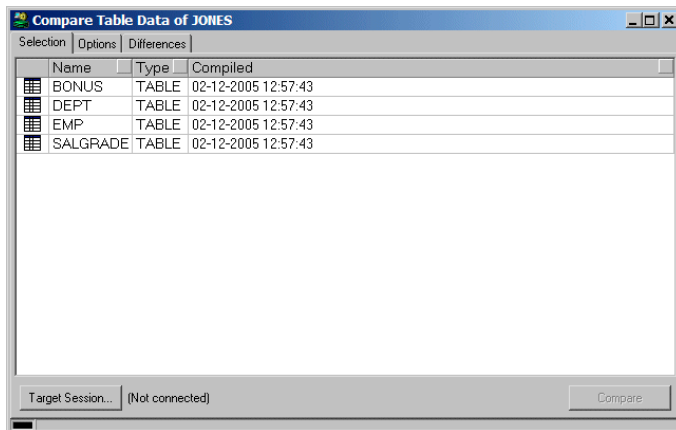
When objects are compared, the following properties are ignored:

- Storage – Properties such as the *next extent* and *pct free* of tables and indexes are not considered relevant for comparison.
- Constraints with system generated names – These constraints will have different names for the 2 users, so they cannot be compared. If a table is new in the target session, these constraints will be generated though.
- Table creation properties – Properties that would require the recreation of the table are ignored.
- Table data – To export table data, use the *Export Table* function (see chapter 17.4).
- Sequence values – The current value of a sequence is considered data.

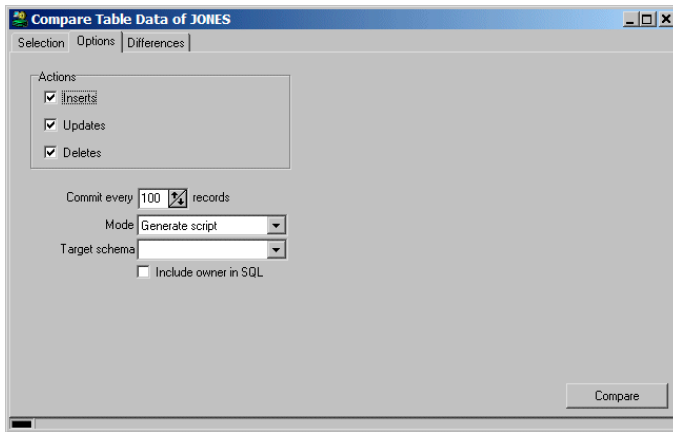
## 17.11 Compare Table Data

After inserting, updating and deleting records in one or more tables during development, it may be useful to propagate these changes to the same table owned by a different user. This may be a propagation from a development to a test database, or to a production database, or to another project member.

To do so, you can use the Compare Table Data tool from the Tools menu. This will bring up the following dialog:



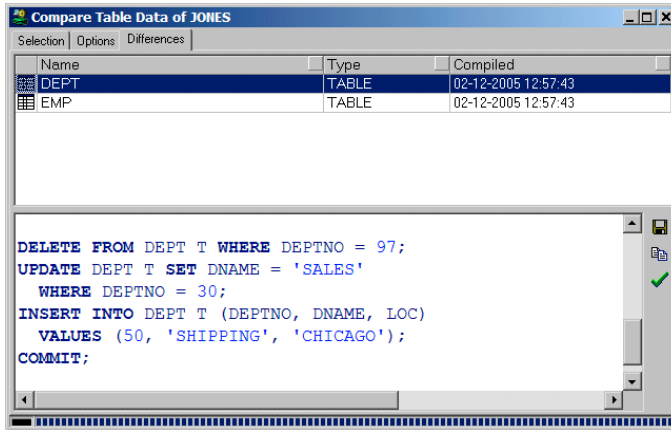
On the Selection tab page you can select the table(s) you want to compare. After making this selection, you can press the Target Session button, to select the user and database that you want to compare. This will enable the Compare button, which you can press to start the compare operation, but you can also go to the Options tab page first:



On this page you specify various options for the compare process.

- **Actions** – You can select the various actions you want to be performed in the target session. If, for example, you omit the Delete action, only inserts and updates will be performed.
- **Commit every N records** – Controls after how many records a commit will occur. If you specify 0, only one commit will occur at the end of the process.
- **Mode** – Select Generate script to generate a SQL script with inserts, updates, deletes and commits. You can save this script later and run it in the Command Window or in SQL\*Plus. Select Update database to immediately apply the changes in the target session.
- **Target schema** – If the tables are located in a different schema than the user of the target session, you can specify this schema here. The target user will need the necessary privileges on the selected tables in the target schema.
- **Include owner in SQL** – This option is only useful in script mode, and will prefix the table names in the generated SQL with the owner. Doing so will allow you to connect as a different user later when you run the script, and still apply the changes in the correct schema.

After selecting the appropriate options, you can press the Compare button to start the compare process. After completion, the Differences will be displayed:



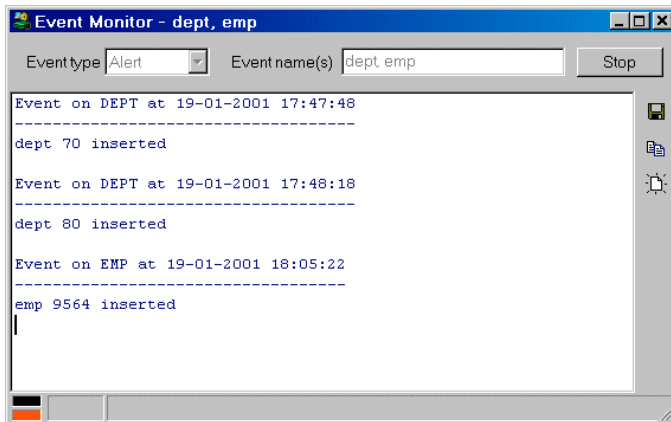
In update mode, this tab page will merely display the total number of inserted, updated, and deleted records for each table.

In script mode, you can review all differences and save the resulting script by pressing the Save SQL to file button to the right. Note that you can limit the script to specific tables by making a table selection in the upper pane. You can also copy the SQL to the clipboard or apply the selected changes in the target session by pressing the corresponding buttons to the right.

## 17.12 Event Monitor

The event monitor allows you to capture and view dbms\_pipe messages or dbms\_alert signals generated by program units in other sessions. Just specify the *event type* (Alert or Pipe), the *name(s)* of the alert or pipe, and press the *Start* button.

The following screenshot shows an Event Monitor that waits for an alert named 'DEPT' or 'EMP':



Whenever such an event occurs, the name of the pipe or alert and the timestamp is written to the output page, followed by the information of the event:

- Pipe: all items of the message are displayed

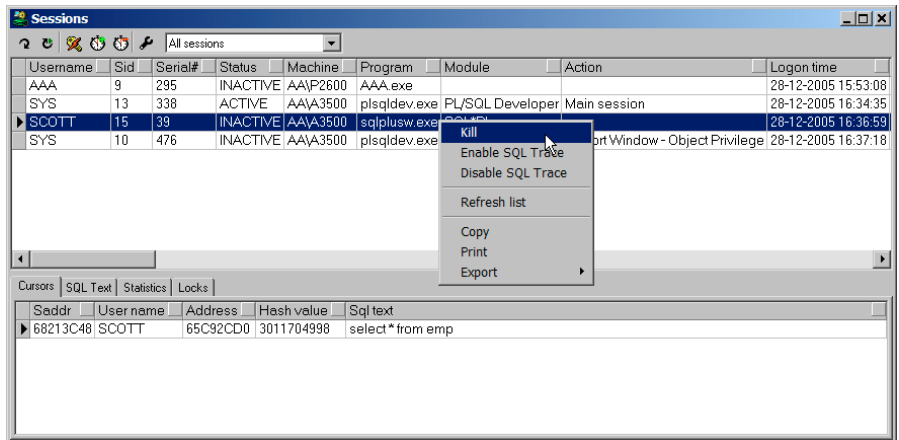


- Alert: the value of the message parameter of the dbms\_alert.signal call is displayed

For alert events you can specify multiple names, separated by colons. For pipe events you can specify just one pipe name.

## 17.13 Sessions

The *Sessions* tool displays all sessions in the database instance you are currently connected to:



In the upper half of the window you see information about the sessions, such as the username, the sid and serial number (which identify the session), the status, and so on. You can sort the session list by clicking on the header button of a column. For example: to quickly find all active sessions, click on the header button of the status column.

You can also select a *Session Filter* from the toolbar to limit the session list to a specific selection of sessions, or to limit the columns displayed. A filter can also define the order of the session and the columns.

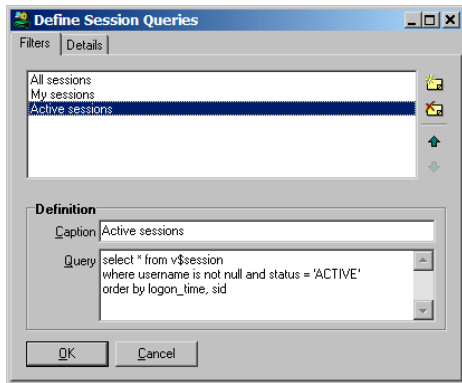
If you right-click on a session, this will bring up a popup menu with the following items:

- Kill – Allows you to kill the selected session.
- Enable SQL Trace – Enables SQL Trace for the selected session. This option will be disabled if your Oracle Server version does not support this feature, or if you do not have privileges on the dbms\_system package.
- Disable SQL Trace – Disables SQL Trace for the selected session.
- Refresh list – Refreshes the session list.
- Copy – Copies the session list to the clipboard.
- Print – Prints the session list.
- Export – Export the session list in CSV, TSV, HTML or XML format.

The Refresh, Kill, Enable SQL Trace and Disable SQL Trace functions are also accessible through the toolbar at the top of the window. The toolbar additionally includes an *Auto refresh* button. When pressed, the session list and session information will periodically be refreshed. You can define the refresh period by right-clicking on the button.

The bottom half of the window contains several tab pages with information about the selected session. The information on these tab pages can be refreshed, copied or printed by right-clicking on the list and selecting the corresponding item from the popup menu. For more information about the individual columns displayed in the lists, see the “*Oracle Server Reference*” guide.

You can create, modify or delete the *Session Queries* for the session list and the session detail tab pages. To do so, press the *Define Session Queries* button on the toolbar. This will bring up the following dialog:



At the top you can switch between the *Filters* and *Detail* queries. The filters control what is displayed in the session list, and the details control what is displayed on the detail tab pages.

At the right side you see four buttons to create a new query, to delete a query, or to move a query up or down in the list. When you create or modify a query you must supply the following information:

## Filters

The *Caption* shows up in the filter selection list in the toolbar.

The *Query* is the select statement that fetches the session information for the main session list. You can use this query to:

1. Limit the sessions displayed (e.g. only the active sessions).
2. Define the order in which the sessions are displayed.
3. Define which columns are displayed, and in which order.

## Details

The *Caption* shows up on the tabs of the detail tab pages. The position in the list determines the tab-position in the session window.

The *Query* is the SQL select statement that fetches the session information. This select statement should usually contain the *:sid* bind variable, which will contain the sid of the selected session when the query is executed. You can use any of the columns of the *v\$session* view though, such as the *:username*, *:sql\_address*, or *:sql\_hash\_value*. If your query leads to an error at run time, it will be displayed on the corresponding tab page after execution.

To concatenate the value(s) of multiple rows of a session detail query, you can use the `/* concatenate */` hint in the SQL text. For example:

```
select sql_text from v$sqltext_with_newlines
where address = hextoraw(:sql_address)
and hash_value = :sql_hash_value
order by piece
/* concatenate */
```

The `sql_text` column of all rows returned by this query will be concatenated, and displayed as one value in the session detail grid.

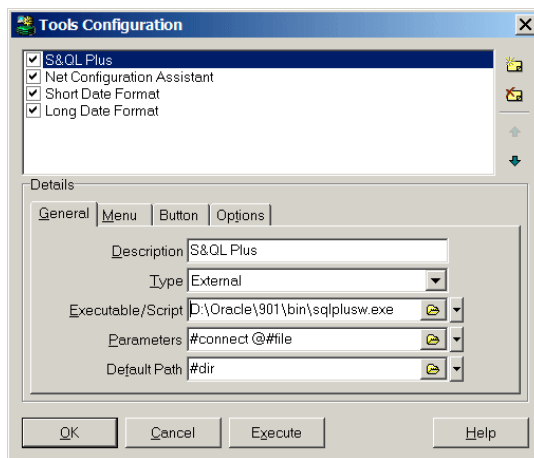
## 17.14 User Defined Tools

You can define your own tools for integration in the PL/SQL Developer IDE. These tools can be added as menu items to start external applications or to perform SQL or PL/SQL scripts in the database for the current session. You can configure parameters so you can pass a filename and connect-string to the application.

A good example of this would be SQL\*Plus. You could add a menu to start SQL\*Plus and with the appropriate settings you could let SQL\*Plus log on and even let it execute the currently opened file.

Another example is a tool to set the NLS settings to a specific set of values. You can create a script to execute the correct "alter session" statements, and add this script to the menu from where it can be invoked for the session of the SQL Window, Test Window and so on.

To configure the external tools, you'll have to select the *Configure Tools* menu item in the *Tools* menu. The following dialog will appear:



The four buttons on the right allow you to insert and delete items, as well as to move items up or down in the list. If you hold down the ctrl key when you press the insert button, the new created item will be copied from the currently selected one.

The *Execute* button can be used to execute the selected tool. If you hold down ctrl when this button is selected, a message will popup containing information (with replaced parameters) about what would have been executed.

The list shows all configured tools and the bottom half shows the configuration of the selected tool. The configuration is split into three sections:

- General – to define the executable/script that is to be run and its parameters
- Menu – to define the place where the corresponding menu-item should appear
- Button – to define a button image and description for the toolbar
- Options – for some additional tool settings

To explain the external tool configuration we will add a SQL\*Plus menu to PL/SQL Developer as an example.

## The General Tab

The first thing to do after you have created a new item with the insert button, is to define the tool type. If you select *External*, the tool will be launched as an external application. This is applicable for SQL\*Plus. If you select *Session*, the tool will run a SQL script for the current session when the tool is invoked.

Next you will need to enter a description in the General tab page. The description is the name that will be displayed in the menu and in the list. You can enter an & in front of the character you want as a shortcut key (S&QL\*Plus to get SQL\*Plus). If you enter – as a description, a separator line will be displayed.

The third crucial thing is the filename of the program or script to execute. For external tools you can enter any executable file or even documents if you wish, in which case the associated application will be started. The browse button opens a file dialog allowing you to look for files. Most Oracle tools are located in Oracle's bin directory and SQL\*Plus can be found at <oracle\_home>\bin\sqlplusw.exe.

These three steps are enough to get SQL\*Plus to work from a menu. You can additionally define parameters to be passed to the application and you can define a default path. You can use the default path as an alternative to a full path for the executable or file parameter. For SQL\*Plus you could add a connection string (#connect) and a link to a file to execute (#file).

The little button with the down arrow on the right allows you to pick a variable that you can be inserted in any of the fields. These variables are all related to the current connection and opened file. They will be replaced by corresponding information when the tool is executed. The following variables are supported:

Variable	Meaning
#file	Represents the filename (without path) of the file in the current window
#path	The filename with path
#dir	The directory of the file in the current window
#object	The selected browser object (e.g. SCOTT.EMP)
#otype	The selected browser object type (e.g. TABLE)
#owner	The object owner (SCOTT)
#oname	The object name (EMP)
#connect	The full connect string of the current PL/SQL Developer connection (e.g. scott/tiger@demo)
#username	The username
#password	The password
#database	The database

It turned out that SQL\*Plus didn't like a full path with possible spaces as file parameter, that's why #dir is specified as default directory so only the filename had to be passed to SQL\*Plus.

## Session tools

For session tools you need to specify a SQL script which can contain multiple SQL statements or PL/SQL blocks. SQL statements can be separated by semicolons or slashes, PL/SQL blocks need to be terminated with a slash. This is the same syntax as in SQL\*Plus.

Example 1 – Germany.sql:

```
alter session set nls_language = 'german';
alter session set nls_territory = 'germany';
```

Example 2 – SpecialRole.sql:

```
begin
  dbms_session.set_role(role_cmd => 'special_role identified by &Password');
end;
```

Note that you can add substitution variables to the script to make it even more flexible. You will be prompted for these variable values when the tool is invoked. When the SpecialRole script above is invoked, you will be prompted for the password.

