16

Using the PL/SQL Hierarchical Profiler

You can use the PL/SQL hierarchical profiler to identify bottlenecks and performance-tuning opportunities in PL/SQL applications.

The profiler reports the dynamic execution profile of a PL/SQL program organized by function calls, and accounts for SQL and PL/SQL execution times separately. No special source or compile-time preparation is required; any PL/SQL program can be profiled.

This chapter describes the PL/SQL hierarchical profiler and explains how to use it to collect and analyze profile data for a PL/SQL program.

Topics:

16.1 Overview of PL/SQL Hierarchical Profiler

Nonhierarchical (**flat**) profilers record the time that a program spends within each subprogram —the **function time** or **self time** of each subprogram. Function time is helpful, but often inadequate. For example, it is helpful to know that a program spends 40% of its time in the subprogram `INSERT_ORDER`, but it is more helpful to know which subprograms call `INSERT_ORDER` often and the total time the program spends under `INSERT_ORDER` (including its descendant subprograms). Hierarchical profilers provide such information.

The PL/SQL hierarchical profiler:

- Reports the dynamic execution profile of your PL/SQL program, organized by subprogram calls
- Accounts for SQL and PL/SQL execution times separately
- Requires no special source or compile-time preparation
- Stores results in database tables (**hierarchical profiler tables**) for custom report generation by integrated development environment (IDE) tools (such as SQL Developer and third-party tools)
- Provides subprogram-level execution summary information, such as:
  - Number of calls to the subprogram
  - Time spent in the subprogram itself (**function time** or **self time**)
  - Time spent in the subprogram itself and in its descendent subprograms (**subtree time**)
  - Detailed parent-children information, for example:
    - All callers of a given subprogram (parents)

* All subprograms that a given subprogram called (children)
* How much time was spent in subprogram *x* when called from *y*
* How many calls to subprogram *x* were from *y*

The PL/SQL hierarchical profiler is implemented by the `DBMS_HPROF` package and has two components:

- Data collection

  The data collection component is an intrinsic part of the PL/SQL Virtual Machine. The `DBMS_HPROF` package provides APIs to turn hierarchical profiling on and off and write the **raw profiler output** to a file or raw profiler data table.

- Analyzer

  The analyzer component processes the raw profiler output and produce analyzed results. The analyzer component analyzes:

  – Raw profiler data located in the raw profiler data file and raw profiler data table into HTML CLOB report, analyzed report file, and hierarchical profiler analysis tables.

> **Note:**
>
> To generate simple HTML reports directly from raw profiler output, without using the Analyzer, you can use the `plshprof` command-line utility.

16.2 Collecting Profile Data

To collect profile data from your PL/SQL program for the PL/SQL hierarchical profiler, follow these steps:

1. Ensure that you have these privileges:
   - `EXECUTE` privilege on the `DBMS_HPROF` package
   - `WRITE` privilege on the directory that you specify when you call `DBMS_HPROF.START_PROFILING`
2. Use the `DBMS_HPROF.START_PROFILING` PL/SQL API to start hierarchical profiler data collection in a session.
3. Run your PL/SQL program long enough to get adequate code coverage.

   To get the most accurate measurements of elapsed time, avoid unrelated activity on the system on which your PL/SQL program is running.
4. Use the `DBMS_HPROF.STOP_PROFILING` PL/SQL API to stop hierarchical profiler data collection.

Example 16-1 Profiling a PL/SQL Procedure

```
BEGIN
  /* Start profiling.
     Write raw profiler output to file test.trc in a directory
     that is mapped to directory object PLSHPROF_DIR
     (see note after example). */

  DBMS_HPROF.START_PROFILING('PLSHPROF_DIR', 'test.trc');
END;
```

```
/
-- Run procedure to be profiled
BEGIN
  test;
END;
/
BEGIN
  -- Stop profiling
  DBMS_HPROF.STOP_PROFILING;
END;
/
```

Example 16-2 Profiling with Raw Profiler Data Table

```
DECLARE
analyze_runid number;
trace_id number;
BEGIN
-- create raw profiler data and analysis tables
-- call create_tables with force_it =>FALSE ( default) when
-- raw profiler data and analysis tables do not exist already
DBMS_HPROF . CREATE_TABLES ;
-- Start profiling .
-- Write raw profiler data in raw profiler data table
trace_id := DBMS_HPROF . START_PROFILING ;
-- Run procedure to be profiled
test ;
-- Stop profiling
DBMS_HPROF . STOP_PROFILING ;
-- analyzes trace_id entry in raw profiler data table and writes
-- hierarchical profiler information in hprof 's analysis tables.
analyze_runid := DBMS_HPROF . ANALYZE(trace_id );
END;
/
```

Consider this PL/SQL procedure, `test`:

```
CREATE OR REPLACE PROCEDURE test AUTHID DEFINER IS
  n NUMBER;

  PROCEDURE foo IS
  BEGIN
    SELECT COUNT(*) INTO n FROM EMPLOYEES;
  END foo;

BEGIN  -- test
  FOR i IN 1..3 LOOP
    foo;
  END LOOP;
END test;
/
```

Consider the PL/SQL procedure that analyzes and generates HTML CLOB report from raw profiler data table

```
declare
reportclob clob ;
trace_id number;
begin
-- create raw profiler data and analysis tables
-- force_it =>TRUE will dropped the tables if table existed
DBMS_HPROF . CREATE_TABLES (force_it =>TRUE );
-- Start profiling .
```

```
-- Write raw profiler data in raw profiler data table
trace_id := DBMS_HPROF . START_PROFILING ;
-- Run procedure to be profiled
test ;
-- Stop profiling
DBMS_HPROF . STOP_PROFILING ;
-- analyzes trace_id entry in raw profiler data table and produce
-- analyzed HTML report in reportclob .
DBMS_HPROF .ANALYZE (trace_id , reportclob );
end;
/
```

The SQL script in Example 16-1 profiles the execution of the PL/SQL procedure `test`.

> **Note:**
>
> A directory object is an alias for a file system path name. For example, if you are connected to the database `AS SYSDBA`, this `CREATE DIRECTORY` statement creates the directory object `PLSHPROF_DIR` and maps it to the file system directory `/private/plshprof/results`:
>
> ```
> CREATE DIRECTORY PLSHPROF_DIR as '/private/plshprof/results';
> ```
>
> To run the SQL script in Example 16-1, you must have `READ` and `WRITE` privileges on both `PLSHPROF_DIR` and the directory to which it is mapped. if you are connected to the database `AS SYSDBA`, this `GRANT` statement grants `READ` and `WRITE` privileges on `PLSHPROF_DIR` to `HR`:
>
> ```
> GRANT READ, WRITE ON DIRECTORY PLSHPROF_DIR TO HR;
> ```

> **See Also:**
>
> - *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about using directory objects
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about `DBMS_HPROF.START_PROFILING` and `DBMS_HPROF.STOP_PROFILING`

16.3 Understanding Raw Profiler Output

Raw profiler output is intended to be processed by the analyzer component of the PL/SQL hierarchical profiler. However, even without such processing, it provides a simple function-level trace of the program. This topic explains how to understand raw profiler output.

