

# Sending Email from within Oracle's Database

*Gregory Williams*

As you know, email is a way of life for most people. Oracle has provided a mechanism to allow users to send email directly from the database. In this article, Gregory Williams will explain how to setup and send email in the database using two Oracle-supplied PL/SQL packages, UTL\_SMTP and UTL\_MAIL.

The first package that I will show you, is called UTL\_SMTP, which was released in version 8i. The second package is new to 10g and is called UTL\_MAIL. This package is so much easier to use than UTL\_SMTP, that you may want to skip the section on UTL\_SMTP and go straight to the section titled "The Installation and Setup for UTL\_MAIL" if you are using Oracle 10g. Finally, I will show you examples of both packages.

## The Installation and Setup for UTL\_SMTP

UTL\_SMTP is installed when catproc.sql is executed. Since catproc.sql is executed when the database is created, UTL\_SMTP is ready to be used when the database is started; no installation is needed. To use UTL\_SMTP to generate email, there are several steps that must be taken, using procedures within the package. In order to reuse the code that generates email, I created a package called Mail (see Listing 2). Before an email can be sent, you must establish a connection to the SMTP server. To establish a connection, you need the server's ip-address or name and the port number. Instead of hard-coding the SMTP server ip-address/name and port number in a package, I stored this information in a table called system\_parameters (see Listing 1). In listing 1, I create and insert data into the system\_parameters table. The reason I decided not to hard-code the ip-address/name and port number is because either one can change. In the 2 years that I have been running this package, the ip-address of the SMTP server changed 3 times. Therefore, it is easier to change the table values then it is to change the package.

*Listing 1. Create and Insert Data in Table system\_parameters.*

```
CREATE TABLE SYSTEM_PARAMETERS
(
  ID      NUMBER NOT NULL,
  NAME    VARCHAR2(40 BYTE) NOT NULL,
  VALUE   VARCHAR2(2000 BYTE) NOT NULL
);

Insert into SYSTEM_PARAMETERS
  (ID, NAME, VALUE)
Values
  (1, 'smtp_server', '<ip-address>');

Insert into SYSTEM_PARAMETERS
  (ID, NAME, VALUE)
Values
  (2, 'smtp_server_port', '25');
COMMIT;
```

In listing 2, there are 6 UTL\_SMTP procedures or functions used:

UTL\_SMTP.OPEN\_CONNECTION() – This function is used to establish connection with the SMTP server.

The parameters for this function are the SMTP server ip-address/name and port number.

UTL\_SMTP.HELO() – This procedure performs the initial handshake with the SMTP server.

UTL\_SMTP.MAIL() - This procedure initiates a mail transaction with the server. This is the From mail ID.

UTL\_SMTP.RCPT() – This procedure sends email to recipients.

UTL\_SMTP.DATA() – The body is assembled through this function. Note: The recipient must be added in the body.

UTL\_SMTP.QUIT() – This function closes the SMTP connection.

*Listing 2. Package for Sending Email*

```
CREATE OR REPLACE PACKAGE mail IS
  g_smtp_server      varchar2(2000);
  g_smtp_server_port pls_integer;
```

```

FUNCTION get_sys_parameter (p_name varchar2)
    Return varchar2;
PROCEDURE send (p_sender varchar2,
    p_recipient varchar2,
    p_message varchar2,
    p_subject varchar2
    );
END mail;
/
CREATE OR REPLACE PACKAGE BODY Mail AS

FUNCTION get_sys_parameter ( p_name VARCHAR2)
    RETURN VARCHAR2 IS
    v_return VARCHAR2(2000);

BEGIN
    SELECT      VALUE
    INTO        v_return
    FROM        SYSTEM_PARAMETERS
    WHERE       NAME = p_name;

    RETURN v_return;

EXCEPTION
    WHEN OTHERS THEN
        NULL;
END get_sys_parameter;

PROCEDURE send ( p_sender VARCHAR2,
    p_recipient VARCHAR2,
    p_message VARCHAR2,
    p_subject VARCHAR2
    )
    IS
    mail_conn utl_smtp.connection;
BEGIN

    mail_conn :=
        utl_smtp.open_connection
            ( g_smtp_server,
              g_smtp_server_port
            );
    utl_smtp.helo( mail_conn,
        g_smtp_server
        );
    utl_smtp.Mail( mail_conn,
        p_sender
        );
    utl_smtp.rcpt( mail_conn,
        p_recipient
        );
    utl_smtp.DATA( mail_conn,
        'Subject: ' || p_subject ||
        CHR (13) ||
        CHR (10) ||
        'To: ' ||
        p_recipient ||
        CHR (13) ||
        CHR (10) ||
        p_message ||
        CHR (13) ||
        CHR (10)
        );
    utl_smtp.quit (mail_conn);

END send;

BEGIN

    g_smtp_server := get_sys_parameter
        ( 'smtp_server' );

    IF g_smtp_server IS NULL THEN
        RAISE_APPLICATION_ERROR
            (
                -20031, 'Could not find system ' ||
                ' parameter smtp_server in ' ||

```

```

        'table system_parameters.'
    );
END IF;
g_smtp_server_port :=
    CAST(get_sys_parameter( 'smtp_server_port' )
        AS NUMBER);
IF g_smtp_server_port IS NULL THEN
    RAISE_APPLICATION_ERROR(-20031,
        'Could not find system '||
        'parameter smtp_server_port '||
        'in table system_parameters.'
    );
END IF;
END Mail;
/

```

There are several more functions/procedures in the package UTL\_SMTP. These six are the basic functions that are needed to send email from the database.

### The Installation and Setup for UTL\_MAIL

UTL\_MAIL is not installed when the database is installed because the SMTP\_OUT\_SERVER must be configured. Listing 3 shows how to install UTL\_MAIL and the results from the script. You must connect to the database as user SYS and run the two scripts identified in the listing.

#### *Listing 3. Installation of UTL\_MAIL.*

```

SQL> connect sys/password as sysdba
Connected.

SQL> @$ORACLE_HOME/rdbms/admin/utlmail.sql

Package created.

Synonym created.

SQL> @$ORACLE_HOME /rdbms/admin/prvtmail.plb

Package body created.

No errors.

```

Next the SMTP\_OUT\_SERVER must be configured. You must connect to SYS then use the alter system command to configure SMTP\_OUT\_SERVER as below:

```

SQL> alter system set smtp_out_server =
    '<ip-address:port' scope=Both;

System altered.

```

That's it! The installation and setup for UTL\_MAIL is complete. The setup for UTL\_MAIL is a lot simpler then setting up UTL\_SMTP. The UTL\_MAIL package only has 1 procedure called Send, for sending email (see Listing 4) for the syntax for the send procedure), compared to the UTL\_SMTP package which has 6 require procedures and many more procedures that are not required (see Listing 2 for the required procedures). This package bundles the email message and sends it to the UTL\_SMTP package and then the email message is sent to the SMTP server.

#### *Listing 4 Syntax for UTL\_MAIL.send*

```

UTL_MAIL.SEND (
    sender      IN VARCHAR2,
    recipients  IN VARCHAR2,
    cc          IN VARCHAR2 DEFAULT NULL,
    bcc         IN VARCHAR2 DEFAULT NULL,
    subject     IN VARCHAR2 DEFAULT NULL,
    message     IN VARCHAR2,
    mime_type   IN VARCHAR2 DEFAULT
        'text/plain; charset=us-ascii',
    priority    IN PLS_INTEGER DEFAULT NULL);

```

Since this package only has 1 procedure, you do not need to create a package as I did for UTL\_SMTP (see Listing 2). Also, there is no need to create a table to store the SMTP server name or ip-address and the port number as was needed in UTL\_SMTP (see Listing 1), because the ip-address and port number are stored in the init parameter SMTP\_OUT\_SERVER. Therefore, the setup of UTL\_MAIL is complete and ready for use.

## Examples

I have created two database triggers. Both of them will send an email just before the database is shut down. The first database trigger will send an email using UTL\_SMTP (see Listing 5). The second database trigger will send an email using UTL\_MAIL (see Listing 6). Notice that, because I have created a package called mail, the call to the email server is very similar in both database triggers.

### Listing 5. Database Shutdown using UTL\_SMTP

```
CREATE OR REPLACE TRIGGER db_smtp_shutdown
before shutdown on database
begin
    mail.send(
        p_sender => 'williams.greg@dol.gov',
        p_recipient => 'williams.greg@dol.gov',
        p_subject => 'Testing utl_smtp',
        p_message => 'The receipt of this email means'||
            ' that shutting down the database'||
            ' works for UTL_SMTP '
    );
end;
/
```

### Listing 6. Database Shutdown using UTL\_MAIL

```
CREATE OR REPLACE TRIGGER SCOTT.db_shutdown
before shutdown on database
begin
    utl_mail.send(
        sender => 'gjwilliams01@yahoo.com',
        recipients => ' gjwilliams01@yahoo.com',
        subject => 'Testing utl_mail',
        message => 'The receipt of this email means'||
            ' that shutting down the database'||
            ' works for UTL_MAIL '
    );
end;
/
```

## UTL\_SMTP vs. UTL\_MAIL Recap

Let's recap. UTL\_SMTP is installed when the database is installed. Two scripts, utlmail.sql and prvtmail.plb must be run to install UTL\_MAIL, thus I give the installation advantage to UTL\_SMTP; although, it's a very small advantage. To send an email message, a developer has to execute at least 6 procedures with the package UTL\_SMTP. Instead of calling those 6 procedures, a user package may be created that will perform that task for you. This user package may be reused over and over again. The package UTL\_MAIL requires you to execute only 1 procedure in order to send an email message. Therefore, no extra user packages will be required, so the advantage here is UTL\_MAIL. In addition, there are 2 other procedures within the UTL\_MAIL package that makes life easy: 1) UTL\_MAIL.ATTACH\_VARCHAR2; and 2) UTL\_MAIL.ATTACH\_RAW. These procedures are for varchar2 and raw attachments respectively.

*Gregory Williams is an Oracle9i Database Administrator Certified Professional. He has over 20 years of IT experience. He has worked with Oracle products for over 15 years. Gregory as worked on multiple projects as an Oracle developer, DBA, and MIS Director. He lives in the Washington D. C. area. gjwilliams01@yahoo.com.*

TOAD has to be one of the most popular Oracle development tools on the market today. TOAD continues to enjoy more than 250,000 users, with more than 25,000 monthly downloads of the freeware version of the tool. Dan Hotka highlights some features of the product and discusses why it's become so widely used.

Some of the information in this article is taken from my latest book, *Oracle9i Development by Example*, from Que. (All rights reserved.) Many thanks to Allen Sofley of Quest Software for providing much of the detail for this article.

TOAD began with one window, the SQL Editor. Jim McDaniel developed it to improve the efficiency of developing and testing code. The alternative at the time was using SQL\*Plus only, or using SQL\*Plus in conjunction with a text editor. Both methods were cumbersome. So the SQL Editor was developed. It's still the window where developers spend most of their time.

Quest Software purchased TOAD in November 1998, when TOAD was already a full-featured product. Additional developers were added to the project. TOAD continues to

TOAD continues to grow with new features and windows in each release. Many TOAD users will discover useful time-savers and features, sometimes by accident. This article covers some of the more compelling time-savers that some TOAD users might not know about.

When TOAD starts up, it puts you in the SQL Editor window (see Figure 1). The SQL Editor is the heart of TOAD, and it lets you type, edit, recall, execute, and tune. The window contains an editor to compose SQL statements or scripts and a results grid to display the results from SELECT SQL statements. A horizontal splitter between the editor and the results grid lets you size each panel accordingly.