If you want to use any of the specified parameters in the script, you must use &1, &2 and so on. The number indicates the order of the parameter on the command-line. You will not be prompted for these command-line substitution variables.

## The Menu Tab

By default all configured tools are added at the bottom of the Tool menu. If you want to create them elsewhere or even in their own main menu, you can use the menu tab page:

If you want your tool menu item to be created in a specific main menu you can select one from the Main Menu edit list or you can enter a new menu name if you want to create a new main menu.

If you have entered a Main Menu, you can use the Sub Menu and Sub-sub Menu to indicate the exact location where you want to create your new menu item.

You can use the three radio buttons (Below, Above & At End) to specify where the new menu should be placed relative to the specified menu. If you specify a new (sub) menu, you should always select 'At End' because 'Below' and 'Above' only have a meaning if you refer to an existing (sub) menu.

Example 1:

Main menu	Tools
Sub menu	Oracle tools
Sub-sub menu	
Location	At End

This will create an Oracle tools sub menu in the Tools main menu. If you were going to add multiple Oracle related tools, a specific submenu might be a good idea.

Example 2:

Main menu	Oracle
Sub menu	
Sub-sub menu	
Location	At End

This will create an Oracle main menu. You should probably only create a new main menu if you are using the tools regularly and you want them 'close-by'.

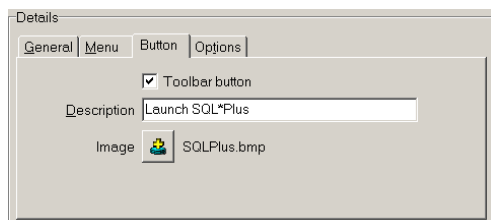
If you want to add your tool as first item in the Tools menu you should enter:

Main menu	Tools
Sub menu	Preferences...
Sub-sub menu	
Location	Above

This will create a menu above the existing Preferences menu item.

## The Button Tab

You can define if and how an external tool should be included on the toolbar:



- **Toolbar button**

When this option is enabled, the external tool can be included on the toolbar.

- **Description**

The description will be displayed as a hint when you hold the mouse cursor over the toolbar button. If you leave this description empty, the description of the external tool (as displayed in the menu) will be used.

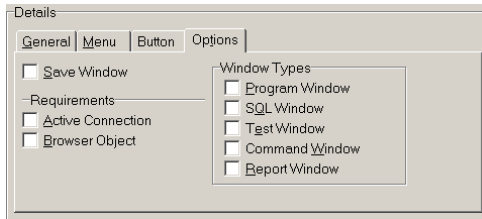
- **Image**

Press the image button to select a Windows bitmap file (\*.bmp) for the toolbar button. The size of this image should preferably be 20 x 20 pixels. Note that PL/SQL Developer will always load the bitmap file from the original location, so you should not remove or rename this file without also changing the corresponding external tool.

PL/SQL Developer comes with a number of standard bitmap files that you can choose from. These files are located in the Icons subdirectory in the PL/SQL Developer installation directory. This is the default directory of the image selector.

## The Options Tab

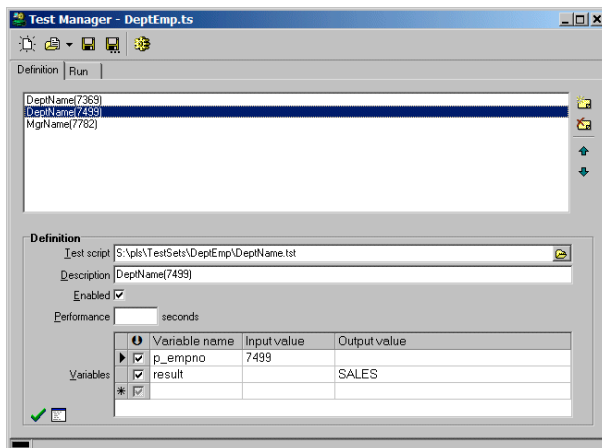
The Options tab page allow you to make the following settings:



- **Save Window**  
When this option is set, the active Window will be saved before the tool is executed. You should probably do this when the tool gets the file as a parameter and you want to be sure it uses the current data.
- **Active Connection**  
If the tool is only useful if PL/SQL Developer is connected (if you pass the connect string as parameter), you should set the Active Connection setting. This will disable the menu if PL/SQL Developer is not connected.
- **Browser Object**  
Set this option if the tool works on selected objects in the browser
- **Window Types**  
If the tool requires a specific window, you can specify these in this section. If one or more are selected, the menu will only be enabled if the active window type is specified.

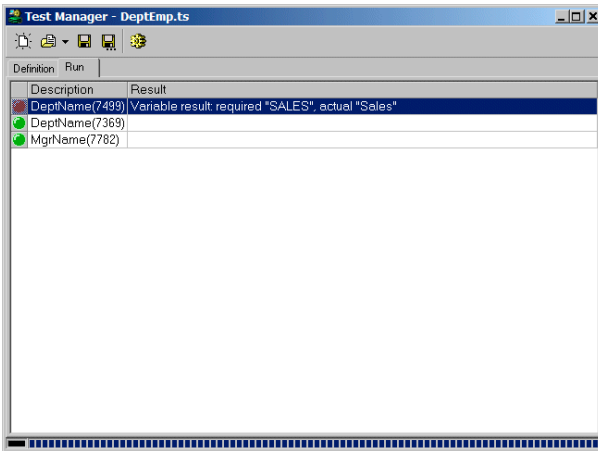
## 17.15 Test Manager

To perform regression testing for your Oracle stored program units you can use the Test Manager. It allows you to define a Test Set, which is a collection of Test Scripts with input variable values and required output. It additionally allows you to specify required performance:



For this example the Test Set consists of 3 Test Scripts. For the selected *DeptName tst* script the *p\_empno* variable will get the value 7499 on input, and after execution the *result* variable value must be *SALES*.

Running a Test Set will quickly reveal if the tested program units still function correctly and/or with the required performance:



All failed Test Scripts will have a red indicator and will be placed at the top. In this case the *DeptName(7499)* script failed, because the specification required that the result is “SALES” instead of the actual “Sales” value.

## Creating a Test Set

To create a Test Set, select the *Test Manager* item from the *Tools* menu. An empty Test Manager screen appears, for which you can create new Test Scripts or add existing Test Scripts. At the top of the Test Manager you see a list of Test Scripts, where you can add scripts to the set, remove scripts from the set, or change the order of the scripts. At the bottom you see the definition of the selected script.

Before creating a new Test Script, it is a good idea to save the Test Set by pressing the *Save Test Set* button on the toolbar. All new Test Scripts will be created in the same directory as the Test Set by default, and it may be a good idea to use a separate directory for each Test Set, so that all related test assets can be managed from this single directory.

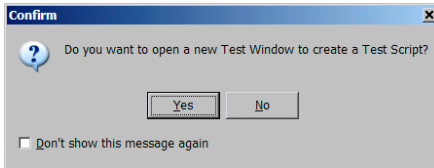
## Adding a Test Script to the Test Set

There are 2 methods to add a Test Script to the Test Set:

1. Drag & drop a procedure or function from the Object Browser to the Test Manager. A new Test Window will be opened with a call to this program unit, and variables for all parameters and the return value. Save the Test Script under a descriptive name, preferably in the same directory as the Test Set.



2. Press the *New* button to the right of the Test Script list. The following dialog will appear:



Click *Yes* to create a new, empty Test Script. You will need to manually program the Test Script and save it before it can be used in the Test Set.

Click *No* if you want to add an existing Test Script to the Test Set.

After the Test Script is saved, you can specify the definition in the Test Set:

- **Test script** – The filename of the Test Script. This will already be filled in if you have created and saved a new Test Script. If you want to add an existing Test Script, press the *Browse* button to the right of this field.
- **Description** – A functional description of the script. If you run a single script multiple times with different input variable values, you should include this in the description as well so that it can quickly be recognized in the list.
- **Enabled** – Use this option to enable or disable the script for the Test Set. Only enabled scripts will be included in the test run, and disabled scripts will be skipped.
- **Performance** – The required performance can be specified as the maximum number of seconds (fractions allowed). If you leave this field empty, the performance will not be tested.
- **Variables** – For each variable in the Test Script, enter the name, input value, and required output value. The input value will be passed to the Test Script before execution. The required output value will be tested after execution. If you leave the output value empty, it will not be tested. To test for null values, enter the word “null”.  
To test for `dbms_output`, enter “`dbms_output`” for the variable name, and enter the required text for the output value.

At the lower left you see 2 buttons that allow you to quickly run and check the selected Test Script, or to open it in a Test Window for editing.

## Duplicating a Test Script

Very often you want to run the same Test Script with different input and output variable values. To accommodate this, you can select an existing Test Script in the Test Set, and hold down the Ctrl-key while pressing the *New* button. The newly created Test Set item will inherit all properties from the selected item, so that you only need to change the description and variable values.

## Running a Test Set

To run a Test Set, press the *Run* button on the toolbar. The Test Manager will switch to the *Run* tab page, and each enabled Test Script will be executed in a separate Test Window. Failures will be indicated with a red indicator, and will be placed at the top of the list. Successful scripts will have a green indicator, and will be placed at the bottom of the list.

To investigate a failure, right-click on the script and select *Debug* from the popup menu, or double-click on the script. A Test Window with the Test Script will be created, and the input variable values will be set accordingly. Now you can debug the Test Script to determine the cause of the error.

Right-click on the test run results to copy them to the clipboard, to print them, or to export them in various formats.

## 18. The Editor

Each window that allows you to edit some SQL or PL/SQL source uses the same editor. This editor has many special features that can make life a bit easier for a programmer.

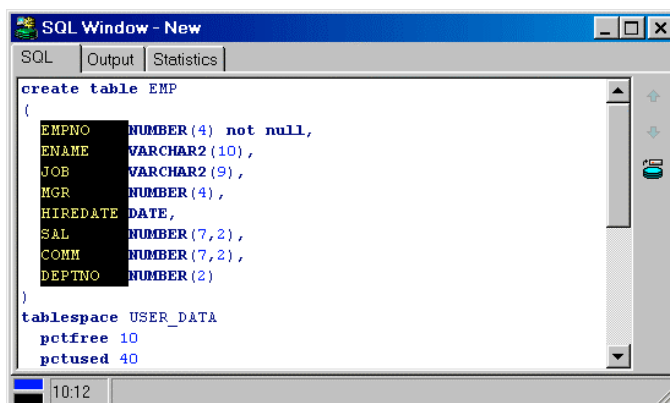
### 18.1 Selection functions

The editor allows you to perform various functions on a selection of text. These functions are available from the *Selection* item in the *Edit* menu, and are only applicable if you have made a text selection. You can assign a function key to these functions through the Key Configuration preferences.

- **Indent / Unindent**  
The number of characters depend on the editor preferences. The *Tab* and *Shift-Tab* keys are always assigned to these 2 functions if a selection exists.
- **Lowercase / Uppercase**
- **Comment / Uncomment**  
Uses */\** and *\*/* to quickly comment a selection.
- **Apply Syntax Case**  
If you have set the Editor Keyword Case preference to Uppercase, Lowercase or Init Caps, then this function will apply this style to all keywords in the selected text.
- **Sort**  
The lines of the selection will be sorted in ascending order. If you invoke the sort function again, the selection will be sorted in descending order.
- **Color mark**  
Applies the current marker color to the selection (see also chapter 18.11).

### 18.2 Column selection

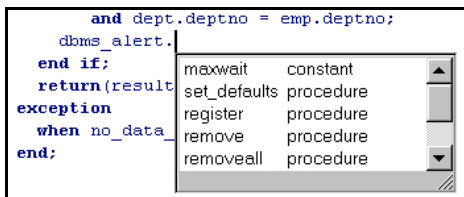
You can select a text column by holding down the *Alt* key when you make a selection:



You can copy, paste, cut, delete, uppercase, lowercase, (un)indent, and (un)comment this selection as usual. If you use the Apply Syntax Case function, and a keyword crosses the column boundary, it will not be processed.

## 18.3 Code Assistant

The code assistant is a very powerful feature that provides help as you type your SQL or PL/SQL code. When for example you type *dbms\_alert.* and hesitate for a little while, the editor will display a list of elements in the *dbms\_alert* package:



As you continue to type the name of the *dbms\_alert* procedure, the list will be reduced. If for example you type an *r* after *dbms\_alert.*, only the *register*, *remove* and *removeall* procedures will still be present in the list. When you press enter while the Code Assistant is visible, the name of the currently selected item will be inserted into the source. You can also use the arrow keys to navigate through the Code Assistant and select the desired item. Pressing the *Escape* key will remove the Code Assistant.

The Code Assistant will provide assistance for the following object types:

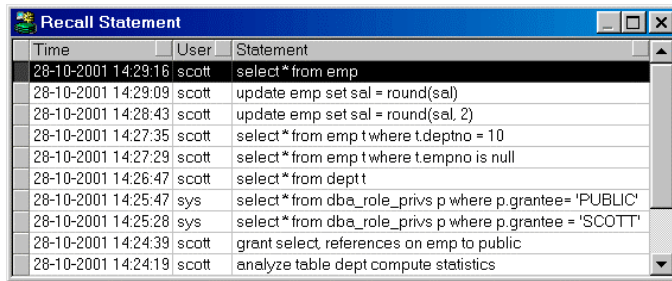
Object type	Elements
Package	Procedures, functions, types, variables, constants and exceptions
Function, Procedure	Parameters, inserted with named notation (param => )
Table, View	Columns
Sequence	nextval and curval
User	Objects owned by the user
Cursor variable	Fields of the select statement of the cursor
Record type variable	Fields of the record type
Table%rowtype variable	Columns of the table
Object type	Attributes (for default constructor)
Object type variable	Attributes and methods
Object type column	Attributes and methods
Collection variable	Collection methods (first, next, limit, and so on)

The Code Assistant can also help you type the names of object (tables, packages, and so on), keywords and PL/SQL identifiers that have meaning within the current context. If, for example, you type *pro* in a SQL Editor and hesitate, the keywords *procedure* and *profile* will be displayed by the Code Assistant, along with any database object name that starts with *pro*. The context depends on the currently connected user (which determines the available object names), the type of editor (which determines the available keywords), and the program unit you are editing (which determines the available PL/SQL identifiers).

The Code Assistant can be automatically and/or manually invoked, and the delay time before the Code Assistant automatically becomes active after typing an object name can be defined by a preference. Furthermore you can define if you want to describe a user, and if so, which object types you want to be included. The context description and the minimum number of characters that need to be typed before it can be activated can be configured as well. All these preferences are described in chapter 16.14.

## 18.4 Recalling statements

Whenever you successfully execute a statement in a SQL Window, Command Window, or Report Window, this statement is stored in the global statement recall buffer. You can recall the most recently executed statements in the editor by selecting *Recall Statement* from the *Edit* menu, or by pressing *Ctrl-E*. This will bring up a selection list like this:



Time	User	Statement
28-10-2001 14:29:16	scott	select * from emp
28-10-2001 14:29:09	scott	update emp set sal = round(sal)
28-10-2001 14:28:43	scott	update emp set sal = round(sal, 2)
28-10-2001 14:27:35	scott	select * from emp t where t.deptno = 10
28-10-2001 14:27:29	scott	select * from emp t where t.empno is null
28-10-2001 14:26:47	scott	select * from dept t
28-10-2001 14:25:47	sys	select * from dba_role_privs p where p.grantee= 'PUBLIC'
28-10-2001 14:25:28	sys	select * from dba_role_privs p where p.grantee = 'SCOTT'
28-10-2001 14:24:39	scott	grant select, references on emp to public
28-10-2001 14:24:19	scott	analyze table dept compute statistics

The statements are displayed in order of execution, with the most recent one displayed first. In the list you see the time, the user, and (part of) the statement text. You can now select a statement and double-click it (or press *Enter*) to insert it at the cursor position in the editor from where the list was invoked.

Note that statements that include passwords are not placed in the global statement recall buffer for security reasons.

You can sort the list by time, by user or by statement text by pressing the corresponding buttons in the header of the list. To display just the statements that were executed by the currently connected user, right click on the list and select the *Current user* item from the popup menu. In the same popup menu you can switch back to *All users*, and you can delete a statement or copy the statement text to the clipboard.

To find a specific statement you can invoke the find function by using the corresponding function key (default *Ctrl-F*), or by right-clicking on the list and selecting *Find statement* from the popup menu. The find function will start after the currently selected statement, and will continue at the first statement if it has not found a match.

