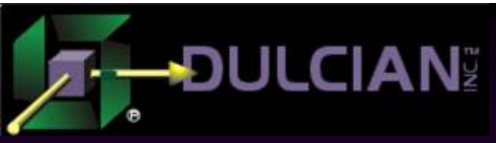


Unduly Forgotten Performance Tuning Hero: PL/SQL Hierarchical Profiler

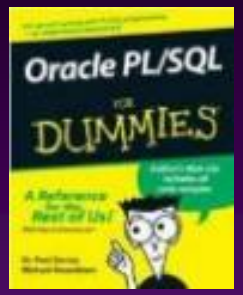
Michael Rosenblum

www.dulcian.com

DULCIAN INC.

Who Am I? – "Misha"

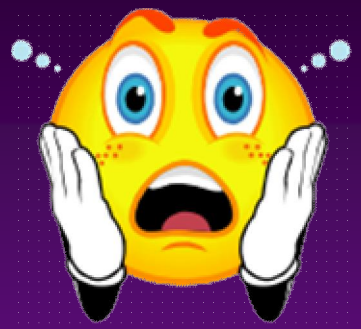
- Oracle ACE
- Co-author of 3 books
  - *PL/SQL for Dummies*
  - *Expert PL/SQL Practices*
  - *Oracle PL/SQL Performance Tuning Tips & Techniques*

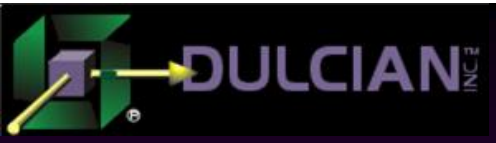


- Known for:
  - SQL and PL/SQL tuning
  - Complex functionality
    - Code generators
    - Repository-based development

Yet another performance presentation???

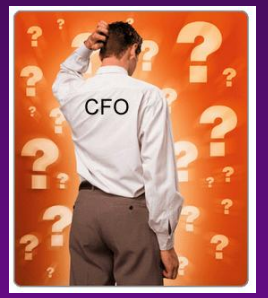
- NO!
- Because:
  - I will NOT talk about bind variables
    - … more than a few [dozen] times ☺
  - I will NOT mention extra paid options/products.
    - Well…I am a [database] doctor, not a [salesman?] (c) Star Trek
  - I will NOT be buzzword-compliant
    - … so you can be [mostly] CLOUD- and EXADATA-free.





Tuning (CFO Level)

- Means:
  - Ensuring that available resources are used in the most efficient way:
    - No wasted resources
    - No under-utilized resources
- Impact:
  - Makes CFO happy when they look at hardware costs
    - …especially in the Cloud [for more, see my session #1454]



DULCIAN INC.

Tuning (Practical Level)

- Means:
  - MAKING END-USERS HAPPY!

