

About the Author

Geoff Ingram (<mailto:geoff@dbcool.com>) is a UK-based ex-Oracle product developer who has worked as an independent Oracle consultant since leaving Oracle Corporation in the mid-nineties. He is the creator of a free Oracle performance tuning tool DbCool downloadable from <http://www.dbcool.com> and the author of a book published by John Wiley and Sons Inc in September 2002, covering Oracle performance and availability features in Oracle8i and Oracle9i (including Release 2):

High-Performance Oracle: Proven Methods for Achieving Optimum Performance and Availability

ISBN: 0-471-22436-7

<http://www.wiley.com/cda/product/0,,0471224367,00.html>

PL/SQL Code Profiling

© Geoff Ingram, Proton Technology Ltd 2002. No part of this document may be reproduced for any purpose without the express written permission of Proton Technology Limited.

No matter what programming language you choose for your Oracle client or server side programming, it's important that you use the appropriate tools in your development environment to check for things like code coverage, code performance, and memory usage. Code coverage involves recording how many times each line of code is executed and how long was spent on execution; it's useful for both performance analysis and code quality analysis. Code coverage should be measured during regression testing of your application. If you don't explicitly measure whether your application code is covered by regression tests, then it's difficult to guarantee that the code actually works as designed, and whether it will be affected (that is, regress) as result of future code changes. The tools available are determined by the programmatic interface you use and whether you're prepared to spend money to buy them.

For PL/SQL, Oracle provides the DBMS_PROFILER package, which can be used to identify PL/SQL performance bottlenecks and code coverage by reporting execution counts and times for statements in PL/SQL procedures. The following scripts need to be run as SYS to install the profiler tables and packages:

```
$ORACLE_HOME/rdbms/admin/profload.sql
```

```
$ORACLE_HOME/rdbms/admin/proftab.sql
```

In the simplest case, profiling can be started and stopped in a session as follows:

```
begin

  sys.dbms_profiler.start_profiler;

  -- run some PL/SQL to profile here...
  sp_profile_me;

  sys.dbms_profiler.stop_profiler;

end;
```

Usually, additional arguments are provided in calls to `START_PROFILER`, to tag profiled time periods with string identifiers through the `RUN_COMMENT` and `RUN_COMMENT1` arguments. These tags can be used later to recall performance profile information from previous runs. By default, `SYSDATE` is used as the value of `RUN_COMMENT`, if it's not supplied. Each profiler run generates a unique numeric code called the `RUNID`, which is returned by some of the overloaded versions of `START_PROFILER` and `STOP_PROFILER`. `RUNID` is useful as a system-generated unique key for each run.

NOTE

Overloading is a language feature that enables two or more procedures to exist with the same names but different parameter lists. The calling context determines which version is used.

The `PLSQL_PROFILER_RUNS` table contains top-level details of each run, including the total time, the session owner who started profiling, the `RUNID`, and the `RUN_COMMENT` as follows:

```
select RUNID, RUN_OWNER, RUN_DATE, RUN_COMMENT,
       round(RUN_TOTAL_TIME/1E9, 3) seconds
from SYS.PLSQL_PROFILER_RUNS;
```

RUNID	RUN_OWNER	RUN_DATE	RUN_COMMENT	SECONDS
1	SYS	28-DEC-2001 20:26:59	28-DEC-01	1065.049

Triggers defined using `AFTER LOGON` and `AFTER LOGOFF` triggers can be used to start and stop profiling at the session level for complete sessions. The following triggers start and stop profiling for all of `SCOTT`'s sessions:

```
create or replace trigger on_logon
after logon on scott.schema
declare
err number;
begin
err:=dbms_profiler.start_profiler ('SCOTT: '||
to_char(sysdate, 'dd-mon-yyyy hh24:mi:ss'));
end;
/
```

```
create or replace trigger on_logoff
before logoff on scott.schema
declare
err number;
begin
err:=dbms_profiler.stop_profiler ;
end;
/
```

Several scripts are provided with the Oracle DBMS set to demonstrate profiling, including a simple example and some useful scripts to present the profiling information. Detailed information is stored in the `PLSQL_PROFILER_DATA` and `PLSQL_PROFILER_UNITS` tables. The scripts can be found in the following locations:

```
$ORACLE_HOME/plsql/demo/profdemo.sql
$ORACLE_HOME/plsql/demo/profrep.sql
```