The Procedure Editor (second button, upper left) lets you create or modify procedures, functions, packages, triggers, types, and type bodies. Right-clicking on an object brings up many options, as shown in Figure 2. Click

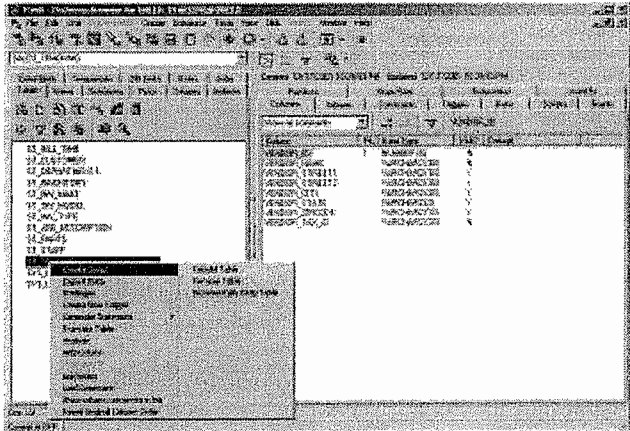


Figure 2. The TOAD Procedure Editor.

around on the tabs on the right-hand part of the screen. I particularly like the Scripts tab. Being a developer, I like to see all the code that's associated with this table in one place. Any SQL procedure, function, table, and code can be created or changed from this interface. Click on the Modify Table button (third from the left of the top row of buttons), and you'll see how easy it is to make object changes, for example.

### TOAD tips and techniques

The remainder of this article will illustrate many of the features that make the TOAD tool so popular.

F4 is a favorite among developers. You place the cursor on the name of a table, procedure, function, package, or view while in the Procedure Editor or SQL Editor, press the F4 key, and get a pop-up Object Describe window that describes the object.

For example, while in the SQL Editor, highlight the function or simply leave the mouse over it and press F4 to display a function pop-up Describe window that shows details including the type of argument it is, the code for the argument, and the grants (see Figure 3). You could select and copy the code into your own script. This works equally well for table names to get a description of the object.

*TOAD Tip!* Here's a lesser-known F4 feature. While in the Schema Browser | Tables | Referential tab, you can select an item from the Referential list, press F4, and get a Describe window for the object.

### Ctrl-Enter or F9 (execute current SQL statement at cursor)

TOAD users can choose Ctrl-Enter or F9 to execute the current SQL statement at the cursor. For example, in the SQL Editor, if your table name is "dept" and you type the following and press Ctrl-Enter, the results panel will display the EMP data (and tabs for the Explain Plan, Auto Trace, DBMS Output, and Script Output), as illustrated in Figure 1:

```
select ename from scott.emp
```

In order to display the Row ID column in the Data tab, you must have the View | Options | Data Grids—Data |

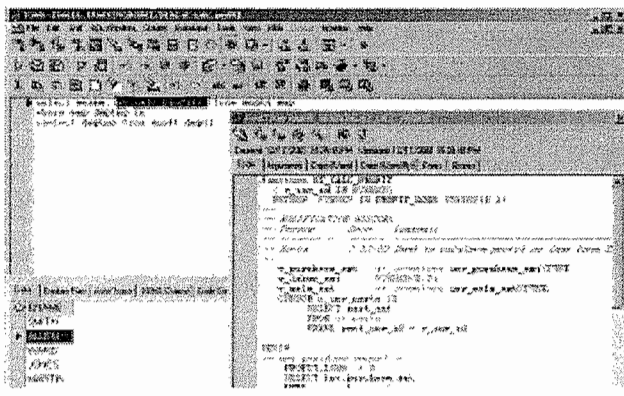


Figure 3. The TOAD F4 feature.

Show ROWID in Data Grids option checked.

### Ctrl-T (display columns drop-down)

Put the cursor on the table name and press Ctrl-T, and a pop-up window lists the columns in that table. You can also type the table name, press the period, and wait a few seconds. A drop-down list of columns for that table will display. So you can easily look up table columns as you construct your query.

### Look up table columns while constructing a query

This applies to any table or view that you can access:

1. After you type the table name (or view name) and the period, press Ctrl-T or wait a few seconds. A list of columns displays.
2. Click the column you wish to select. (Hold down the Ctrl key to select multiple columns.) Press Enter.

TOAD places the selected column(s) into the SQL Editor in your query.

After a query populates the SQL Results Grid, you can press Ctrl-T to display a list of the columns from the SQL Results Grid.

### Auto-replace substitution

TOAD can automatically replace text with pre-defined substitution text as you type. In fact, it comes with a few auto-replace substitutions. (A substitution is a text phrase that corresponds to replacement text.) If you type NDF and press a delimiter, NDF is replaced with NO\_DATA\_FOUND. And the common "teh" typo will be replaced with "the."

Many people don't realize that they can set their own auto-replace pairs. To access auto-replace, go to the Edit | Editor Options | Auto-Replace tab. You'll see the built-in auto-replace substitutions specified.

The TOAD parser scripts come with a handful of substitution pairs, but you can edit and add to the list in the Editor Options window. For example, if you specify a substitution pair of ACCT = ACCOUNT\_ID, when you type ACCT and press the space bar (or other configurable word delimiters), it's automatically replaced with ACCOUNT\_ID. Once saved from this window, the substitution pairs will be saved to an ASCII file named [Language]SUB.TXT in the TEMPS directory.

The auto-replace substitutions for each language type are stored in separate files in the TOAD\TEMPS folder (see Table 1).

Table 1. The TOAD auto-replace substitutions.

Language	Auto-replace substitutions file
HTML	HTMLSUB.TXT
INI	INISUB.TXT
JAVA	JAVASUB.TXT
PL/SQL	PLSQLSUB.TXT
TEXT	TEXTSUB.TXT

## Code completion templates

Code completion templates expand upon the auto-replace substitution concept, but a keystroke (Ctrl-Space Bar) is required to load the code completion templates. Code templates are more than a single phrase and can contain line feeds. If a vertical pipe character is in the code template, the cursor will be placed at that point in the template. Code templates are loaded from the text file [Language].DCI from the TOAD\TEMPS folder, where [Language] can be HTML, INI, JAVA, PLSQL, or TEXT.

For example, one of the code templates defined in PLSQL.DCI is:

```
[crbl | entire cursor block]
DECLARE
  CURSOR c1 IS
    SELECT | FROM WHERE;
  clrec IS c1%ROWTYPE;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO clrec;
    EXIT WHEN cl%NOTFOUND;
  END LOOP;
  CLOSE c1;
END;
```

where "crbl" is the macro for the template (the text *you* type), "entire cursor block" is the description of the template, and everything following until the next template is the body of the template

*TOAD Tip!* Don't leave spaces between the end of the template description and the final right bracket! NT4.0 API calls to manage profile strings have a bug that will cause reading of the templates file to fail.

If you type "crbl" and press Ctrl-Space Bar, TOAD will load the body of the template and place the cursor at the position of the vertical pipe character. If the word or phrase under the cursor doesn't match an existing macro verbatim, a drop-down list of all macros is displayed.

\TEMPS\PLSQL.DCI contains sample templates that you can alter to suit your needs.

You can edit the code completion templates directly in the Edit | Editor Options | Code Templates tab, or via a text editor on the \*.DCI files.

## SQL history

Did you know that you can scroll through the SQL history in TOAD while in the SQL Editor Edit panel?

Alt-Up Arrow displays the previously executed statement that you've run. Alt-Down Arrow is for use after Alt-Up Arrow. It displays the next statement in the SQL history.

If you want your SQL history list to contain only statements that executed successfully, you can check the "Save only statements that are valid" option via View | Options | SQL Editor.

## Object search

You can open the Object Search window by either clicking the toolbar button or selecting the Tools | Object Search

menu item.

Object Search searches all database object names, table column names, index column names, constraint column names, trigger column names, and procedure source code for a keyword or phrase. Each of these items can be searched or excluded from the search via check box options.

Many capabilities of the Object Search window can be found on the Schema Browser Filters, allowing results of the search to be viewed in a browser.

You can search across all users, including or excluding SYS and SYSTEM schemas, or you can search just one schema by selecting the schema in the Schemas/Owners drop-down list. To search across all schemas, pick the first entry in the Schemas/Owners drop-down list ("\* ALL USERS").

To use the Object Search window:

1. Select the Schema/Owner from the drop-down. The default is the current schema.
2. Check or uncheck the Exclude Sys and System check box, as appropriate.
3. Type the search word(s) in the Search for box.
4. Check the appropriate Object Search check boxes and choose the type of filter from the drop-down box. You have a choice of Text occurs anywhere, Starts with text, or Exact match.
5. Check the appropriate Source Search check boxes. You can specify case-sensitive.
6. Click the Search button to begin the search.

The results display in the results grid. If no matches are found, the results grid will be empty.

In the results grid, you can double-click on a table, view, or procedure, and a pop-up window displays details about the object.

## SQL Modeler

Sometimes it's nice to see the relationships between objects. TOAD contains a simple SQL modeling tool that illustrates the objects you have access to. Double-click on the objects in the right window or simply drag-and-drop the objects into the left modeling window, and any relationships will be instantly illustrated. These SQL models are quite convenient to have at your fingertips during the development and testing process of any application. Figure 4 shows some of the Sales Tracking database relationships. Right-clicking on any of the objects again gives you many available options.

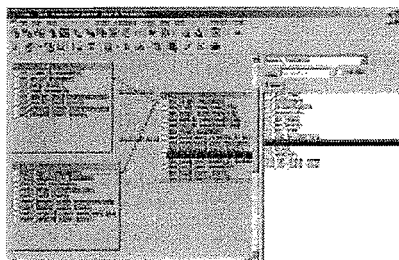


Figure 4. SQL Modeler.

## Ribbit!

I'll end this discussion with a cool feature that you can turn on. If you want TOAD to "croak" after a procedure compiles, go to View | Options | Procedure Editor and check the "Notification when compile process is complete" option. TOAD will play the croak sound TOADLOAD.WAV when the compile of a procedure has completed!

## Summary

TOAD is a very useful tool when working with the Oracle database. It easily allows you to see and manipulate the data, create scripts, create and change objects/procedures/functions/triggers, and much more. TOAD contains many keystrokes that simplify the development process. What used to be a time-consuming process using tools like SQL\*Plus to get the correct spelling of table columns is base

functionality of TOAD. A tool with 250,000 users worldwide can't be a bad thing!

For additional examples and more information on TOAD, check chapter 16 in my current book, *Oracle9i Development by Example*, from Que. ▲

Dan Hotka is a senior technical advisor for Quest Software. He has more than 23 years of experience in the computer industry and more than 18 years of experience with Oracle products. He's an acknowledged Oracle expert with Oracle experience dating back to the Oracle v4.0 days. He's authored the books *Oracle9i Development By Example* and *Oracle8i from Scratch* (Que) and has co-authored the popular books *Oracle Unleashed*, *Oracle8 Server Unleashed*, *Oracle Development Unleashed* (all from Sams) and *Special Edition using Oracle8/8i* (Que). Dan is frequently published in *Oracle Professional* and regularly speaks at Oracle conferences and user groups around the world. dhotka@earthlink.net or dhotka@quest.com.

## Table Functions...

Continued from page 10

to be invoked (as if they were a table) in the FROM list of a SELECT clause. For convenience, they can be used to define a VIEW, giving new functionality.