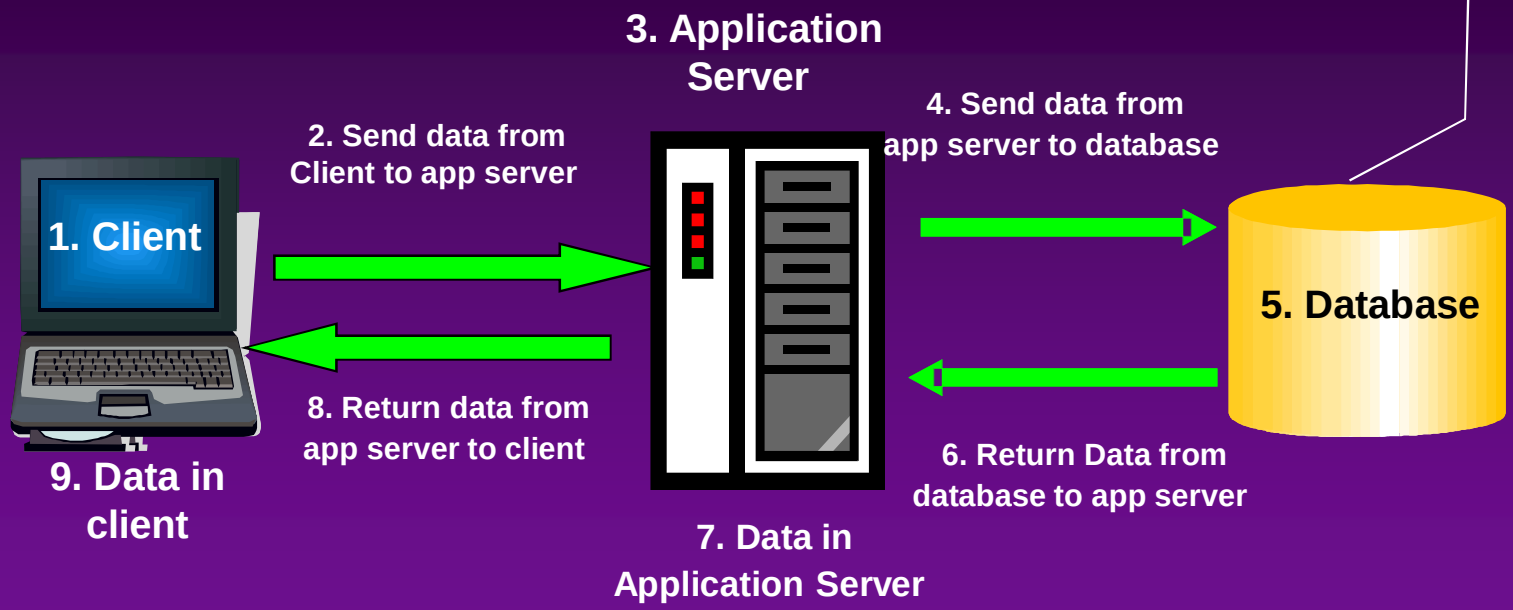
Reality Check

- End-users
  - DON'T CARE ABOUT:
    - CPU utilization/disk workload/etc.
    - Being buzzword-compliant by using the coolest technology stack
  - DO CARE ABOUT:
    - Being able to run their business
      - … i.e. monthly report should not take two months to prepare!
    - Time wasted looking at an hourglass on the screen
      - … although the notion of "wasted time" can be managed by using various psychological tricks (managing expectations!).

So?