\$ORACLE_HOME/plsql/demo/profsum.sql

The following SQL shows a summary of the time spent in seconds on each code line of a stored procedure SP_INDEX_REBUILD (defined in Appendix A) that is used to rebuild indexes:

```
select  u.unit_name, d.LINE#, d.TOTAL_OCCUR,
        round((d.TOTAL_TIME/1E9),4) total_time,
        round((d.MIN_TIME/1E9),4) min_time,
        round((d.MAX_TIME/1E9),4) max_time
  from    plsqli_profiler_units u, plsqli_profiler_data d
 where   d.RUNID=u.runid and d.UNIT_NUMBER = u.unit_number
        and d.total_occur >0
        and u.runid=4 and UNIT_NAME='SP_INDEX_REBUILD'
 order  by u.unit_number, d.line#;
```

UNIT_NAME	LINE#	TOTAL_OCCUR	TOTAL_TIME	MIN_TIME	MAX_TIME
SP_INDEX_REBUILD	4	1	.001	.001	.001
SP_INDEX_REBUILD	7	8	.5656	.0006	.5244
SP_INDEX_REBUILD	18	7	0	0	0
SP_INDEX_REBUILD	19	7	47.2495	4.3977	8.263
SP_INDEX_REBUILD	22	14	.3163	.0006	.2484
SP_INDEX_REBUILD	23	7	.0002	0	0
SP_INDEX_REBUILD	25	15	.0001	0	0
SP_INDEX_REBUILD	30	1	0	0	0

The following SQL shows more detailed information for the code lines, including the first 42 characters of the PL/SQL code:

```
select p.total_occur howmany, p.total_time time, p.line# line,
       substr(s.text, 1,42) text
  from
    (select u.unit_name, d.TOTAL_OCCUR, round((d.TOTAL_TIME/1E9),4)
     total_time, d.line#
     from plsqli_profiler_units u, plsqli_profiler_data d
     where d.RUNID=u.runid and d.UNIT_NUMBER = u.unit_number
           and d.TOTAL_OCCUR >0
           and u.runid=4) p, user_source s
 where p.unit_name = s.name(+) and p.line# = s.line (+)
 and p.unit_name='SP_INDEX_REBUILD'
 order by p.unit_name, p.line#;
```

HOWMANY	TIME	LINE	TEXT
1	.001	4	execute immediate 'alter session set sort
8	.5656	7	for rec in (
7	0	18	l_sql := rec.validate_sql;
7	47.2495	19	execute immediate l_sql;
14	.3163	22	for rec2 in (select pct_used from inde
7	.0002	23	if (100-rec2.pct_used) < p_in_rebui
15	.0001	25	execute immediate l_sql;
1	0	30	END SP_INDEX_REBUILD;

Summary

In order to ensure best performance and robustness of PL/SQL stored procedures in your production environments, Oracle provides the DBMS_PROFILER package. During system testing, the package can be used to identify both performance bottlenecks for executed code and code that is not executed. Code that is not executed during system testing represents a potential risk to your production environment because if it's not executed, then there's no guarantee that it will work correctly whenever it's actually used. On the other hand, code that is redundant and never used should be removed, and DBMS_PROFILER can help to identify it.

Appendix A SP_INDEX_REBUILD

```
create or replace
procedure sp_index_rebuild(p_in_rebuild_threshold_pct integer
default 0) AS
l_sql varchar2(32000);
BEGIN
    execute immediate 'alter session set sort_area_size=10000000';

    -- consider rebuild of all indexes owned by APP with > 1 extent
    for rec in (
        select 'alter index '||owner||'.'||index_name||
            ' rebuild unrecoverable' rebuild_sql,
            'analyze index '||owner||'.'||index_name||
            ' validate structure' validate_sql
        from dba_indexes where owner='APP'
        and index_type='NORMAL'
        and (owner,index_name) in
        (select owner,segment_name from dba_segments where extents > 1)
    ) loop

        -- create the index stats...
        l_sql := rec.validate_sql;
        execute immediate l_sql;

        -- only rebuild indexes where pct_used < threshold
        for rec2 in (select pct_used from index_stats) loop
            if (100-rec2.pct_used) < p_in_rebuild_threshold_pct then
                l_sql := rec.rebuild_sql;
                execute immediate l_sql;
            end if;
        end loop;
    end loop;

end sp_index_rebuild;
/
```