> **Note:**
>
> The raw profiler format shown in this chapter is intended only to illustrate conceptual features of raw profiler output. Format specifics are subject to change at each Oracle Database release.

The SQL script in Example 16-1 wrote this raw profiler output to the file `test.trc`:

```
P#V PLSHPROF Internal Version 1.0
P#! PL/SQL Timer Started
P#C PLSQL."".""."__plsql_vm"
P#X 4
P#C PLSQL."".""."__anonymous_block"
P#X 77
P#C PLSQL."HR"."TEST"::7."TEST"#980980e97e42f8ec #1
P#X 4
P#C PLSQL."HR"."TEST"::7."TEST.FOO"#980980e97e42f8ec #4
P#X 47
P#C SQL."HR"."TEST"::7."__static_sql_exec_line6" #6."3r6qf2qhr3cm1"
P#! SELECT COUNT(*) FROM EMPLOYEES
P#X 279
P#R
P#X 58
P#R
P#X 3
P#C PLSQL."HR"."TEST"::7."TEST.FOO"#980980e97e42f8ec #4
P#X 4
P#C SQL."HR"."TEST"::7."__static_sql_exec_line6" #6."3r6qf2qhr3cm1"
P#! SELECT COUNT(*) FROM EMPLOYEES
P#X 121
P#R
P#X 5
P#R
P#X 2
P#C PLSQL."HR"."TEST"::7."TEST.FOO"#980980e97e42f8ec #4
P#X 3
P#C SQL."HR"."TEST"::7."__static_sql_exec_line6" #6."3r6qf2qhr3cm1"
P#! SELECT COUNT(*) FROM EMPLOYEES
P#X 117
P#R
P#X 4
P#R
P#X 2
P#R
P#X 2
P#R
P#X 3
P#R
P#C PLSQL."".""."__plsql_vm"
P#X 3
P#C PLSQL."".""."__anonymous_block"
P#X 86
P#C PLSQL."SYS"."DBMS_HPROF"::11."STOP_PROFILING"#980980e97e42f8ec #453
P#R
P#R
P#R
P#! PL/SQL Timer Stopped
```

Table 16-1 Raw Profiler Output File Indicators

Indicator	Meaning
`P#V`	`PLSHPROF` banner with version number
`P#C`	Call to a subprogram (call event)
`P#R`	Return from a subprogram (return event)

Table 16-1 (Cont.) Raw Profiler Output File Indicators

Indicator	Meaning
P#X	Elapsed time between preceding and following events
P#!	Comment

Call events (P#C) and return events (P#R) are properly nested (like matched parentheses). If an unhandled exception causes a called subprogram to exit, the profiler still reports a matching return event.

Each call event (P#C) entry in the raw profiler output includes this information:

- **Namespace** to which the called subprogram belongs
- **Name of the PL/SQL module** in which the called subprogram is defined
- **Type of the PL/SQL module** in which the called subprogram is defined
- **Name of the called subprogram**

  This name might be one of the special function names in Special Function Names.
- Hexadecimal value that represents the hash algorithm of the **signature** of the called subprogram

  The DBMS_HPROF.analyze PL/SQL API stores the hash value in a hierarchical profiler table, which allows both you and DBMS_HPROF.analyze to distinguish between **overloaded subprograms** (subprograms with the same name).
- **Line number** at which the called subprogram is defined in the PL/SQL module

  PL/SQL hierarchical profiler does not measure time spent at individual lines within modules, but you can use line numbers to identify the source locations of subprograms in the module (as IDE/Tools do) and to distinguish between overloaded subprograms.

For example, consider this entry in the preceding example of raw profiler output:

```
P#C PLSQL."HR"."TEST"::7."TEST.FOO"#980980e97e42f8ec #4
```

The components of the preceding entry have these meanings:

Component	Meaning
PLSQL	PLSQL is the **namespace** to which the called subprogram belongs.
"HR"."TEST"	HR.TEST is the **name of the PL/SQL module** in which the called subprogram is defined.
7	7 is the internal enumerator for the **module type** of HR.TEST. Examples of module types are procedure, package, and package body.
"TEST.FOO"	TEST.FOO is the **name of the called subprogram**.
#980980e97e42f8ec	#980980e97e42f8ec is a hexadecimal value that represents the hash algorithm of the **signature** of TEST.FOO.
#4	4 is the **line number** in the PL/SQL module HR.TEST at which TEST.FOO is defined.

> **Note:**
>
> When a subprogram is inlined, it is not reported in the profiler output.
>
> When a call to a `DETERMINISTIC` function is "optimized away," it is not reported in the profiler output.

> **See Also:**
>
> - Namespaces of Tracked Subprograms
> - Analyzing Profile Data for more information about `analyze` PL/SQL API
> - *Oracle Database PL/SQL Language Reference* for information about subprogram inlining
> - *Oracle Database PL/SQL Language Reference* for information about `DETERMINISTIC` functions

16.3.1 Namespaces of Tracked Subprograms

The subprograms that the PL/SQL hierarchical profiler tracks are classified into the namespaces PLSQL and SQL, as follows:

- Namespace PLSQL includes:
  - PL/SQL subprogram calls
  - PL/SQL triggers
  - PL/SQL anonymous blocks
  - Remote subprogram calls
  - Package initialization blocks
- Namespace SQL includes SQL statements executed from PL/SQL, such as queries, data manipulation language (DML) statements, data definition language (DDL) statements, and native dynamic SQL statements

16.3.2 Special Function Names

PL/SQL hierarchical profiler tracks certain operations as if they were functions with the names and namespaces shown in Table 16-2.