Furthermore the popup menu contains functions to delete the selected statement from the list, to copy it to the clipboard, or to export the entire list to a CSV file.

The recall buffer can hold up to 200 statements, which will be stored in the application data directory of your windows profile. You can change this statement limit and the directory through the preferences (see chapter 16.25).

## 18.5 Special Copy

If you are using PL/SQL Developer to write SQL and PL/SQL code that you subsequently want to use in other tools, such as a 3GL programming language, it may be that you need the code in a slightly different format. Let us assume that you wrote and tested a SQL statement like this in PL/SQL Developer:

```
select deptno, sum(sal) mgr_sal from emp
where job = 'MANAGER'
group by deptno
order by mgr_sal desc
```

If, for example, you want to use this statement in Borland Delphi, you may need a format like this:

```
SQL := 'select deptno, sum(sal) mgr_sal from emp' + #13#10 +
'where job = 'MANAGER'' + #13#10 +
'group by deptno' + #13#10 +
'order by mgr_sal desc';
```

For this purpose you can use the *Special Copy* function from the *Edit* menu or from the popup menu after right clicking on a selection in an editor. This function has a submenu, which shows all special copy formats that are defined. After selecting a format, the converted code is stored on the clipboard, so that you can paste it in the editor of the corresponding tool.

The special copy formats are defined in the SpecialCopy subdirectory in the PL/SQL Developer installation directory. You can change the pre-defined copy formats, or add new copy formats. Simply add a text file with a .copy extension that contains a variable for the first line of the PL/SQL code (<line\_1>), the last line of the PL/SQL code (<line\_N>) and a variable for all other lines (<line\_\*>). The example for Borland Delphi is as follows:

```
;PL/SQL Developer SpecialCopy definition for Borland Delphi
;<line_1> for first line
;<line_*> for all other lines
;<line_N> for last line
;
SQL := '<line_1>' + #13#10 +
'<line_*>' + #13#10 +
'<line_n>';
```

The first line needs to be preceded by the assignment to the SQL variable, and needs to be followed by a CR/LF pair. The last line does not need the CR/LF pair, but needs to be terminated by a semi-colon. All other lines merely need to be followed by CR/LF.

You can omit <line\_1> and <line\_n> if they are the same as <line\_\*>.

In some languages you need to use escape sequences for special characters. For example, in C++ you would use \t for a tab character (ASCII code 9). To define these escape sequences, use the #define keyword:

```
#define char(9) = \t
#define \ = \\
String("<line_1>\n") +
String("<line_*>\n") +
String("<line_n>");
```

You can additionally use “#define compress” to indicate that you want to remove all redundant blank characters from the result (spaces, tabs, and linefeeds).

Note that the name of the .copy file will be included in the menu, so you should use a descriptive filename.

## 18.6 Context sensitive help

Most programmers frequently access reference information on the functions that they are using in their programs. To find this reference information as quickly as possible, placing the cursor on a keyword in the source text and pressing *F1* will search for this keyword in Oracle's online manuals.

You can read chapter 27 for more information about PL/SQL Developer's Help system.

## 18.7 Database object popup menu

When your program accesses a database object, you very often need to get some information from this object, or you want to manipulate it. You may want to know its description or its properties. For a table or view you may want to query or edit the data. For a function or procedure you perhaps want to test a certain scenario.

If you right-click on an object name in the editor, you will see the same popup-menu that you see when you right-click on it in the Object Browser. Therefore, all relevant functions for this object are immediately available from within the editor.

## 18.8 Explain Plan

To invoke an Explain Plan Window for a SQL statement in an editor, select the text of the statement, right-click on it, and select the *Explain Plan* item from the popup menu. If no text is selected, the entire source will be taken to the Explain Plan Window. If the SQL statement contains PL/SQL variables, you must alter the statement in the Explain Plan Window as described in chapter 5.1.

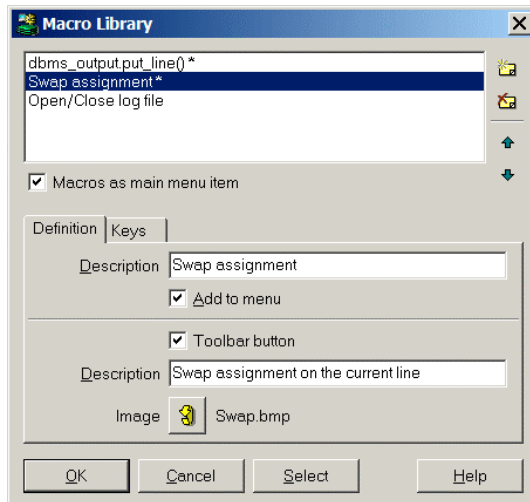
## 18.9 Macros

The editor has a powerful macro function that allows for quick and easy macro recording and playback, which can be used to automate a specific repetitive task.

A recorded macro can be stored in the Macro Library to be executed later. Macros in the library will usually be some recurring repetitive task (like swap assignment: `a := b; -> b := a;`), or may contain code snippets. For example: why continuously look up how a cursor type must be declared when you can type it once and record for later use?

To record a macro, press *F11* or select the *Record* item in the *Macro* menu of the *Tools* menu. The toolbar now displays a flashing recorder icon, indicating that you are currently recording a macro. All keyboard strokes are recorded for playback, which implies that you should not use the mouse during macro recording. By pressing *F11* again, macro recording is ended. Pressing *F12* will playback the macro.

To store the currently recorded macro in the Macro Library, select the *Library* item in the *Macro* menu. This will open the Macro Library dialog:



Press the *New* button to add a new macro with the currently recorded definition. You can enter a description for the macro and indicate whether it should be included in the Macro menu. This way you can make some macros easily available, and you can additionally assign a function key to each macro through the Key Configuration preferences. If the *Macros as main menu item* option is checked, the *Macros* menu is located in the main menu. If it is not checked, it will be located under the *Tools* menu.

When the *Toolbar button* option is enabled, the external tool can additionally be included on the toolbar. The corresponding *Description* will be displayed as a hint when you hold the mouse cursor over the toolbar button. If you leave this description empty, the main description of the macro (as displayed in the library and in the menu) will be used. Press the *Image* button to select a Windows bitmap file (\*.bmp) for the toolbar button. The size of this image should preferably be 20 x 20 pixels. Note that PL/SQL Developer will always load the bitmap file from the original location, so you should not remove or rename this file without also changing the corresponding macro. PL/SQL Developer comes with a number of standard bitmap files that you can choose from. These files are located in the Icons subdirectory in the PL/SQL Developer installation directory. This is the default directory of the image selector.

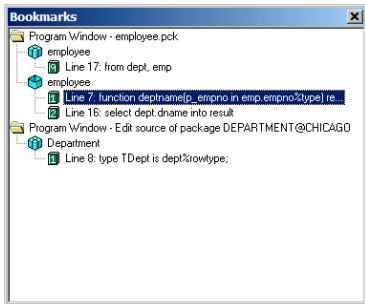
On the *Keys* tab page you can view and change the keyboard actions for the macro.

To execute a stored macro, open the Macro Library and double-click on its name. The selected macro can now be executed by pressing *F12*. When you select a macro in the menu, it will immediately be executed, and it can be re-executed by pressing *F12*.

## 18.10 Bookmarks

To mark a location in an editor you can add a bookmark. Simply press Ctrl-Kn (where n is a number from 0 .. 9) to add a bookmark at the current location. A green bookmark will appear in the gutter, with the corresponding number. To navigate to this bookmark later, you can press Ctrl-Qn. Alternatively, you can use the *Set bookmark* and *Go to bookmark* submenus from the *Edit* menu.

The *Bookmark list* item of the *Edit* menu will bring up an global hierarchical display of all editor bookmarks:



In this tree view you can see the windows, editors, and lines where a bookmark is located. Clicking on a bookmark will bring the window and editor to the front, will navigate to that bookmark. You can dock the bookmark list to make it accessible at all times without blocking any other windows.

## 18.11 Color marks

You can add color marks in an editor to highlight specific sections. To do so, select the text you want to mark and press the *Color marker* button on the toolbar:



The background color of the selected text will change to the current marker color:

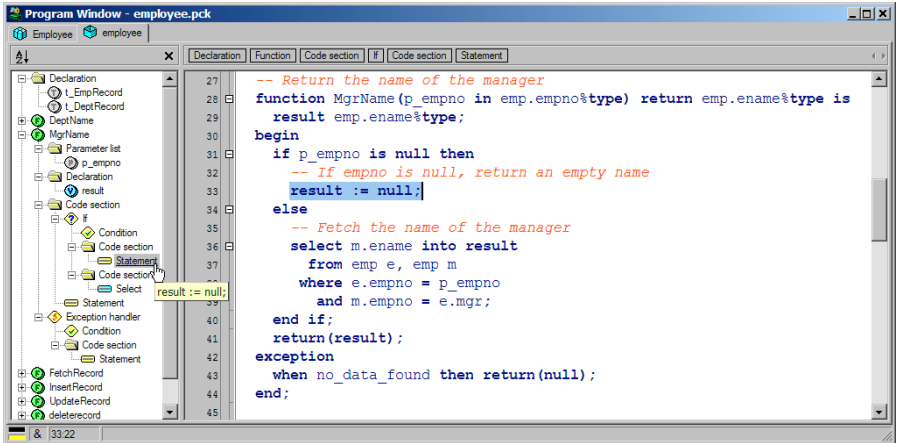
```
-- Fetch the name of the manager
select m.ename into result
  from emp e, emp m
 where e.empno = xa
        and m.empno = e.mgr;
```

To change the marker color, press the arrow next to the button and select a new color. To clear a color mark, place the cursor within the marked text without making a selection, and press the *Color marker* button.



## 18.12 Code Contents

The Code Contents feature is restricted to the Program Editor. It displays the complete structure of a program unit. This is most useful in large package bodies and type bodies, which can span thousands of lines of PL/SQL code. The Code Contents are displayed in a tree at the left side of a Program Window:



The Code Contents keep track of your current location in the source code. As you navigate through the source code, the selected item will indicate where exactly you are located.

If you move the mouse over an item in the Code Contents, the corresponding source code in the editor will be highlighted as illustrated above.

Clicking on an item in the Code Contents will immediately navigate to its declaration. Double-clicking on the item will expand or collapse it. Right clicking on an item will display a popup menu that allows you to select the corresponding source code in the editor, to cut or copy it to the clipboard, to paste over it from the clipboard, to convert it to a comment, to expand all child items, or to create a Test Script for it (if it's a public program unit).

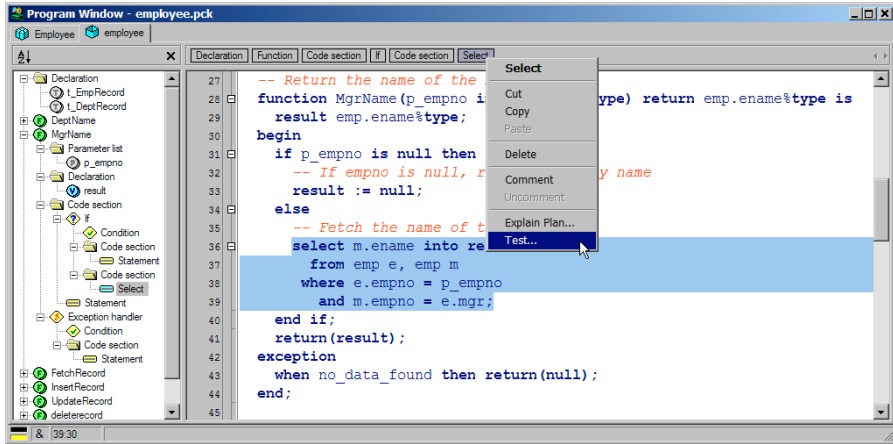
The contents can be alphabetically sorted by pressing the *Sort contents* button at the top of the content pane. Only the main contents will be sorted: parameters, local variables and so on will not be sorted. The button will stay down, and pressing it again will revert the contents to their original order.

To close the Code Contents you can click on the *Close* button at the upper right corner. You can also right click in the editor and select the *Contents* item to make it disappear or appear, or select the *Code Contents* item in the *Tools* menu.

By default the Code Contents are visible. To change this default you can close the Code Contents in the current Program Window and select the *Save Layout* item in the *Window* menu.

## 18.13 Code Hierarchy

The Code Hierarchy is restricted to the Program Editor, and displays the hierarchical structure of the PL/SQL code at the cursor location. It is located above the editor:

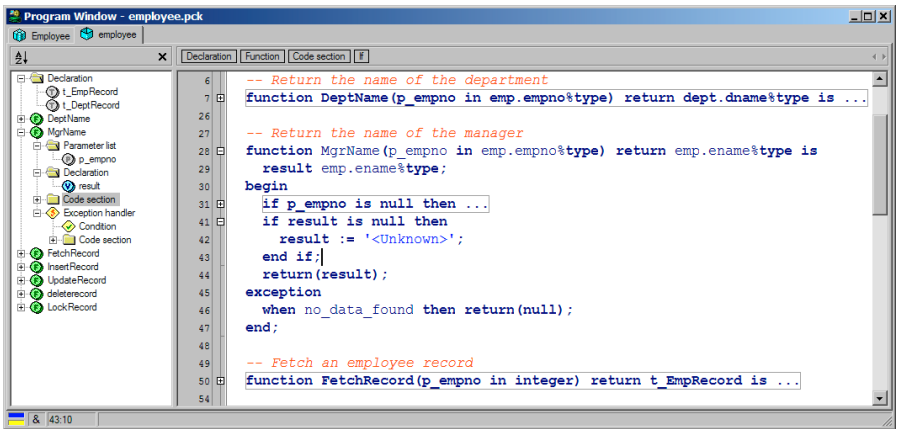


In the example above, the cursor is located at a **Select** statement within a **Code section** within an **if** statement within a **Code section** within a **Function**. If you move the mouse over the items of the Code Hierarchy, the corresponding source code will be highlighted. Clicking on an item will select the corresponding source code.

If you right-click on an item (as illustrated above), a popup menu will be displayed that allows you to select the source code, to cut or copy the source code to the clipboard, to paste the clipboard contents over the item, to delete the source code, or to convert the source code to or from a comment.

## 18.14 Code Folding

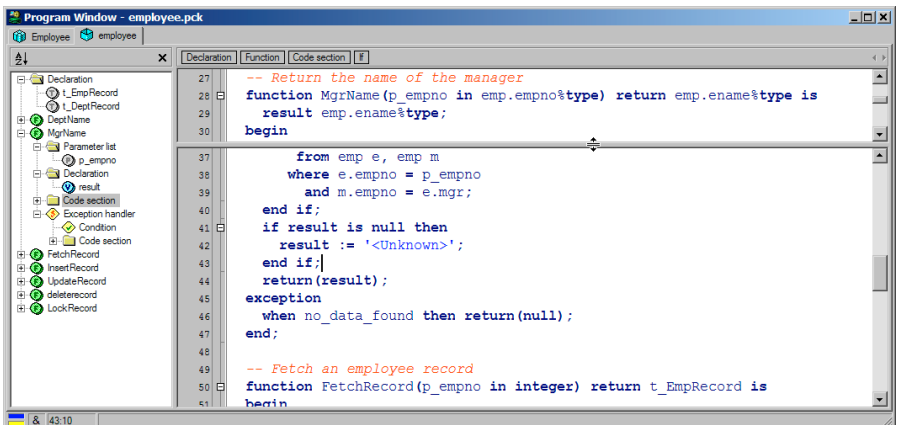
Code Folding allows you to show or hide specific sections of PL/SQL code. You could for example fold all procedures and functions within a package, and unfold just the one that you are interested in. As another example you can fold structures like loops and if/then/else statements to get a better overview within a long program unit. In the following example, several functions have been folded, as well as an if-statement:



To fold a code section, press the minus sign in the left margin of the editor. To unfold it, press the plus sign in the left margin. You can also right-click in the left margin and select *Fold All* or *Unfold All* from the popup menu. If you move the mouse cursor over a folded line in the editor, the folded section is temporarily displayed in a separate pane.

## 18.15 Split Editing

To edit or view 2 separate sections of the same source, you can use the Split Editing feature. Simply use the mouse to drag the splitter at the top of the editor down:



You can now navigate, view, and edit both sections simultaneously.

## 18.16 Hyperlink navigation

In your PL/SQL code you very often have references to elements that are defined elsewhere. You can for example reference a PL/SQL type within the same package body, or within the specification of the same package, or within another package. The same goes for function or procedure calls, variables, views, and so on.

