PL/SQL Tuning

accenture

consulting | technology | outsourcing

Introduction

Radu Pârvu

Speaker Introduction

Radu Pârvu
Manager
Oracle DTE

accenture

Areas of Deep Expertise

Database Expertise:
Data migrations, database upgrades, backups, cloning and recovery
Oracle installation, configuration, patching and database tuning
Data Replication: Golden Gate
Enterprise Manager Grid Control (OEM 12c)
Business Intelligence and Data Warehousing:
Engineered Systems: Exadata
MAA and DR: Maximum Availability Architecture and Disaster Recovery

Professional Background

Radu has over 13 years of experience in various database projects (mainly Oracle but other DBMS, too).
Radu holds License (one degree over Bachelor) of Engineering in Technological Physics from University of Bucharest.

Oracle Community

- Speaker at OUGF, BGOUG, OOW 2012
- Frequent Speaker at Accenture Finland Oracle Community Sessions
- Speaker for the Accenture WorldWide Oracle Community of Practice
- DB related blog: http://db.parvu.org/

Agenda

- Performance tuning overview
- PL/SQL Profiler
- PL/SQL Hierarchical Profiler
  - configuring the schema
  - collecting profiler data
  - understanding and interpreting the profiler data
  - using the plshprof utility
- A few other optimizer techniques
  - Use the DBMS_UTILITY.GET_TIME Function
  - quick best practices (avoid procedural code, use triggers with care and only for small code, avoid type conversions etc.)

Performance tuning overview

General steps

- Tune the access to code and data in SGA
- SQL Optimize
- Adjust the compiler optimization level
- Apply best practices and standards
- Check the execution profile
- Algorithms
- Use PL/SQL performance features
- Validate the memory consumption. Done together with DBA. Implementation of PL/SQL features require sometimes a lot of PGA and maybe SGA. The application memory consumption have to be reasonable from system point of view.

Main Question: What runs slowly?

Profilers

- DBMS_PROFILER
  - Legacy profiler
  - Switches on execution profiling at session level
- DBMS_HPROF (hierarchical profiler)
  - Hierarchical profiler introduced on 11g
  - Rolls up results through the execution stack

DBMS_PROFILER

DBMS_PROFILER

Setup

- Install the program. Run the script as SYSDBA account:
  `$ORACLE_HOME/rdbms/admin/profload.sql`
- Verify. The statement below should not return error.
  `SQL> DESC DBMS_PROFILER`
- Create in own schema the tables populated by the profiler. Run:
  `$ORACLE_HOME/rdbms/admin/proftab.sql`
- three tables populated by DBMS_PROFILER will be created:
  - PLSQL_PROFILER_RUNS - Parent table of runs
  - PLSQL_PROFILER_UNITS - Program units executed in run
  - PLSQL_PROFILER_DATA -Profiling data for each line in a program unit

DBMS_PROFILER

Usage

- After the setup is completed, you start gathering profiling information for your application by writing code like this:

```
BEGIN
  DBMS_PROFILER.START_PROFILER (
  'Radus app' || TO_CHAR (SYSDATE, 'YYYYMMDD
  HH24:MI:SS')
  );
  radu_procedural_code;
  DBMS_PROFILER.STOP_PROFILER;
END;
```

DBMS_PROFILER

Usage

- Find the profiler run of interest by checking the run_comment_column, make a note of the run_id:

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

- Next, enter the RUNID from the prior SQL statement. Oracle will place several lines of ‘’ in the UNIT_OWNER column. This information is the overhead that Oracle incurred executing the code, not the code itself. We are not interested in this:

DBMS_PROFILER

Usage

```
select runid, unit_number, unit_type, unit_owner,
unit_name, unit_timestamp

from plsql_profiler_units

where unit_owner <> ''

and runid = <collected run id>;
```

DBMS_PROFILER

Usage

- Next check the run data with a query like:

```
select pu.unit_name, pd.line#, pd.total_occur passes,
round(pd.total_time / 1000000000,5) total_time, us.text text

from plsql_profiler_data pd, plsql_profiler_units pu, user_source
us

where pd.runid = <collected run id>

and pd.unit_number = <unit ids of interest>

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');
```

DBMS_HPROF

DBMS_HPROF, the hierarchical profiler

Intro

- Oracle DB 11g introduced a second profiling mechanism: DBMS_HPROF
- Used to obtain the execution profile of PL/SQL code, organized by the distinct subprogram calls in your application
- Unlike DBMS_PROFILER that record the time that your application spends within each subprogram, down to the execution time of each individual line of code, sometimes you also want to know how much time the application spends within a particular subprogram

