Tuning PL/SQL with DBMS_PROFILER

Christian Antognini
Trivadis AG
Zürich, Switzerland

Introduction

Oracle8i provides an integrated profiling and coverage tool (referred to in this document as profiler) that can be used to find performance bottlenecks in PL/SQL code and help quality assurance tests. For example it can show how many times each line was executed and the time required to execute it. For quality assurance tests, it can be used to see if all lines of a stored object have been executed.

The goal of this article is to show how to install the profiler and to give some basic information about its use.

For a complete description refer to *Oracle8i Supplied PL/SQL Packages Reference* manual.

Distribution

The following files, stored in `$ORACLE_HOME/rdbms/admin`, are needed to install the profiler:

`profload.sql`	installs the profiler server side package (the scripts executes `dbmspbp.sql` and `prvtpbp.sql`) and check the installation, if isn't successful the package is de-installed
`proftab.sql`	creates the tables where the profiling data are stored
`dbmspbp.sql`	installs the DBMS_PROFILER package specification
`Prvtpbp.plb`	installs the DBMS_PROFILER package body

The scripts are missing on some UNIX 8.1.5 distributions. You can copy the files from an NT distribution (you have to remove the ^M characters and some blank lines) or ask Oracle Support.

Installation

The script `profload.sql` must be started as SYS.

The script `proftab.sql` must be started by each user that want to use the profiler.

Upgrade

If you upgrade your database from 8.1.5 to 8.1.6, you have to manually re-install the package and re-create the tables, as both components have changed.

Tuning/Proof Session

A typical tuning/proof session is an iterative process. Each iteration is composed from the following steps:

1. start data collection
2. execute the PL/SQL code
3. stop data collection
4. analyze the collected data
5. detect and solve the problem or check the code coverage

How to execute these steps depends on which tool is used. The only condition is that steps from 1 to 3 have to be executed in the same session.

To analyse the data stored in the profiler tables you have 3 possibilities:

1. write your own report
2. use the sample report `profrep.sql` and `profsum.sql` provided by Oracle (both are stored in `$ORACLE_HOME/plsql/demo`)
3. use a third party tool

Example

To see in practice how to use the profiler let's look at a couple of examples.

The first one shows how to use it with SQL*Plus, the second one with SQL Navigator from Quest Software.

As example a typical recursive function is used. The code is the following:

```
FUNCTION factorial ( p_n IN NUMBER ) RETURN NUMBER IS
BEGIN
    IF p_n IS NULL OR p_n < 0
    THEN
        RAISE INVALID_NUMBER;
    ELSIF p_n <= 1
    THEN
        RETURN 1;
    ELSE
        RETURN factorial(p_n-1) * p_n;
    END IF;
END;
```

SQL*Plus

To profile the function in SQL*Plus the following statements have to be executed:

```
rem
rem Start data collection
rem

SELECT decode(dbms_profiler.start_profiler, 0, ’OK’, ’ERROR’) status
FROM dual;

rem
rem execute the PL/SQL code
rem

SELECT factorial(20) FROM dual;

rem
rem Stop data collection
rem

SELECT decode(dbms_profiler.stop_profiler, 0, ’OK’, ’ERROR’) status
FROM dual;

SELECT plsql_profiler_runnumber.currval runid FROM dual;
```

The return values must be checked, in fact no ORA-????? error is generated.

The following codes can be returned:

0	successful
1	incorrect parameter
2	data flush operation failed
-1	version mismatch between package and tables

When the function STOP_PROFILER is called, the data is stored in the profiler tables (they are not discussed in this article) and a RUNID (tuning session identifier) is assigned to it.

With the RUNID, retrieved with the last statement, it’s possible to select the data from the profiler tables with the following script:

```
set scan on
set verify off
set feedback off
set pagesize 50000
set linesize 120
col line format 999999 heading LINE#
col total_occur format 999,999 heading EXEC#
col total_time format 999,990.999 heading 'TIME[ms]'
col text format a80
col coverage format 90.9 heading 'COVERAGE%'

rem
rem statement to find bottlenecks
rem

select s.line, p.total_occur, p.total_time, s.text
from all_source s, (
     select u.unit_owner, u.unit_name, u.unit_type, d.line#,
            d.total_occur, d.total_time/1000000 total_time
     from plsql_profiler_data d, plsql_profiler_units u
     where u.runid = &&runid
     and u.runid = d.runid
     and u.unit_number = d.unit_number) p
where s.owner = p.unit_owner (+)
and s.name = p.unit_name (+)
and s.type = p.unit_type (+)
and s.line = p.line# (+)
and s.name = upper('&&name')
and s.owner = upper('&&owner')
order by s.line;

rem
rem statement to show coverage in %
rem

select exec.nbr/total.nbr*100 coverage
from (select count(*) nbr
      from plsql_profiler_data d, plsql_profiler_units u
      where d.runid = &&runid
      and u.runid = d.runid
      and u.unit_number = d.unit_number
      and u.unit_name = upper('&&name')
      and u.unit_owner = upper('&&owner')) total,
     (select count(*) nbr
      from plsql_profiler_data d, plsql_profiler_units u
      where d.runid = &&runid
      and u.runid = d.runid
      and u.unit_number = d.unit_number
      and u.unit_name = upper('&&name')
      and u.unit_owner = upper('&&owner')
      and d.total_occur > 0) exec;

undef runid
undef owner
undef name
```