To easily find a declaration from within a Program Window, you can move the mouse pointer over such a reference, hold down the Ctrl key, and press the left mouse button. After holding down the Ctrl key, the identifier under the mouse pointer will become light blue and underlined, resembling the familiar hyperlinks in an HTML document:

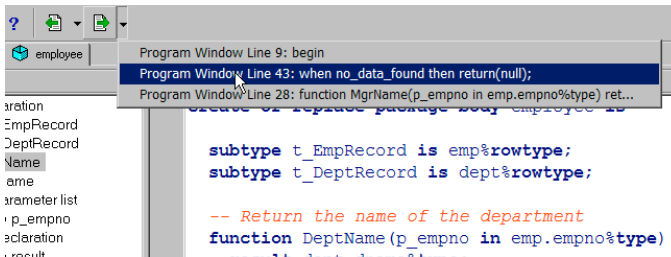
```
-- Created : 17-12-1999 17:19:26
-- Purpose : Lock an employee record
procedure lockrecord(p_empno in emp.empno%type) is
  dummy varchar2(1);
begin
  if p_empno != g_lastrecord.empno then
    select null into dummy from emp
    where empno = p_empno
    for update;
  end if;
end lockrecord;
```

If the declaration is located within the same source, PL/SQL Developer will navigate there. After that, the rest of the program units within the same Program Window will be searched. Then it will search all other Program Windows that are currently open. Finally, it will try to open a new Program Window or SQL Window for the object that contains the declaration and will navigate there.

By default hyperlink navigation will locate a declaration in a package or type specification, if appropriate. To open the package or type body instead of the specification, you can right-click on a hyperlink.

## 18.17 Navigation buttons

Whenever you jump to an absolute location from within an editor in a window, PL/SQL Developer will make a bookmark for the previous location. These bookmarks can be accessed with the Navigate back and Navigate forward buttons on the main toolbar:



Bookmarks will be added when you navigate in the Code Contents, use the Find function, click on a compilation error, go to the top or end of the editor, use hyperlink navigation, open a new window, and so on.

## 18.18 Refactoring

The refactoring function allows you to quickly reorganize your PL/SQL code. It works on the selected code, or – if no selection is made – on the current statement. Right-clicking on a statement or selection provides the following refactoring functions in the corresponding submenu:

- **Rename item**  
Renames the current variable, parameter, constant or program unit. Both the declaration and usage will be renamed.
- **Extract procedure**  
If a program unit has become too large or too complex, you can make a selection and convert it to a separate procedure. All variables used within and outside of the selection will be converted to parameters. All variables used only within the selection will be moved from the current program unit to the new program unit. The selection will be replaced by a call to the new program unit.
- **Extract local constant**  
If a certain expression should be converted to a local constant, you can select it and provide the constant name. A local constant will be created within the current subprogram, of a type that is determined from the expression. All occurrences of the expression in the current subprogram will be replaced by the constant name.
- **Extract global constant**  
If a certain expression should be converted to a global constant, you can select it and provide the constant name. A global constant will be created within the current package, of a type that is determined from the expression. All occurrences of the expression in the current package will be replaced by the constant name.
- **Replace assignment with initialization**  
If a local variable assignment is purely for initialization, you can move it to the declaration of the variable. You can right-click on a statement or select multiple statements.

If the results of a refactoring action are not satisfactory, you can simply undo it.

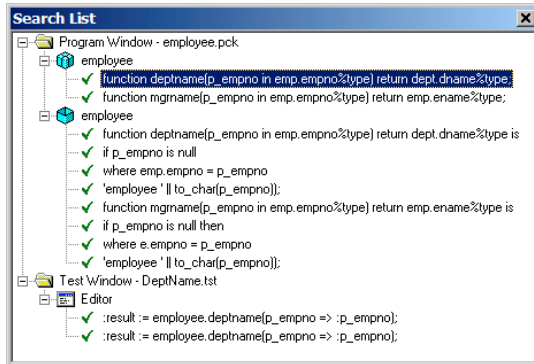
## 18.19 Search Bar

The Search Bar is available from the Edit menu and can be displayed as a floating tool, or docked at the top or bottom of the work area. In its docked position, it can be permanently available to perform searches across multiple editors and multiple windows:



After entering a search word, you can press the *Search* button or press *Enter*. The search results will be highlighted in the editor(s), and you can press the *Go to next/previous occurrence* buttons to navigate the search results. You can press *Esc* to move the focus from the search bar to the editor. Press the *Hide occurrences* button to clear the highlighted search results in the editors.

The *Search list* button will bring up an hierarchical display of the search results:



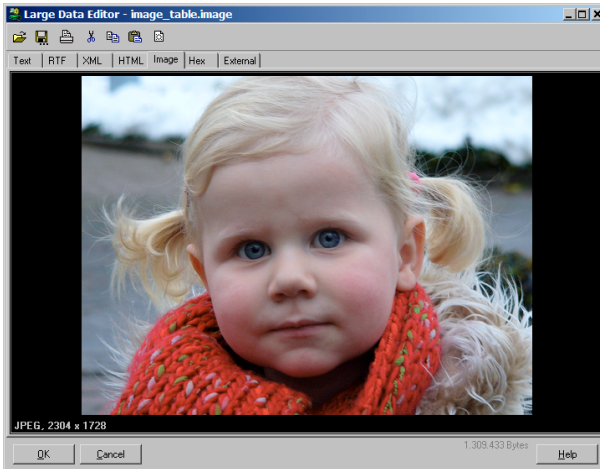
In this tree view you can see the windows, editors, and lines where the search results are found. Clicking on a search result will bring the window and editor to the front, will navigate to the search result, and will select it. You can dock the search list to make it accessible at all times without blocking any other windows.

On the search bar you can additionally enable options to search all editors or just the current editor, to search case sensitive, to search for whole words only, and to use regular expressions.

## 19. The Large Data Editor

The Large Data Editor will be invoked whenever you need to view or edit long, long raw, CLOB, BLOB or BFILE column or variable values. It can additionally be used to edit varchar2 data. The Large Data Editor will analyze the data, and will attempt to display it in the correct format. It can display values as text, RTF, XML, HTML, Image (bmp, jpg, gif, tiff and others), and Hex data. Furthermore it can launch an external application (e.g. MS Word or Acrobat Reader) to view or edit the data.

Invoking the Large Data Editor for an image will result in the following screen:



The toolbar at the top of the editor includes the following buttons:

- Open – Opens the contents of a file into the editor.
- Save – Saves the contents of the editor to a file.
- Print – Prints the editor contents in the currently selected format.
- Cut – Place the editor contents on the clipboard in the currently selected format, and subsequently clear (null) the editor contents.
- Copy – Place the editor contents on the clipboard in the currently selected format.
- Paste – Paste the clipboard data into the editor in the currently selected format.
- Clear – Clear (null) the editor contents.

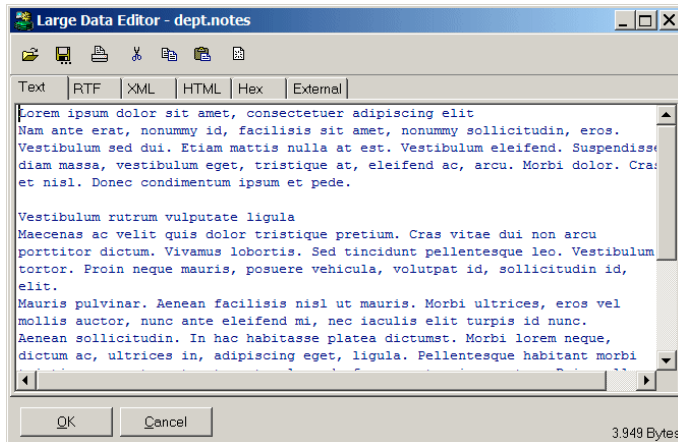
Below the toolbar you can select a tab page for the correct format of the editor contents. Multiple formats may be applicable for some data. For example, an HTML document can be viewed in HTML format or in Text format. All data can be viewed in Hex (binary) format.

When making changes to the editor contents in a specific format, you can immediately switch to a different format to view these changes. For example, when viewing an HTML document you can switch to Text format to make changes and return to HTML format to view these changes.

After pressing the OK button your changes will be propagated to the function from where the Large Data Editor was invoked.

## 19.1 Editing plain text

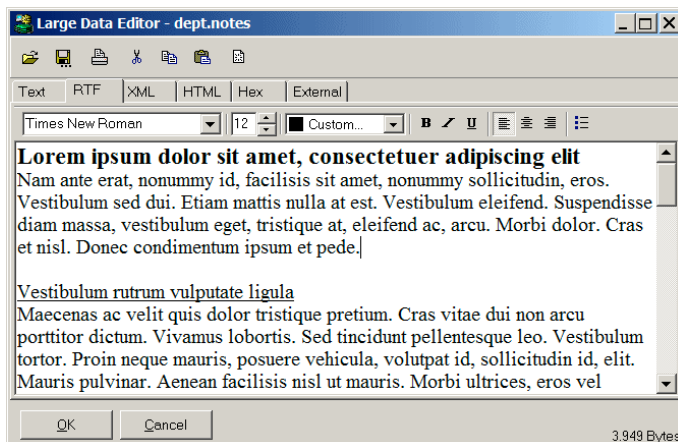
The text editor can be used to view or edit any kind of plain text data:



It can also be used to edit XML and HTML text. Switching to the XML or HTML tab page will immediately show the results of these changes.

## 19.2 Editing RTF

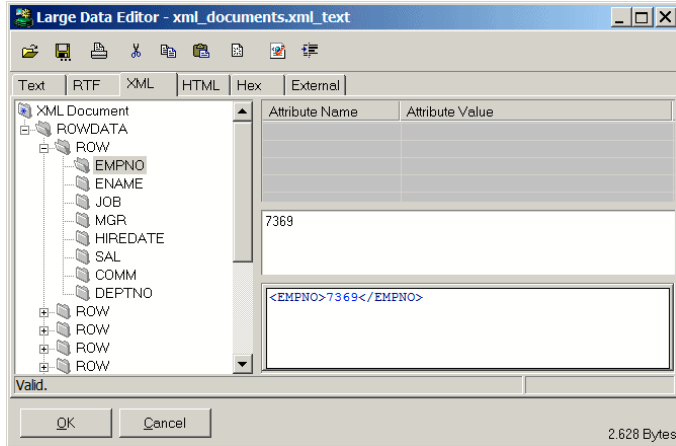
The RTF editor can be used to view and edit text in Rich Text Format. It allows you to easily change font type, size, color, style, and alignment:





## 19.3 Editing XML

When viewing XML content, the editor will initially switch to the XML tab page:

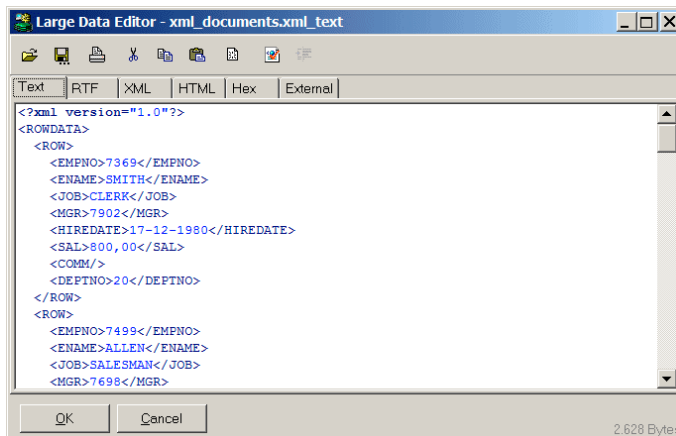


Here you can see the structure of the XML document in a tree view, and edit individual elements and attributes.

Pressing the *Parse XML* button will parse the XML document, and will report any errors. This requires that you have Net8 8.1 or later, and that the user has access to the SYS.XMLTYPE type or the XMLPARSER package.

You can press the *Format XML* button on the toolbar to apply standard formatting rules on the document.

If you switch to the plain text format tab page for an XML document, then the appropriate syntax highlighting will be applied.



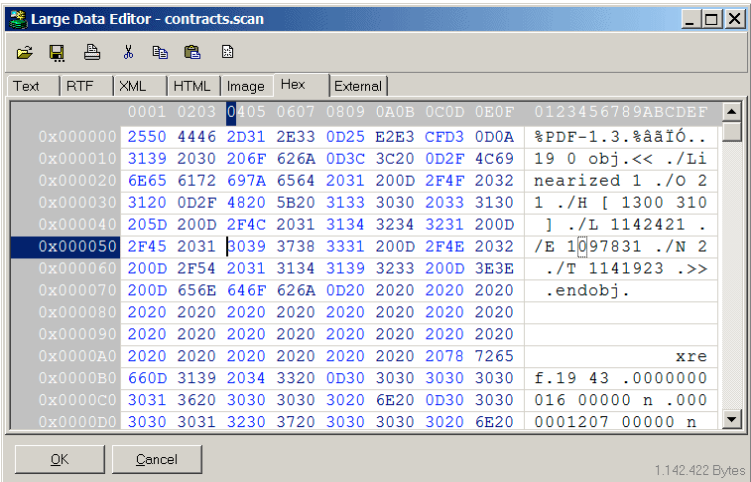
## 19.4 Editing images

The image editor will display images in most popular formats, such as JPG, BMP, GIF, and so on. If the image is larger than the editor can display, the image will be resized to fit. This does not affect the contents, it is merely a visual effect. Below the image you will see the image format and the image size in pixels.

To change an image, you can either press the *Open* button on the toolbar to load it from a file, or copy the image from another source to the clipboard and press the *Paste* button. Note that the image format will change to BMP when copying/pasting via the clipboard! To ensure that the original format is preserved you should load it from a file.

## 19.5 Editing Hex data

To view or edit the editor contents in hexadecimal format you can switch to the Hex tab page:

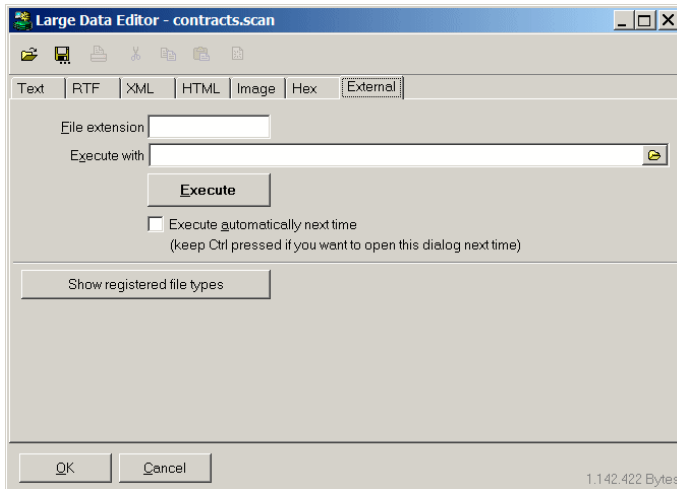


On the left you see 8 columns with words (16 bits) of hexadecimal data. On the right you see the text representation of the hexadecimal data. You can edit both the hexadecimal and textual representation.

If you right-click on the hex editor you can select a different column layout.

## 19.6 Invoking an external viewer or editor

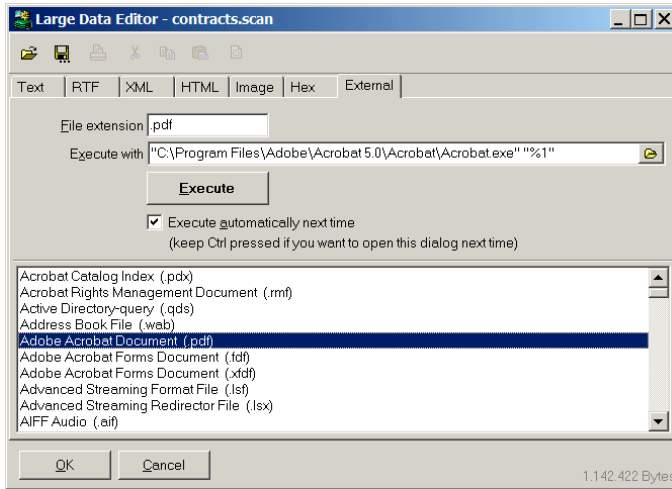
For many large objects that are stored in the database it may be necessary to invoke an external viewer or editor. For example, if a BLOB column contains the contents an Acrobat PDF file (Portable Document File), it will be necessary to invoke the Acrobat Reader. When invoking the Large Data Editor for such content, you will initially be taken to the Text or Hex tab page. After that, you can switch to the External tab page:



Here you can define which program should be executed. First you will need to define the file extension. Before executing the external program, a temporary file with the editor contents will be saved with that extension. Next you will need to define which program should be executed. After that you can press the *Execute* button to view the content in the external program. If you change the file in the external program and save it, the Large Data Editor will notice this change and will allow you to import these changes back into the editor.