- Table functions can be used to deliver the rows from an arbitrarily complex PL/SQL transformation sourced from Oracle tables (including, therefore, other table functions) as a "VIEW," without storage of the calculated rows. This gives increased speed and scalability, and increased developer productivity and application reliability.
- Taking the "VIEW" metaphor a step further, the input parameters to the table function allow the "VIEW" to be parameterizable, increasing code reusability and therefore increasing developer productivity and application reliability.
- A table function with a ref cursor input parameter can be invoked with another table function as the data source. Thus table functions can be daisy-chained, allowing modular program design and hence increased ease of programming, reuse, and application robustness.
- Table function execution can be parallelized giving improved speed and scalability. This, combined with the daisy-chaining feature, makes table functions particularly suitable in data warehouse applications for implementing Extraction, Transformation, and Load operations.
- Fanout (DML from an autonomous transaction in the table function) adds functionality of particular interest in data warehouse applications.
- A table function allows data stored in nested tables to be queried as if it were stored relationally, and data stored relationally to be queried as if it were stored as

nested tables. (This will be illustrated in the code samples for the next article.) This allows genuine independence between the format for the persistent storage of data and the design of the applications that access it. (A VIEW can be defined on a table function, and INSTEAD OF triggers can be created on the VIEW to complete the picture.)

Table functions and cursor expressions expand in important new ways our ability as PL/SQL developers both to improve performance and reuse critical business logic. These features are particularly important if you rely on parallelization in your application and want to fully exploit PL/SQL-based functionality. ▲



FEUER.ZIP at [www.oracleprofessionalnewsletter.com](http://www.oracleprofessionalnewsletter.com)

Steven Feuerstein is considered one of the world's leading experts on the Oracle PL/SQL language. He's the author or co-author of six books on PL/SQL, including the now-classic *Oracle PL/SQL Programming* and *Oracle PL/SQL Best Practices* (all from O'Reilly & Associates). Steven is a Senior Technology Advisor with Quest Software, has been developing software since 1980, and worked for Oracle Corporation from 1987 to 1992. Steven is president of the Board of Directors of the Crossroads Fund, which makes grants to Chicagoland organizations working for social, racial, and economic justice ([www.CrossroadsFund.org](http://www.CrossroadsFund.org)). [steven.feuerstein@quest.com](mailto:steven.feuerstein@quest.com).

Bryn Llewellyn is PL/SQL Product Manager, Database and Application Server Technologies Development Group, at Oracle Corporation Headquarters. Bryn has worked in the software field for 25 years. He joined Oracle UK in 1990 at the European Development Center in the Oracle Designer team. He transferred to the Oracle Text team and from there moved into Consulting as EMEA's Text specialist. He relocated to Redwood Shores in 1996 to join the Text Technical Marketing Group. His brief has recently been extended to cover Product Manager responsibility for PL/SQL. [Bryn.Llewellyn@oracle.com](mailto:Bryn.Llewellyn@oracle.com).



# Oracle

Solutions for High-End  
Oracle® DBAs and Developers

## Professional

 PINNACLE

## Debugging PL/SQL with TOAD

Dan Hotka

DOWNLOAD

TOAD is a popular SQL and PL/SQL development tool whose roots go back to its shareware days. TOAD has a real symbolic PL/SQL debugger—this feature makes troubleshooting PL/SQL a snap. In this article, Dan Hotka discusses how to use TOAD's symbolic debugger. His latest book, *TOAD Handbook* (co-authored with Bert Scalzo, published by Sams), is a complete reference for the TOAD tool, including code debugging. Current versions of TOAD (for licensed users and trial versions) can be downloaded from [www.toadsoft.com](http://www.toadsoft.com) or [www.quest.com](http://www.quest.com).

**D**EBUGGING code is usually a tedious job that can be time-consuming. Debugging PL/SQL without a tool reminds me of my COBOL days, when we used to put in display comments to show what variables were at certain places in the process and to indicate when procedures were entered/exited (useful if your code might be looping). Using `DBMS_OUTPUT.PUT_LINE` is the PL/SQL equivalent of performing this kind of character-mode debugging in Oracle.

This article will work with the `TEMPERATURE` package that I already have loaded into my database (the package is included in the Download file that's available at [www.oracleprofessionalnewsletter.com](http://www.oracleprofessionalnewsletter.com)). Using TOAD, I accessed this package right from the database via the SQL Browser | Procs tab—right-click on the `TEMPERATURE` package and select "Load into Procedure Editor." You can also use the "Load source from file" button in the second row of the TOAD Procedure Editor. Compiling this package will put it in your database (use the Compile button or F9).

TOAD allows you to easily debug using this display method. Right-click in the edit window of the Procedure Editor and select "Blank Output Statement." This will create a blank `DBMS_OUTPUT.PUT_LINE` statement in the clipboard. Now return to your code, click where you'd like to put the display, and Ctrl+V (or

*Continues on page 3*

May 2003

Volume 10, Number 5

- 1 Debugging PL/SQL  
with TOAD  
Dan Hotka
- 5 The Keyword Indextype,  
Part 3  
Bryn Llewellyn and  
Steven Feuerstein
- 10 Transaction, Heal Thyself!  
Darryl Hurley
- 16 May 2003 Downloads

DOWNLOAD

Indicates accompanying files are available online at  
[www.oracleprofessionalnewsletter.com](http://www.oracleprofessionalnewsletter.com).

# Debugging PL/SQL with TOAD...

Continued from page 1

Edit | Paste) to put the blank statement in your code, as in Figure 1. Notice that all you have to do is put into the syntax the message and variables you'd like to see at this point.

Debugging in this method is slow and tedious, and, more importantly, you have to go back and remove all these extra statements after you've solved your coding issues.

## Symbolic debugging

Symbolic debugging has been around in many languages for years. This kind of debugging allows you to interactively change the contents of variables, stop the code and check the contents of variables, and process until certain conditions exist, then stop so that the contents of variables can be examined. Symbolic debuggers also allow you to walk through the code one step at a time. This is convenient for finding loops and to see exactly the logic flow that's being executed. The TOAD debugger does all of these things.

Before I dive into a detailed discussion of the TOAD debugger, I'll first discuss some debugging terminology.

Breakpoints are a pre-determined, unconditional stopping point. Code is executed and processing is stopped when this line of code is encountered. The rough equivalent of this would be to put a "pause" statement into your code. Using a symbolic debugger, you're able to examine and change variable content as needed for your test.

Conditional breakpoints are similar to breakpoints, but they're based on the content of a variable. When the condition is met, code processing is suspended at the point where this breakpoint is inserted.

Watches display the contents of variables while processing is occurring and when processing is stopped via a breakpoint or when the routine comes to an end.

## TOAD debugging preliminaries

Figure 2 shows the debugging toolbar, located on the right end of the second line of buttons. These functions are also available from the Debug menu option and are associated with a single keystroke as well (see Figure 3). These

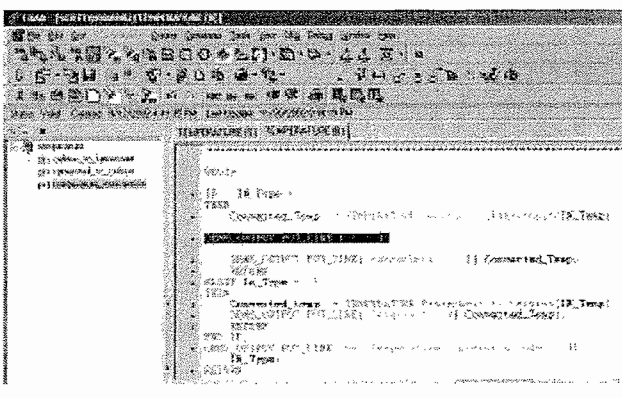


Figure 1. Debugging via DBMS\_OUTPUT.PUT\_LINE statements.

functions are also available by right-clicking in the Procedure Editor window.

Following are descriptions of the options shown in Figure 2, from left to right:

- The lightning bolt is the Run button. This button will execute the procedure in debug mode.
- The (...) parameter allows you to set any input variables to the package and/or procedure.
- Next is the Step Over button, which allows you to walk through the PL/SQL code one line at a time. This button will skip any procedures that are called.
- The fourth button is the Trace Into button. Both this button and the Step Over button walk through PL/SQL code one line at a time, but Trace Into will enter any called procedures or functions and execute that code one line at a time.
- Next is the Trace Out button; it exits Trace Into code.
- The sixth button is the Run to Cursor button. This is an alternative to setting breakpoints. PL/SQL is executed until it reaches the line where the cursor is, and then halts execution much like a breakpoint.
- The hand button is the Halt Execution key, and it will stop execution of the PL/SQL routine.
- The eyeglass button (Add Watch at Cursor) allows you to easily set watches on variables where the cursor is located.
- The final button compiles the entire procedure call tree for debugging.

Programmers who use TOAD like the hot keys. Figure 3 illustrates the aforementioned functions and their associated keystrokes; you can access this list by selecting Debug from the top menu bar.

## Setting breakpoints/visualizing variable contents

Breakpoints suspend the execution of the code. This allows you to view the contents of various variables.

Figure 4 shows a simple looping example. You could set the breakpoint by simply clicking to the left of the Procedure Window. Notice that the Break Points tab at the bottom shows the line location of the breakpoint. To work this code, click where the breakpoint is desired and then press the Run button. This will invoke the debugging routine and suspend code execution when the breakpoint is reached.

After the code is suspended, you can check the contents of variables and implicit cursors. The line with the breakpoint set is now highlighted in blue. Simply hover over a variable and the contents will pop up after a second or so. As shown in Figure 5, when you



Figure 2. Debugging control buttons.

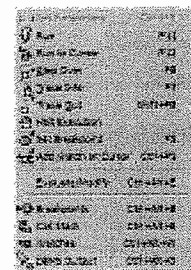


Figure 3. Hot keys.

place the mouse cursor over the loop\_counter variable, the debugger automatically displays the current value stored in the variable.

In fact, TOAD can also display the contents of implicit cursors. In Figure 6, the mouse is hovering over the REC implicit cursor name, and all of the columns from the SELECT \* FROM emp query are displayed.

## Watches

A watch in TOAD allows you to watch the changes to variables as the code is executed. You can also change the contents of a "watched" variable. Watches are easy to set. You can highlight the variable and press the Add Watch button on the Watch tab (the Ins key is the TOAD hot key). You can also right-click on the highlighted variable and select Debug | Add Watch at Cursor. Any of these options will add the variable to the Watch tab as shown in Figure 7. This example shows that the variable watch is being set prior to executing the procedure.

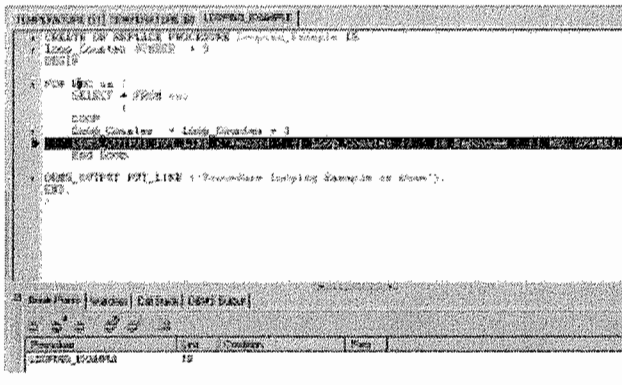


Figure 4. Looping example with breakpoint set at line 10.