Table 16-2 Function Names of Operations that the PL/SQL Hierarchical Profiler Tracks

Tracked Operation	Function Name	Namespace
Call to PL/SQL Virtual Machine	`__plsql_vm`	PL/SQL
PL/SQL anonymous block	`__anonymous_block`	PL/SQL
Package initialization block	`__pkg_init`	PL/SQL
Static SQL statement at line *line#*	`__static_sql_exec_line`*line#*	SQL

Table 16-2 (Cont.) Function Names of Operations that the PL/SQL Hierarchical Profiler Tracks

Tracked Operation	Function Name	Namespace
Dynamic SQL statement at line *line#*	__dyn_sql_exec_line*line#*	SQL
SQL FETCH statement at line *line#*	__sql_fetch_line*line#*	SQL

16.4 Analyzing Profile Data

The analyzer component of the PL/SQL hierarchical profiler, DBMS_HPROF.analyze, processes the raw profiler output and stores the results in the **hierarchical database tables** described in Table 16-3.

Table 16-3 PL/SQL Hierarchical Profiler Database Tables

Table	Description
DBMSHP_RUNS	Top-level information for this run of DBMS_HPROF.analyze. For column descriptions, see Table 16-4.
DBMSHP_FUNCTION_INFO	Information for each subprogram profiled in this run of DBMS_HPROF.analyze. For column descriptions, see Table 16-5.
DBMSHP_PARENT_CHILD_INFO	Parent-child information for each subprogram profiled in this run of DBMS_HPROF.analyze. For column descriptions, see Table 16-6.

Topics:

- Creating Hierarchical Profiler Tables
- Understanding Hierarchical Profiler Tables

> **Note:**
>
> To generate simple HTML reports directly from raw profiler output, without using the Analyzer, you can use the plshprof command-line utility. For details, see plshprof Utility.

16.4.1 Creating Hierarchical Profiler Tables

The following steps explain how to create hierarchical profiler tables in Table 16-3 and the other data structures required for persistently storing profile data, privileges required to run the DBMS_HPROF package, and generate custom reports:

1. Hierarchical profiler tables in Table 16-3 and other data structures required for persistently storing profile data can be created in the following ways.
   a. Call the DBMS_HPROF.CREATE_TABLES procedure.
   b. Run the script dbmshptab.sql (located in the directory rdbms/admin).

> **Note:**
>
> Running the script `dbmshptab.sql` drops any previously created hierarchical profiler tables.

> **Note:**
>
> The `dbmshptab.sql` (located in the directory `rdbms/admin`) has been deprecated. This script contains the statements to drop the tables and sequences along with the deprecation notes.

2. Ensure that you have these privileges:
   - `EXECUTE` privilege on the `DBMS_HPROF` package
   - `READ` privilege on the directory that `DBMS_HPROF.analyze` specifies
3. Use the PL/SQL API `DBMS_HPROF.analyze` to analyze a single raw profiler output file and store the results in hierarchical profiler tables.

   (For an example of a raw profiler output file, see `test.trc` in Understanding Raw Profiler Output.)

   For more information about `DBMS_HPROF.analyze`, see *Oracle Database PL/SQL Packages and Types Reference*.
4. Use the hierarchical profiler tables to generate custom reports.

Example 16-3 Invoking DBMS_HPROF.analyze

```
DECLARE
  runid NUMBER;
BEGIN
  runid := DBMS_HPROF.analyze(LOCATION=>'PLSHPROF_DIR',
                              FILENAME=>'test.trc');
  DBMS_OUTPUT.PUT_LINE('runid = ' || runid);
END;
/
```

The anonymous block in Example 16-3:

- Invokes the function `DBMS_HPROF.analyze` function, which:
  - Analyzes the profile data in the raw profiler output file `test.trc` (from Understanding Raw Profiler Output), which is in the directory that is mapped to the directory object `PLSHPROF_DIR`, and stores the data in the hierarchical profiler tables in Table 16-3.
  - Returns a unique identifier that you can use to query the hierarchical profiler tables in Table 16-3. (By querying these hierarchical profiler tables, you can produce customized reports.)
- Stores the unique identifier in the variable `runid`, which it prints.

16.4.2 Understanding Hierarchical Profiler Tables

These topics explain how to use the hierarchical profiler tables in Table 16-3:

16.4.2.1 Hierarchical Profiler Database Table Columns

Table 16-4 describes the columns of the hierarchical profiler table `DBMSHP_RUNS`, which contains one row of top-level information for each run of `DBMS_HPROF.analyze`.

The primary key for the hierarchical profiler table `DBMSHP_RUNS` is `RUNID`.

Table 16-4 DBMSHP_RUNS Table Columns

Column Name	Column Data Type	Column Contents
`RUNID`	`NUMBER`	Unique identifier for this run of `DBMS_HPROF.analyze`, generated from `DBMSHP_RUNNUMBER` sequence.
`RUN_TIMESTAMP`	`TIMESTAMP(6)`	Time stamp for this run of `DBMS_HPROF.analyze`.
`RUN_COMMENT`	`VARCHAR2(2047)`	Comment that you provided for this run of `DBMS_HPROF.analyze`.
`TOTAL_ELAPSED_TIME`	`NUMBER(38)`	Total elapsed time for this run of `DBMS_HPROF.analyze`.

Table 16-5 describes the columns of the hierarchical profiler table `DBMSHP_FUNCTION_INFO`, which contains one row of information for each subprogram profiled in this run of `DBMS_HPROF.analyze`. If a subprogram is overloaded, Table 16-5 has a row for each variation of that subprogram. Each variation has its own `LINE#` and `HASH` (see Distinguishing Between Overloaded Subprograms.)

The primary key for the hierarchical profiler table `DBMSHP_FUNCTION_INFO` is `RUNID, SYMBOLID`.

Table 16-5 DBMSHP_FUNCTION_INFO Table Columns

Column Name	Column Data Type	Column Contents
`RUNID`	`NUMBER`	References `RUNID` column of `DBMSHP_RUNS` table. For a description of that column, see Table 16-4.
`SYMBOLID`	`NUMBER`	Symbol identifier for subprogram (unique for this run of `DBMS_HPROF.analyze`).
`OWNER`	`VARCHAR2(128)`	Owner of module in which each subprogram is defined (for example, `SYS` or `HR`).
`MODULE`	`VARCHAR2(128)`	Module in which subprogram is defined (for example, `DBMS_LOB`, `UTL_HTTP`, or `MY_PACKAGE`).
`TYPE`	`VARCHAR2(32)`	Type of module in which subprogram is defined (for example, `PACKAGE`, `PACKAGE_BODY`, or `PROCEDURE`).