When viewing column data, the association between the table/column and the external program will be remembered for the next time you want to view it. If you additionally enable the *Execute automatically next time* option, then the Large Data Editor will immediately invoke the external program. To override this automatic behavior, hold down the *Ctrl* key when invoking the editor.

Most programs will have registered their file types, and instead of manually defining the extension and program you can simply select it from a list. First you will need to press the *Show registered file types* button. After that you can search for the correct file type and double-click on it. For a PDF file this would be the *Adobe Acrobat File*:



If you right-click on the file type list, you can sort it by extension or description (default).

## 20. The Query Builder

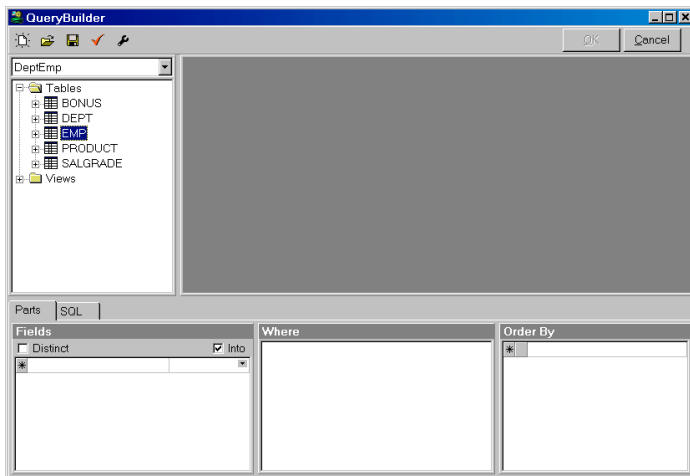
The Query Builder allows you to create and modify select statements in your PL/SQL and SQL source files. It provides a simple point and click interface to build the field list, the table list, where clause and order by clause of the select statement.

### 20.1 Creating a new select statement

For this example we will assume that we want to create a join statement between the dept and emp table, and show some columns from both tables:

```
select e.empno, e.ename, d.deptno, d.dname
from emp e, dept d
order by e.empno
```

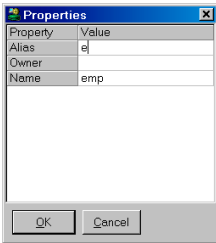
To create a new select statement, place the editor cursor at the text position where you want it to be inserted, and press the *Query Builder* button on the toolbar (or select the *Query Builder* item from the *Tools* menu). This will bring up an empty Query Builder Form:



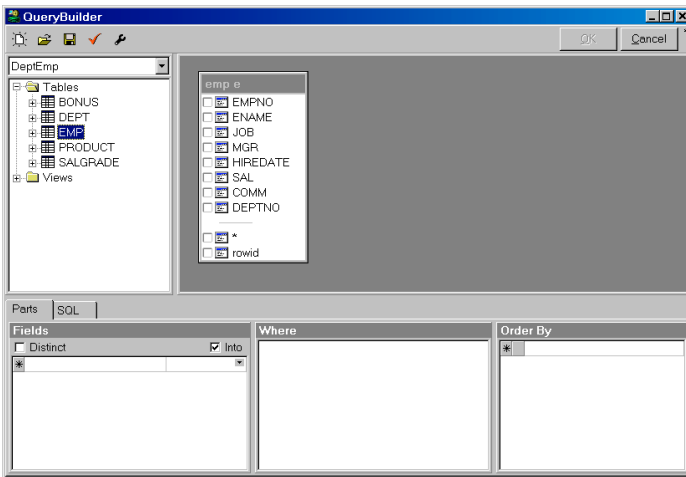
On the left side you see an Object Browser with just the tables and views. On the right you see the work area, which is empty right now. At the bottom you see 3 panels with the field list (and optionally the *into* item list), the where clause and the order by clause. The bottom section contains a tab page that allows you to switch to the actual SQL text that would be created from the current query definition.

At the top of the window you see a toolbar with a *New* button to create a new query, an *Open* button to open a previously saved query definition from a file, a *Save* button to save the current query definition to a file, a *Parse* button to parse the current query definition, and a *Preferences* button. The *OK* button returns you to the PL/SQL or SQL Editor and inserts the SQL text you have built.

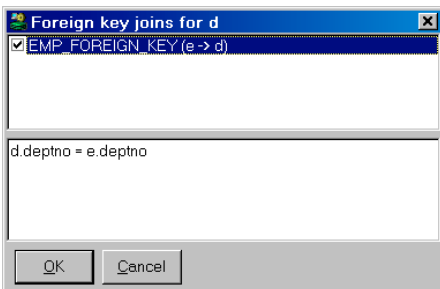
To include tables or views in the select statement, drag them from the Object Browser into the work area. Each time you add a new table, the *Table Properties* dialog appears:



Here you can enter an alias for the table. For our example we wanted to use 'e' for the *emp* table:

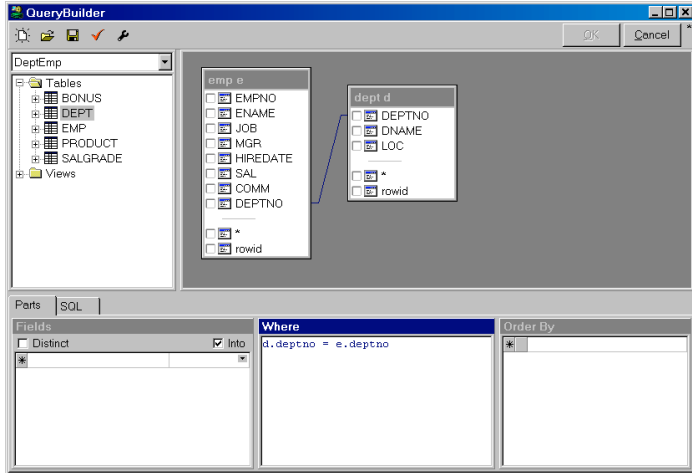


After adding the *Dept* table in the same way, you are asked if you want to join the 2 tables using the columns of the foreign key named *EMP\_FOREIGN\_KEY*:

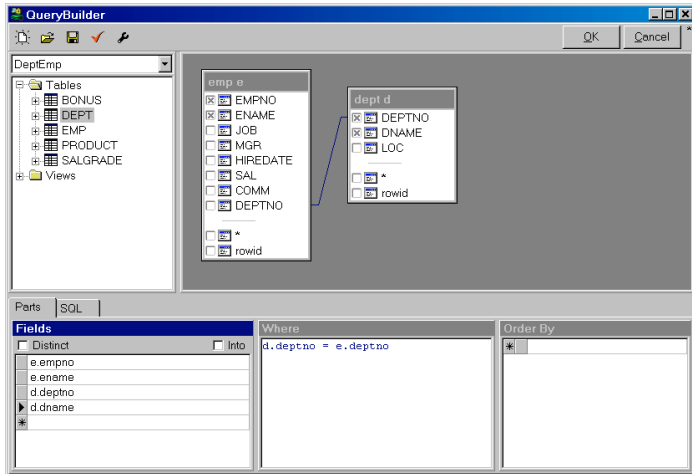


After checking this option, you will see the new join condition in the bottom half of the form.

Now the *Dept* table is added, and the join condition is visualized in the work area:

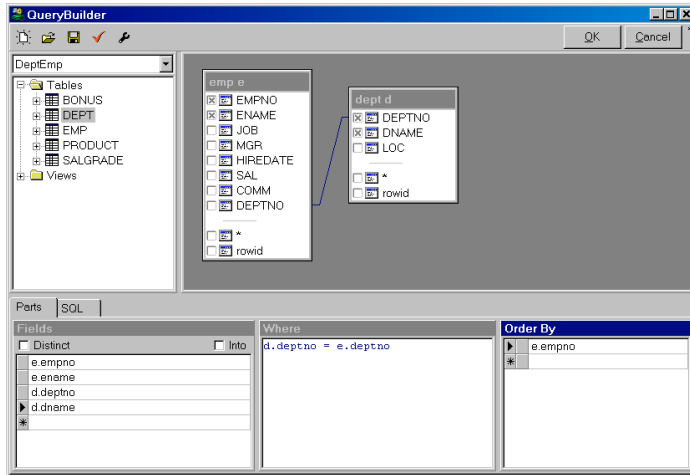


What we need to do now is include the columns in the field list of the select statement. Just click on the checkbox next to the column names. The columns will be displayed in the *Fields* panel:



If you want to create a PL/SQL *select .. into* statement, you would also need to specify the items for the *Into* list next to the fields. The selection list will display the PL/SQL variables that are defined in the PL/SQL code where you insert the statement. In this case we do not want to create an *Into* list, so we can leave these items empty.

To create the order by clause, drag the *empno* column to the *Order By* panel:



Now you can press the *OK* button to insert the SQL statement into the editor.

## 20.2 Modifying an existing select statement

To modify an existing select statement in your PL/SQL or SQL source code, simply right-click on the select statement and select the *Query Builder* item from the popup menu. The complete select statement will be marked in the editor, and the Query Builder will be displayed. Now you can change the query definition as described in the previous chapter.

## 20.3 Manipulating the query definition

The following paragraphs describe how you can manipulate the query definition.

### Changing the field list

To add a field from a table in the work area to the query, you can click on the checkbox next to the column, or you can drag the column to the field list. You can alternatively double-click on the column if the field list has the focus.

You can also drag a column or column folder directly from the Object Browser to the field list. Column folders are located under the table, its constraints and its indexes.

You can type column names and other expressions directly in the field list.

To delete a field from the field list, right click on it and select *Delete selected items* from the popup menu. If you select multiple fields, you can remove them all at once.

You can move fields to a different position by selecting them, and dragging them to their new position.

If you want a *distinct* result set, check the corresponding option above the field list.

### Using field aliases

To use an alias for a field, simply type the alias after the expression, just like you would do if you were typing the select statement by hand.



## Changing the where clause

The where clause is a normal text editor, where you can enter the conditions. To include a column, you can drag it from the table in the work area, or you can double-click on the column if the where clause has the focus.

To add a join condition based on a foreign key constraint definition, right-click on the table in the work area and select the *Foreign keys* item from the popup menu. You cannot use this function to remove an existing join condition, but need to remove the text in the editor instead.

## Changing the order by clause

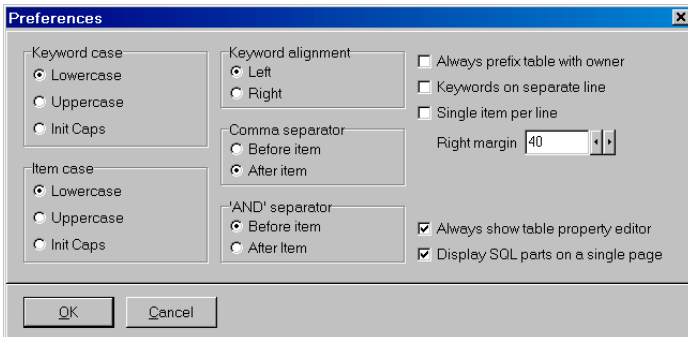
The order by clause can be manipulated in the same way as the field list, except that you cannot use the checkboxes next to the column names. To change an item between ascending and descending order, you can click on the button next to the item, which will display a corresponding arrow.

## Using a synonym for a table or view

To use a table or view through a synonym, right-click on the work area and select the *Add Table* item from the popup menu. Type the synonym name in the *Name* property, provide an alias if necessary, and press the *OK* button.

# 20.4 Query Builder Preferences

By pressing the *Preferences* button on the toolbar, you can change the behavior of the Query Builder. The following form will appear:



Most of these preferences affect the layout of the generated select statement:

- **Keyword case**  
Determines how the SQL keywords (select, from, into, where, order by) are inserted into the generated SQL statement.
- **Item case**  
Determines how the items (table names, column names) are inserted into the generated SQL statement.
- **Keyword alignment**  
Determines whether the SQL keywords are right or left aligned.

- Comma separator  
Determines whether commas are placed before the next item, or after the current item. This is primarily of interest if the *Single item per line* option is enabled.
- 'AND' separator  
Determines whether the 'AND' operator is placed before the next condition, or after the current condition in the where clause.
- Always prefix table with owner  
When enabled, table names are prefixed with the owner (e.g. scott.emp instead of emp). When disabled, tables are only prefixed with the owner if they are not owned by the current user.
- Keywords on separate line  
When enabled, a SQL keyword is placed on a separate line and all items are placed on subsequent lines with a 2 character indent. If this option is disabled, the items will immediately follow the keyword on the same line.
- Single item per line  
When enabled, each item will be placed on a new line. When disabled, items will be placed on a single line, until the *Right margin* is reached.
- Right margin  
Determines the maximum length of each line when the *Single item per line* preference is disabled.
- Always show table property editor  
When this option is enabled, the table property editor will always be displayed after adding a new table to the query definition. When this option is disabled, the property will only be displayed if the newly added table is already used in the query, and therefore needs to have an alias.
- Display SQL parts on a single page  
Determines if the 3 SQL parts (Field list, Where clause, Order By clause) are displayed on a single page. If this option is disabled, each part is displayed on a separate tab page, next to the tab page of the SQL text.

## 20.5 Query Builder Plug-Ins

If a Query Builder Plug-In is installed, then the standard *Query Builder* function will invoke the Plug-In. Right-clicking on the *Query Builder* button allows you to select a query builder from a list of all installed query builders. The most recently used query builder will be invoked by default when you subsequently use the *Query Builder* function again.

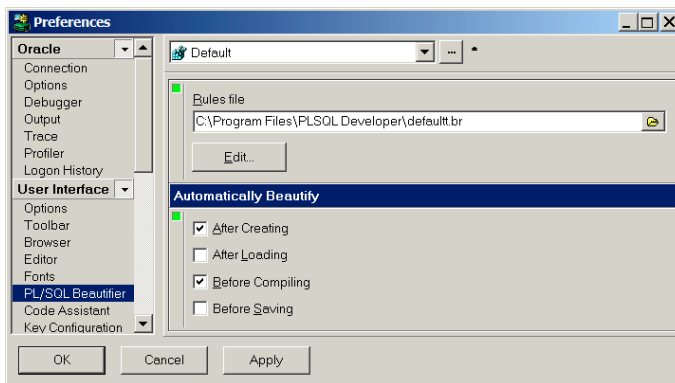
## 21. The PL/SQL Beautifier

If you develop, maintain, and support PL/SQL code in project teams with several developers, it is useful that all project members use the same coding style. This will be beneficial for both quality and productivity. To enforce this, PL/SQL Developer includes a PL/SQL Beautifier.

All you need to do is setup the PL/SQL Beautifier options, and optionally define the coding style through the PL/SQL Beautifier rules (you can also simply adopt the default rules). After that, your PL/SQL code will automatically be beautified as you work, or will be explicitly beautified by activating the beautifier function.

### 21.1 Defining the options

To define the PL/SQL Beautifier options, select the corresponding item from the *Edit* menu. The following preference dialog will appear:



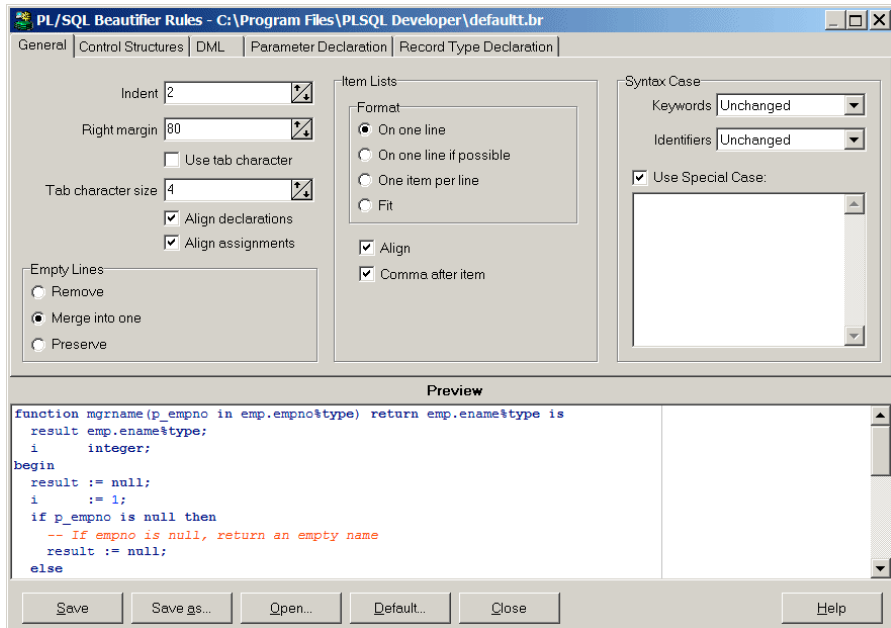
The *Rules file* determines the rules that will be followed when beautifying your code. You can leave it empty to adopt the default rules, which can be defined through a *default.br* file in the PL/SQL Developer installation directory. See the next chapter for more information about these rules.