- This talk is all about end-user requests…

… when time is lost here

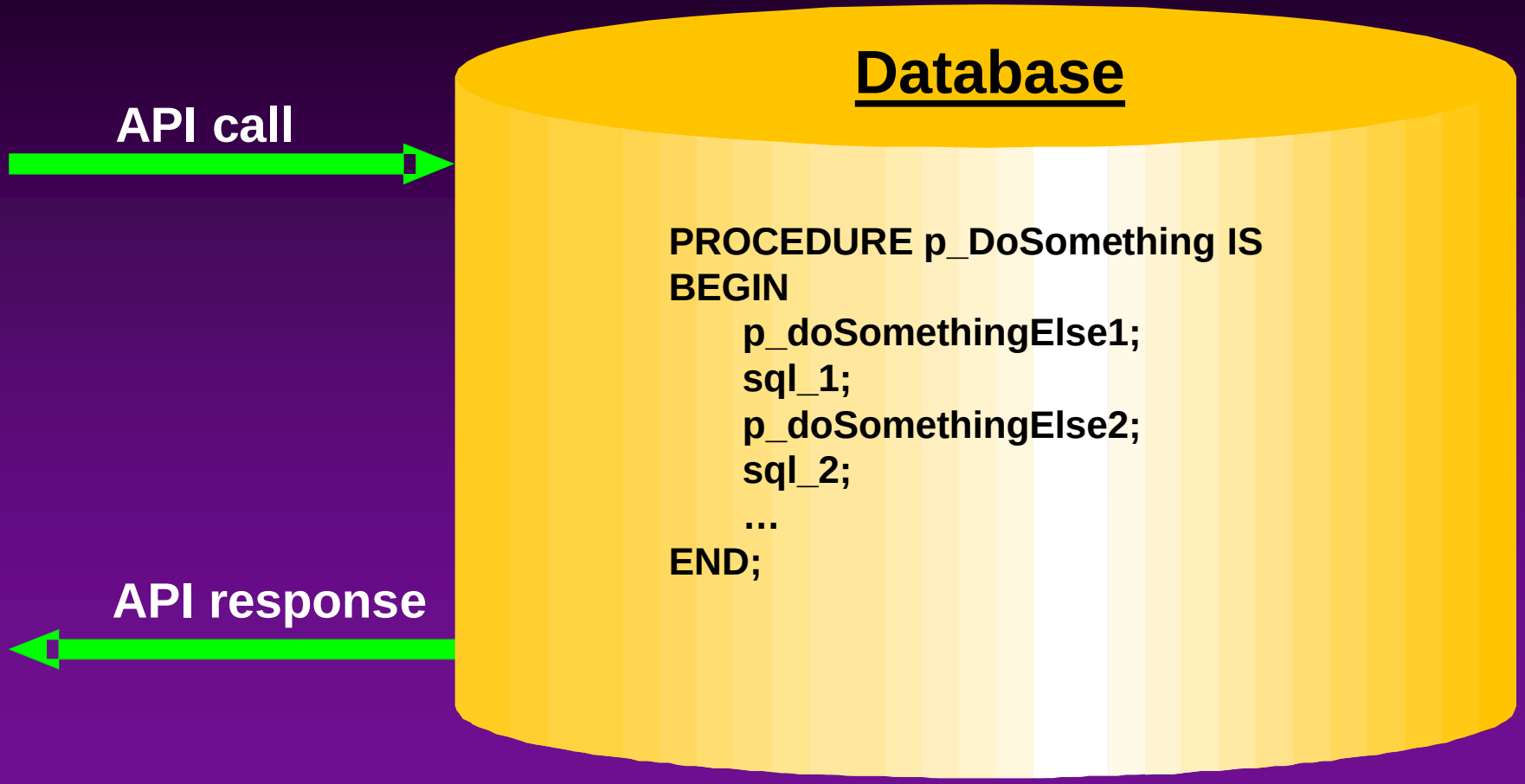


DULCIAN INC.

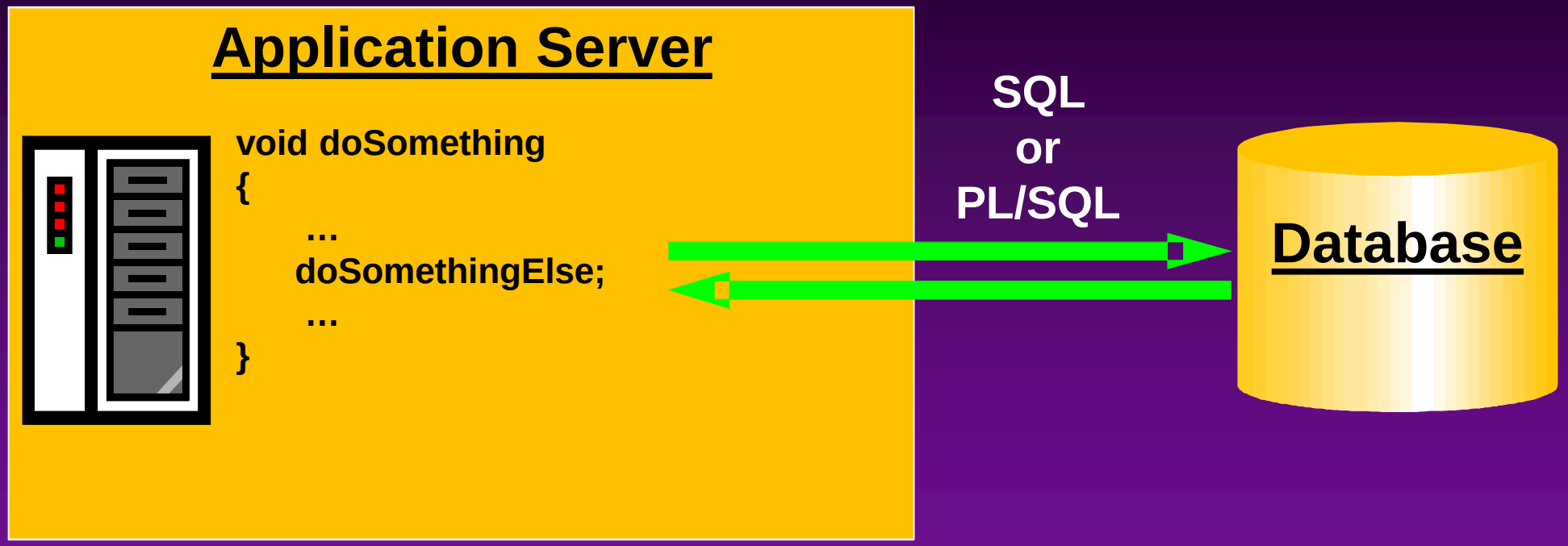
Let's assume….