DBMS_HPROF, the hierarchical profiler

Intro

- Reports performance information about each subprogram in your application that is profiled, keeping SQL and PL/SQL execution times distinct.
- The profiler tracks a wide variety of information, including:
  - the number of calls to the subprogram
  - the amount of time spent in that subprogram
  - the time spent in the subprogram's subtree (that is, in its descendent subprograms)
  - detailed parent children info

DBMS_HPROF, the hierarchical profiler

Components

- **Data collector**
  - Turns hierarchical profiling on and off
  - The PL/SQL runtime engine writes the “raw” profiler output to the specified file.
- **Analyzer**
- Processes the raw profiler output and stores the results in hierarchical profiler tables from which profiler information can be displayed.

DBMS_HPROF, the hierarchical profiler

Usage

- execute rights on the DBMS_HPROF package

```
SQL> conn sys as sysdba
Enter password:
Connected.
SQL> grant execute on dbms_hprof to scott;
```

- WRITE privileges on the directory that you specify when you call DBMS_HPROF.START_PROFILING
- create the three profiler tables (see details below)
- call the DBMS_HPROF.START_PROFILING procedure to start the hierarchical profiler data collection in your session.

DBMS_HPROF, the hierarchical profiler

```
SQL> desc dbms_hprof
FUNCTION ANALYZE RETURNS NUMBER
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 LOCATION                       VARCHAR2                IN
 FILENAME                       VARCHAR2                IN
 SUMMARY_MODE                   BOOLEAN                 IN     DEFAULT
 TRACE                          VARCHAR2                IN     DEFAULT
 SKIP                           BINARY_INTEGER          IN     DEFAULT
 COLLECT                        BINARY_INTEGER          IN     DEFAULT
 RUN_COMMENT                    VARCHAR2                IN     DEFAULT
 PROFILE_UGA                    BOOLEAN                 IN     DEFAULT
 PROFILE_PGA                    BOOLEAN                 IN     DEFAULT
PROCEDURE START_PROFILING
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 LOCATION                       VARCHAR2                IN     DEFAULT
 FILENAME                       VARCHAR2                IN     DEFAULT
 MAX_DEPTH                      BINARY_INTEGER          IN     DEFAULT
 PROFILE_UGA                    BOOLEAN                 IN     DEFAULT
 PROFILE_PGA                    BOOLEAN                 IN     DEFAULT
PROCEDURE STOP_PROFILING

SQL> create directory tmp_dir as '/home/oracle/tmp';

Directory created.
```

DBMS_HPROF, the hierarchical profiler

Usage

- run your code long and repetitively enough
- call the DBMS_HPROF.STOP_PROFILING procedure to terminate the gathering of profile data
- analyze the contents and then run queries against the profiler tables

DBMS_HPROF, the hierarchical profiler

Create the three profiler tables

- Run the dbmshptab.sql script (located in the rdbms/admin directory)
- This script will create these three tables:
  - DBMSHP_RUNS: Top-level information about each run of the ANALYZE utility of DBMS_HPROF.
  - DBMSHP_FUNCTION_INFO: Detailed information about the execution of each subprogram profiled in a particular run of the ANALYZE utility
  - DBMSHP_PARENT_CHILD_INFO: Parentchild info for each subprog profiled in DBMSHP_FUNCTION_INFO.

DBMS_HPROF, the hierarchical profiler

```
[oracle@localhost admin]$ pwd
/home/oracle/app/oracle/product/11.2.0/dbhome_2/rdbms/admin
[oracle@localhost admin]$ ls dbmsh*
dbmshae.sql   dbmshm.sql    dbmshpro.sql   dbmshrep.sql  dbmshsld.sql  dbmshssn.sql  dbmshsxp.sql
dbmshias.sql  dbmshord.sql  dbmshptab.sql  dbmshrmg.sql  dbmshsna.sql  dbmshs.sql    dbmshtdb.sql
[oracle@localhost admin]$ exit
exit

SQL> ho pwd
/home/oracle

SQL> @/home/oracle/app/oracle/product/11.2.0/dbhome_2/rdbms/admin/dbmshptab.sql
drop table dbmshp_runs                      cascade constraints
           *
ERROR at line 1:
ORA-00942: table or view does not exist


drop table dbmshp_function_info             cascade constraints
           *
ERROR at line 1:
ORA-00942: table or view does not exist


drop table dbmshp_parent_child_info         cascade constraints
           *
ERROR at line 1:
ORA-00942: table or view does not exist
```

DBMS_HPROF, the hierarchical profiler

Test the performance of my procedure

- Start profiling:

```
BEGIN
  DBMS_HPROF.start_profiling ('TEMP_DIR', 'my_plsql_trace.txt');
END;
/
```

- Call my PLSQL program:

```
BEGIN
  my_proc ('param1');
END;
/
```

DBMS_HPROF, the hierarchical profiler

Test the performance of my procedure

- Stop profiling session:

```
BEGIN
  DBMS_HPROF.stop_profiling;
END;
/
```

- Now the trace file is updated. It might be possible to open it and check it, but the most clever way would be to use the ANALYZE utility of the package DBMS_HPROF. This will take the trace file contents, tranforms it and places it into the three profiler tables. **Important:** it will return a run number to be used when querying those tables.

