Using Oracle® DBMS_SESSION
And
DBMS_APPLICATION_INFO
Built-in Packages
# TABLE OF CONTENTS

**INTRODUCTION** .................................................................................................................................... 3  
**THE DBMS_SESSION PACKAGE** ........................................................................................................... 3  
  - ROLES and Security .......................................................................................................................... 5  
  - User NLS Customization .................................................................................................................... 6  
  - Managing Session Resources ............................................................................................................ 7  
  - Who am I? ......................................................................................................................................... 7  
  - Closing Database Links ......................................................................................................................... 8  
  - Session Memory and Package State ..................................................................................................... 9  
**THE DBMS_APPLICATION_INFO PACKAGE** ...................................................................................... 12  
  - Registering Modules and Actions ......................................................................................................... 13  
  - Monitoring Application SQL Resource Consumption ........................................................................ 14  
  - Client Info and Three-tier Architectures .......................................................................................... 15  
**CONCLUSION** ....................................................................................................................................... 16  
**REFERENCES** ......................................................................................................................................... 16
INTRODUCTION
Each new release of Oracle adds to the already rich set of built-in packages as a standard part of PL/SQL functionality. DBAs and developers can benefit greatly from using packages, but a lack of time or other job pressures often prevents them from exploring the value and employing packages on a regular basis. The Oracle PL/SQL built-in packages (a.k.a. the DBMS packages) are a virtual treasure trove of useful and powerful constructs that can greatly enhance the quality and value of any PL/SQL programming effort. Understanding the Oracle built-in packages is critical to developing proficiency and productivity in PL/SQL programming.

Two of these built-in packages, DBMS_SESSION and DBMS_APPLICATION_INFO, are particularly useful for setting session parameters and monitoring performance and resource utilization. DBAs and developers are often surprised by how incredibly useful they can be.

THE DBMS_SESSION PACKAGE
The DBMS_SESSION package lets PL/SQL programmers access and change session-level settings in SQL. Among the session modifications that can be made using DBMS_SESSION are:

- Enabling and disabling roles
- Setting National Language Support (NLS) characteristics
- Resetting package states and releasing session package memory
- Setting Trusted Oracle label characteristics

DBMS_SESSION is frequently used to set preferences and security levels for a user's current database session. For example, the DBMS_SESSION.SET_ROLE procedure can be used to set required roles before performing certain tasks.

Most of the programs in DBMS_SESSION are not typically called deep in the processing layers of application code. They are more commonly called early in the application to establish an appropriate context for the end-user.

Figures 1 and 2 summarize the use and call structure of the DBMS_SESSION programs.
<table>
<thead>
<tr>
<th>Name</th>
<th>Program Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_SESSION.SET_ROLE</td>
<td>Procedure</td>
<td>Enable or disable roles for the session</td>
</tr>
<tr>
<td>DBMS_SESSION.SET_SQL_TRACE</td>
<td>Procedure</td>
<td>Turn session SQL tracing on or off</td>
</tr>
<tr>
<td>DBMS_SESSION.SET_NLS</td>
<td>Procedure</td>
<td>Set National Language Support characteristics for the session</td>
</tr>
<tr>
<td>DBMS_SESSION.CLOSE_DATABASE_LINK</td>
<td>Procedure</td>
<td>Close an inactive but open database link</td>
</tr>
<tr>
<td>DBMS_SESSION.SET_LABEL</td>
<td>Procedure</td>
<td>Set Trusted Oracle label</td>
</tr>
<tr>
<td>DBMS_SESSION.SET_MLS_LABEL_FORMAT</td>
<td>Procedure</td>
<td>Set Trusted Oracle MLS label format</td>
</tr>
<tr>
<td>DBMS_SESSION.RESET_PACKAGE</td>
<td>Procedure</td>
<td>Clear all persistent package state</td>
</tr>
<tr>
<td>DBMS_SESSION.UNIQUE_SESSION_ID</td>
<td>Function</td>
<td>Returns a unique character string for the session</td>
</tr>
<tr>
<td>DBMS_SESSION.IS_ROLE_ENABLED</td>
<td>Function</td>
<td>Returns TRUE if role enabled</td>
</tr>
<tr>
<td>DBMS_SESSION.SET_CLOSE_CACHED_OPEN_CURSORS</td>
<td>Procedure</td>
<td>Turns automatic closing of cached cursors on or off</td>
</tr>
<tr>
<td>DBMS_SESSION.FREE_UNUSED_USER_MEMORY</td>
<td>Procedure</td>
<td>Release unused session memory</td>
</tr>
</tbody>
</table>