- You've proven that IT IS a database problem
  - ... and not network traffic/slow client/etc.
  - … and not the number of round trips from the application server!
- You can modify database-related code
  - Best case: You know how to use a "thick database approach"
    - … i.e. you have high level PL/SQL APIs (that call various SQL queries)
    - …and these APIs are called by everybody else (UI/reports/BI/etc.)
  - Worst case: If needed, you can add diagnostic PL/SQL calls around SQL.

A Perfect World



Less Than Perfect World



DULCIAN INC.

THE Problem

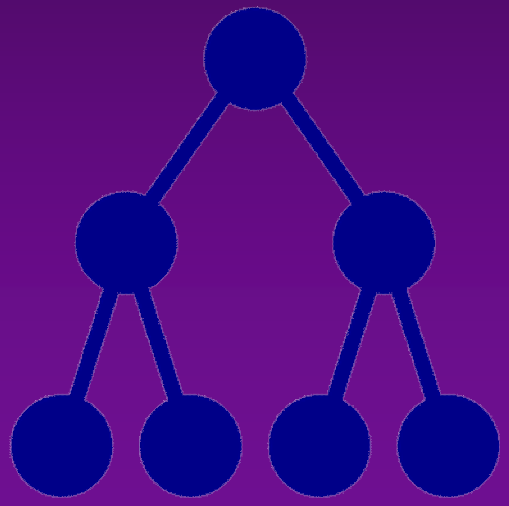
- Database is spending too much time doing something:
  - Perfect Case [one SQL statement that does not contain any user-defined functions]
    - Many monitoring mechanisms
    - Many ways to adjust
    - Lots of coverage

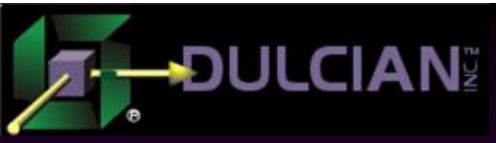
  - Real case [combination of SQL and PL/SQL]
    - Hierarchical in its nature → something is calling something that is calling something else
    - Cannot be represented as a sequence of simple cases!

DULCIAN INC.

The Hero

PL/SQL Hierarchical Profiler





What can it do for you?

- PL/SQL Hierarchical Profiler:
  - Gathers hierarchical statistics of all calls (both SQL and PL/SQL) for the duration of the monitoring
    - … into a portable trace file
  - Has powerful aggregation utilities
    - … both within the database and using a command-line interface
  - Available since Oracle 11.1 [replaced PL/SQL Profiler]
    - … and constantly improved/adjusted even in 18c

DULCIAN INC.

Introductory Case

- Background:
  - You have multiple PL/SQL program units calling each other that have SQL statements within them.
- Problem:
  - You need to know where time is wasted and where it would be best to spend time on tuning.

Intro (1)

```
SQL> CREATE DIRECTORY IO AS 'C:\IO';
SQL> exec dbms_hprof.start_profiling
                  (location=>'IO',filename=>'HProf.txt');

SQL> DECLARE
2       PROCEDURE p_doSomething (pi_empno NUMBER) IS
3       BEGIN
4           dbms_lock.sleep(0.1);
5       END;
6       PROCEDURE p_main IS
7       BEGIN
8           dbms_lock.sleep(0.5);
9           FOR c IN (SELECT * FROM emp) LOOP
10              p_doSomething(c.empno);
11          END LOOP;
12      END;
13 BEGIN
14      p_main();
15 END;
16 /
SQL> exec dbms_hprof.stop_profiling;
```

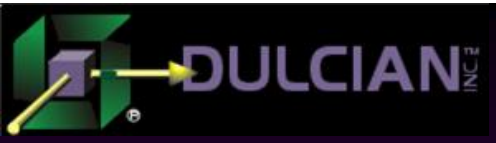
Destination folder:
WRITE is enough

Spend time

Intro (2)