Table 16-5 (Cont.) DBMSHP_FUNCTION_INFO Table Columns

Column Name	Column Data Type	Column Contents
FUNCTION	VARCHAR2(4000)	Name of subprogram (not necessarily a function) (for example, INSERT_ORDER, PROCESS_ITEMS, or TEST). This name might be one of the special function names in Special Function Names. For subprogram B defined within subprogram A, this name is A.B. A recursive call to function X is denoted X@*n*, where *n* is the recursion depth. For example, X@1 is the first recursive call to X.
LINE#	NUMBER	Line number in OWNER.MODULE at which FUNCTION is defined.
HASH	RAW(32)	Hash code for signature of subprogram (unique for this run of DBMS_HPROF.analyze).
NAMESPACE	VARCHAR2(32)	Namespace of subprogram. For details, see Namespaces of Tracked Subprograms.
SUBTREE_ELAPSED_TIME	NUMBER(38)	Elapsed time, in microseconds, for subprogram, including time spent in descendant subprograms.
FUNCTION_ELAPSED_TIME	NUMBER(38)	Elapsed time, in microseconds, for subprogram, excluding time spent in descendant subprograms.
CALLS	NUMBER(38)	Number of calls to subprogram.
SQL_ID	VARCHAR2(13)	SQL Identifier of the SQL statement.
SQL_TEXT	VARCHAR2(4000)	First 50 characters of the SQL statement.

Table 16-6 describes the columns of the hierarchical profiler table DBMSHP_PARENT_CHILD_INFO, which contains one row of parent-child information for each unique parent-child subprogram combination profiled in this run of DBMS_HPROF.analyze.

Table 16-6 DBMSHP_PARENT_CHILD_INFO Table Columns

Column Name	Column Data Type	Column Contents
RUNID	NUMBER	References RUNID column of DBMSHP_FUNCTION_INFO table. For a description of that column, see Table 16-5.
PARENTSYMID	NUMBER	Parent symbol ID. RUNID, PARENTSYMID references DBMSHP_FUNCTION_INFO(RUNID, SYMBOLID).
CHILDSYMID	NUMBER	Child symbol ID. RUNID, CHILDSYMID references DBMSHP_FUNCTION_INFO(RUNID, SYMBOLID).

Table 16-6 (Cont.) DBMSHP_PARENT_CHILD_INFO Table Columns

Column Name	Column Data Type	Column Contents
SUBTREE_ELAPSED_TIME	NUMBER(38)	Elapsed time, in microseconds, for RUNID, CHILDSYMID when called from RUNID, PARENTSYMID, including time spent in descendant subprograms.
FUNCTION_ELAPSED_TIME	NUMBER(38)	Elapsed time, in microseconds, for RUNID, CHILDSYMID when called from RUNID, PARENTSYMID, excluding time spent in descendant subprograms.
CALLS	NUMBER(38)	Number of calls to RUNID, CHILDSYMID from RUNID, PARENTSYMID.

16.4.2.2 Distinguishing Between Overloaded Subprograms

Overloaded subprograms are different subprograms with the same name.

Suppose that a program declares three subprograms named `compute`—the first at line 50, the second at line 76, and the third at line 100. In the `DBMSHP_FUNCTION_INFO` table, each of these subprograms has `compute` in the `FUNCTION` column. To distinguish between the three subprograms, use either the `LINE#` column (which has 50 for the first subprogram, 76 for the second, and 100 for the third) or the `HASH` column (which has a unique value for each subprogram).

In the profile data for two different runs, the `LINE#` and `HASH` values for a function might differ. The `LINE#` value of a function changes if you insert or delete lines before the function definition. The `HASH` value changes only if the signature of the function changes; for example, if you change the parameter list.

> **See Also:**
>
> *Oracle Database PL/SQL Language Reference* for more information about overloaded subprograms

16.4.2.3 Hierarchical Profiler Tables for Sample PL/SQL Procedure

The hierarchical profiler tables for the PL/SQL procedure `test` in Collecting Profile Data are shown in Example 16-4 through Example 16-6.

Consider the third row of the table `DBMSHP_PARENT_CHILD_INFO` (Example 16-6). The `RUNID` column shows that this row corresponds to the third run. The columns `PARENTSYMID` and `CHILDSYMID` show that the symbol IDs of the parent (caller) and child (called subprogram) are `2` and `1`, respectively. The table `DBMSHP_FUNCTION_INFO` (Example 16-5) shows that for the third run, the symbol IDs `2` and `1` correspond to procedures `__plsql_vm` and `__anonymous_block`, respectively. Therefore, the information in this row is about calls from the procedure `__plsql_vm` to the `__anonymous_block` (defined within `__plsql_vm`) in the module `HR.TEST`. This row shows that, when called from the procedure `__plsql_vm`, the function time for the procedure `__anonymous_block` is 44 microseconds, and the time spent in the `__anonymous_block` subtree (including descendants) is 121 microseconds.

Example 16-4 DBMSHP_RUNS Table for Sample PL/SQL Procedure

```
     RUNID RUN_TIMESTAMP                 TOTAL_ELAPSED_TIME RUN_COMMENT
---------- ----------------------------- ------------------ -----------
         1 20-DEC-12 11.37.26.688381 AM                   7
         2 20-DEC-12 11.37.26.700523 AM                   9
         3 20-DEC-12 11.37.26.706824 AM                 123
```

Example 16-5 DBMSHP_FUNCTION_INFO Table for Sample PL/SQL Procedure

```
     RUNID    SYMBOLID OWNER   MODULE       TYPE            FUNCTION
---------- ----------- ------- ------------ --------------- -------------------------
         1           1 HR      PKG          PACKAGE BODY    MYPROC
         2           1 HR      PKG          PACKAGE BODY    MYFUNC
         2           2 HR      PKG          PACKAGE BODY    MYPROC
         3           1                                      __anonymous_block
         3           2                                      __plsql_vm
         3           3 HR      PKG          PACKAGE BODY    MYFUNC
         3           4 HR      PKG          PACKAGE BODY    MYPROC
         3           5 HR      TEST2        PROCEDURE       TEST2

LINE#  CALLS HASH                                                             NAMESPACE
----- ------ -------------------------------------------------------------- -----------
    2      1 9689BA467A19CD19                                               PLSQL
    7      1 28DC3402BAEB2B0D                                               PLSQL
    2      1 9689BA467A19CD19                                               PLSQL
    0      1                                                                PLSQL
    0      1                                                                PLSQL
    7      1 28DC3402BAEB2B0D                                               PLSQL
    2      2 9689BA467A19CD19                                               PLSQL
    1      1 980980E97E42F8EC                                               PLSQL

NAMESPACE      SUBTREE_ELAPSED_TIME    FUNCTION_ELAPSED_TIME
---------- ----------------------- ------------------------
PLSQL                            7                        7
PLSQL                            9                        8
PLSQL                            1                        1
PLSQL                          121                       44
PLSQL                          123                        2
PLSQL                            9                        8
PLSQL                            8                        8
PLSQL                           77                       61
```

Example 16-6 DBMSHP_PARENT_CHILD_INFO Table for Sample PL/SQL Procedure

```
     RUNID PARENTSYMID CHILDSYMID    SUBTREE_ELAPSED_TIME    FUNCTION_ELAPSED_TIME  CALLS
---------- ----------- ---------- ----------------------- ------------------------ ------
         2           1          2                       1                        1      1
         3           1          5                      77                       61      1
         3           2          1                     121                       44      1
         3           3          4                       1                        1      1
         3           5          3                       9                        8      1
         3           5          4                       7                        7      1
```

16.4.2.4 Examples of Calls to DBMS_HPROF.analyze with Options

For an example of a call to `DBMS_HPROF.analyze` without options, see Example 16-3.