DBMS_HPROF, the hierarchical profiler

```
SQL> ed
Wrote file afiedt.buf

  1  begin
  2             dbms_hprof.start_profiling ('TMP_DIR','my_plsql_trace.txt');
  3* end;
SQL> l
  1  begin
  2             dbms_hprof.start_profiling ('TMP_DIR','my_plsql_trace.txt');
  3* end;
SQL> /

PL/SQL procedure successfully completed.

SQL> begin
  2  test;
  3  end;
  4  /

PL/SQL procedure successfully completed.

SQL> begin
  2  dbms_hprof.stop_profiling;
  3  end;
  4  /

PL/SQL procedure successfully completed.
```

DBMS_HPROF, the hierarchical profiler

```
P#V PLSHPROF Internal Version 1.0
P#! PL/SQL Timer Started
P#C PLSQL."".""."__plsql_vm"
P#X 3
P#C PLSQL."".""."__anonymous_block"
P#X 45
P#C PLSQL."SCOTT"."TEST"::7."TEST"#980980e97e42f8ec #1
P#X 3
P#C PLSQL."SCOTT"."TEST"::7."TEST.FOO"#980980e97e42f8ec #7
P#X 30
P#C SQL."SCOTT"."TEST"::7."__static_sql_exec_line11" #11
P#X 688200
P#R
P#X 30
P#R
P#X 3
P#C PLSQL."SCOTT"."TEST"::7."TEST.FOO"#980980e97e42f8ec #7
P#X 6
P#C SQL."SCOTT"."TEST"::7."__static_sql_exec_line11" #11
P#X 72
P#R
P#X 2
P#R
P#X 0
P#C PLSQL."SCOTT"."TEST"::7."TEST.FOO"#980980e97e42f8ec #7
P#X 2
P#C SQL."SCOTT"."TEST"::7."__static_sql_exec_line11" #11
P#X 28
--More--(73%)
```

DBMS_HPROF, the hierarchical profiler

Interpreting the profiler data

- Analyze:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE (
  DBMS_HPROF.ANALYZE ('TEMP_DIR', 'my_plsql_trace.txt'));
END;
/
```