- Raw file (C:\IO\HProf.txt) is not very readable…


```
P#V PLSHPROF Internal Version 1.0
P#! PL/SQL Timer Started
P#C PLSQL."".""."__plsql_vm"
P#X 8
P#C PLSQL."".""."__anonymous_block"
P#X 6
P#C PLSQL."".""."__anonymous_block.P_MAIN"#980980e97e42f8ec #6
P#X 63
P#C PLSQL."SYS"."DBMS_LOCK"::9."__pkg_init"
P#X 7
P#R
P#X 119
P#C PLSQL."SYS"."DBMS_LOCK"::11."SLEEP"#e17d780a3c3eae3d #197
P#X 500373
P#R
P#X 586
P#C SQL.""."."__sql_fetch_line9" #9."4ay6mhcbhvbf2"
P#! SELECT * FROM SCOTT.EMP
P#X 3791
P#R
P#X 17
<<… and so on …>>
```



Intro (3)

- … but you can and make it readable via the command-line utility:

```
C:\Utl_File\IO>plshprof -output hprof_intro HProf.txt
PLSHPROF: Oracle Database 12c Enterprise Edition Release 12.2.0.1.0
  - 64bit Production
[8 symbols processed]
[Report written to 'hprof_intro.html']
```

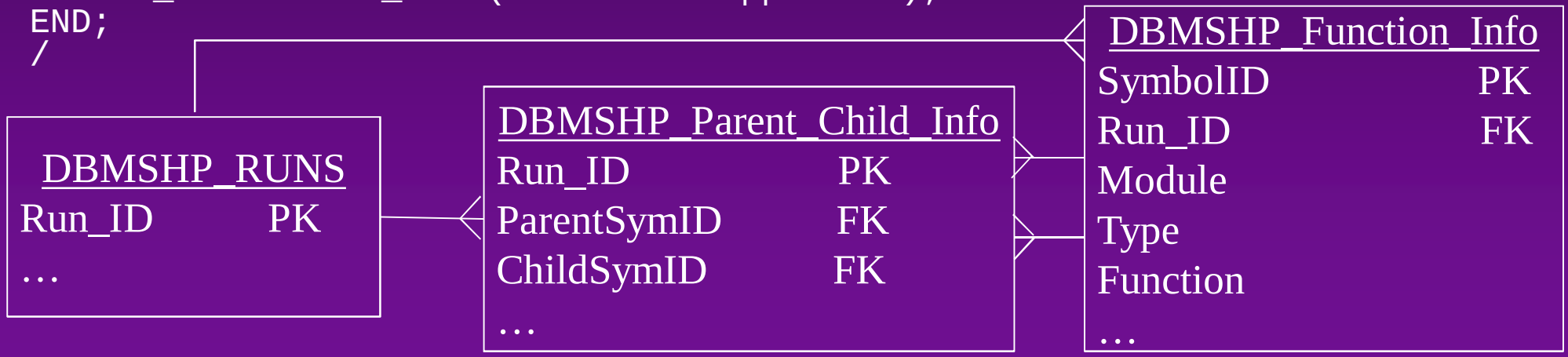
Intro Findings

- The results are:
  - All of the time is spent in DBMS_LOCK.SLEEP
    - …There are no descendants!
  - When we drill down, the SLEEP procedure was called from multiple parent modules!
    - This is important because, in one case, time spent is 0.1 per call and in the other is 0.5 per call.
  - Oracle 12.2+ → SQL ID and first 50 characters of SQL text
    - Very nice, especially in the case of Dynamic SQL
- Many sorting/reporting options!

Intro (4)

- … and also you can analyze the trace file via PL/SQL APIs
  - Pro: easier to link with SQL statistics
  - Contra: need extra READ privilege on the directory + need to create tables beforehand

```
DECLARE
  runid NUMBER;
BEGIN
  runid := DBMS_HPROF.analyze('IO','HProf.txt');
  DBMS_OUTPUT.PUT_LINE('runid = ' || runid);
END;
/
```



Intro (5)

- … btw, ANALYZE has some nice options:
  - Trace only specific entries

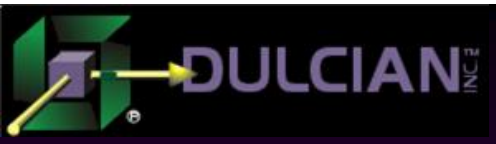
```
runid := DBMS_HPROF.analyze('IO','HProf.txt',
                            trace=> '"SCOTT"."F_CHANGE_TX"');
```

  - Trace up to N occurrences

```
runid := DBMS_HPROF.analyze('IO','HProf.txt',
                            collect => 20,
                            trace=> '"SCOTT"."F_CHANGE_TX"');
```

  - Trace starting from N-th occurrence

```
runid := DBMS_HPROF.analyze('IO','HProf.txt',
                           skip =>1,
                            trace=> '"SCOTT"."F_CHANGE_TX"');
```

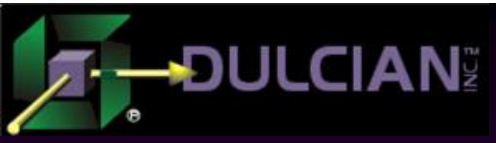


True Story #1:
Typical Hierarchical Profiler Use

DULCIAN INC.