Example 16-7 creates a package, creates a procedure that invokes subprograms in the package, profiles the procedure, and uses `DBMS_HRPROF.analyze` to analyze the raw profiler output file. The raw profiler output file is in the directory corresponding to the `PLSHPROF_DIR` directory object.

Example 16-7 Invoking DBMS_HPROF.analyze with Options

```
-- Create package

CREATE OR REPLACE PACKAGE pkg AUTHID DEFINER IS
  PROCEDURE myproc (n IN out NUMBER);
  FUNCTION myfunc (v VARCHAR2) RETURN VARCHAR2;
  FUNCTION myfunc (n PLS_INTEGER) RETURN PLS_INTEGER;
END pkg;
/
CREATE OR REPLACE PACKAGE BODY pkg IS
  PROCEDURE myproc (n IN OUT NUMBER) IS
  BEGIN
    n := n + 5;
  END;

  FUNCTION myfunc (v VARCHAR2) RETURN VARCHAR2 IS
    n NUMBER;
  BEGIN
    n := LENGTH(v);
    myproc(n);
    IF n > 20 THEN
      RETURN SUBSTR(v, 1, 20);
    ELSE
      RETURN v || '...';
    END IF;
  END;

  FUNCTION myfunc (n PLS_INTEGER) RETURN PLS_INTEGER IS
    i PLS_INTEGER;
    PROCEDURE myproc (n IN out PLS_INTEGER) IS
    BEGIN
      n := n + 1;
    END;
  BEGIN
    i := n;
    myproc(i);
    RETURN i;
  END;
END pkg;
/

-- Create procedure that invokes package subprograms

CREATE OR REPLACE PROCEDURE test2 AUTHID DEFINER IS
  x NUMBER := 5;
  y VARCHAR2(32767);
BEGIN
  pkg.myproc(x);
  y := pkg.myfunc('hello');
END;

-- Profile test2

BEGIN
  DBMS_HPROF.START_PROFILING('PLSHPROF_DIR', 'test2.trc');
END;
/
BEGIN
  test2;
END;
/
```

```
BEGIN
  DBMS_HPROF.STOP_PROFILING;
END;
/
-- If not done, create hierarchical profiler tables (see Creating Hierarchical Profiler Tables).

-- Call DBMS_HPROF.analyze with options

DECLARE
  runid NUMBER;
BEGIN
  -- Analyze only subtrees rooted at trace entry "HR"."PKG"."MYPROC"

  runid := DBMS_HPROF.analyze('PLSHPROF_DIR', 'test2.trc',
                              trace => '"HR"."PKG"."MYPROC"');

  -- Analyze up to 20 calls to subtrees rooted at trace entry
  -- "HR"."PKG"."MYFUNC".  Because "HR"."PKG"."MYFUNC" is overloaded,
  -- both overloads are considered for analysis.

  runid := DBMS_HPROF.analyze('PLSHPROF_DIR', 'test2.trc',
                              collect => 20,
                              trace => '"HR"."PKG"."MYFUNC"');

  -- Analyze second call to PL/SQL virtual machine

  runid := DBMS_HPROF.analyze('PLSHPROF_DIR', 'test2.trc',
                              skip => 1, collect => 1,
                              trace => '"".""."".""__plsql_vm"');
END;
/
```

16.5 plshprof Utility

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 `test.trc` in Collecting Profile Data.)

You can browse the generated HTML reports in any browser. The browser's navigational capabilities, combined with well chosen links, provide a powerful way to analyze performance of large applications, improve application performance, and lower development costs.

Topics:

- plshprof Options
- HTML Report from a Single Raw Profiler Output File
- HTML Difference Report from Two Raw Profiler Output Files

16.5.1 plshprof Options

The command to run the `plshprof` utility is:

```
plshprof [option...] profiler_output_filename_1 profiler_output_filename_2
```

Each *option* is one of these:

Option	Description	Default
`-skip count`	Skips first *count* calls. Use only with `-trace symbol`.	0
`-collect count`	Collects information for *count* calls. Use only with `-trace symbol`.	1
`-output filename`	Specifies name of output file	*symbol*`.html` or *tracefile1*`.html`
`-summary`	Prints only elapsed time	None
`-trace symbol`	Specifies function name of tree root	Not applicable

Suppose that your raw profiler output file, `test.trc`, is in the current directory. You want to analyze and generate HTML reports, and you want the root file of the HTML report to be named `report.html`. Use this command (`%` is the prompt):

```
% plshprof -output report test.trc
```

16.5.2 HTML Report from a Single Raw Profiler Output File

To generate a PL/SQL hierarchical profiler HTML report from a single raw profiler output file, use these commands:

```
% cd target_directory
% plshprof -output html_root_filename profiler_output_filename
```

target_directory is the directory in which you want the HTML files to be created.

html_root_filename is the name of the root HTML file to be created.

profiler_output_filename is the name of a raw profiler output file.

The preceding `plshprof` command generates a set of HTML files. Start browsing them from *html_root_filename*`.html`.

Topics:

- First Page of Report
- Function-Level Reports
- Understanding PL/SQL Hierarchical Profiler SQL-Level Reports
- Module-Level Reports
- Namespace-Level Reports
- Parents and Children Report for a Function

16.5.2.1 First Page of Report

The first page of an HTML report from a single raw profiler output file includes summary information and hyperlinks to other pages of the report.