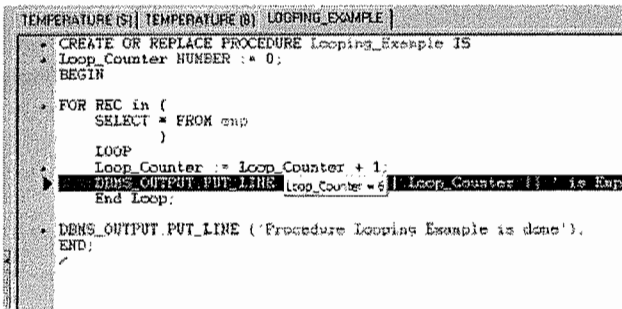


Figure 5. Mouse hovering over the loop\_counter variable.



Figure 6. Mouse hovering over an implicit cursor name.

Figure 8 shows that the variable can be changed when a breakpoint is utilized to suspend the execution of the code. Notice that the content of the variable is 50 and is being changed, on the fly, to 40. This is accomplished by pressing the Evaluate/Modify Watch button (the button on the Watch tab that looks like a calculator). Make sure to highlight the variable in the Watch tab prior to pressing this button.

Figure 9 shows an implicit cursor with a watch set on it.

*Continues on page 15*

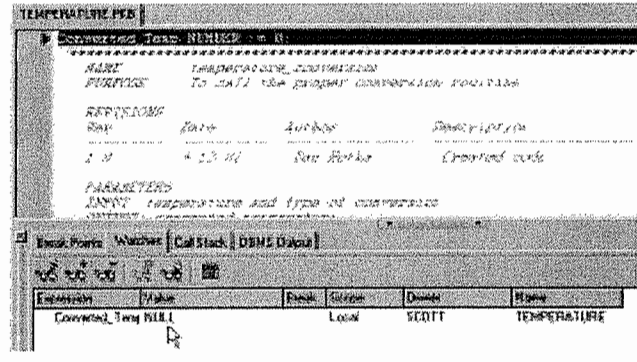


Figure 7. Watching variables.

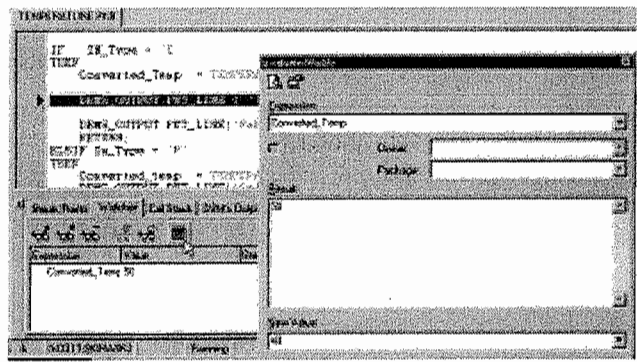


Figure 8. Modifying variable contents.

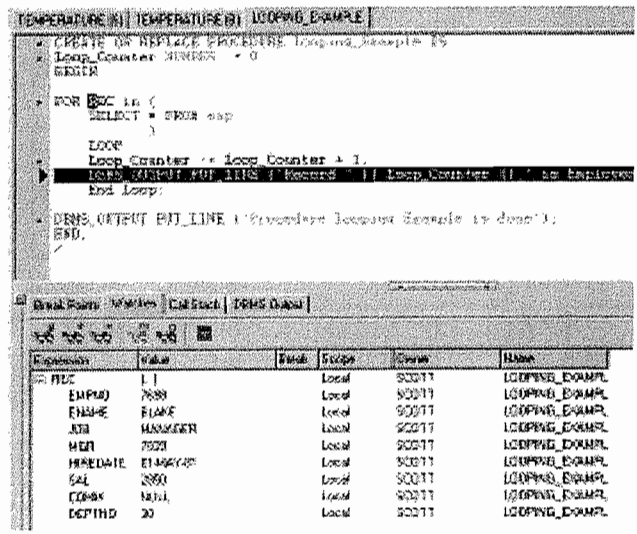


Figure 9. Implicit cursor being watched.

```

        error_type,
        username,
        object_type,
        object_name,
        done)
VALUES (fixer_info_seq.nextval,
       v_error_type,
       ORA_LOGIN_USER,
       v_object_type,
       v_object_name,
       'N');
END IF;

```

Now the fixer package shown in Listing 7 has enough information to free up suspended statements.

Listing 7. Expanded FIXER package.

```

CREATE OR REPLACE PACKAGE BODY fixer AS

/*-----*/
PROCEDURE fix_it IS
/*-----*/

    CURSOR curs_get_sql IS
    SELECT *
    FROM fixer_info
    WHERE done = 'N';

BEGIN

    OPEN curs_get_sql;

    LOOP

        FETCH curs_get_sql INTO v_fixer_info_rec;

        EXIT WHEN curs_get_sql%NOTFOUND;
        /* Call the appropriate procedure to fix the
        error by replacing spaces
        with underscores eg :
        SPACE QUOTA EXCEEDED = SPACE_QUOTA_EXCEEDED
        */
        EXECUTE IMMEDIATE 'BEGIN ' ||

```

```

        REPLACE(v_fixer_info_rec.error_type, ' ','_')
        || ';' || 'END;';

        UPDATE fixer_info
        SET done = 'Y'
        WHERE fixer_id = v_fixer_info_rec.fixer_id;

    END LOOP;

    COMMIT;

    END fix_it;

END fixer;

```

Future installments of this series will detail the code required to build a complete procedure for handling space quota errors, as well as flush out more details of the FIXER package.

### Coming up next time

The next article in this series details error and exception-handling concepts for suspended statements—specifically, how to determine why a statement was suspended and ensure future statements behave accordingly. ▲

**DOWNLOAD** HURLEY.ZIP at [www.oracleprofessionalnewsletter.com](http://www.oracleprofessionalnewsletter.com)

Darryl Hurley has been working with Oracle technology for 15 years, with a significant focus on database administration and PL/SQL development. His days are spent as the database manager for MDSI Mobile Data Solutions Inc. ([www.mdsi-advantex.com](http://www.mdsi-advantex.com)), and his spare time is spent writing articles and teaching under the moniker of ImpleStrat Solutions ([www.implestrat.com](http://www.implestrat.com)). He's written several articles for the Oracle Development Tools User Group ([www.odtug.com](http://www.odtug.com)) and contributed to several Oracle books from O'Reilly & Associates ([www.ora.com](http://www.ora.com)). [dhurley@mdsi.ca](mailto:dhurley@mdsi.ca) or [darryl.hurley@implestrat.com](mailto:darryl.hurley@implestrat.com).

Is there to be a Download file?

## Debugging PL/SQL with TOAD...

Continued from page 4

Once again, it's easy to see data value changes as the procedure or function executes.

### Conditional breakpoints

TOAD allows for breakpoints to occur when the process has performed certain events (such as looping x number of

times) or when a certain data condition exists. This is easily accomplished by first setting the breakpoint. Click to the left of the Procedure Window, and then right-click and select Debug | Set Breakpoint). Then, go to the Break Points tab, highlight the breakpoint for which you want to define a condition, and press the Edit Breakpoint button (the left-most button). Figure 10 shows how you can set a count on the number of times the breakpoint is to be passed before suspending code execution. Figure 11 shows that code execution suspension can occur with just about any condition valid within the procedure.

### Using the debugger

Here are the steps for using the debugger:

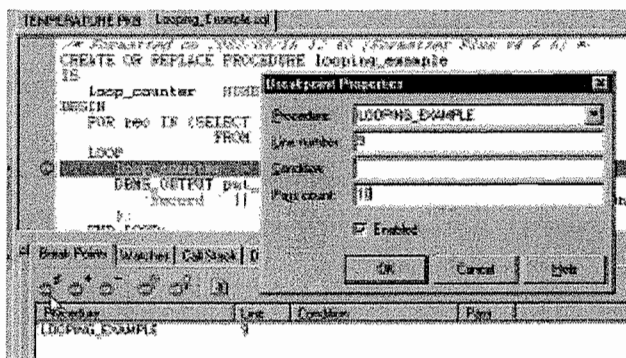


Figure 10. Looping conditional breakpoint.

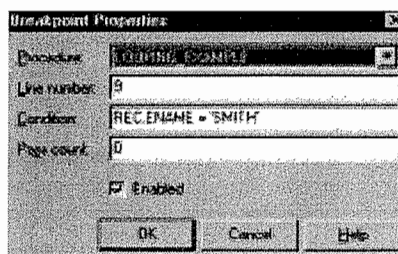


Figure 11. Conditional breakpoint.

1. Access the package/procedure/function that you want to debug.
2. Make sure you set any necessary procedure parameters.
3. Set watches on variables you wish to monitor during execution.
4. Set breakpoints where you'd like to examine variable content.
5. Set conditional breakpoints to speed up the debugging process, allowing code execution to occur until the desired problematic code or routine is accessed.
6. Use the Run button if breakpoints are set.
7. Use the Step Over, Trace Into, and Trace Out buttons to walk through code one line at a time (this is useful from the beginning, or from a breakpoint).
8. Modify variable contents to alter the code to drop into routines you want to debug.

### Conclusion

PL/SQL packages, procedures, and functions can be lengthy

and contain many calls to other procedures/functions, making them difficult to troubleshoot. PL/SQL debugging is made easy with the new features that are available in TOAD. I hope the information in this article is helpful to you in your work with PL/SQL. ▲

### DOWNLOAD

HOTKA.ZIP at [www.oracleprofessionalnewsletter.com](http://www.oracleprofessionalnewsletter.com)

Dan Hotka is a training specialist who has more than 24 years of experience in the computer industry and more than 19 years of experience with Oracle products. He's an internationally recognized Oracle expert with Oracle experience dating back to the Oracle v4.0 days. His current book, *The TOAD Handbook* (from Sams), is now available on [www.Amazon.com](http://www.Amazon.com). He's also the author of *Oracle9i Development By Example* and *Oracle8i from Scratch* (Que) and co-authored the popular books *Oracle Unleashed*, *Oracle8 Server Unleashed*, *Oracle Development Unleashed* (all from Sams) and *Special Edition using Oracle8/8i* (Que). He's frequently published in Oracle trade journals and regularly speaks at Oracle conferences and user groups around the world. Visit his Web site at [www.DanHotka.com](http://www.DanHotka.com). [dhotka@earthlink.net](mailto:dhotka@earthlink.net).

## May 2003 Downloads

- HOTKA.ZIP—Source code to accompany Dan Hotka's article, "Debugging PL/SQL with TOAD."
- BRYN.ZIP—Source code to accompany Bryn Llewellyn and

Steven Feuerstein's article, "The Keyword Indextype, Part 3."

- HURLEY.ZIP—Source code to accompany Darryl Hurley's article, "Transaction, Heal Thyself!"

For access to all current and archive content and source code, log in at [www.oracleprofessionalnewsletter.com](http://www.oracleprofessionalnewsletter.com) with your unique subscriber user name and password. For access to this issue's Downloads only, click on the "Source Code" button, select the file(s) you want from this issue, and enter the User name and Password at right when prompted.