```
SQL> ed
Wrote file afiedt.buf

  1  BEGIN
  2  DBMS_OUTPUT.Put_LINE (
  3  DBMS_HPROF.ANALYZE ('TMP_DIR', 'my_plsql_trace.txt'));
  4* END;
  5  /

PL/SQL procedure successfully completed.
```

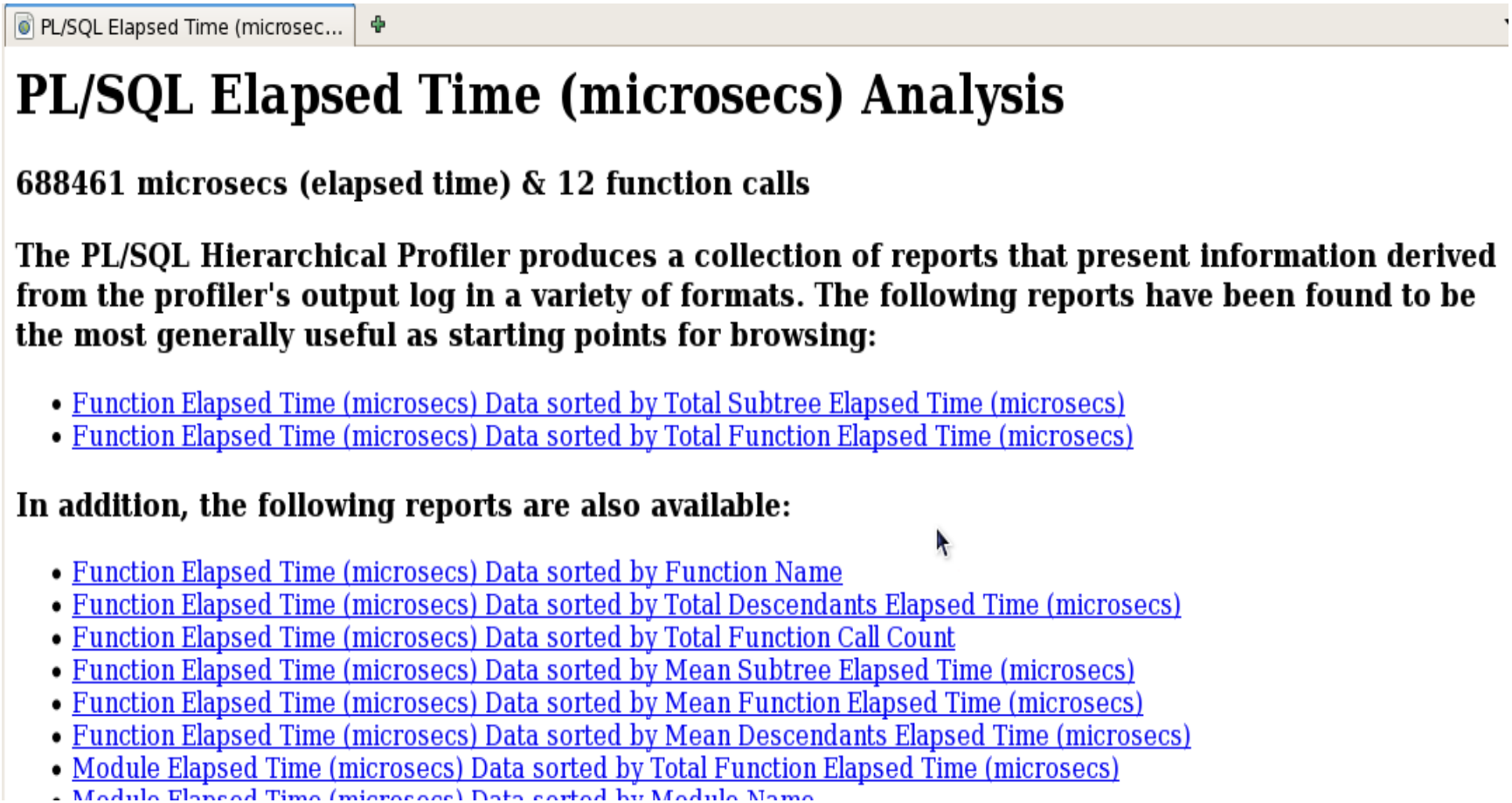
DBMS_HPROF, the hierarchical profiler

Generate simple HTML report

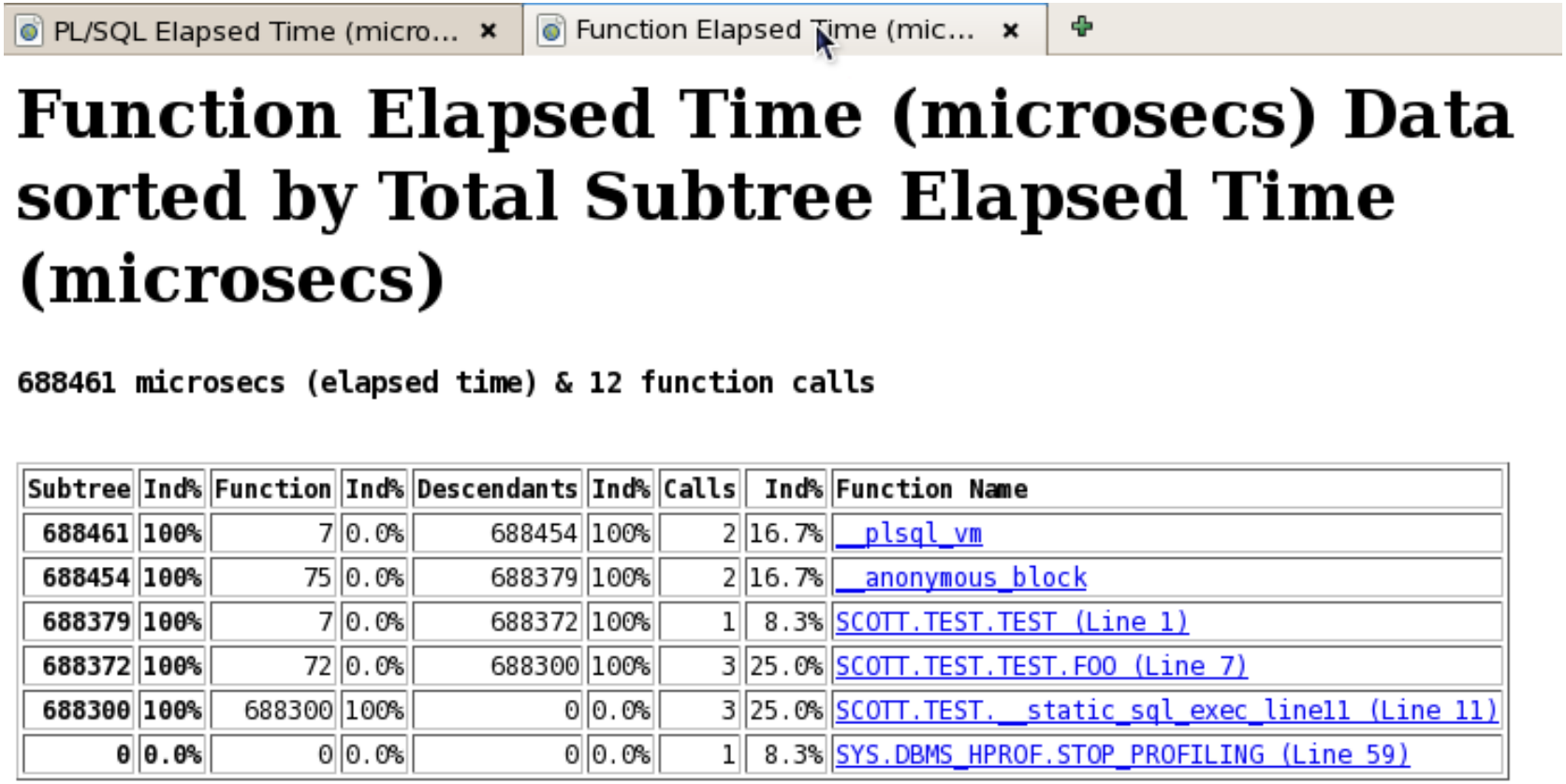
- Achieved by running the plshprof command-line utility
- Located in the directory $ORACLE_HOME/bin/
- Generates simple HTML reports from either one or two raw profiler output files
- For an example of a raw profiler output file, see the section titled “Collecting Profile Data” in the Oracle Database Advanced Application Developer’s Guide
- We can use the generated HTML reports in any browser