Sample First Page

PL/SQL Elapsed Time (microsecs) Analysis

824 microsecs (elapsed time) & 12 function calls

The PL/SQL Hierarchical Profiler produces a collection of reports that present information derived from the profiler output log in a variety of formats. These reports have been found to be the most generally useful as starting points for browsing:

- Function Elapsed Time (microsecs) Data sorted by Total Subtree Elapsed Time (microsecs)
- Function Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs)
- SQL ID Elapsed Time (microsecs) Data sorted by SQL ID

In addition, the following reports are also available:

- Function Elapsed Time (microsecs) Data sorted by Function Name
- Function Elapsed Time (microsecs) Data sorted by Total Descendants Elapsed Time (microsecs)
- Function Elapsed Time (microsecs) Data sorted by Total Function Call Count
- Function Elapsed Time (microsecs) Data sorted by Mean Subtree Elapsed Time (microsecs)
- Function Elapsed Time (microsecs) Data sorted by Mean Function Elapsed Time (microsecs)
- Function Elapsed Time (microsecs) Data sorted by Mean Descendants Elapsed Time (microsecs)
- Module Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs)
- Module Elapsed Time (microsecs) Data sorted by Module Name
- Module Elapsed Time (microsecs) Data sorted by Total Function Call Count
- Namespace Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs)
- Namespace Elapsed Time (microsecs) Data sorted by Namespace
- Namespace Elapsed Time (microsecs) Data sorted by Total Function Call Count
- Parents and Children Elapsed Time (microsecs) Data

16.5.2.2 Function-Level Reports

The function-level reports provide a flat view of the profile information. Each function-level report includes this information for each function:

- Function time (time spent in the function itself, also called "self time")
- Descendants time (time spent in the descendants of the function)
- Subtree time (time spent in the subtree of the function—function time plus descendants time)
- Number of calls to the function
- Function name

  The function name is hyperlinked to the Parents and Children Report for the function.
- SQL ID
- SQL Text (First 50 characters of the SQL text).

Each function-level report is sorted on a particular attribute; for example, function time or subtree time.

This sample report is sorted in descending order of the total subtree elapsed time for the function, which is why information in the Subtree and Ind% columns is in **bold type**:

Sample Report

Function Elapsed Time (microsecs) Data sorted by Total Subtree Elapsed Time (microsecs)

824 microsecs (elapsed time) & 12 function calls

Subtree	Ind %	Function	Descendant	Ind %	Calls	Ind %	Function Name	SQL ID	SQL TEXT
824	**100 %**	10	814	98.8 %	2	16.7 %	__plsq_vm		
814	**98.8 %**	165	649	78.8 %	2	16.7 %	__anonymous_block		
649	**78.8 %**	11	638	77.4 %	1	8.3 %	HR.TEST.TEST (Line 1)		
638	**77.4 %**	121	517	62.7 %	3	25.0 %	HR.TEST.TEST.FOO (Line 4)		
517	**62.7 %**	517	0	0.0 %	3	25.0 %	HR.TEST.__static_sql_exec_line5 (Line 6)	3r6qf2qhr3cm1	SELECT COUNT(*) FROM EMPLOYEES
0	**0.0 %**	0	0	0.0 %	1	8.3 %	SYS.DBMS_HPROF.STOP_PROFILING (Line 453)		

16.5.2.3 Module-Level Reports

Each module-level report includes this information for each module (for example, package or type):

- Module time (time spent in the module—sum of the function times of all functions in the module)
- Number of calls to functions in the module

Each module-level report is sorted on a particular attribute; for example, module time or module name.

This sample report is sorted by module time, which is why information in the Module, Ind%, and Cum% columns is in **bold type**:

Sample Report

Module Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs)

166878 microsecs (elapsed time) & 1099 function calls

Module	Ind%	Cum%	Calls	Ind%	Module Name
84932	**50.9%**	**50.9%**	6	0.5%	HR.P
67749	**40.6%**	**91.5%**	216	19.7%	SYS.DBMS_LOB
13340	**8.0%**	**99.5%**	660	60.1%	SYS.UTL_FILE
839	**0.5%**	**100%**	214	19.5%	SYS.UTL_RAW
18	**0.0%**	**100%**	2	0.2%	HR.UTILS
0	**0.0%**	**100%**	1	0.1%	SYS.DBMS_HPROF

16.5.2.4 Namespace-Level Reports

Each namespace-level report includes this information for each namespace:

- Namespace time (time spent in the namespace—sum of the function times of all functions in the namespace)
- Number of calls to functions in the namespace

Each namespace-level report is sorted on a particular attribute; for example, namespace time or number of calls to functions in the namespace.

This sample report is sorted by function time, which is why information in the Function, Ind%, and Cum% columns is in **bold type**:

Sample Report

Namespace Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs)

166878 microsecs (elapsed time) & 1099 function calls

Function	Ind%	Cum%	Calls	Ind%	Namespace
93659	**56.1%**	**56.1%**	1095	99.6%	PLSQL
73219	**43.9%**	**100%**	4	0.4%	SQL

16.5.2.5 Parents and Children Report for a Function

For each function tracked by the profiler, the Parents and Children Report provides information about parents (functions that call it) and children (functions that it calls). For each parent, the report gives the function's execution profile (subtree time, function time, descendants time, and number of calls). For each child, the report gives the execution profile for the child when called from this function (but not when called from other functions).

The execution profile for a function includes the same information for that function as a function-level report includes for each function.

This Sample Report is a fragment of a Parents and Children Report that corresponds to a function named `HR.P.UPLOAD`. The first row has this summary information:

- There are two calls to the function `HR.P.UPLOAD`.
- The total subtree time for the function is 166,860 microseconds—11,713 microseconds (7.0%) in the function itself and 155,147 microseconds (93.0%) in its descendants.

After the row "Parents" are the execution profile rows for the two parents of HR.P.UPLOAD, which are HR.UTILS.COPY_IMAGE and HR.UTILS.COPY_FILE.

The first parent execution profile row, for HR.UTILS.COPY_IMAGE, shows:

- HR.UTILS.COPY_IMAGE calls HR.P.UPLOAD once, which is 50% of the number of calls to HR.P.UPLOAD.
- The subtree time for HR.P.UPLOAD when called from HR.UTILS.COPY_IMAGE is 106,325 microseconds, which is 63.7% of the total subtree time for HR.P.UPLOAD.
- The function time for HR.P.UPLOAD when called from HR.UTILS.COPY_IMAGE is 6,434 microseconds, which is 54.9% of the total function time for HR.P.UPLOAD.

After the row "Children" are the execution profile rows for the children of HR.P.UPLOAD when called from HR.P.UPLOAD.

The third child execution profile row, for SYS.UTL_FILE.GET_RAW, shows:

- HR.P.UPLOAD calls SYS.UTL_FILE.GET_RAW 216 times.
- The subtree time, function time and descendants time for SYS.UTL_FILE.GET_RAW when called from HR.P.UPLOAD are 12,487 microseconds, 3,969 microseconds, and 8,518 microseconds, respectively.
- Of the total descendants time for HR.P.UPLOAD (155,147 microseconds), the child SYS.UTL_FILE.GET_RAW is responsible for 12,487 microsecs (8.0%).