**Figure 1: Table of programs in DBMS_SESSION**

```sql
PROCEDURE DBMS_SESSION.SET_ROLE
  (role_cmd IN VARCHAR2);
PROCEDURE DBMS_SESSION.SET_NLS
  (param IN VARCHAR2,
   value IN VARCHAR2);
PROCEDURE DBMS_SESSION.SET_SQL_TRACE
  (sql_trace IN BOOLEAN);
PROCEDURE DBMS_SESSION.SET_LABEL
  (lbl IN VARCHAR2);
PROCEDURE DBMS_SESSION.SET_MLS_LABEL_FORMAT
  (fmt IN VARCHAR2);
FUNCTION DBMS_SESSION.UNIQUE_SESSION_ID
  RETURN VARCHAR2;
FUNCTION DBMS_SESSION.IS_ROLE_ENABLED
  (rolename IN VARCHAR2)
  RETURN BOOLEAN;
PROCEDURE DBMS_SESSION.SET_CLOSE_CACHED_OPEN_CURSORS
  (close_cursors IN BOOLEAN);
PROCEDURE DBMS_SESSION.CLOSE_DATABASE_LINK
  (dblink IN VARCHAR2);
PROCEDURE DBMS_SESSION.RESET_PACKAGE;
PROCEDURE DBMS_SESSION.FREE_UNUSED_USER_MEMORY;
```

**Figure 2: DBMS_SESSION program signatures**
DBMS_SESSION programs are a kind of grab-bag of ways to alter the user’s current session characteristics. They can be roughly categorized as follows:

**SESSION SETTINGS**
- **SET_ROLE**
- **SET_NLS**
- **SET_LABEL**
- **SET_MLS_LABEL_FORMAT**

**SESSION INFORMATION**
- **UNIQUE_SESSION_ID**
- **IS_ROLE_ENABLED**

**SESSION RESOURCE UTILIZATION**
- **SET_SQL_TRACE**
- **CLOSE_DATABASE_LINK**
- **RESET_PACKAGE**
- **FREE_UNUSED_USER_MEMORY**
- **SET_CLOSE_CACHED_OPEN_CURSORS**

**ROLES AND SECURITY**
Roles are an important security component of an ORACLE application environment. The **SET_ROLE** procedure and **IS_ROLE_ENABLED** function of DBMS_SESSION provide the means to integrate role-based security into applications using PL/SQL. The **SET_ROLE** procedure is used to enable and disable roles for the current session. It is equivalent to executing the SET ROLE SQL command. The procedure call simply appends the text of the single parameter role_cmd to the string ‘SET ROLE’ and executes the resulting SQL command. In order to successfully enable a role, the user must already have been granted the role. After executing the **SET_ROLE** procedure, only the roles specified in the procedure call will be enabled.

Note the following restrictions on DBMS_SESSION.SET_ROLE:
- Cannot be called from triggers
- Cannot be called from stored procedures

These restrictions are for security purposes in the sense that they prevent users from obtaining role privileges indirectly through privileges on other objects (tables and stored procedures). In practice, this means that applications will have to set session roles using anonymous PL/SQL blocks. A few examples follow:

To set the password protected role SPECIAL_ROLE in the current session:
```
BEGIN
  DBMS_SESSION.SET_ROLE ('SPECIAL_ROLE '||
    'IDENTIFIED BY password');
END;
```

To disable all roles in the current session:
```
BEGIN
  DBMS_SESSION.SET_ROLE('NONE');
END;
```

To set multiple roles for the current session:
```
BEGIN
  DBMS_SESSION.SET_ROLE ('ROLE1, ROLE2, ROLE3');
END;
```
The role_cmd parameter is not case-sensitive, which is the default behavior for role names in the ORACLE catalog. To set a case-sensitive role, the role name should be double-quoted in the role_cmd parameter. For example:

```
BEGIN
    DBMS_SESSION.SET_ROLE
        ("special_role");
END;
```