Furthermore you can define when your PL/SQL code will be automatically beautified:

- **After Creating** – Whenever a new program file is created through a template, or a DML statement is dragged and dropped from the Object Browser into a Program Window.
- **After Loading** – After a program file is loaded from the file system. This does not affect code that is opened from the Oracle database.
- **Before Compiling** – When you compile a program in the Program Window. This ensures that all PL/SQL code in the Oracle database conforms to the rules.
- **Before Saving** – Before a program file is saved to the file system.

## 21.2 Defining the rules

If you press the *Edit* button in the Options dialog, you can view or edit the rules of the selected file:



After making changes to these rules, you can press the *Save* or *Save as* button to save these changes. You can also open another rules file for viewing or editing, and you can revert to the default rules by pressing the corresponding button. If you save a file under a new name, or open a different file, you need to select that rules file on the Options dialog to make these rules effective.

On the *General* tab page you can change the rules that apply to general aspects of your PL/SQL code. All changes you make are immediately visible in the *Preview* pane at the bottom of the dialog, and most options are self-explanatory. The following options need some explanations though:

- **Indent** – The number of spaces that will be indented for nested structures such as *begin/end*, *if/then/else*, loops, and so on.
- **Right margin** – Whenever code needs to wrap to a following line, the right margin will be used as a guideline. There may be situations where the right margin is exceeded though, for example when long strings have been used.
- **Use tab character**– When enabled, the resulting code will contain hard tabs (character 9) to indent the code. When disabled, spaces will be used.
- **Tab character size** – The number of characters that a hard tab represents. This is not only useful when the *Use tab character* option is enabled, but can also be useful to correctly align comments that contain hard tabs.
- **Empty Lines** – You can define that empty lines should be removed, and that the beautifier will insert empty lines by its own rules, or that groups of subsequent empty lines should be merged into one empty line, or that all empty lines are preserved.

- **Item Lists** – These rules apply to all item lists that do not fall into the categories of the other tab pages.
- **Syntax Case** – Determines how PL/SQL keywords and identifiers are capitalized. The *Use Special Case* option allows you to define a number of words that will be included with the capitalization as specified. If, for example, you include *DBMS\_Output* in this list, this will be the exact capitalization of that word in your PL/SQL code.

The other tab pages contain rules that apply to specific aspects of your PL/SQL code. These rules are again self-explanatory, and are demonstrated in the preview pane.

## 21.3 Using the beautifier

If you have not selected any of the beautify events on the Options dialog, you can explicitly beautify your code by selecting the *PL/SQL Beautifier* item of the *Edit* menu, by pressing the corresponding button on the toolbar, or by using a function key that is assigned to it. By default all code in the current PL/SQL editor will be beautified. If a critical syntax error is present in the code, it will not be beautified, and you will receive an error message.

You can alternatively select a complete piece of PL/SQL code (e.g. a local function or procedure, or a DML statement), and activate the beautifier.

## 22. Templates

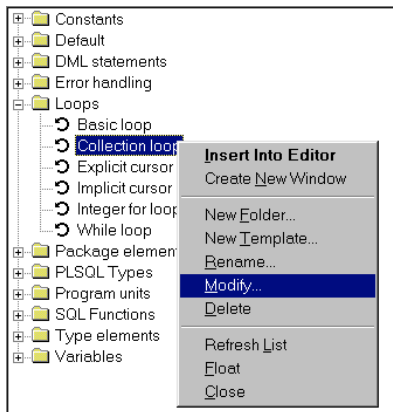
Whenever you create a new program unit, its initial contents are based upon a template which contains pre-defined text and variables. When selecting a template, you will be prompted for the values of these variables, which are then replaced in the template text.

Templates can also be used to insert pieces of SQL or PL/SQL code into existing source. You could for example create a template for a cursor-for-loop, and use it whenever you need to program such a control structure.

Templates are plain text files located in the *Template* directory and have a *.tpl* extension. Templates for all program unit types are pre-defined, as well as templates for common SQL and PL/SQL constructs. You can modify and create templates for your own specific requirements.

### 22.1 The Template Window

The Template Window displays the hierarchical structure of the template directory. This makes it easy to organize your templates. When you start PL/SQL Developer for the first time, the Template Window will be docked under the Object Browser. This way it is always available and never obstructs the view of other document windows:



There are a number of templates that are pre-defined, and they are organized in the following template folders:

- Constants – Templates for PL/SQL constant declarations.
- Default – Templates with the default contents of a SQL Window, Test Window, Command Window or Explain Plan Window. It also contains the template for a new view object. You probably won't use these templates directly, but you can maintain them here.
- DML Statements – Templates for DML statements that you can use in PL/SQL.
- Loops – Templates for various kinds of PL/SQL loops.
- Package elements – Templates for elements that can be used in package specifications or bodies.
- PLSQL Types – Templates for all types that can be declared in PL/SQL.

- **Program Units** – This is a special folder that contains templates for new program units. Whenever you create a new program unit, the template from this folder with the same name as the program unit type will be used.
- **SQL Functions** – Templates with all SQL functions, like `add_months`, `substr`, and so on. These templates can be useful if you don't know the exact name or parameter meanings of a specific function.
- **Type elements** – Templates for elements that can be used in type specifications or bodies.
- **Variables** – Templates for PL/SQL variable declarations.

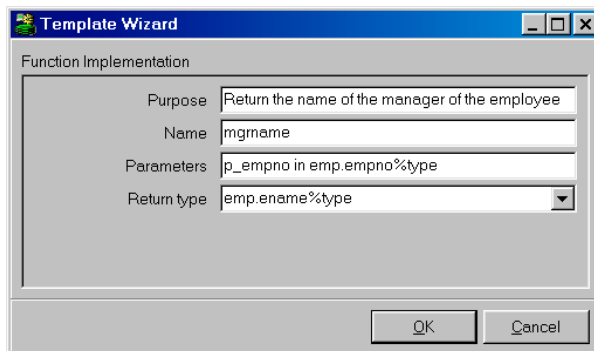
By default the Template Window is docked under the Object Browser, but you can turn it into a floating window by right clicking on it and selecting the *Float* item from the popup menu. To dock the window again, follow the same procedure.

You can close the Template Window by selecting the *Close* item in the popup menu. To make the Template Window visible again, select the *Templates* item in the *Tools* menu.

You can save this layout for the next time you start PL/SQL Developer by using the *Save Layout* item in the *Window* menu, which will save the current situation. This function is described in more detail in chapter 28.2.

## 22.2 Using a template

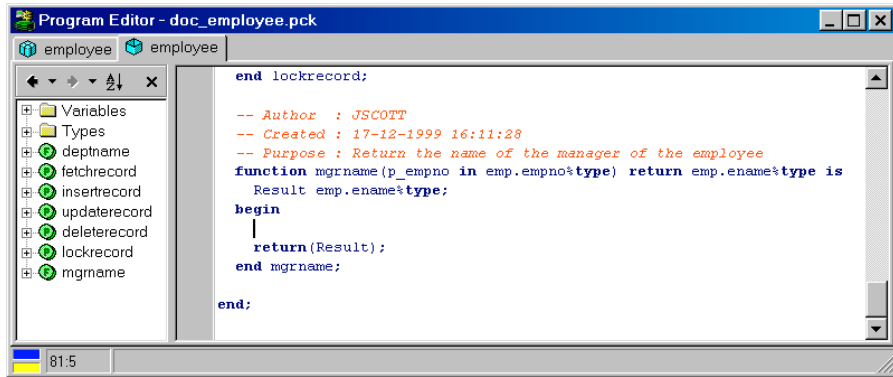
To insert a template into an existing source, you can simply double-click on it. If the template contains variables that must be specified by the user, an entry form will appear before the text is inserted. If, for example, you double-click on the *Function implementation* template, the following entry form will appear:



The screenshot shows a dialog box titled "Template Wizard" with a standard Windows window border. The main area is labeled "Function Implementation". It contains four input fields: "Purpose" with the text "Return the name of the manager of the employee", "Name" with "mgmname", "Parameters" with "p\_empno in emp.empno%type", and "Return type" with a dropdown menu showing "emp.ename%type". At the bottom right are "OK" and "Cancel" buttons.

Function Implementation	
Purpose	Return the name of the manager of the employee
Name	mgmname
Parameters	p_empno in emp.empno%type
Return type	emp.ename%type

After filling the form and pressing the *OK* button, the template text will be inserted into the source, at the current cursor position:



You can also drag-and-drop a template from the Template Window to a specific position in an editor. This position will also determine the indentation of the template text. Similarly you can right-click at a location in the editor, and select a template item from the *Templates* submenu from the popup menu.

## 22.3 Creating and modifying templates

To modify an existing template, right-click on it and select the *Modify* item from the popup menu. This will bring up a text editor with the text of the template. This text contains the complete specification of the template, including variables, queries, and so on. The format of the template text is described in the following chapter.

To create a new template, right-click on the folder in which you want to create it, and select the *New Template* item from the popup menu. You will first be prompted for the name of the template, after which the template text editor will appear. You can also create a new folder by selecting the *New Folder* item from the popup menu.

### Template text

The template specification contains literal text and variables. The literal text will simply be copied into the source file without any modification. Consider a template called *Commit* that only contains the literal text *commit;*. When this template is double-clicked, the text *commit;* is inserted into the source at the cursor position. Square brackets indicate a variable in a template (see below), so if you want to use a square bracket as literal text, use 2 brackets instead. For example: `[[option]]`

### User variables

Template variables are codes in the template text that will be replaced with a substitution text. This substitution text can be an implicit value, like the current date or the username, or can be user-specified. These user variables will appear in the entry form when the template is invoked. The following example defines the variables *Name* and *Type* for a function template:

```

create or replace function [Name] return [Type] is
begin
    return(result);
end [Name];
```



When this template is invoked, the user can specify the values for the *Name* and *Type* variables. As you can see, the *Name* variable is used twice. The user will be prompted for it only once though, and both occurrences will be replaced with the same value.

The following functionality is available for user variables:

- **Default value.**  
A default value for a variable can simply be placed after the variable name. To define varchar2 as the default function type, you can use the following declaration:  
[Type = varchar2]
- **Restricted list.**  
To restrict the possible values for a variable, you can define it as a list of values. To limit the function types to varchar2, number and date, you can use the following declaration:  
[Type = varchar2, \*number, date]  
The value preceded with an asterisk will be used as default.
- **Suggestion list.**  
To provide a user with a list of suggested values, allowing other values for the variable as well, use ... as the last value in the list:  
[Type = varchar2, number, date, ...]
- **Descriptive list**  
Instead of using the same text for the list items and the substitution text, you can also use a description for each item. Just follow the description with a colon and the value:  
[Level = Write No Database State:WNPS, Read No Database State:RNDS, ...]
- **Check box.**  
For variables that are actually options, you can use a check box. Place a slash between the values for an unchecked (left) and checked (right) check box. The following variable will insert the text *for each row* when the check box is unchecked, and will insert *for each statement* when the check box is checked:  
[Statement level? = for each row / for each statement]

If you want to use special characters (like comma's, brackets, and so on) in a substitution text, you can place the text between double quotes.

The following is an example of a template for a trigger, using just user variables:

```
create or replace trigger [Name]
  [Fires = before, after, instead of] [Event = insert, update, delete, ...]
  on [Table or view]
  [Statement level? = for each row/for each statement]
declare
  -- local variables here
begin
  ;
end [Name];
```

Note that you can create multiple program units in one template by separating them with a line with just a slash (/) character. This way you can create a templates for a package specification and body in a single program file.

When this template is used, the user will be prompted for the variable values in the following way:



## Implicit variables

Besides the user variables you can additionally use implicit variables. The substitution text for these variables are not specified by the user, but are defined by the system (date, username), or the template developer (queries, text).

### System variables

The following 4 system variables are defined:

- `$OSUSER` – The name of the Operating System user.
- `$DBUSER` – The name of the database user that is currently logged on.
- `$DATE` – The current date.
- `$TIME` – The current time.

The first 2 lines of the following example template inserts the Windows user and the current date/time into a source file:

```
-- Author   : $OSUSER
-- Created  : $DATE $TIME
-- Purpose  : [Purpose]
procedure [Name] is
begin
;
end [Name];
```

Note that system variables do not use square brackets in the template text, but are preceded by a \$ sign.

If you want to use the original name of a system variable to appear in the resulting text instead of the substitution value, you can escape this by placing a second \$ sign before the name. For example:

```
-- $$Date$ $Revision$
```

The resulting text would be `$Date$ $Revision$`.

**Cursor position**

The cursor position variable determines where the text cursor will be located in the editor after the template text is inserted into an editor. Just place [#] at the desired location:

```
loop
  [#]
end loop;
```

**Query variables**

You can use queries to populate selection lists in a template. The following example defines the query *seq\_query*, which is subsequently used as a suggestion list for the *Sequence* variable:

```
[$QUERY seq_query =
  select lower(object_name) from user_objects
  where object_type = 'SEQUENCE'
  order by object_name]
select [Sequence=$seq_query,...].nextval into [Variable name] from dual;
```

The *Sequence* variable is a suggestion list, because the query is followed by "...", implying that other values can be entered manually by the user. Query results can simply be viewed as comma separated lists of values.

**Included and excluded text**

You can include or exclude text from a template, depending on the value of another variable. In the following example, the word *where* is added if the *Search condition* is entered by the user:

```
select [Item list]
  into [Variable list]
  from [Table list]
  [+Search condition=where] [Search condition];
```

As a result, the user doesn't have to type 'where' in the search condition. To exclude a piece of text, use the *-Variable name* syntax instead of the *+Variable name* syntax.

**Text variables**

You can define text variables in a template and refer to these variables in other parts of the template. This is particularly useful if you want to conditionally include large pieces of text. The following example includes an exception block if the user indicates so:

```
[$TEXT exception_block=
exception
  when no_data_found then ...
  when too_many_rows then ...
  when others then ...
end;]
select [Item list]
  from [Table list]
  into [Variable list]
  where [Search condition];
[Exception block =/$exception_block]
```

## Template icons

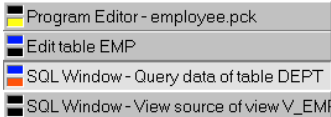
The tree view in the Template Window displays an icon for each template. The Template Window first looks for a bitmap with the same filename as the template. For template `Cursor.tpl`, it will display a `Cursor.bmp` from the same directory, if present. If this bitmap is not present, it will look for a `Cursor.bmp` in directories below the current directory. If this bitmap is not found either, it will look for a `default.bmp` file in the current directory and the directories below.

The bitmaps for the template icons should have a size of 16x16 pixels with 16 colors.

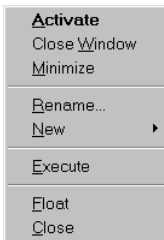
## 23. Window List

The Window List allows direct and easy navigation between multiple document windows. This feature is especially useful if you like to work with maximized document windows, but if you have a lot of windows open, such a list will always make navigation easier.

By default the Window List is docked under the Object Browser and Template Window. It is only visible if one or more windows are open, and displays the titles and status of these windows:



If you click on one of these items the corresponding window will be activated. If you right click on an item in the Window List, the following popup will be displayed:



From this popup menu you can activate, close, and resize the window to a normal or minimized state. You can also close a window by shift-clicking on an item in the list.

Furthermore you can rename a window. This can be useful if you just created a new window and want to provide a meaningful name instead of, for example, *SQL Window - New*.

The *New* menu item allows you to create a new Program Window, SQL Window, Test Window, Command Window or Explain Plan Window.