Sample Report

HR.P.UPLOAD (Line 3)

Subtree	Ind%	Function	Ind%	Descendant	Ind%	Calls	Ind%	Function Name
166860	**100%**	11713	7.0%	155147	93.0%	2	0.2%	HR.P.UPLOAD (Line 3)
Parents:								
106325	**63.7%**	6434	54.9%	99891	64.4%	1	50.0%	HR.UTILS.COPY_IMAGE (Line 3)
60535	**36.3%**	5279	45.1%	55256	35.6%	1	50.0%	HR.UTILS.COPY_FILE (Line 8))
Children:								
71818	**46.3%**	71818	100%	0	N/A	2	100%	HR.P.__static_sql_exec_line38 (Line 38)
67649	**43.6%**	67649	100%	0	N/A	214	100%	SYS.DBMS_LOB.WRITEAPPEND (Line 926)
12487	**8.0%**	3969	100%	8518	100%	216	100%	SYS.UTL_FILE.GET_RAW (Line 1089)
1401	**0.9%**	1401	100%	0	N/A	2	100%	HR.P.__static_sql_exec_line39 (Line 39)
839	**0.5%**	839	100%	0	N/A	214	100%	SYS.UTL_FILE.GET_RAW (Line 246)
740	**0.5%**	73	100%	667	100%	2	100%	SYS.UTL_FILE.FOPEN (Line 422)
113	**0.1%**	11	100%	102	100%	2	100%	SYS.UTL_FILE.FCLOSE (Line 585)

Subtree	Ind%	Function	Ind%	Descendant	Ind%	Calls	Ind%	Function Name
100	**0.1%**	100	100%	0	N/A	2	100%	SYS.DBMS_LOB.CREATETEMPORARY (Line 536)

> **See Also:**
>
> Function-Level Reports

16.5.2.6 Understanding PL/SQL Hierarchical Profiler SQL-Level Reports

Understanding DBMS_HPROF.ANALYZE SQL-level reports.

The PL/SQL Hierarchical Profiler SQL-level report provides the list of all the SQLs collected during profiling. along with the abbreviated SQL text, and the elapsed time (microsecs) sorted by SQL ID. The SQL ID is useful if other SQL statistics must be retrieved in other tables, for example, for SQL tuning purpose. The first 50 characters of the SQL text is included in the report. You can use the Function-Level reports to get the details surrounding where the SQL is called, and its location in the source code if needed.

Sample Report

SQL ID Elapsed Time (microsecs) Data sorted by SQL ID

824 microsecs (elapsed time) & 12 function calls

SQL ID	SQL TEXT	Function	Ind%	Calls	Ind%
3r6qf2qhr3cm1	SELECT COUNT(*) FROM EMPLOYEES	679	82.4%	3	25.0%

16.5.3 HTML Difference Report from Two Raw Profiler Output Files

To generate a PL/SQL hierarchical profiler HTML difference report from two raw profiler output files, use these commands:

```
% cd target_directory
% plshprof -output html_root_filename profiler_output_filename_1 profiler_output_filename_2
```

target_directory is the directory in which you want the HTML files to be created.

html_root_filename is the name of the root HTML file to be created.

profiler_output_filename_1 and *profiler_output_filename_2* are the names of raw profiler output files.

The preceding `plshprof` command generates a set of HTML files. Start browsing them from *html_root_filename*.html.

Topics:

16.5.3.1 Difference Report Conventions

Difference reports use these conventions:

- In a report title, **Delta** means **difference**, or **change**.
- A **positive value** indicates that the number increased (**regressed**) from the first run to the second run.
- A **negative value** for a difference indicates that the number decreased (**improved**) from the first run to the second run.
- The symbol **#** after a function name means that the function was called in only one run.

16.5.3.2 First Page of Difference Report

The first page of an HTML difference report from two raw profiler output files includes summary information and hyperlinks to other pages of the report.

Sample First Page

PL/SQL Elapsed Time (microsecs) Analysis – Summary Page

This analysis finds a net **regression** of **2709589** microsecs (elapsed time) or **80%** (**3393719** versus **6103308**). Here is a summary of the 7 most important individual function regressions and improvements:

Regressions: 3399382 microsecs (elapsed time)

Function	Rel%	Ind%	Calls	Rel%	Function Name
2075627	**+941%**	**61.1%**	0		HR.P.G (Line 35)
1101384	**+54.6%**	**32.4%**	5	+55.6%	HR.P.H (Line 18)
222371		**6.5%**	1		HR.P.J (Line 10)

Improvements: 689793 microsecs (elapsed time)

Function	Rel%	Ind%	Calls	Rel%	Function Name
-467051	**-50.0%**	**67.7%**	-2	-50.0%	HR.P.F (Line 25)
-222737		**32.3%**	-1		HR.P.I (Line 2)#
-5	**-21.7%**	**0.0%**	0		HR.P.TEST (Line 46)

The PL/SQL Timing Analyzer produces a collection of reports that present information derived from the profiler's output logs in a variety of formats. The following reports have been found to be the most generally useful as starting points for browsing:

- Function Elapsed Time (microsecs) Data for Performance Regressions
- Function Elapsed Time (microsecs) Data for Performance Improvements

Also, the following reports are also available:

- Function Elapsed Time (microsecs) Data sorted by Function Name
- Function Elapsed Time (microsecs) Data sorted by Total Subtree Elapsed Time (microsecs) Delta
- Function Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs) Delta
- Function Elapsed Time (microsecs) Data sorted by Total Descendants Elapsed Time (microsecs) Delta
- Function Elapsed Time (microsecs) Data sorted by Total Function Call Count Delta
- Module Elapsed Time (microsecs) Data sorted by Module Name
- Module Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs) Delta
- Module Elapsed Time (microsecs) Data sorted by Total Function Call Count Delta
- Namespace Elapsed Time (microsecs) Data sorted by Namespace
- Namespace Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs)
- Namespace Elapsed Time (microsecs) Data sorted by Total Function Call Count
- File Elapsed Time (microsecs) Data Comparison with Parents and Children

16.5.3.3 Function-Level Difference Reports

Each function-level difference report includes, for each function, the change in these values from the first run to the second run:

- Function time (time spent in the function itself, also called "self time")
- Descendants time (time spent in the descendants of the function)
- Subtree time (time spent in the subtree of the function—function time plus descendants time)
- Number of calls to the function
- Mean function time

  The mean function time is the function time divided by number of calls to the function.
- Function name

  The function name is hyperlinked to the Parents and Children Difference Report for the function.