Notice that "special_role" is double-quoted and also contained in a string literal delimited by single quotes. The IS_ROLE_ENABLED function is used to implement role-sensitive functionality in PL/SQL programs. It returns TRUE or FALSE depending on whether the role specified in the rolename parameter is enabled or not. Unlike SET_ROLE, the function can be called from triggers or stored procedures. Role-dependent logic can be implemented in PL/SQL programs with code like that in Figure 3. IS_ROLE_ENABLED is case-sensitive on the rolename parameter, so beware of unexpected results. Default behavior in the ORACLE catalog is for role names to be uppercase, so it is probably best to cast the rolename to uppercase prior to calling this function.

```
BEGIN
    IF DBMS_SESSION.IS_ROLE_ENABLED('ADMIN');
    THEN
        /*
        || do the special functions available to admin
        || users here
        */
        DBMS_OUTPUT.PUT_LINE('ADMIN IS ENABLED');
    ELSE
        /*
        || do the functions for non-admins here
        */
        DBMS_OUTPUT.PUT_LINE('ADMIN IS NOT ENABLED');
    END IF;
END;
```

Figure 3: Role dependent program logic

User NLS Customization
Oracle supports a great deal of flexibility with respect to languages, character sets and formatting preferences for dates and currencies. National Language Support (NLS) settings can be set at the session level to allow different users of the same application to view and manipulate data using their own preferred settings. The DBMS_SESSION.SET_NLS procedure is used to set or alter NLS characteristics for the current session. The following NLS session characteristics can be set using this procedure:

- NLSCALENDAR
- NLSCURRENCY
- NLSDATE_FORMAT
- NLSDATE_LANGUAGE
- NLS_ISO_CURRENCY
- NLSLANGUAGE
- NLS_SPECIAL_CHARS
- NLSSORT
- NLS_TERRITORY
DBMS_SESSION.SET_NLS cannot be called from triggers or stored procedures, so applications need to establish proper NLS settings by calling the procedure from anonymous PL/SQL blocks. In practice applications could begin by prompting the user for NLS preferences, issue the appropriate DBMS_SESSION.SET_NLS procedure calls, and then move on to the real work.

**CAUTION:** Figure 4 shows sample code and output that reveal a problem with the Oracle 7.3 version of DBMS_SESSION.SET_NLS. The NLS_DATE_FORMAT passed in is valid yet the procedure generates an unhandled exception. This problem appears to have been fixed and is no longer a problem in more recent Oracle versions.

The NLS_DATE format is particularly important in that it serves as the default date format mask for all TO_DATE and TO_CHAR functions called by the session. Applications which convert dates to and from strings using these functions may be affected by NLS setting changes. The DBMS_JOB package makes use of NLS settings, so users of that package may want to condition their job execution environments by using SET_NLS in sessions submitting jobs.

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT=’YYYY:MM:DD:HH24:MI:SS’;
Session altered.
SQL> BEGIN
2    DBMS_SESSION.SET_NLS(’NLS_DATE_FORMAT’,’”YYYY:MM:DD:HH24:MI:SS”’);
3 END;
4 /
5 BEGIN
6    *
7    ERROR at line 1:
8    ORA-06510: PL/SQL: unhandled user-defined exception
9    ORA-06512: at ”SYS.DBMS_SESSION”, line 46
10   ORA-06512: at line 2
```

**Figure 4: A bug in DBMS_SESSION under Oracle 7.3**

**MANAGING SESSION RESOURCES**

DBAs may find the most interesting programs in DBMS_SESSION to be the ones that allow manipulation and tracing of session resources. Some interesting and useful notes are presented below as a result of an in depth exploration of these programs.

**WHO AM I?**

A question that seems to come up quite often is "how can I tell what is my current session id (sid) in the database?" One reason why people want to know this is that many of the V$ tables have a sid column and they want to query the tables for statistics and information on their session. I’ve created a package called MY_Session from which many of the examples in this paper are drawn. Figure 5 shows one of the most useful programs from this package. It is called simply SID, and returns the current session ID.

The SID function has its purity declared using the PRAGMA RESTRICT_REFERENCES compiler directive, which enables its use directly in SQL statements, such as:

```
SELECT name, value
FROM v$sysstat
WHERE sid = my_session.sid;
```

As you can see, SID is a handy little function to have.
CLOSING DATABASE LINKS

The database initialization parameter OPEN_LINKS controls the maximum number of concurrent open connections to remote databases per user session. When a session exceeds this number, the following ORACLE error is raised:

**ORA-02020: too many database links in use**

Generally, the DBA should set OPEN_LINKS to a value which will accommodate the needs of distributed applications accessing the database. However, in highly distributed environments with multiple applications, it's possible that users could receive the ORA-02020 error. Presumably, this is the purpose of the CLOSE_DATABASE_LINK procedure; however there is a serious problem. Quite simply, users should not have to know anything about database links, nor for that matter, should applications. Database links are an implementation detail which should be kept transparent to users and applications. So the real question is, when an ORA-02020 is incurred, how is a user or application supposed to know which links are open and which links can be closed?

Clearly, users should not have to know about specific database links and yet should be able to do something in case of an ORA-02020 error. In order to address this situation, the CLOSE_LINKS procedure was written. CLOSE_LINKS enhances DBMS_SESSION.CLOSE_DATABASE_LINK as follows:

- Close any open links which can be closed
- Close all open links (when force_with_commit_TF is TRUE)

Figure 6 contains the source code for CLOSE_LINKS. There are a few things to notice in this procedure. First, exceptions are declared and assigned to the two ORACLE errors which can be raised by DBMS_SESSION.CLOSE_DATABASE_LINK. This is done using PRAGMA EXCEPTION_INIT compiler directives. Next is a loop through all database links available to the user. For each link in the loop, we execute DBMS_SESSION.CLOSE_DATABASE_LINK in a BEGIN...END block and trap the exceptions raised by links which are not open or in use. Trapping the exceptions allows the loop to continue until all links have been processed. Database links are considered in use if they have been referenced in the current transaction. Therefore, CLOSE_LINKS can close all links by ending the current transaction with a COMMIT. Since a COMMIT is not something that should be done unconditionally, this behavior is controlled by the force_with_commit_TF parameter.
SESSION MEMORY AND PACKAGE STATE
The FREE_UNUSED_USER_MEMORY procedure of DBMS_SESSION is used to reclaim user session memory, especially when memory intensive operations have been performed in the session. Oracle recommends that this procedure be used only where memory is at a premium and the session has allocated large (i.e. > 100 Kb) amounts. Calling the procedure forces Oracle to de-allocate any session memory previously allocated but currently unused. Some examples of operations which can lead to large memory allocations are:

- In-memory sorts where the SORT_AREA_SIZE initialization parameter is large
- Compilation of large PL/SQL packages, procedures, and functions

```sql
PROCEDURE close_links
(force_with_commit_TF IN BOOLEAN DEFAULT FALSE)
/
|| closes all open database links not in use by session,
|| or all if force_with_commit_TF is TRUE
*/

IS
/* declare exceptions for ORA errors */
dblink_in_use EXCEPTION;
PRAGMA EXCEPTION_INIT(dblink_in_use,-2080);

dblink_not_open EXCEPTION;
PRAGMA EXCEPTION_INIT(dblink_not_open,-2081);