```
[oracle@localhost tmp]$ /home/oracle/app/oracle/product/11.2.0/dbhome_2/bin/plshprof -output test.html my_plsql_trace.txt
PLSHPROF: Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - Production
[6 symbols processed]
[Report written to 'test.html.html']
[oracle@localhost tmp]$
```

DBMS_HPROF, the hierarchical profiler

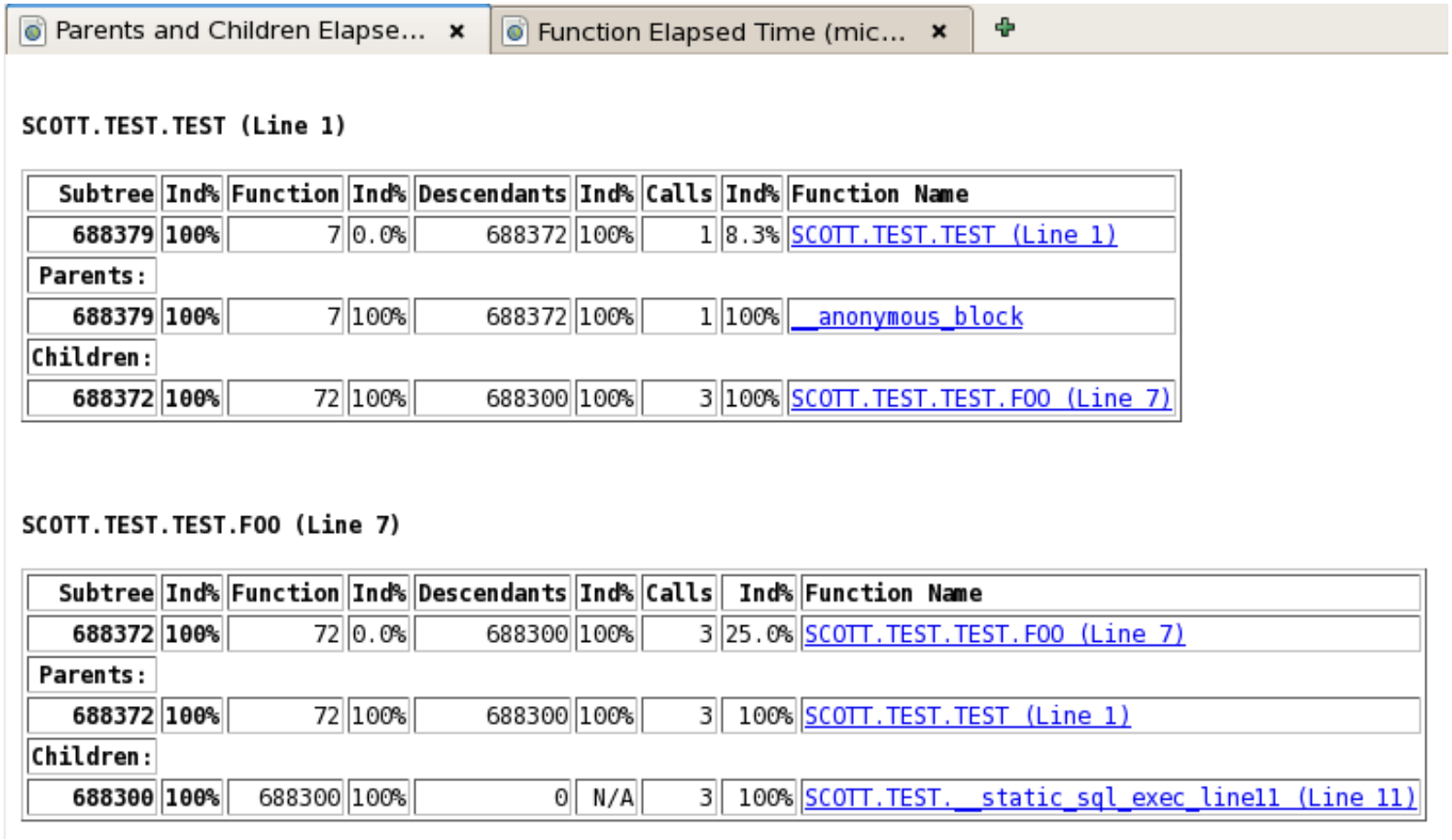


DBMS_HPROF, the hierarchical profiler



Subtree	Ind%	Function	Ind%	Descendants	Ind%	Calls	Ind%	Function Name
688461	100%	7	0.0%	688454	100%	2	16.7%	__plsql_vm
688454	100%	75	0.0%	688379	100%	2	16.7%	__anonymous_block
688379	100%	7	0.0%	688372	100%	1	8.3%	SCOTT.TEST.TEST (Line 1)
688372	100%	72	0.0%	688300	100%	3	25.0%	SCOTT.TEST.TEST.FOO (Line 7)
688300	100%	688300	100%	0	0.0%	3	25.0%	SCOTT.TEST.__static_sql_exec_line11 (Line 11)
0	0.0%	0	0.0%	0	0.0%	1	8.3%	SYS.DBMS_HPROF.STOP_PROFILING (Line 59)

DBMS_HPROF, the hierarchical profiler



Parents and Children Elapse... | Function Elapsed Time (mic...

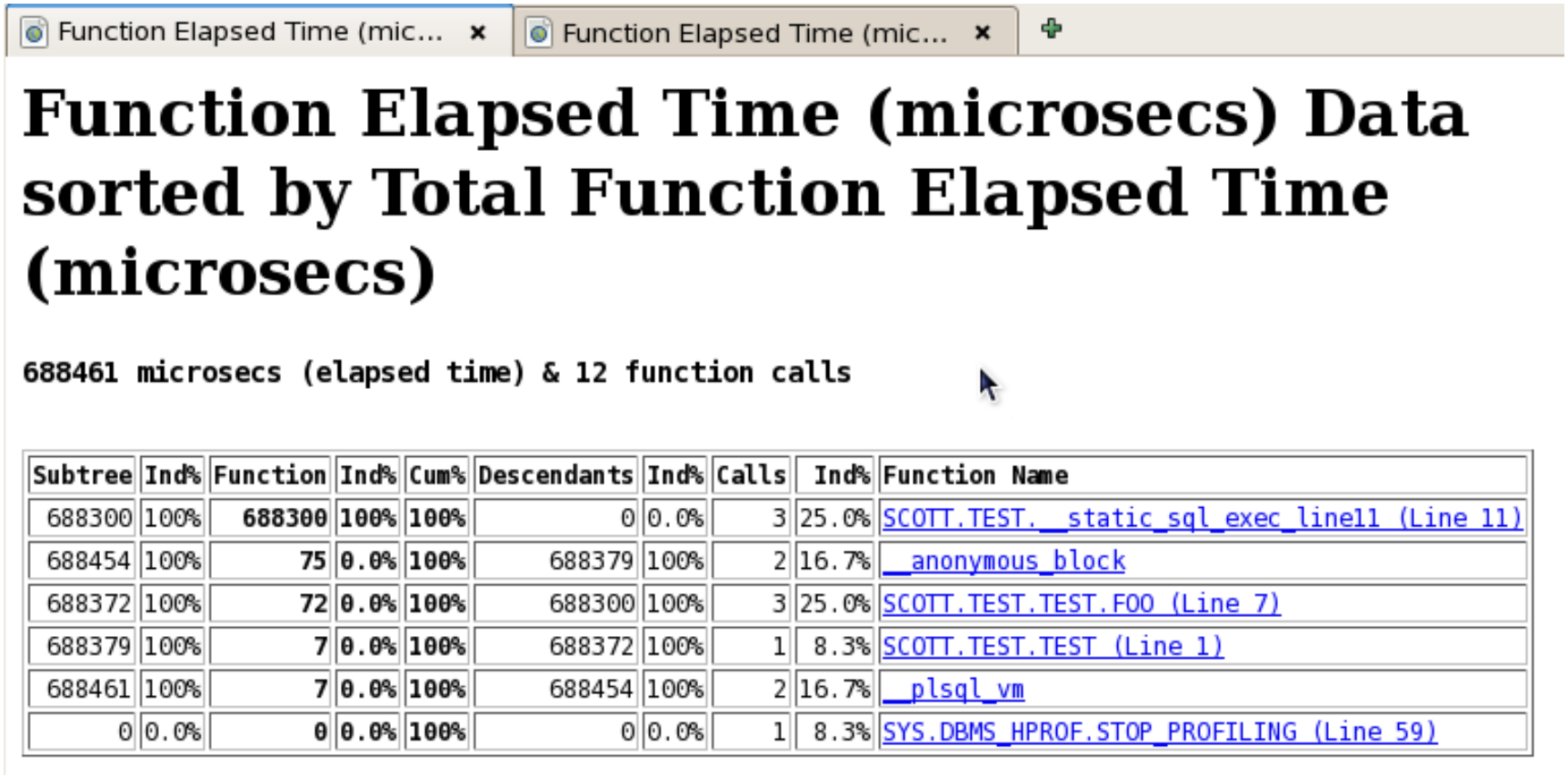
SCOTT.TEST.TEST (Line 1)

Subtree	Ind%	Function	Ind%	Descendants	Ind%	Calls	Ind%	Function Name
688379	100%	7	0.0%	688372	100%	1	8.3%	SCOTT.TEST.TEST (Line 1)
Parents:								
688379	100%	7	100%	688372	100%	1	100%	__anonymous_block
Children:								
688372	100%	72	100%	688300	100%	3	100%	SCOTT.TEST.TEST.FOO (Line 7)

SCOTT.TEST.TEST.FOO (Line 7)

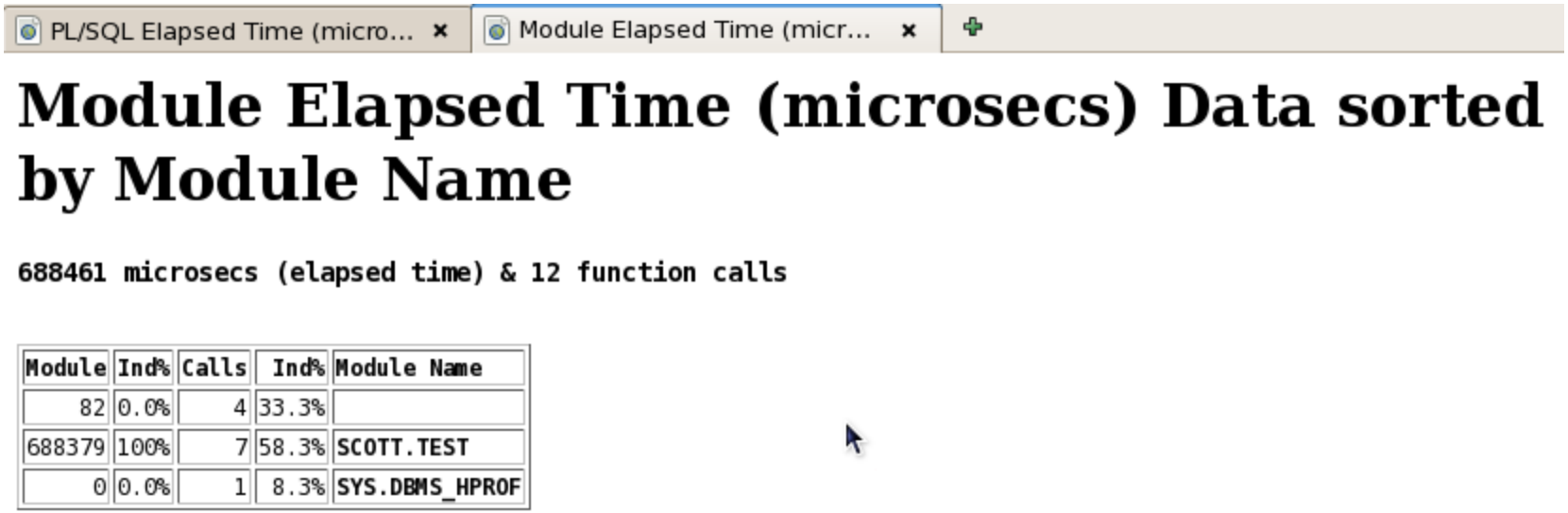
Subtree	Ind%	Function	Ind%	Descendants	Ind%	Calls	Ind%	Function Name
688372	100%	72	0.0%	688300	100%	3	25.0%	SCOTT.TEST.TEST.FOO (Line 7)