The report in Sample Report 1 shows the difference information for all functions that performed better in the first run than they did in the second run. Note that:

- For `HR.P.G`, the function time increased by 2,075,627 microseconds (941%), which accounts for 61.1% of all regressions.

- For `HR.P.H`, the function time and number of calls increased by 1,101,384 microseconds (54.6%) and 5 (55.6%), respectively, but the mean function time improved by 1,346 microseconds (-0.6%).
- `HR.P.J` was called in only one run.

Sample Report 1

Function Elapsed Time (microsecs) Data for Performance Regressions

Subtree	Function	Rel%	Ind%	Cum%	Descendant	Calls	Rel%	Mean Function	Rel%	Function Name
4075787	**2075627**	**+941%**	**61.1%**	61.1%	2000160	0		2075627	+941%	HR.P.G (Line 35)
1101384	**1101384**	**+54.6%**	**32.4%**	93.5%	0	5	+55.6%	-1346	-0.6%	HR.P.H (Line 18)
222371	**222371**		6.5%	100%	0	1				HR.P.J (Line 10)#

The report in Sample Report 2 shows the difference information for all functions that performed better in the second run than they did in the first run.

Sample Report 2

Function Elapsed Time (microsecs) Data for Performance Improvements

Subtree	Function	Rel%	Ind%	Cum%	Descendant	Calls	Rel%	Mean Function	Rel%	Function Name
-1365827	**-467051**	**-50.0%**	**67.7%**	67.7%	-898776	-2	-50.0%	-32	0.0%	HR.P.F (Line 25)
-222737	**-222737**		**32.3%**	100%	0	-1				HR.P.I (Line 2)
2709589	**-5**	**-21.7%**	**0.0%**	100%	2709594	0		-5	-20.8	HR.P.TEST (Line 46)#

The report in Sample Report 3 summarizes the difference information for all functions.

Sample Report 3

Function Elapsed Time (microsecs) Data sorted by Total Function Call Count Delta

Subtree	Function	Rel%	Ind%	Descendant	Calls	Rel%	Mean Function	Rel%	Function Name
1101384	1101384	+54.6%	32.4%	0	**5**	**+55.6%**	-1346	-0.6%	HR.P.H (Line 18)
-1365827	-467051	+50.0%	67.7%	-898776	**-2**	**-50.0%**	-32	-0.0%	HR.P.F (Line 25)
-222377	-222377		32.3%	0	**-1**				HR.P.I (Line 2)#
222371	222371		6.5%	0	**1**				HR.P.J(Line 10)#
4075787	2075627	+941%	61.1%	2000160	**0**		2075627	+941%	HR.P.G (Line 35)
2709589	-5	-21.7%	0.0%	2709594	**0**		-5	-20.8%	HR.P.TEST (Line 46)

Subtree	Function	Rel%	Ind%	Descendant	Calls	Rel%	Mean Function	Rel%	Function Name
0	0			0	**0**				SYS.DBMS_HPROF.STOP_PROFILING (Line 53)

16.5.3.4 Module-Level Difference Reports

Each module-level report includes, for each module, the change in these values from the first run to the second run:

- Module time (time spent in the module—sum of the function times of all functions in the module)
- Number of calls to functions in the module

Sample Report

Module Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs) Delta

Module	Calls	Module Name
2709589	3	HR.P
0	0	SYS.DBMS_HPROF

16.5.3.5 Namespace-Level Difference Reports

Each namespace-level report includes, for each namespace, the change in these values from the first run to the second run:

- Namespace time (time spent in the namespace—sum of the function times of all functions in the namespace)
- Number of calls to functions in the namespace

Sample Report

Namespace Elapsed Time (microsecs) Data sorted by Namespace

Function	Calls	Namespace
2709589	3	**PLSQL**

16.5.3.6 Parents and Children Difference Report for a Function

The Parents and Children Difference Report for a function shows changes in the execution profiles of these from the first run to the second run:

- Parents (functions that call the function)
- Children (functions that the function calls)

  Execution profiles for children include only information from when this function calls them, not for when other functions call them.

The execution profile for a function includes this information:

- Function time (time spent in the function itself, also called "self time")
- Descendants time (time spent in the descendants of the function)
- Subtree time (time spent in the subtree of the function—function time plus descendants time)
- Number of calls to the function
- Function name

The sample report is a fragment of a Parents and Children Difference Report that corresponds to a function named `HR.P.X`.

The first row, a summary of the difference between the first and second runs, shows regression: function time increased by 1,094,099 microseconds (probably because the function was called five more times).

The "Parents" rows show that `HR.P.G` called `HR.P.X` nine more times in the second run than it did in the first run, while `HR.P.F` called it four fewer times.

The "Children" rows show that `HR.P.X` called each child five more times in the second run than it did in the first run.

Sample Report

HR.P.X (Line 11)

Subtree	Function	Descendant	Calls	Function Name
3322196	1094099	2228097	5	**HR.P.X (Line 11**)
Parents:				
6037490	1993169	4044321	9	**HR.P.G (Line 38)**
-2715294	-899070	-1816224	-4	**HR.P.F (Line 28)**
Children:				
1125489	1125489	0	5	**HR.P.J (Line 10)**
1102608	1102608	0	5	**HR.P.I (Line 2)**

The Parents and Children Difference Report for a function is accompanied by a Function Comparison Report, which shows the execution profile of the function for the first and second runs and the difference between them. This example is the Function Comparison Report for the function `HR.P.X`:

Sample Report

Elapsed Time (microsecs) for HR.P.X (Line 11) (20.1% of total regression)

HR.P.X (Line 11)	First Trace	Ind%	Second Trace	Ind%	Diff	Diff%
Function Elapsed Time (microsecs)	1999509	26.9%	3093608	24.9%	1094099	+54.7%
Descendants Elapsed Time (microsecs)	4095943	55.1%	6324040	50.9%	2228097	+54.4%
Subtree Elapsed Time (microsecs)	6095452	81.9%	9417648	75.7%	3322196	+54.5%
Function Calls	9	25.0%	14	28.6%	5	+55.6%
Mean Function Elapsed Time (microsecs)	222167.7		220972.0		-1195.7	-0.5%

HR.P.X (Line 11)	First Trace	Ind%	Second Trace	Ind%	Diff	Diff%
Mean Descendants Elapsed Time (microsecs)	455104.8		451717.1		-3387.6	-0.7%
Mean Subtree Elapsed Time (microsecs)	677272.4		672689.1		-4583.3	-0.7%