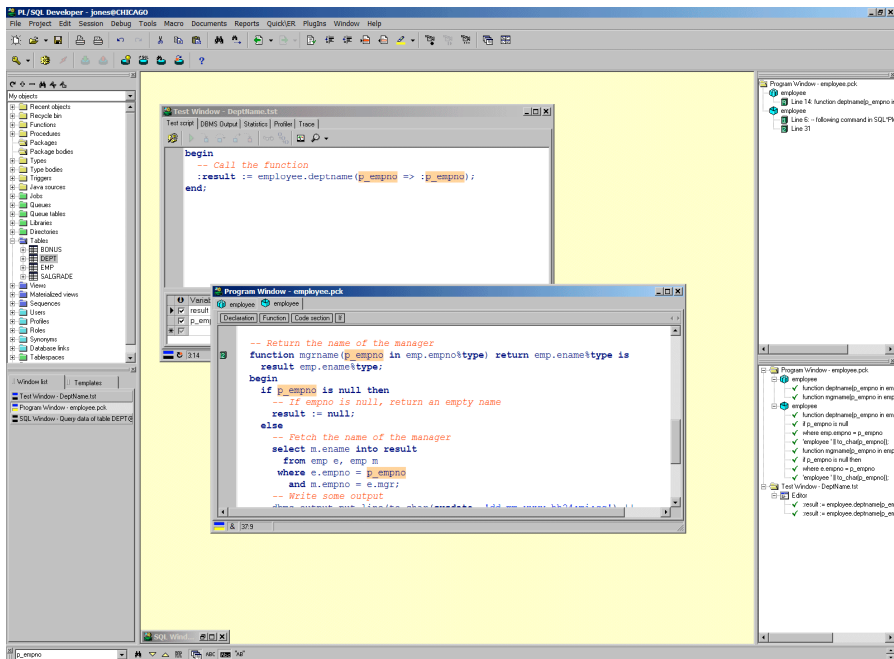
To execute a certain window, you can select the *Execute* menu item.

You can turn the Window List into a floating window by selecting the *Float* item. To dock the window again, follow the same procedure. To close the Window List, select the *Close* item. To make the Window List visible again select the *List* item in the *Window* menu. You can save this layout for the next time you start PL/SQL Developer by using the *Save Layout* item in the *Window* menu, which will save the current situation. This function is described in more detail in chapter 28.2.

## 24. Dockable and Floating Tools

The following tools can be docked or can be presented in a floating window:

- Object Browser
- Window List
- Template List
- Bookmark List
- Search Bar
- Search Results



To dock a tool, simply drag it to the left, right, top or bottom of the work area. As soon as you come close to the side of the work area, a frame will be displayed, indicating the docking position. If another tool is already docked at this location, you can drag the new tool above, below, to the left, to the right, or within the other tool. If you drag one tool within the other, they will become available at the same position on different tab pages (see the Window List and Template List in the screenshot above).

To make a tool float, simply drag it from its docking position to a floating position.

## 25. Authorization

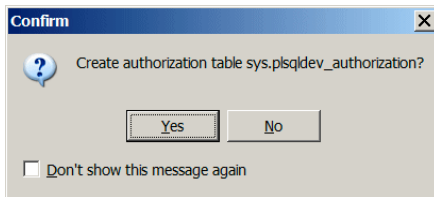
After installing PL/SQL Developer all users can use all PL/SQL Developer functionality, within the limits of the system privileges and object privileges that are granted to the Oracle user that is connected to the database. For example, if the Oracle user does not have the *create user* system privilege, the PL/SQL Developer user can start the *New user* function in PL/SQL Developer, but will eventually get an “ORA-01031, insufficient privileges” error message from Oracle.

You can explicitly authorize all relevant PL/SQL Developer functionality to specific Oracle users and roles. In a development database you will allow all developers all functionality, whereas in a production database you would typically disable all functions for most users that could alter the database or take up too much resources and would affect performance.

By granting PL/SQL Developer privileges to roles you can customize authorization for specific groups of people. You can make use of existing roles that implicitly map to a user group (such as DBA and RESOURCE) or you can create roles specifically for PL/SQL Developer user groups.

### 25.1 Enabling authorization

The authorization information is stored in the *plsqdev\_authorization* table in the *sys* schema. As long as this table does not exist in a database, all users are authorized to use all PL/SQL Developer functionality. If you are connected to a database where authorization is not yet enabled and select the *Authorization* item from the *File* menu, you will get the following question:

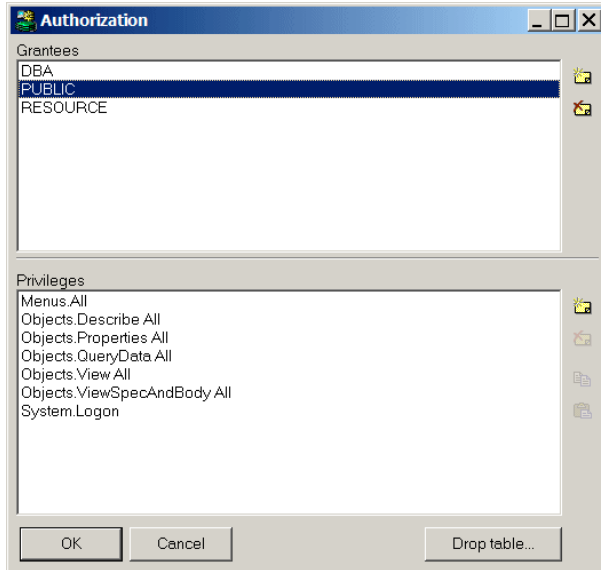


If you select *Yes*, an empty authorization table is created. Note that you must be connected as a DBA to create this table. As soon as you create one or more privileges in this table the authorization is enabled, and only privileged users can use PL/SQL Developer in this database.

By default only the owner of the table (*sys*) or another DBA can alter the table or update its contents. All users have select privileges on the table.

## 25.2 Defining authorization

To define the authorization you must be connected as a DBA and start the *Authorization* function from the *File* menu. The following dialog will appear:



At the top you will see all users and roles for which one or more PL/SQL Developer privileges have been granted. At the bottom you see the actual privileges for the currently selected grantee.

To add a grantee, press the *New grantee* button to the right of the grantee list. This will bring up a list of all potential grantees (users and roles). You can select one or more (Ctrl-click or Shift-click) grantees and press OK to add them. At this point, these new grantees do not have any privileges yet.

To remove a grantee and its privileges, press the *Remove grantee* button. This will not affect the actual user or role in the database, but will merely remove its privileges from the authorization table.

To grant privileges, select the grantee from the list and press the *Grant privileges* button to the right of the privileges list. A list of all PL/SQL Developer privileges is displayed, where you can select one or more privileges. There are 3 categories of privileges:

- **System privileges** – PL/SQL Developer system functions (e.g. Logon).
- **Object privileges** – Database object type specific privileges (e.g. Drop Table).
- **Menu privileges** – Privileges to use PL/SQL Developer menu functions (e.g. Tools > Sessions).

**Note:** If a user does not have the *System.Logon* or *All* privilege, he or she cannot use PL/SQL Developer on the current database instance.

You can select individual privileges or select from a hierarchical set of privileges. For example:

- **All** – Grant all PL/SQL Developer privileges.
- **Objects.All** – Grant all PL/SQL Developer privileges related to database objects.
- **Objects.Rename All** – Grant all PL/SQL Developer privileges related to renaming database objects (rename table, view, sequence and synonym).



- **Objects.Rename Table** – Grant the PL/SQL Developer privilege to rename a table.

Note that for database objects, the Oracle user or role must still have the necessary system privilege to perform the operation. Granting *Objects.Drop User* still requires the *drop user* system privilege.

Also note that if a user has the *Drop User* system privilege, he can still execute the *drop user* command from a SQL or PL/SQL script.

## 25.3 Disabling authorization

To permanently disable authorization, you can drop the *sys.plsqldev\_authorization* table or delete all records. To temporarily disable authorization, you can either revoke the privileges on the table or rename it (making it invisible to the PL/SQL Developer users). To enable it again later, you can grant select privileges again.

## 26. Oracle File System (OFS)

The OFS allows you to store all your files (sources, scripts, reports, and so on) in the Oracle database. This has the benefit that both the database objects as the files that operate on these objects can be stored in the same database, and can be accessed by all users that have access to this database. Furthermore, it is guaranteed that the database objects and files are consistent, and are backed up and recovered in a consistent way.

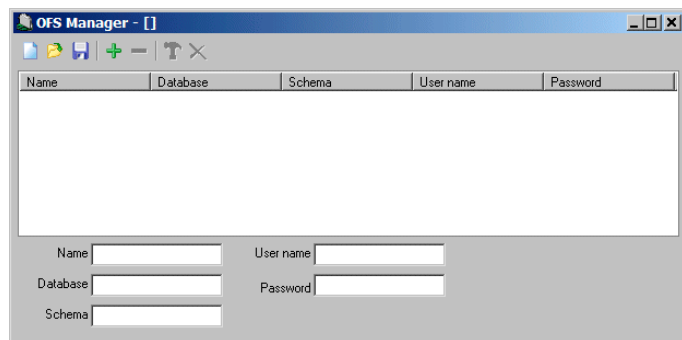
Files can be saved and opened in the usual way, with a standard file open/save dialog with some extensions.

### 26.1 OFS Manager

Before anybody can use the OFS from within PL/SQL Developer, you have to use the OFS Manager to define an OFS Location Directory with one or more OFS Locations. For each location you need to install the OFS database objects.

#### Creating an OFS Location Directory

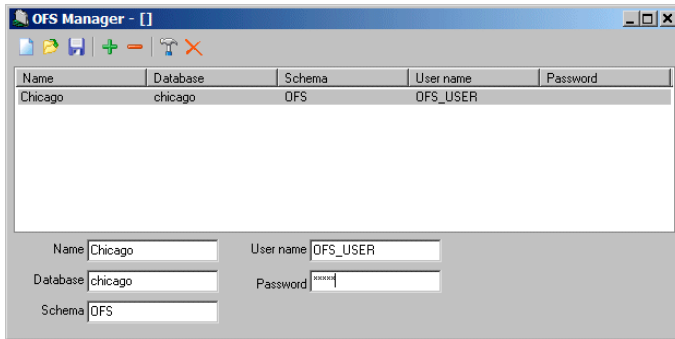
To start the OFS Manager, go to PL/SQL Developer's *Preferences* and select the *Directories* page. At the bottom of this page you can find the OFS Manager button. After starting it you get the following screen:



This is an empty OFS Location Directory, for which you can add locations, and which you can save in an OFS Location Directory file (.ldf). This file is used from within PL/SQL Developer to browse the directory and to connect to locations and browse the files.

## Adding OFS Locations

To add an OFS Location, press the *Add location* (+) button. Now you can enter the following information:



Name	Database	Schema	User name	Password
Chicago	chicago	OFS	OFS_USER	

Name:       User name:

Database:       Password:

Schema:

- Name – Will be displayed in the location list in the file selector.
- Database – The database where the OFS files will be stored.
- Schema – The schema that holds the OFS database objects (tables, packages, and so on). This user requires the *resource* role privilege and the *query rewrite* system privilege to create its objects.
- User name – The name of the Oracle user that will be used to access the OFS. If you do not specify an OFS user name, then the user that accesses the OFS will need to specify a user name and password, or the OFS will be accessed as the current PL/SQL Developer Oracle user.
- Password – The password of the Oracle user that will be used to access the OFS. Only useful if the *User name* field is entered.

After creating one or more locations, you can save the OFS Location Directory by pressing the *Save* button. If you save the file in the PL/SQL Developer directory under the name *OFS.ldf*, then PL/SQL Developer will automatically pick-up this directory. Otherwise the PL/SQL Developer user will need to point to the correct OFS Location Directory file in the preferences.

## Installing the OFS database objects

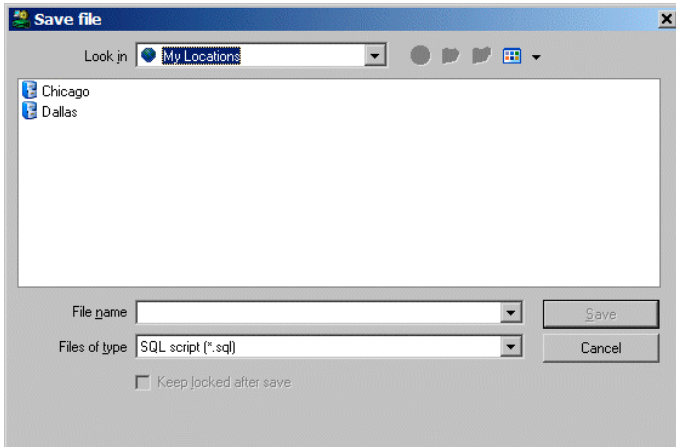
After adding a location to the directory, you need to install its database objects. If the OFS schema and (optional) OFS user do not yet exist in the database, you will first need to create them:

```
create user OFS identified by <password>;
grant connect, resource, query rewrite to OFS;
create user OFS_USER identified by <password>;
grant connect to OFS_USER;
```

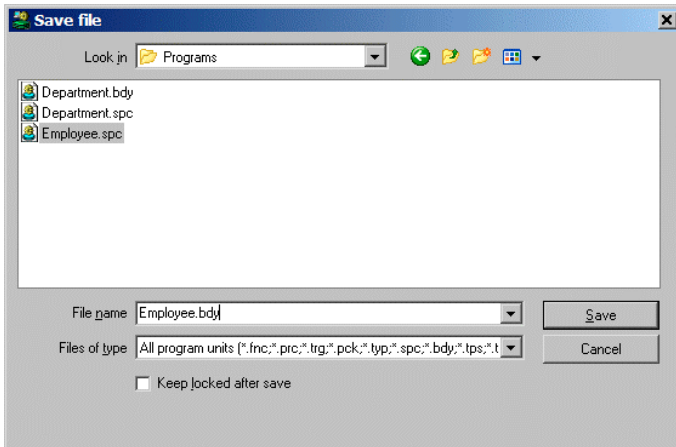
Next you can install the OFS database objects by pressing the *Install database objects* button. After supplying the password of the schema user, the database objects will be created, and the location is ready for use.

## 26.2 OFS Usage

After creating an OFS Location Directory, installing the locations, and making the directory file available to PL/SQL Developer, you can use the *OFS Save As* item from the *File* menu to save a file into the OFS. The following dialog will appear:



You first need to open a location, after which you can save the file or create a directory:



To open a file from the OFS, you can use the *OFS Open* item from the *File* menu, or use the *Reopen* menu. OFS files have the path name OFS:\Location\Path (e.g. OFS:\Chicago\Programs\Employee.bdy). A previously opened OFS file can be saved by using the standard *Save* function. You can use the *Save As* function to save an OFS file to the standard file system. The *OFS Save As* function can be used to save it in the OFS under a different name or in a different directory.

When saving or opening a file, you can select the *Keep locked* option to lock it. Other users cannot lock or overwrite a file that is locked by you.

Right-clicking on a file in the file selector gives you the option to Cut, Copy, Paste, Delete or Rename the file. You can also view and change object properties from this popup menu. Properties you can change include the Read-only status, the Locked status, and the Compressed status. Files can only be compressed on Oracle10g and later, since it uses the *utl\_compress* package which is only available since 10g.

## 27. Help systems

You can configure PL/SQL Developer to integrate with the help files and manuals that are provided with the Oracle Server software. This chapter describes how to setup and use these help systems.

### 27.1 MS Help files

Up to Oracle 7.2 the online manuals were provided in MS Help format. These help files can be used to provide context sensitive help in the editor, either by right clicking on a word, by selecting the *SQL help* or *PL/SQL help* items in the *Help* menu, or by pressing a function key that you have associated with these two menu items.

PL/SQL Developer searches the help files in the following two directories:

1. In the directory where PL/SQL Developer is located.
2. In the directory where SQL\*Plus is located.

The SQL help file name is assumed to be *sqlhelp.hlp*, the PL/SQL help file is *plshelp.hlp*. If these help files are not available, but you have some other help files you wish to use for SQL or PL/SQL help, you can place these files in the PL/SQL Developer directory and rename them as mentioned above.