Typical Situation

- Help-desk client's performance complaints:
  - Developer checked 10046 trace and couldn't find anything suspicious
  - I noticed that the core query contains a user-defined PL/SQL function.
- Action:
  - Wrap suspicious call in HProf start/stop in TEST instance (with the same volume of data)



Suspect

```
SQL> exec dbms_hprof.start_profiling ('IO', 'HProf_Case1.txt');

SQL> declare
  2      v_tx varchar2(32767);
  3  begin
  4      select listagg(owner_tx,',') within group (order by 1)
  5      into v_tx
  6      from (
  7          select distinct scott.f_change_tx(owner) owner_tx
  8          from scott.test_tab
  9          );
 10  end;
 11  /

SQL> exec dbms_hprof.stop_profiling;
```

1. Only 26 owners!
2. Function is doing basic formatting

Profile

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

508391 microsecs (elapsed time) & 100006 function calls

Subtree	Ind%	Function	Ind%	Descendants	Ind%	Calls	Ind%	Function Name	SQL ID	SQL TEXT
508391	100%	14	0.0%	508377	100%	2	0.0%	__plsql_vm		
508377	100%	171	0.0%	508206	100%	2	0.0%	__anonymous_block		
508206	100%	328430	64.6%	179776	35.4%	1	0.0%	__static_sql_exec_line4 (Line 4)	27t27npwd3n0j	SELECT LISTAGG(OWNER_TX,',') WITHIN GROUP (ORDER B
179776	35.4%	66436	13.1%	113340	22.3%	50000	50.0%	__plsql_vm@1		
113340	22.3%	113340	22.3%	0	0.0%	50000	50.0%	SCOTT.F_CHANGE_TX.F_CHANGE_TX (Line 1)		
0	0.0%	0	0.0%	0	0.0%	1	0.0%	SYS.DBMS_HPROF.STOP_PROFILING (Line 453)		

50k calls?!

Here is my time!

Findings

- Problem:
  - Time is wasted on very cheap function which is fired lots and lots of times
  - … because the original developer "guessed" at the query behavior
  - … i.e. he knew function was doing basic formatting, so the output would also be distinct
  - … but forgot to tell that to the CBO ➔ GIGO!
- Solution:
  - Rewrite query in a way that helps the CBO
  - … and remind all developers:
    - The number of function calls in SQL will surprise you if you don't measure them.

Fix

```
SQL> exec dbms_hprof.start_profiling ('IO', 'HProf_Case1_fix.txt');

SQL> declare
  2      v_tx varchar2(32767);
  3  begin
  4      select listagg(owner_tx,',') within group (order by 1)
  5      into v_tx
  6      from (
  7          select  scott.f_change_tx(owner) owner_tx
  8          from (select distinct owner
  9                from scott.test_tab)
 10           );
 11  end;
 12  /

SQL> exec dbms_hprof.stop_profiling
```

Filter first!

<Show files>

Updated Profile

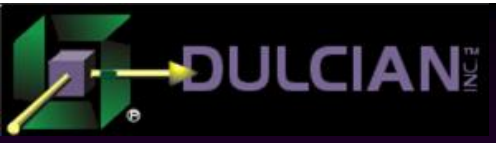
Function Elapsed Time (microsecs) Data sorted by Total Subtree Elapsed Time (microsecs)