/* cursor of all db links available to user */
CURSOR all_links_cur
IS
SELECT db_link
FROM all_db_links;
BEGIN
/* try all links and close the ones you can */
FOR dblink_rec IN all_links_cur
LOOP
BEGIN
DBMS_SESSION.CLOSE_DATABASE_LINK(dblink_rec.db_link);
EXCEPTION
WHEN dblink_not_open
THEN null;
WHEN dblink_in_use
THEN
IF force_with_commit_TF
THEN
COMMIT;
DBMS_SESSION.CLOSE_DATABASE_LINK(dblink_rec.db_link);
END IF;
WHEN OTHERS
THEN null;
END;
END LOOP;
END close_links;
```

Figure 6: Procedure MY_SESSION.CLOSE_LINKS
- Use of PL/SQL tables to store large amounts of data

In each of the above cases, memory allocated for a specific purpose can only be reused by ORACLE for that same purpose. For example, memory allocated to the session for large PL/SQL compilations will not be reused later for anything but compilation of PL/SQL. If no more compilation is anticipated and memory is scarce, this is a good time to call FREE_UNUSED_USER_MEMORY.

In the case of PL/SQL tables, the scope of the table determines when memory allocated to the table is no longer in use and can be freed. Memory allocated for tables declared locally to modules (i.e. procedures and functions) becomes unused once the module completes execution. Other PL/SQL tables, local or global, can then make use of this memory, however it remains allocated to the session. Global tables (i.e. tables declared at the package level) have persistent scope and the memory associated with them can only be made reusable by assigning a null table to them. In either case, once the memory is made reusable it is also eligible to be freed (de-allocated form the session) using the FREE_UNUSED_USER_MEMORY procedure. Memory freed by calling FREE_UNUSED_USER_MEMORY is returned either to the operating system or to the ORACLE shared pool. This depends on the ORACLE configuration and session connection as follows:

- Sessions connected through multi-threaded servers return memory to the shared pool
- Sessions with dedicated server connections return memory to the operating system

Session memory can be monitored through the statistics for "session uga memory" and "session pga memory" found in the V$SESSTAT virtual table. Figure 7 shows a useful procedure from the MY_SESSION package that can be used to monitor my session memory in SQL*Plus.

```sql
PROCEDURE memory
/*
 || displays session uga and pga using dbms_output
 */
IS
BEGIN
    DBMS_OUTPUT.ENABLE;
    DBMS_OUTPUT.PUT_LINE('session UGA: ' || TO_CHAR(my_session.statval('session uga memory') ) );
    DBMS_OUTPUT.PUT_LINE('session PGA: ' || TO_CHAR(my_session.statval('session pga memory') ) );
END memory;
```

**Figure 7: Procedure MY_SESSION.MEMORY**

The DBMS_SESSION.RESET_PACKAGE procedure invalidates all package states, including all global variables and PL/SQL tables. This procedure should be used with great caution! It will cause the loss of package state in all packages for the current session. Applications making use of persistent package variables may be compromised and fail to work properly after calls to RESET_PACKAGE. Once the session's package state has been invalidated, memory allocated for persistent package variables can be freed using FREE_UNUSED_USER_MEMORY. If you're going to blow away package state, you might as well free memory also, so the RESET procedure (Figure 8) combines the two.
Figure 8: MY_SESSION.RESET

Note that no package states can be reestablished until the outermost PL/SQL calling scope within which
RESET_PACKAGE was called ends. The PL/SQL block in Figure 9 will never print either message because both