User name **instant**

Password **jargon**

**Editor:** Garry Chan ([gchan@procaseconsulting.com](mailto:gchan@procaseconsulting.com))  
**CEO & Publisher:** Mark Ragan  
**Group Publisher:** Connie Austin  
**Executive Editor:** Farion Grove  
**Production Editor:** Andrew McMillan

### Questions?

Customer Service:  
 Phone: 800-493-4867 x.4209 or 312-960-4100  
 Fax: 312-960-4106  
 Email: [PinPub@Ragan.com](mailto:PinPub@Ragan.com)

Editorial: [FarionG@Ragan.com](mailto:FarionG@Ragan.com)

Advertising: [HowardF@Ragan.com](mailto:HowardF@Ragan.com)

Pinnacle Web Site: [www.pinnaclepublishing.com](http://www.pinnaclepublishing.com)

### Subscription rates

United States: One year (12 issues): \$229; two years (24 issues): \$389  
 Other:\* One year: \$254; two years: \$432

**Single issue rate:**  
 \$27.50 (\$32.50 outside United States)\*

\* Funds must be in U.S. currency.

*Oracle Professional* (ISSN 1525-1756)  
 is published monthly (12 times per year) by:

Pinnacle  
 A division of Lawrence Ragan Communications, Inc.  
 316 N. Michigan Ave., Suite 400  
 Chicago, IL 60601

**POSTMASTER:** Send address changes to Lawrence Ragan Communications, Inc., 316 N. Michigan Ave., Suite 400, Chicago, IL 60601.

Copyright © 2003 by Lawrence Ragan Communications, Inc. All rights reserved. No part of this periodical may be used or reproduced in any fashion whatsoever (except in the case of brief quotations embodied in critical articles and reviews) without the prior written consent of Lawrence Ragan Communications, Inc. Printed in the United States of America.

Oracle, Oracle 8i, Oracle 9i, PL/SQL, and SQL\*Plus are trademarks or registered trademarks of Oracle Corporation. Other brand and product names are trademarks or registered trademarks of their respective holders. *Oracle Professional* is an independent publication not affiliated with Oracle Corporation. Oracle Corporation is not responsible in any way for the editorial policy or other contents of the publication.

This publication is intended as a general guide. It covers a highly technical and complex subject and should not be used for making decisions concerning specific products or applications. This publication is sold as is, without warranty of any kind, either express or implied, respecting the contents of this publication, including but not limited to implied warranties for the publication, performance, quality, merchantability, or fitness for any particular purpose. Lawrence Ragan Communications, Inc., shall not be liable to the purchaser or any other person or entity with respect to any liability, loss, or damage caused or alleged to be caused directly or indirectly by this publication. Articles published in *Oracle Professional* reflect the views of their authors; they may or may not reflect the view of Lawrence Ragan Communications, Inc. Inclusion of advertising inserts does not constitute an endorsement by Lawrence Ragan Communications, Inc., or *Oracle Professional*.





# Oracle

Solutions for High-End  
Oracle® DBAs and Developers

## Professional

# Project Raptor/SQL Developer Review

Dan Hotka

The Oracle development world hasn't had a quality and free development tool since Quest Software purchased TOAD. Well, that is until Project Raptor came along! In this article, Dan Hotka introduces us to this new Oracle graphical interface tool.

**P**ROJECT Raptor is Oracle's newly released free graphical database query and object development/debugging tool. This tool is in its infancy and already has quality features found in many available development tools.

Project Raptor is Oracle's internal name for this project. When it's released for production, it will be named SQL Developer. In the meantime, you might hear both names used for this tool.

This article will start with Project Raptor's system requirements and illustrate the ease of installation. I'll then demonstrate the tool's main features, including connecting to databases, database object browsing, and running SQL and scripts.

### Raptor capabilities

Project Raptor gives the user a nice graphical interface to the Oracle RDBMS environment (see Figure 1). Navigation is easy via the tree browser on the left. Raptor can run SQL statements or SQL scripts. If the scripts contain variables, it will prompt the user for each variable. Raptor gives a simple execution plan.

Tables, indexes, and database objects can easily be created, data viewed, relationships illustrated, dependencies listed, and Raptor can even create a script for the object selected.

The SQL area has a history. There are code "snippets" to aid in the fine details of syntax. These snippets include various date format options, number formats, character formats, hints, and even PL/SQL code examples. The code can be dragged and dropped from the snippets window.

All kinds of PL/SQL can be created and maintained. Raptor even includes

**March 2006**

Volume 13, Number 3

- 1** Project Raptor/SQL Developer Review  
Dan Hotka
- 6** Conditional Compilation in PL/SQL, Part 3  
Steven Feuerstein
- 12** March 2006 Downloads

**DOWNLOAD**

Indicates accompanying files are available online at  
[www.pinnaclepublishing.com](http://www.pinnaclepublishing.com).

a symbolic debugger that allows for break points and variable manipulation.

Raptor contains a variety of reports. These reports include a wealth of information about the database, top SQL statements, current cursor information, users, objects, storage parameters, tablespace storage, scheduling, and more. You can even add your own reports to Raptor.

Raptor appears to be a pretty complete development and research tool for the developer and DBA. The tool is only in its third release in about a month and it already contains all of this functionality.

There are no server-side objects to install, and no SYSTEM password is required. Raptor will allow you to see and do what your Oracle permissions allow you to do.

This article will focus on table objects and SQL. Watch for future articles on the Raptor debugger, problem-solving using Raptor, a Raptor report review, and of course new features and updates.

### Raptor system requirements

The version of Raptor used for this article is version 0795. This version supports both Windows (2000, NT, and XP) and Linux (Redhat and SuSE) operating systems. Oracle plans an Apple MAC version.

Raptor requires Java SDK 1.5 or greater. If your system has this version of Java, then the hard disk requirements are 42MB. The download contains the Java SDK 1.5 software and takes a disk footprint of 110MB.

Raptor is certified (tested and supported) against Oracle 9.2.4.0, Oracle10g, and Oracle10g Express database environments.

### Raptor download and installation

Project Raptor can be downloaded as a ZIP file from the Oracle Technology Network at [www.oracle.com/technology/products/database/project\\_raptor](http://www.oracle.com/technology/products/database/project_raptor).

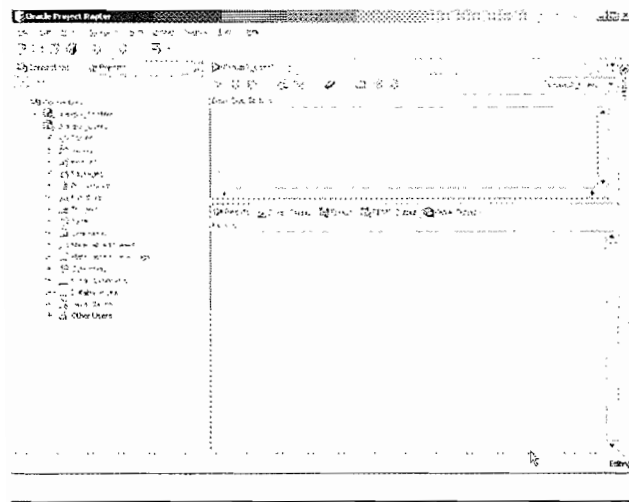


Figure 1. The Project Raptor interface.

Installation is a snap. Simply make a folder on your computer and unzip the contents of the download. I always make an icon for my desktop.

The nice part of this installation is that the ZIP file contains the Java.exe required to run Raptor. The first time you run Raptor, it might ask for the location of the Java.exe. If your system doesn't have the Java SDK version 1.5 installed, simply point this information box to the directory that you unzipped Raptor into followed by `jdk/jre/bin/java.exe`. You can also search for "java.exe" from the same Raptor install directory.

Raptor doesn't require the use of the TNSNAMES.ora file for its connections. Having one of these TNSNAMES.ora files handy when creating your connections will be helpful.

The first time you run Raptor, you'll need to create a "gallery" that will also contain your database connections. These galleries are a way of grouping your databases and connections together.

Connections require a valid userid, password, machine name, and an Oracle Sid. Raptor doesn't have an Oracle Home concept. The advanced option accepts JDBC connection information. I got my Raptor working using the host name, port, and service names found in my TNSNAMES.ora file, as shown in Figure 2.

*Note:* The Raptor documentation indicates that Raptor will display the TNSNAMES.ora information, but I wasn't able to get this version of Raptor to display any existing connection information.

Once the connection is made, Raptor maintains it. When starting Raptor again, all you'll see is "Connections" in the left navigator pane. Opening this item will display all the connections you've established.

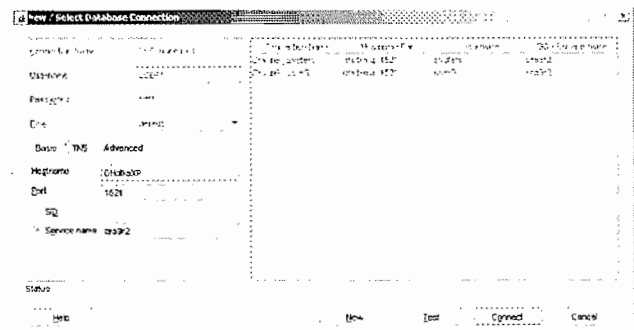


Figure 2. Database connection setup.

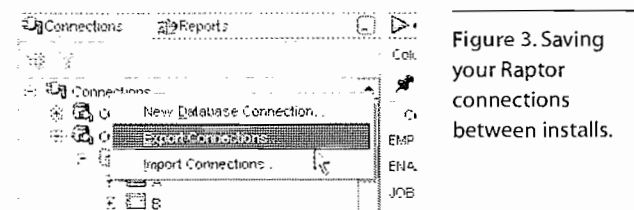


Figure 3. Saving your Raptor connections between installs.



When you click on one of the listed connections, Raptor will connect you and display the object navigation tree for that connection.

A shortcoming I noticed in this connection scenario is that Raptor does save passwords encrypted, but it would be nice if you didn't have to give it a password and Raptor would then prompt you for a password. I feel this might present a security risk for some sites.

When upgrading to newer versions of Raptor, you may want to export your connections. Right-click on the Connections item and select Export Connections, as shown in Figure 3. After installing the new Raptor, follow this same procedure except select Import Connections. This will save all of your connections and make them available in the new Raptor environment.

Each connection maintains one login to the database. If you have several SQL windows open using one connection, each SQL window shares this one connection so a commit in one SQL window commits all work across the windows using the same connection. If you want separate sessions to the same database, create additional connections, using one connection for one SQL window and another connection to the same database in another SQL window.

## Using Raptor

Raptor is an intuitive tool. There are buttons that will cause Raptor to do work, and there's an accelerator key (key stroke combination, or hotkey) that will also activate most features. Click on the top menu bar to see the various common options and their associated keystrokes. Figure 4 shows the Edit menu item and its tasks.

The Edit dropdown menu supports all of the common cut/save/paste type keystrokes. The File menu allows you to get/save the SQL you're working on, among other things. The View dropdown menu allows you to open or close various views—for instance, you can close the Connections pane, open the Code Snippets pane (which can stay active but hidden along the right-hand side!)—and it allows you to navigate to the Connections or

Reports tab in the left pane. The Navigate dropdown menu allows you to view the previous Raptor view you had up, and place and navigate bookmarks you placed in your code (while using Raptor). The Run dropdown menu allows you to execute your code via a menu item. The Debug dropdown menu allows you to debug your PL/SQL routines. Again, notice most of these action items are associated with an accelerator key sequence. The Source dropdown menu allows additional code functionality and insights when coding PL/SQL routines. The Tools dropdown menu gives you access to predefined tools such as SQL\*Plus. More importantly, the Raptor Preferences access is on this dropdown menu. I'll discuss these options later in the article. There's also context-sensitive help throughout the tool.