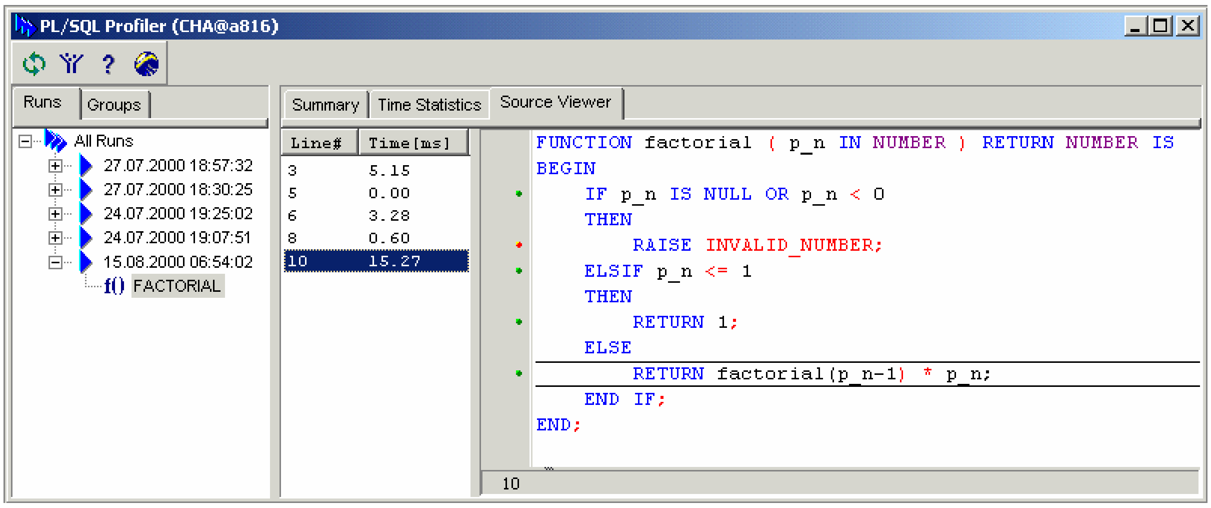
The output reports the number of execution and the time spent on each line.

```
  LINE#   EXEC# TIME[ms] TEXT
------- ------- -------- ----------------------------------------------------
      1                  FUNCTION factorial ( p_n IN NUMBER ) RETURN NUMBER IS
      2                  BEGIN
      3      20    6.066     IF p_n IS NULL OR p_n < 0
      4                      THEN
      5       0    0.000         RAISE INVALID_NUMBER;
      6      20    3.550     ELSIF p_n <= 1
      7                      THEN
      8       1    1.025         RETURN 1;
      9                      ELSE
     10      20   12.584         RETURN factorial(p_n-1) * p_n;
     11                      END IF;
     12                  END;

COVERAGE%
---------
     80.0
```

SQL Navigator

The second and more comfortable way is to use a tool like SQL Navigator from Quest Software. With version 3.2 when you execute a stored object you can directly chose if you want to enable the profiler (via the check box *Enable Profiling*). If the profiler is enabled, at the end of the execution the tool *PL/SQL Profiler* is automatically started. The following figure shows this tool after the execution of the function FACTORIAL.



Line#	Time[ms]
3	5.15
5	0.00
6	3.28
8	0.60
10	15.27

```
FUNCTION factorial ( p_n IN NUMBER ) RETURN NUMBER IS
BEGIN
    IF p_n IS NULL OR p_n < 0
    THEN
        RAISE INVALID_NUMBER;
    ELSIF p_n <= 1
    THEN
        RETURN 1;
    ELSE
        RETURN factorial(p_n-1) * p_n;
    END IF;
END;
```

10

Remarks

Sometimes the elapsed time stored in the profiler tables contains wrong (or at least very strange) values. It seams that Oracle has some problems collecting the elapsed time. Therefore I suggest you only use these values to find where the code takes more time compared with the other lines.

If the profiler must be started/stopped automatically while the users are testing and without changing the application, you can create a logon and logoff trigger like this:

```
CREATE OR REPLACE TRIGGER on_logon_trg AFTER LOGON ON DATABASE
DECLARE
  l_err NUMBER;
BEGIN
  l_err := DBMS_PROFILER.START_PROFILER;
END;
/

CREATE OR REPLACE TRIGGER on_logoff_trg BEFORE LOGOFF ON DATABASE
DECLARE
  l_err NUMBER;
BEGIN
  l_err := DBMS_PROFILER.STOP_PROFILER;
END;
/
```

Conclusion

PL/SQL developers have waited for long time such a utility. Although the implementation is not perfect, it seems that Oracle recognizes that they must provide the developers a better programming environment. Unfortunately it has taken a long time…