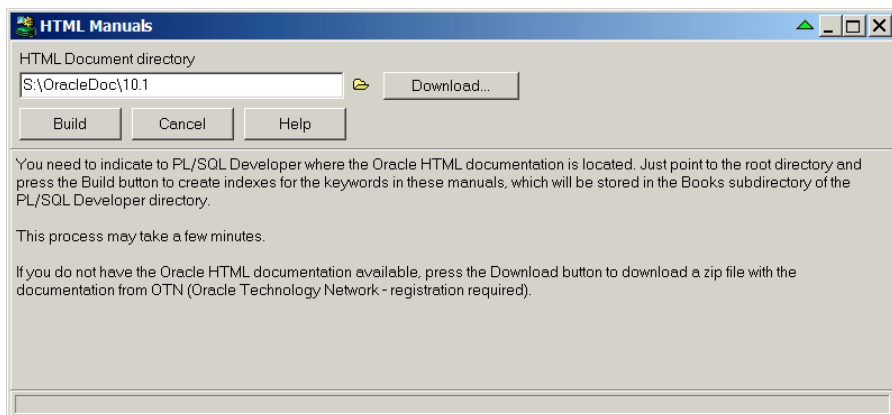
### 27.2 HTML Manuals

Since Oracle 7.3 the online manuals have been provided in Acrobat Reader and HTML format. You can configure PL/SQL Developer to make use of the HTML manuals, either by right clicking on a word in an editor, by selecting the *HTML Manuals* item in the *Help* menu, or by pressing *F1*. Furthermore you can press the *Help* button on an error message in case of an Oracle exception, which will automatically take you to the appropriate paragraph in the *Oracle Error Messages* manual. Double-clicking on a compilation error in a Program Window will also display this information.

**Note:** the online documentation of some Oracle Server versions contains HTML files that are over 1MB in size. Loading these files can take a very long time, in which case you are advised to use the online documentation of an Oracle Server version where the chapters are divided into smaller files. Oracle has reduced file sizes again for the 8.1.5 documentation.

#### Configuring the HTML Manuals

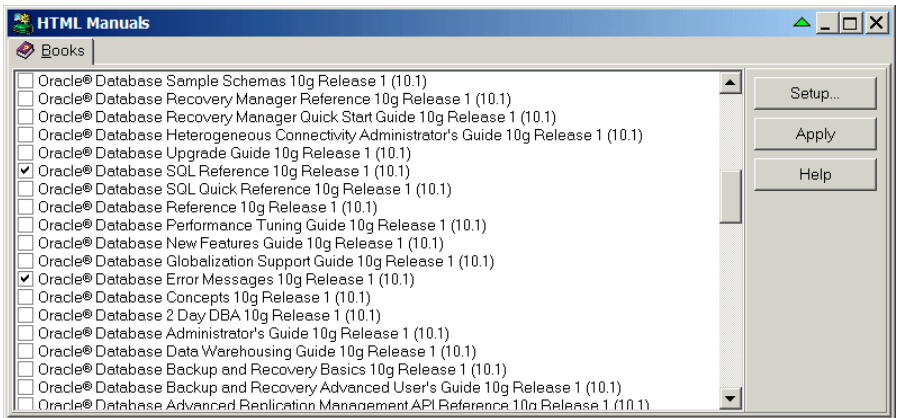
The first time you invoke the HTML manuals in one of the ways described above, you need to indicate to PL/SQL Developer where your HTML files are located:



Just point to the root directory and press the *Build* button to create indexes for the keywords in these manuals, which will be stored in the *Books* subdirectory of the PL/SQL Developer directory. This process may take a few minutes.

If you do not have the Oracle HTML documentation available, you can press the *Download* button. This will take you to a local html page where you can select the appropriate Oracle Server version and download a zip file with the HTML documentation. This download requires an Oracle Technology Network account. If you do not have an OTN account, you can first create one for free.

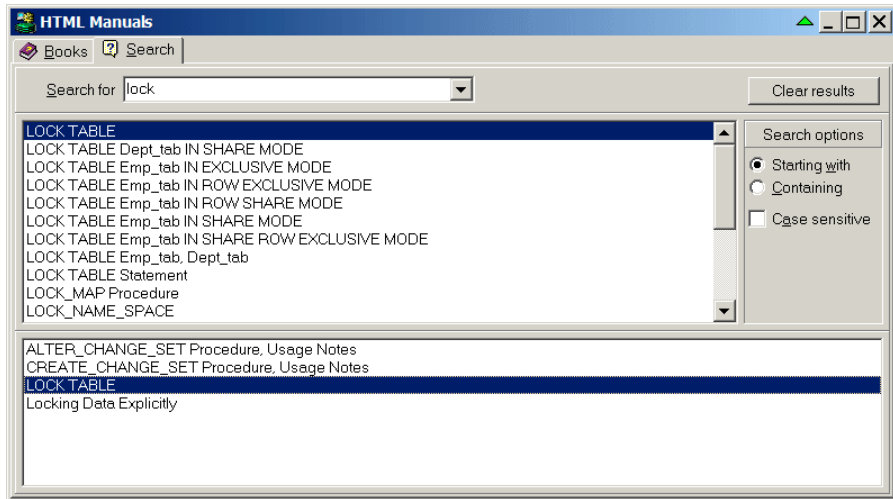
After this index build is finished, you will be presented with a list of manuals that have been found. You can now select the books you want to use for context sensitive help. By default this selection will include the *Error Messages*, *SQL Reference* and *PL/SQL Reference* manuals, which all contain relevant information during PL/SQL development. The *Error Messages* manual is required if you want to automatically have the *cause* and *action* available in case of an Oracle exception or compilation error. The current Oracle7 manual is not properly indexed to allow this feature, the Oracle8 version works without a problem though.



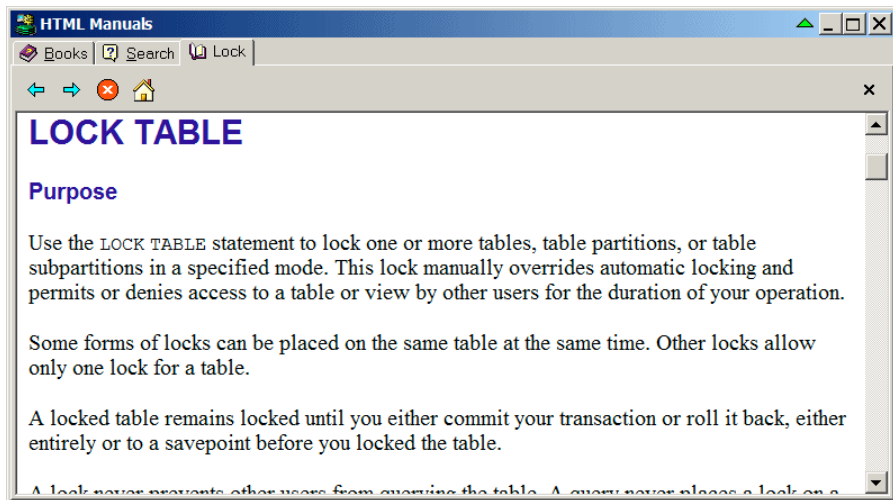
Press the *Apply* button to make the selection effective. After this you are taken to the search page, which is discussed in the following chapter. By pressing the *Setup* button you can scan a different directory for HTML manuals.

## Using the HTML manuals

After invoking the HTML manuals you can type the keyword you want to search for on the *Search* page. The two lists display the keywords that match this search and the topics in the manuals in which this keyword appears:



In the *Search options* pane to the right of the keyword list you can refine the search criterion. Next you can double click on an item in the topic list at the bottom of the search page to display the corresponding paragraph in the HTML manual:



The search result page displays an HTML page in which you can navigate through hyperlinks as usual. The buttons at the top of the page can be used to navigate to the previous or next location that you visited before, to cancel the page buildup, or to go to the original page after navigating.



You can go back to the *Search* page to search for different keywords or go to the *Books* page to select different books. Each search creates a new result tab page, so that you can keep different search results available. To close a result page, press the close button at the upper right of the tab page. To close all pages, press the *Clear results* button on the search page.

By default the HTML window will stay on top of other windows in the PL/SQL Developer IDE, so that the information stays visible. To temporarily hide the window, you can use the green rollup button at the upper right of the window.

## 28. Customization

Some aspects of PL/SQL Developer can be customized to meet your personal needs. These aspects are described in the following paragraphs.

### 28.1 Preferences

In the *Tools* menu, there is a *Preferences* item that allows you to set various preferences for PL/SQL Developer. These preferences are described in detail in chapter 16.

### 28.2 Window layout

There are various window layout settings that can be tailored to your needs. Selecting the *Save Layout* item in the *Window* menu saves these settings. The Layout Settings include:

- Size and position of the application window.
- Presence and size of the Object Browser.
- Presence and size of the Code Contents in the Program Window.
- Presence, status (floating or docked) and position of the Window List and Template Window.
- Size of the Program Window, Test Window, SQL Window, Explain Plan Window, Command Window, Report Window, Compile Invalid Objects Window, Find Database Objects Window, Export User Objects Window, and Compare User Objects Window.

When you save the layout, the size of the last opened window is saved. If a window of a certain type is not present, the size setting of that window type is left unchanged.

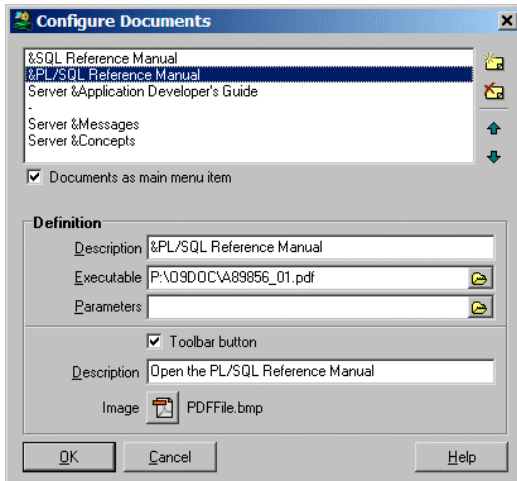
### 28.3 On-line documentation

Most of Oracle's documentation is provided as on-line documents. Through time, these documents have had several formats. So far there have been MS Help files, Oracle Book files, Adobe PDF files and HTML files. In the future, Oracle may choose to use yet another format.

Moreover, you may have some corporate standards and project documents in a word-processor format that you need to access from time to time.

All of these on-line documents can be integrated into PL/SQL Developer's IDE by including them in the Documents menu. This way they are available to you by a simple mouse click.

To configure on-line documents, select the *Configure Documents* item in the *Tools* menu. The following dialog allows you to define a description of each document, and how to view it:



At the right side you see four buttons, to create a new document, to delete a document, and to move a document up or down in the menu. When you create or modify a document you must supply the following information:

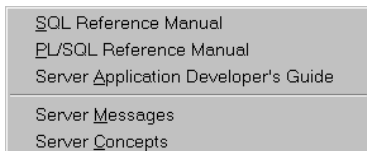
The Description shows up in the menu. To create a shortcut key for the menu item of a document, prefix the letter in the description with an ampersand. If you want an ampersand in the description, use two ampersands instead. If you enter just a '-' into the description, it will become a separator in the menu. This way you can divide the documents into logical groups.

The Executable should be the full path to the program that you want to use to view the document. If the document is a registered file type (like an MS Help file or HTML file), you can enter the path to the document here.

The Parameters can be used to pass information to the executable if it is a program. It should at least include the path to the document, but can also be used to pass options to the program.

When the *Documents as main menu item* option is checked, the *Documents* menu is located in the main menu. If it is not checked, it will be located under the *Tools* menu.

In the above example the PL/SQL Reference Manual is a PDF file (P:\O9DOC\A89856\_01.pdf), which is a registered file type, so that can just enter the file name in the Executable field. The letter 'P' is used as a shortcut in the menu. The Documents menu would look like this:



When the *Toolbar button* option is enabled, the document can additionally be included on the toolbar. The corresponding *Description* will be displayed as a hint when you hold the mouse cursor over the toolbar button. If you leave this description empty, the main description of the document (as displayed in the menu) will be used. Press the *Image* button to select a Windows bitmap file (\*.bmp) for the

toolbar button. The size of this image should preferably be 20 x 20 pixels. Note that PL/SQL Developer will always load the bitmap file from the original location, so you should not remove or rename this file without also changing the corresponding document. PL/SQL Developer comes with a number of standard bitmap files that you can choose from. These files are located in the Icons subdirectory in the PL/SQL Developer installation directory. This is the default directory of the image selector.

## 28.4 Command-line parameters

The following parameters can be used on the PL/SQL Developer command-line. The default shortcut does not include any parameters, but merely starts `plsqldev.exe` from the PL/SQL Developer installation directory.

A default set of parameters can be defined through the *params.ini* file in the PL/SQL Developer installation directory. You can edit the file in a text editor like notepad. The file contains an explanation for each parameter. If a parameter is defined in *params.ini* and on the command-line, then the command-line takes precedence.

### userid

Every time you start PL/SQL Developer, it will prompt you for at least the password. To avoid this, you can supply a userid parameter with the familiar username/password@database format:

```
plsqldev.exe userid=scott/tiger@chicago
```

Note that you can also use the following registry key to supply a default logon:

HKEY\_CURRENT\_USER\Software\Allround Automations\PL/SQL Developer\Logon

Here you can add a Username, Password and Database.

The last method has the advantage that it enables you to be automatically logged on after double clicking a PL/SQL Developer registered file.

### nologon

The nologon parameter suppresses the logon dialog that is displayed when PL/SQL Developer is started. This parameter does not require any arguments:

```
plsqldev.exe nologon
```

### oraclehome

You can specify the name of the Oracle Home on the command line, thereby overriding the Primary Oracle Home (which is used by default) and the Oracle Home preference:

```
plsqldev.exe oraclehome=ora817
```

Note that this is the name of the Oracle Home, as specified in the ORACLE\_HOME\_NAME key in the registry. This name is also used by the Oracle Home Selector. It is not a registry section name or a directory name.

### dontadjustpath

PL/SQL Developer will not temporarily modify the PATH for Oracle Net initialization:

```
plsqldev.exe dontadjustpath
```

**nosplash**

The nosplash parameter suppresses the splash screen that is displayed when PL/SQL Developer is started. This parameter does not require any arguments:

```
plsqldev.exe nosplash
```

**noplugins**

Specifying the noplugins parameter will prevent that any Plug-Ins are loaded:

```
plsqldev.exe noplugins
```

**library**

Specify the location of the style library for reports:

```
plsqldev.exe library=p:\standard.lib
```

**prefpath**

Specify the path for the personal preference sets. For example:

```
plsqldev.exe prefpath=u:\userdata
```

See chapter 16.33 for details.

**commandfile**

Run the specified command file in the Command Window. For example:

```
plsqldev.exe userid=scott/tiger commandfile="u:\sql scripts\demo_build.sql"
```

This command will first connect to the database and subsequently run *demo\_build.sql*.

**viewobject**

View the specified database object. It requires that you also specify the *userid* parameter. For example:

```
plsqldev.exe userid=scott/tiger@chicago viewobject=emp
```

This command will first connect to the database, and subsequently show the table definition of table *EMP*. Note that the object specification can also include the owner (e.g. *SCOTT.EMP*).

**editobject**

Edit the specified database object. It requires that you also specify the *userid* parameter. For example:

```
plsqldev.exe userid=scott/tiger@chicago editobject=emp
```

This command will first connect to the database, and subsequently show the table definition of table *EMP*. Note that the object specification can also include the owner (e.g. *SCOTT.EMP*).

**registry**

To use different sets of IDE settings you can use the registry parameter. For example:

```
plsqldev.exe registry=home
```

This command will cause all IDE settings (window sizes & positions, file history, logon history, and so on) to be loaded from and stored under a *home* name.

## 28.5 SQL, PL/SQL, Command, Java and XML keywords

The keywords that are highlighted in SQL, PL/SQL and Command editor can be defined in keyword files. In the directory where PL/SQL Developer is installed, you can find the text files *sql.kwf*, *plsql.kwf*, *command.kwf*, *java.kwf* and *xml.kwf*. You can simply change, add or remove keywords in the appropriate sections with a text editor like notepad.

## 28.6 Plug-Ins

You can program your own Plug-Ins to extend the functionality of PL/SQL Developer. Functions in a Plug-In can be added to the PL/SQL Developer menu, and can perform any kind of action: access the database, the Object Browser, the current window and editor and so on.

A Plug-In is a DLL that exposes a specific interface to PL/SQL Developer. To create a Plug-In, you can use any programming language that can create DLL's. When a Plug-In DLL is placed in PL/SQL Developer's Plug-In directory, it will automatically be picked up. This makes it very easy to distribute your Plug-Ins.

You can build Plug-Ins for your own or public use. Some standard Plug-Ins are available on the Allround Automations web site (<http://www.allroundautomations.com/plsqldev.html>). Third party Plug-Ins are available here as well.

Plug-In documentation and examples are available in the *PluginDoc* subdirectory in the PL/SQL Developer root directory.