Using Raptor is rather easy. Once you've established your connection, you'll get a SQL window (see the circled tab in Figure 5). If you don't have a tab with your connection name in it, simply go to the Tools dropdown menu and select SQL Worksheet. You can have multiple tabs open and you can have multiple connections open at the same time. This is a convenient way to run the same SQL in one environment and then run it again in another environment. Type or open a file that has your SQL that

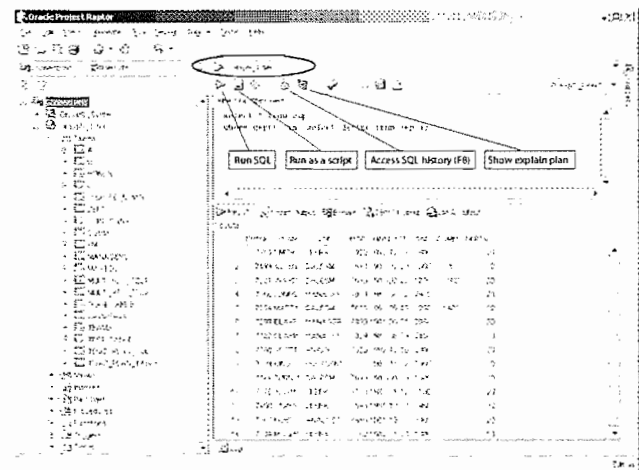


Figure 5. The Raptor SQL window.

Figure 4. Navigating around Raptor.

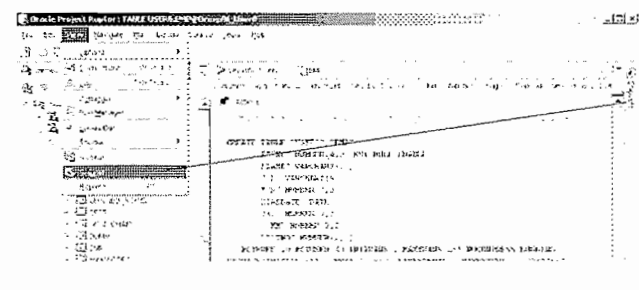
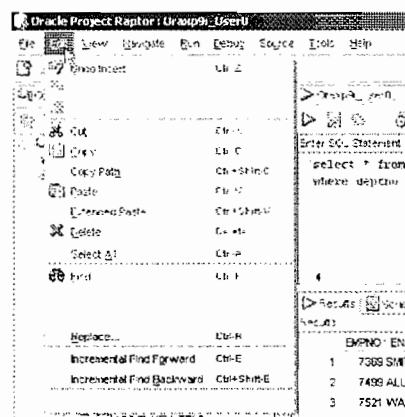


Figure 6. Accessing code snippets.

you'd like to run and press the green button just above the SQL window.

If you type two or more SQL statements into the SQL window, the one that the cursor is on is the one that will be run by pressing the green button. The Results tab will be populated with your results. If you wish for all of the SQL to be executed, press the Run as a Script button. This will populate the Script Output tab.

The explain plan is visible by clicking on the Explain tab. An explain plan can be reached without executing the SQL by pressing the Explain Plan button above the SQL window.

The pencil eraser clears the SQL window, the diskette is the save button for all code in the SQL window, and the printer will send the contents of the SQL window to your default printer.

The Navigator menu bar item allows you to easily jump to various parts of the code in the SQL window.

Raptor also keeps track of all of your SQL. You can use the SQL History button (see Figure 5) or the F8 key to display the SQL you've entered. There are options in this screen to replace or append the desired SQL into the current SQL window.

Code Snippets is a dockable window (see Figure 6) that allows for easy access to various SQL code formats, PL/SQL coding examples (such as a working example of IF/THEN/ELSE, CASE statements, CURSOR management, and so on), SQL hints, and more. Figure 7 shows the current list of available snippets. This area will always be updated with the latest code examples for the supported Oracle database releases.

These code snippets can be dragged and dropped (left-click and hold, move the object to the SQL window, then release) to be easily included in your SQL or PL/SQL code. Watch for more code examples in future releases of Raptor.

The Navigator pane (left side) also allows you to filter or search for specific objects. The "%" is the global search

symbol. Right-click on the navigator item and select Apply Filter (see Figure 8). You can use any character pattern. Figure 9 illustrates selecting just table objects that start with D. The filter criteria for this would be D%. You can visually tell that a filter has been applied as the filter criteria appear in the navigator!

You can easily create and edit any database object for which you have proper permissions allowing you to do so. Right-click on any of the navigator headings (tables, indexes, and so on) to create a new object or right-click on any existing object to edit the object. As shown in Figure 10, each object has many options allowing you to perform the latest supported database tasks allowed for that object type. When creating or editing objects, there's usually a DDL tab that allows you to review and save the syntax generated by the screen options.

If you right-click on your database connection, you'll see different options as shown in Figure 11. Notice the Export option, where you can gather schema stats or

create a DDL script for this entire schema! Individual object scripts are available by clicking on the object and going to the right pane and clicking on the SQL tab.

## Raptor table objects and SQL

Raptor allows for easy access to object information (columns, statistics, scripts

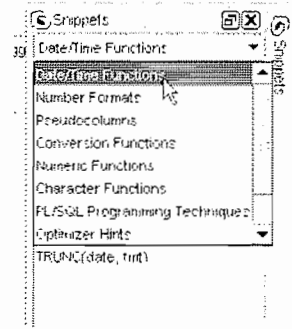


Figure 7. Supported snippets.

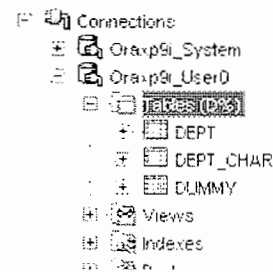


Figure 9. Filtering on D%.

Figure 8. Navigator filtering.

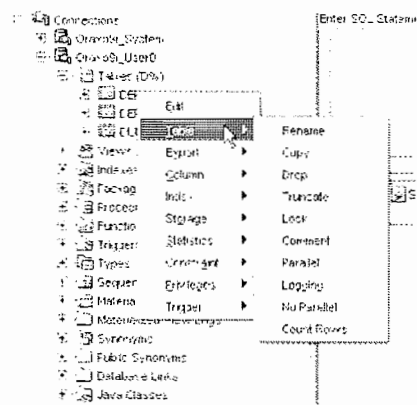
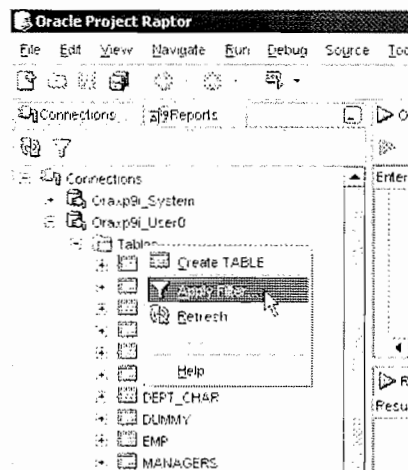
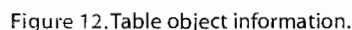
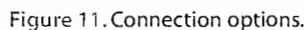


Figure 10. Object maintenance.

This table display allows me to see the columns and data types. Notice that you can easily see indexes, constraints, permissions, and statistics. The Dependencies tab is very nice if you're considering altering or dropping the object. It will display any relationship or reference to the object from within any PL/SQL code stored in the database.

With the buttons along the top edge of this tab you can add a row, delete a row, and commit or rollback your changes. The button with two blue arrows allows you to refresh the data in the pane. The Sort button allows you to order the data, and Filter allows you to just display certain rows. Typing where clause predicates in the Filter



The red thumb pin button appears on many Raptor screens. Clicking this button will keep the screen's information available while you click on other objects.

Being an old DBA, I'm used to reading scripts and working with scripts. I like the SQL tab (shown in Figure 15) that displays the syntax necessary to re-create this object. This syntax can then be copied and saved as part of a test environment script perhaps.

### Raptor preferences

The Environment preferences allow you to set the default behavior of Raptor when starting up and exiting.

*Continues on page 11*

Figure 13. Data tab of table object EMP.

Figure 14.  
Saving data.

allowing us to take full advantage of each version's unique PL/SQL features. ▲

**DOWNLOAD** 603FEUER.ZIP at [www.pinnaclepublishing.com](http://www.pinnaclepublishing.com)

Steven Feuerstein is considered one of the world's leading experts on the Oracle PL/SQL language, having written 10 books on PL/SQL, all from

O'Reilly Media. Steven has been developing software since 1980 and serves as a Senior Technology Advisor to Quest Software. The fourth edition of his popular *Oracle PL/SQL Programming* was released in September 2005, and he's currently working on two products: Qnxo ([www.qnxo.com](http://www.qnxo.com), a code generator and repository manager) and Qute ([www.unit-test.com](http://www.unit-test.com), a unit testing tool). You can sign up for a monthly PL/SQL newsletter at [www.oracleplsqliprogramming.com](http://www.oracleplsqliprogramming.com). [steven@stevenfeuerstein.com](mailto:steven@stevenfeuerstein.com).

## Project Raptor...

*Continued from page 5*

You can easily disable prompts to save code, for example, by having Raptor automatically save all files when exiting. Clicking on the + allows you to modify the default code colors, the SQL history save timings, and more.

The Code Editor allows for default control over the amount of indenting, pasting, default fonts, printing, and so on.

Database Connections allows you to define the locations of other utilities such as Export and SQL\*Plus. This screen also allows you to register your JDBC drivers.

The Debugger allows you to adjust the default behavior of the PL/SQL debugging environment.

You can tell Raptor which file types to display during OS file searches with the File Types screen, and the Global Ignore can't show any OS file that would have otherwise appeared in Raptor selection areas.

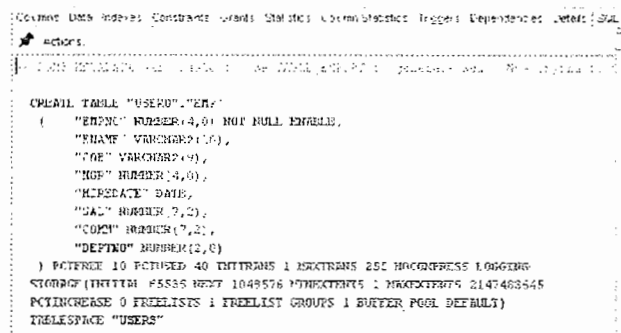


Figure 15. Object syntax.

**Know a clever shortcut? Have an idea for an article for *Oracle Professional*?  
Visit [www.pinnaclepublishing.com](http://www.pinnaclepublishing.com) and click on "Write For Us" to submit your ideas.**

## Don't miss another issue! Subscribe now and save!

**Subscribe to *Oracle Professional* today and receive a special one-year introductory rate:  
Just \$179\* for 12 issues (that's \$20 off the regular rate)**

NAME		
COMPANY		
ADDRESS		
CITY	STATE/PROVINCE	ZIP/POSTAL CODE
COUNTRY IF OTHER THAN U.S.		
E-MAIL		
PHONE (IN CASE WE HAVE A QUESTION ABOUT YOUR ORDER)		

- ☐ Check enclosed (payable to Pinnacle Publishing)  
☐ Purchase order (in U.S. and Canada only); mail or fax copy  
☐ Bill me later  
☐ Credit card: ☐ VISA ☐ MasterCard ☐ American Express

CARD NUMBER EXP. DATE

SIGNATURE (REQUIRED FOR CARD ORDERS)

Detach and return to:  
Pinnacle ▲ 111 E. Wacker Dr., Suite 500 ▲ Chicago, IL 60601  
Or fax to 312-960-4106

\* Outside the U.S. add \$30. Orders payable in U.S. funds drawn on a U.S. or Canadian bank.