are contained within the same scope as the call to RESET_PACKAGE and the DBMS_OUTPUT buffer cannot be
established within the scope of the block.

BEGIN
    DBMS_SESSION.RESET_PACKAGE;
    DBMS_SESSION.FREE_UNUSED_USER_MEMORY;
END reset;

Figure 9: This block of code will never print either message
THE DBMS_APPLICATION_INFO PACKAGE

The DBMS_APPLICATION_INFO package provides procedures that allow applications to “register” their current execution status with the ORACLE database. Once registered, information about the status of an application can be externally monitored through several of the V$ virtual tables. The package can be used to develop applications that can track and monitor:

- Module usage (e.g. where do users spend their time in the application?)
- Resource accounting by transaction and module
- End-users in three-tier architectures
- Incremental changes in long-running process statistics

Applications registered using DBMS_APPLICATION_INFO can be analyzed for performance and resource consumption by DBAs and developers much more closely than is otherwise possible. This facilitates better application tuning, as well as enabling more accurate usage-based cost accounting.

Figures 10 and 11 summarize the use and call structure of the DBMS_APPLICATION_INFO programs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Program Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_APPLICATION_INFO.SET_MODULE</td>
<td>Procedure</td>
<td>Set name of module executing</td>
</tr>
<tr>
<td>DBMS_APPLICATION_INFO.SET_ACTION</td>
<td>Procedure</td>
<td>Set action within module</td>
</tr>
<tr>
<td>DBMS_APPLICATION_INFO.READ_MODULE</td>
<td>Procedure</td>
<td>Read module and action for current session</td>
</tr>
<tr>
<td>DBMS_APPLICATION_INFO.SET_CLIENT_INFO</td>
<td>Procedure</td>
<td>Set client info for session</td>
</tr>
<tr>
<td>DBMS_APPLICATION_INFO.READ_CLIENT_INFO</td>
<td>Procedure</td>
<td>Read client info for session</td>
</tr>
<tr>
<td>DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS</td>
<td>Procedure</td>
<td>Set row in LONGOPS table (v8.0 only)</td>
</tr>
</tbody>
</table>

Figure 10: Table of programs in DBMS_APPLICATION_INFO
REGISTERING MODULES AND ACTIONS
The SET_MODULE and SET_ACTION procedures of DBMS_APPLICATION_INFO are used to set, or register, a module name for the currently executing program and an action name for the current transaction within the program. Registered module and action names appear in the MODULE and ACTION columns of the V$SESSION and V$SQLAREA virtual tables.

Oracle recommends that the module name correspond to a recognizable name for the program or logical application unit which is currently executing. They provide examples, such as the name of the form executing in a Forms application or the name of a sql script executing under SQL*Plus. The action name should be set to a name that can identify the current transaction or logical unit of work within the module.

When the module terminates, DBMS_APPLICATION_INFO.SET_MODULE should be called again and null values passed for both parameters. Doing this prevents subsequent transactions and programs from being incorrectly counted as part of the current module in the event they do not properly register themselves using DBMS_APPLICATION_INFO. If the program has an EXCEPTION section, the module and action information should be reset there as well.

The READ_MODULE procedure is used to read the current module and action settings for the session. Figure 12 illustrates how a procedure to award employee bonuses might register an unregister itself.