Parents:								
688372	100%	72	100%	688300	100%	3	100%	SCOTT.TEST.TEST (Line 1)
Children:								
688300	100%	688300	100%	0	N/A	3	100%	SCOTT.TEST.__static_sql_exec_line11 (Line 11)

DBMS_HPROF, the hierarchical profiler



Subtree	Ind%	Function	Ind%	Cum%	Descendants	Ind%	Calls	Ind%	Function Name
688300	100%	**688300**	**100%**	**100%**	0	0.0%	3	25.0%	SCOTT.TEST.__static_sql_exec_line11 (Line 11)
688454	100%	**75**	**0.0%**	**100%**	688379	100%	2	16.7%	__anonymous_block
688372	100%	**72**	**0.0%**	**100%**	688300	100%	3	25.0%	SCOTT.TEST.TEST.FOO (Line 7)
688379	100%	**7**	**0.0%**	**100%**	688372	100%	1	8.3%	SCOTT.TEST.TEST (Line 1)
688461	100%	**7**	**0.0%**	**100%**	688454	100%	2	16.7%	__plsql_vm
0	0.0%	**0**	**0.0%**	**100%**	0	0.0%	1	8.3%	SYS.DBMS_HPROF.STOP_PROFILING (Line 59)

DBMS_HPROF, the hierarchical profiler



Other Considerations

DBMS_UTILITY.GET_TIME

Check the elapsed time

- Sometimes a simple approach is needed, e.g. to check side by side the two versions of a procedure we work on. We do not wish to go through the whole procedure of profiling.
- The DBMS_UTILITY package comes handy with two functions to help you obtain this information:
  - DBMS_UTILITY.GET_TIME
  - DBMS_UTILITY.GET_CPU_TIME
- Both are available for Oracle Database 10g and later.

DBMS_UTILITY.GET_TIME

Check the elapsed time

- We use these functions to calculate the elapsed time (total and CPU, respectively) of our code down to the 100th of a second
- Here's how:
  - Call DBMS_UTILITY.GET_TIME (or GET_CPU_TIME) before you execute the code. Store this "starting time."