INS6

**Pinnacle, A Division of Lawrence Ragan Communications, Inc. ▲ 800-920-4804 or 312-960-4100 ▲ Fax 312-960-4106**

## Summary

Project Raptor is in its early stages of development and release. It's only been available for about a month now, and Oracle Corporation has already had three code releases. I've found this to be a useful tool, especially considering that it's fully supported by Oracle Corporation and that it's completely free. Oracle has established a forum for Raptor where you can easily make comments, suggestions, or simply monitor the questions and answers others have posed to the Raptor development staff (<http://forums.oracle.com/forums>).

Watch for additional Project Raptor articles on using the PL/SQL Debugger and working with the various reports of Raptor to discover problems, display busy SQL, or gain useful information. ▲

Dan Hotka is a Training Specialist with over 27 years of experience in the computer industry and more than 22 years of experience with Oracle products. He's an internationally recognized Oracle expert with Oracle experience dating back to the V4.0 days. Dan's latest book is *Database Oracle10g Linux Administration* (Oracle Press). He's also the author of *Oracle9i Development By Example* and *Oracle8i from Scratch* (Que) and has co-authored six other popular books. He's frequently published in Oracle trade journals, and regularly speaks at Oracle conferences and user groups around the world. [www.DanHotka.com](http://www.DanHotka.com), [dhotka@earthlink.net](mailto:dhotka@earthlink.net).

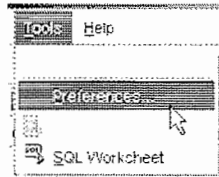


Figure 16. Accessing the Raptor preferences via the Tools menu.

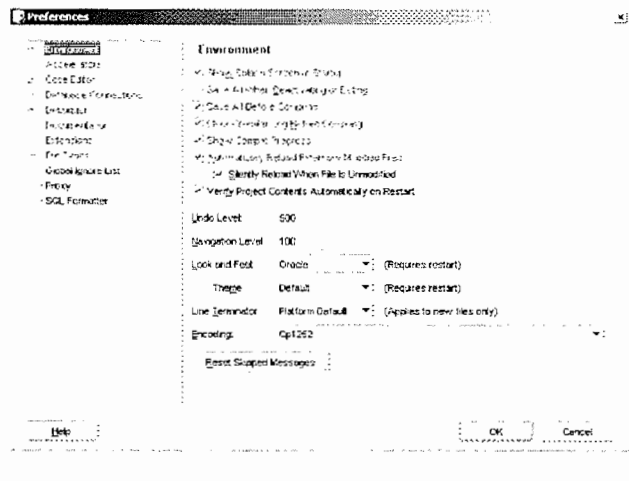


Figure 17. Preferences.

## March 2006 Downloads

- 603FEUER.ZIP—Source code to accompany Steven Feuerstein's article, "Conditional Compilation in PL/SQL, Part 3."

For access to current and archive content and source code, log in at [www.pinnaclepublishing.com](http://www.pinnaclepublishing.com).

**Editor:** Garry Chan ([gchan@procaseconsulting.com](mailto:gchan@procaseconsulting.com))  
**Contributing Editor:** Bryan Boulton  
**CEO & Publisher:** Mark Ragan  
**Group Publisher:** Michael King  
**Executive Editor:** Farion Grove

### Questions?

#### Customer Service:

Phone: 800-920-4804 or 312-960-4100  
Fax: 312-960-4106  
Email: [PinPub@Ragan.com](mailto:PinPub@Ragan.com)

Advertising: [PinPub@Ragan.com](mailto:PinPub@Ragan.com)

Editorial: [FarionG@Ragan.com](mailto:FarionG@Ragan.com)

Pinnacle Web Site: [www.pinnaclepublishing.com](http://www.pinnaclepublishing.com)

### Subscription rates

United States: One year (12 issues): \$199; two years (24 issues): \$348  
Other:\* One year: \$229; two years: \$408

#### Single issue rate:

\$27.50 (\$32.50 outside United States)\*

\* Funds must be in U.S. currency.

Oracle Professional (ISSN 1525-1756)  
is published monthly (12 times per year) by:

Pinnacle Publishing  
A Division of Lawrence Ragan Communications, Inc.  
111 E. Wacker Dr., Suite 500  
Chicago, IL 60601

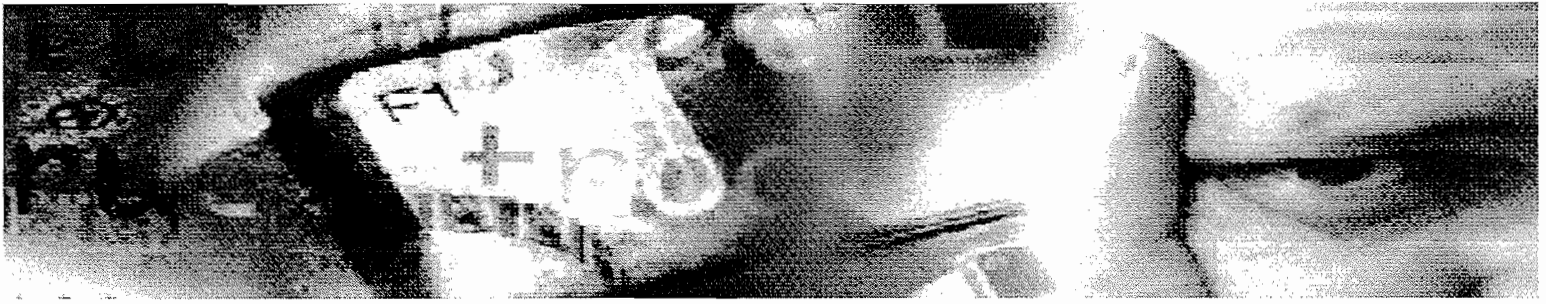
**POSTMASTER:** Send address changes to Lawrence Ragan Communications, Inc., 111 E. Wacker Dr., Suite 500, Chicago, IL 60601.

Copyright © 2006 by Lawrence Ragan Communications, Inc. All rights reserved. No part of this periodical may be used or reproduced in any fashion whatsoever (except in the case of brief quotations embodied in critical articles and reviews) without the prior written consent of Lawrence Ragan Communications, Inc. Printed in the United States of America.

Oracle, Oracle 8i, Oracle 9i, Oracle 10g, PL/SQL, and SQL\*Plus are trademarks or registered trademarks of Oracle Corporation. Other brand and product names are trademarks or registered trademarks of their respective holders. Oracle Professional is an independent publication not affiliated with Oracle Corporation. Oracle Corporation is not responsible in any way for the editorial policy or other contents of the publication.

This publication is intended as a general guide. It covers a highly technical and complex subject and should not be used for making decisions concerning specific products or applications. This publication is sold as is, without warranty of any kind, either express or implied, respecting the contents of this publication, including but not limited to implied warranties for the publication, performance, quality, merchantability, or fitness for any particular purpose. Lawrence Ragan Communications, Inc., shall not be liable to the purchaser or any other person or entity with respect to any liability, loss, or damage caused or alleged to be caused directly or indirectly by this publication. Articles published in Oracle Professional reflect the views of their authors; they may or may not reflect the view of Lawrence Ragan Communications, Inc. Inclusion of advertising inserts does not constitute an endorsement by Lawrence Ragan Communications, Inc., or Oracle Professional.





## Using the Oracle PL/SQL Profiler (Draft)

By Dan Hotka 9/2007

Oracle has provided the ability to see how much time each step of a PL/SQL routine takes since Oracle8i. The environment is easy to setup and the information is easy to retrieve.

### Introduction

Why Profile? When tuning SQL, it is easy because there is just the single SQL statement. With PL/SQL, there are SQL statements, SQL imbedded in implicit and explicit cursors, called routines (functions/procedures), and the PL/SQL code itself. When a PL/SQL routine is taking 5 minutes to run, exactly **what code** is taking how **much time**. This is the information that the PL/SQL profiler provides. Without this information, the person trying to tune the PL/SQL is only shooting in the dark, perhaps pulling out and tuning the SQL within the code, but otherwise has no idea what a PL/SQL routine is doing when it comes to time spent on each line of code.

The profiler is easy to setup and easy to use. Tools like Quest Software's TOAD provides a nice GUI interface to this profiler.

### Installation

The profiler has two scripts that setup the environment. Both are found in the <ORACLE\_HOME>/rdbms/admin folder.

The PROFLOAD.sql script needs to be run as SYS (connect <connect string> AS SYSDBA). This script will create the DBMS\_PROFILER package and create synonyms and permissions for usage.

The PROFTAB.sql script is recommended to be executed for each user desiring to run the DBMS\_PROFILER package. This script sets up the three main tables: PLSQL\_PROFILER\_RUNS, PLSQL\_PROFILER\_UNITS, and the PLSQL\_PROFILER\_DATA. This script can be setup so that all the users share these tables but this topic is beyond the scope of this paper.

### Understanding the Profiler Process

Profiling is initiated with the START\_PROFILER program. Start this process then execute the PL/SQL routine to be profiled. When execution is done, run the

STOP\_PROFILER program. This will stop the profiling process and write the collected information to the profiler tables.

There are two more routines that control the collection of profiler data: PAUSE\_PROFILER and RESUME\_PROFILER. These routines might be useful if only certain statistics are of interest from a rather large PL/SQL program, or perhaps called subroutines are not desired to be profiled. These routines are typically imbedded in the PL/SQL code.

The FLUSH\_DATA routine can also be called to periodically write the collected information to the profiler tables. This might be useful if the STOP\_PROFILER routine is taking an excessive amount of time when profiling larger PL/SQL routines. This routine is typically embedded in the PL/SQL code.

When the START\_PROFILER routine is started, the Oracle RDBMS collects a variety of information about the PL/SQL routine while it is being executed. The STOP\_PROFILER then stops this collection process and writes the collected information to the three profiler tables.

These tables are then examined using SQL to view the results of the profiler collection.

### Using the PL/SQL Profiler

There will be additional profiler information collected if the object being profiled has been compiled using the debug option.

This example will use my simple LOOPING\_EXAMPLE code:

```
CREATE OR REPLACE PROCEDURE looping_example
IS
    loop_counter    NUMBER := 0;
BEGIN
    FOR rec IN (SELECT *
                FROM emp)
    LOOP
        loop_counter := loop_counter + 1;
        DBMS_OUTPUT.put_line (
            'Record ' || loop_counter || ' is Employee ' || rec.ename
        );
    END LOOP;

    DBMS_OUTPUT.put_line ('Procedure Looping Example is done');
END;
```

To capture PL/SQL profile information, execute the following statements. The comment submitted with the START command will be populated into the RUN\_COMMENT in the PLSQL\_PROFILER\_RUNS table, see below.

```
SQL> execute DBMS_PROFILER.START_PROFILER('User0 Looping_Example');
SQL>
```



```
SQL> execute LOOPING_EXAMPLE:
SQL>
SQL> execute DBMS_PROFILER.STOP_PROFILER;
```

This example code and the SQL\*Plus script (runs all 3 SQL statements in an interactive script) are available from [www.DanHotka.com](http://www.DanHotka.com) . Ask me for the PROFILER\_RPT.sql script.