```
PROCEDURE DBMS_APPLICATION_INFO.SET_MODULE
    (module_name IN VARCHAR2
    ,action_name IN VARCHAR2);

PROCEDURE DBMS_APPLICATION_INFO.SET_ACTION
    (action_name IN VARCHAR2);

PROCEDURE DBMS_APPLICATION_INFO.READ_MODULE
    (module_name OUT VARCHAR2
    ,action_name OUT VARCHAR2);

PROCEDURE DBMS_APPLICATION_INFO.SET_CLIENT_INFO
    (client_info IN VARCHAR2);

PROCEDURE DBMS_APPLICATION_INFO.READ_CLIENT_INFO
    (client_info OUT VARCHAR2);

PROCEDURE DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS
    (hint IN OUT BINARY_INTEGER
    ,context IN NUMBER DEFAULT 0
    ,stepid IN NUMBER DEFAULT 0
    ,stepsofar IN NUMBER DEFAULT 0
    ,steptotal IN NUMBER DEFAULT 0
    ,sofar IN NUMBER DEFAULT 0
    ,totalwork IN NUMBER DEFAULT 0
    ,application_data_1 IN NUMBER DEFAULT 0
    ,application_data_2 IN NUMBER DEFAULT 0
    ,application_data_3 IN NUMBER DEFAULT 0);
```

Figure 11: DBMS_APPLICATION_INFO program signatures
MONITORING APPLICATION SQL RESOURCE CONSUMPTION

When applications make use of DBMS_APPLICATION_INFO to register themselves, DBAs can monitor application usage and resource consumption through the V$SESSION and V$SQLAREA virtual tables. The following is a simple report which summarizes SQL resource consumption data by application, module and action. Such reports can serve a number of useful purposes, including:

- Identifying tuning opportunities
- Quantifying utilization levels by application component
- Implementing chargeback schemes

A simple user written SQL script called REGTEST.SQL which performs several DDL and DML operations and registers itself using DBMS_APPLICATION_INFO serves to illustrate this point. REGTEST was run a few times before executing the SQLAREA.SQL script. This script queries the V$SQLAREA virtual table and summarizes several SQL resource consumption metrics by MODULE and ACTION. The output shows how application performance and impact analysis can be accomplished at a fine level of granularity using registration. Figure 13 shows both SQLAREA.SQL and the results. Note that doing resource consumption analysis like that illustrated in Figure 13 is different from tracing sessions and using TKPROF in that resource metrics are aggregated across all sessions executing registered modules, not just a single session.

CREATE OR REPLACE PROCEDURE award_bonus
  (empno_IN IN NUMBER,
pct_IN IN NUMBER)
IS
  BEGIN
    DBMS_APPLICATION_INFO.SET_MODULE
      (module_name => 'EMPLOYEE UPDATE',
       action_name => 'AWARD BONUS');

    UPDATE emp
      SET sal = sal*(1+pct_IN/100)
     WHERE empno = empno_IN;
    COMMIT;

    DBMS_APPLICATION_INFO.SET_MODULE(null,null);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_APPLICATION_INFO.SET_MODULE(null,null);
  END award_bonus;

Figure 12: Registering module and action names
PROCEDURE set user(app user IN IN VARCHAR2)
IS
BEGIN
DBMS_APPLICATION_INFO.SET_CLIENT_INFO(app_user_IN);
END set_user;

CLIENT INFO AND THREE-TIER ARCHITECTURES
It’s not unusual for DBAs to struggle with the difficult issue of tracking down specific users in the following types of applications:

- Three-tier applications where users do not connect to ORACLE directly but through proxy connections
- Applications where all users connect to ORACLE under a common username -- security and user-differentiation are maintained entirely within the application

In both situations, it is a challenge for DBAs to correlate specific end-users with database sessions. In the first case, sessions are persistent and serve different users at different times and sometimes no user at all. In the second, all sessions connect to a common username and are indistinguishable in V$SESSION. Figure 14 illustrates how SET_CLIENT_INFO can provide a simple solution to this problem.

```
rem ==============================================================
rem SQLAREA.SQL
rem Simple report from V$SQLAREA on SQL resource utilization by module and action
rem ==============================================================

col module format a15
col action format a15

SELECT  module,
    action,
    SUM(buffer_gets) buffer_gets,
    SUM(rows_processed) rows_processed,
    SUM(disk_reads) disk_reads
FROM  sys.v_$sqlarea
WHERE  module IS NOT NULL
AND    action IS NOT NULL
GROUP BY module, action;

SQL> @sqlarea

MODULE ACTION BUFFER GETS ROWS_PROCESSED DISK_READS
------------ ----------- ----------- -------------- ------------
REGTEST.SQL BEGIN 0 7 0
REGTEST.SQL CREATE 0 7 0
REGTEST.SQL DELETE 1014 313 0
REGTEST.SQL DROP 0 7 0
REGTEST.SQL UPDATE 6721 308 33
5 rows selected.
```

Figure 13: Module/action resource analysis from V$SQLAREA

Figure 14: A simple solution to the challenge of establishing client identity
Calling the SET_USER procedure when users first connect to the application allows DBAs to externally
distinguish between user sessions in V$SESSION by examining the CLIENT_INFO column. For example,
suppose users JOE SMITH and SALLY DALLY log into the application, which connects to ORACLE as the user
OPBIP. If the application uses the SET_USER procedure at login, the database sessions for Joe and Sally can be
distinguished in V$SESSION as shown below:

```sql
SQL> SELECT sid, username, client_info
2 FROM v$session
3 WHERE username='OPBIP';
SID USERNAME CLIENT_INFO
-------- -------- ------------------------
14 OPBIP JOE SMITH
24 OPBIP SALLY DALLY
```

Once the client's have registered, DBAs can tell one from the other. **DBAs in three-tier or single-login
environments will want to encourage application developers to use client registration.**

**CONCLUSION**

There is a lot more to explore in the Oracle built-in packages and they continue to grow with every new Oracle
release. This paper provides some useful insights and illustrative techniques for using DBMS_SESSION and
DBMS_APPLICATION_INFO to develop more efficient, secure and traceable applications but these are only two
of the many built-in packages that are currently available. If time permits, readers are encouraged to investigate
some of the other powerful and time-saving features of packages on their own. The benefits may far outweigh the
effort.

**REFERENCES**

Material for this paper was adapted from the original author’s contributions to the book *Oracle Built-in Packages*
published by O'Reilly & Associates.