  - Run the code you want to measure
  - Call DBMS_UTILITY.GET_TIME (or GET_CPU_TIME) to get the "end time."
- Subtract start from end; this difference is the number of 100ths of sec. that have elapsed between start and end.

Performance Related Warnings

Compile-time warnings

- Introduced a compile-time warnings framework in Oracle Database 10g
- when you turn on warnings in your session, will give feedback on the quality of the code, and will offer advice for improving readability and performance

- **Recommendation:** use compile-time warnings to help identify areas of your code that could be optimized

Performance Related Warnings

Compile-time warnings

- You enable warnings for the entire set of performance-related warnings with the following statement:

```
ALTER SESSION SET PLSQL_WARNINGS = 'ENABLE:PERFORMANCE'
```

- Performance warnings include the following:
  - PLW-06014: PLSQL_OPTIMIZE_LEVEL <= 1 turns off native code generation
  - PLW-07203: parameter “string” may benefit from use of the NOCOPY compiler hint
  - PLW-07204: conversion away from column type may result in suboptimal query plan

Last but not least

A few simple things to always consider

- avoid procedural code when possible
- use triggers with care and only for small code
- avoid type conversions
- Use parallelism

References

- Oracle® Database PL/SQL Packages and Types Reference 11g Release 2 (11.2) E40758-03
- Oracle® Database Advanced Application Developer's Guide 11g Release 2 (11.2) E41502-02

Thank You!