The profiler populates three tables with related information. PLSQL\_PROFILER\_RUNS has information about each time the profiler is started, including the comment entered when the profiler session was initiated. The PLSQL\_PROFILE\_UNITS contains information about the PL/SQL code executed during the run. Each procedure, function, and package will have its own line in this table. The PLSQL\_PROFILE\_DATA contains the executed lines of code, code execution time, and more. The following SQL is useful in extracting the profiler information.

First, find the profiler run of interest. The RUN\_COMMENT column has the

```
select runid, run_owner, run_date, run_comment
from plsql_profiler_runs;
```

RUNID	RUN_OWNER	RUN_DATE	RUN_COMMENT
1	USER0	21-SEP-07	User0 Looping_Example

```
SQL> |
```

In this SQL, enter the RUNID from the prior SQL statement. Oracle will place several lines of '<anonymous>' in the UNIT\_OWNER column. This information is the overhead that Oracle incurred executing the code, not the code itself. Since I am not interested in this clutter, I coded the SQL to just show me the profiler information of interest to me.

```
select runid, unit_number, unit_type, unit_owner, unit_name,
unit_timestamp
from plsql_profiler_units
where unit_owner <> '<anonymous>'
and runid = &rpt_runid;
```

RUNID	UNIT NUMBER	UNIT TYPE	UNIT OWNER	UNIT NAME	UNIT TIME
1	3	PROCEDURE	USER0	LOOPING_EXAMPLE	20-SEP-07

```
SQL>
```

```
select pu.unit_name, pd.line#, pd.total_time / 1000000000 total_time,
us.text text
from plsql_profiler_data pd, plsql_profiler_units pu, user_source us
where pd.runid = &rpt_runid
```

```

and pd.unit_number = &rpt_unitid
and pd.runid = pu.runid
and pd.unit_number = pu.unit_number
and us.name = pu.unit_name
and us.line = pd.line#
and us.type in ('PACKAGE BODY','PROCEDURE','FUNCTION');

```

UNIT_NAME	LINE#	TOTAL_TIME	TEXT
LOOPING_EXAMPLE	3	0	loop_counter NUMBER := 0;
LOOPING_EXAMPLE	5	.024	FOR rec IN (SELECT *
LOOPING_EXAMPLE	8	0	loop_counter := loop_counter + 1;
LOOPING_EXAMPLE	9	.001	DBMS_OUTPUT.put_line (
LOOPING_EXAMPLE	14	0	DBMS_OUTPUT.put_line ('Procedure Loop ing Example is done');
LOOPING_EXAMPLE	15	0	END;

6 rows selected.

SQL>

This code cleans up the profiler tables.


```

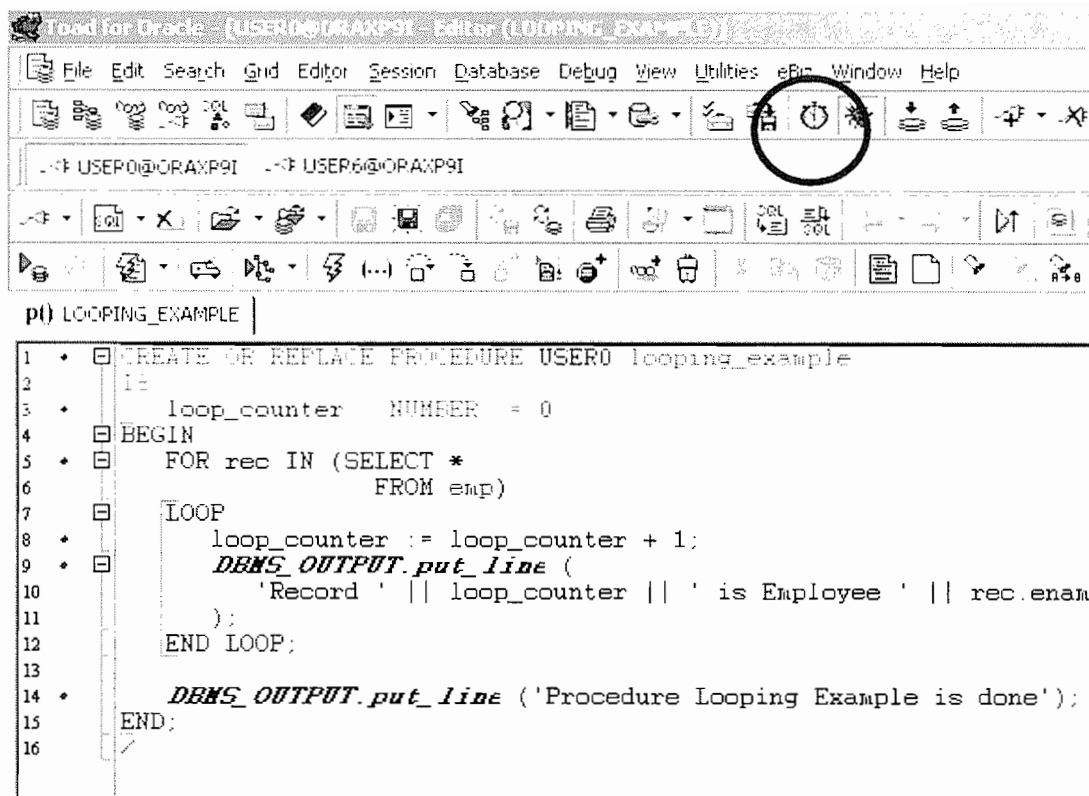
delete from plsql_profiler_data;
delete from plsql_profiler_units;
delete from plsql_profiler_runs;

```

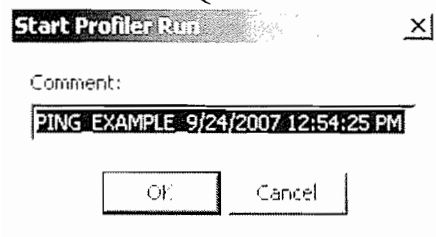
## TOAD Users

Quest Software has implemented the PL/SQL Profiler into the TOAD tool. This option has been available for quite a while. It too is easy to use and the whole process is easily handled from the TOAD environment.

Starting and stopping the profiler is easily accomplished by clicking on the Toggle PL/SQL Profiler button  .



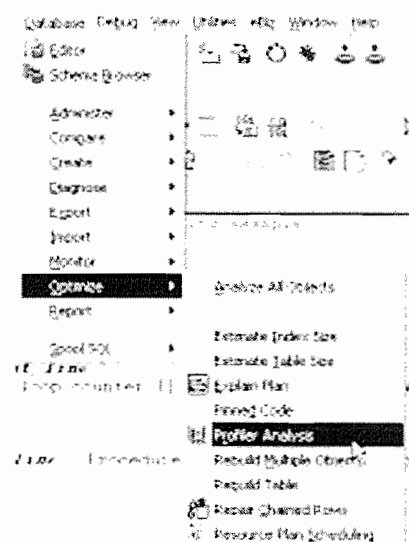
When the PL/SQL code is executed, a dialog box will popup for the start comment.

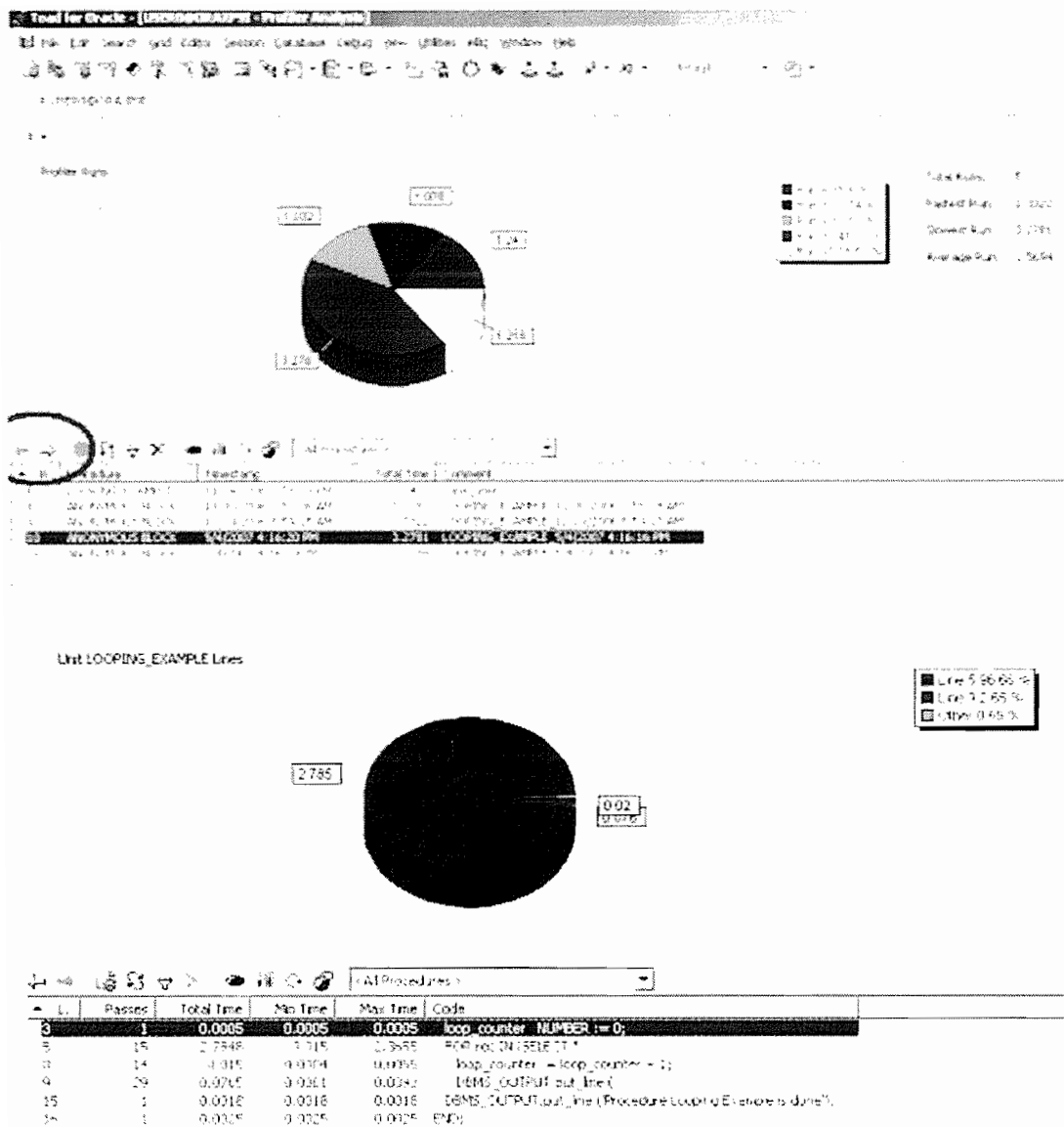


Click the Toggle PL/SQL Profiler button again to stop profiling.

The same profile tables are populated but TOAD also formats this same outout using a nice interactive wizard. Use Database → Optimize → Profiler Analysis menu item to access the Profiler Analysis.

The output is easily visible. Select your executed code from the list and click on the arrow button in the circle.





## Summary

The PL/SQL Profiler is an essential tool when tuning PL/SQL and the SQL coded into these same routines. Without something like this profiler process, it is impossible to tell where the time is spend when tuning PL/SQL code.

Dan Hotka

Dan Hotka is a Training Specialist who has over 29 years in the computer industry and over 24 years of experience with Oracle products. He is an internationally recognized Oracle expert with Oracle experience dating back to the Oracle V4.0 days. Dan's latest