18230 microsecs (elapsed time) & 58 function calls

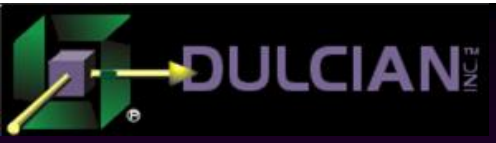
Subtree	Ind%	Function	Ind%	Descendants	Ind%	Calls	Ind%	Function Name	SQL ID	SQL TEXT
18230	100%	15	0.1%	18215	100%	2	3.4%	__plsql_vm		
18215	100%	139	0.8%	18076	99.2%	2	3.4%	__anonymous_block		
18076	99.2%	17954	98.5%	122	0.7%	1	1.7%	__static_sql_exec_line4 (Line 4)	b4pduc9z5xybc	SELECT LISTAGG(OWNER_TX,',') WITHIN GROUP (ORDER B
122	0.7%	42	0.2%	80	0.4%	26	44.8%	__plsql_vm@1		
80	0.4%	80	0.4%	0	0.0%	26	44.8%	SCOTT.F_CHANGE_TX.F_CHANGE_TX (Line 1)		
0	0.0%	0	0.0%	0	0.0%	1	1.7%	SYS.DBMS_HPROF.STOP_PROFILING (Line 453)		

28 times faster!

26 calls



Extra Test:
SQL in Java and SQL*Plus



Running directly from Java?

- Good news:
  - It works!
  - You can run multiple statements between START and STOP
- Bad news:
  - No SQL IDs if they run directly (at least we couldn't get it) ➔ confused statistics ☹
    - Environment: JDeveloper 11g

Java Sample

```
String sql =
"begin dbms_hprof.start_profiling (location=>'IO',filename=>'Case1a.txt'); end;";
CallableStatement stmt = conn.prepareCall(sql);
stmt.execute();

PreparedStatement stmt2 =
conn.prepareStatement("select listagg(owner_tx,',') within group (order by 1) result \n" +
  "from (select distinct  scott.f_change_tx(owner) owner_tx\n" +
  "    from scott.test_tab) A ");
stmt2.execute();

stmt2 = conn.prepareStatement("select listagg(owner_tx,',') within group (order by 1) \n" +
    "from (select distinct  scott.f_change_tx(owner) owner_tx\n" +
    "    from scott.test_tab) B ");
stmt2.execute();

sql = "begin dbms_hprof.stop_profiling; end;";
stmt = conn.prepareCall(sql);
stmt.execute();
```

Difference!

<Show files>

Impact - Java

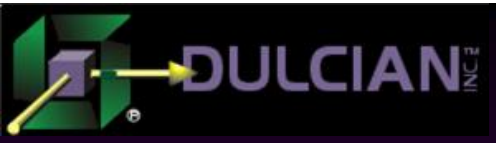
Function Elapsed Time (microsecs) Data sorted by Total Subtree Elapsed Time (microsecs)

368188 microsecs (elapsed time) & 200003 function calls

Subtree	Ind%	Function	Ind%	Descendants	Ind%	Calls	Ind%	Function Name	SQL ID	SQL TEXT
368188	100%	138805	37.7%	229383	62.3%	100001	50.0%	__plsql_vm		
229279	62.3%	229279	62.3%	0	0.0%	100000	50.0%	SCOTT.F_CHANGE_TX.F_CHANGE_TX (Line 1)		
104	0.0%	104	0.0%	0	0.0%	1	0.0%	__anonymous_block		
0	0.0%	0	0.0%	0	0.0%	1	0.0%	SYS.DBMS_HPROF.STOP_PROFILING (Line 453)		

No SQL IDs

100k Calls



Running directly from SQL*Plus?

- Bad news: the same problem with multiple statements:

```
SQL> exec dbms_hprof.start_profiling
  2          (location=>'IO',filename=>'Case1b_SQLPlus.txt');

SQL> select listagg(owner_tx,',') within group (order by 1)
  2  from (select distinct  scott.f_change_tx(owner) owner_tx
  4      from scott.test_tab a);
...
SQL> select listagg(owner_tx,',') within group (order by 1)
  2  from (select distinct  scott.f_change_tx(owner) owner_tx
  4      from scott.test_tab b);
...

SQL> exec dbms_hprof.stop_profiling;
```

Impact – SQL*Plus

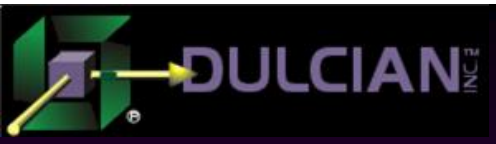
Function Elapsed Time (microsecs) Data sorted by Total Subtree Elapsed Time (microsecs)

360092 microsecs (elapsed time) & 200003 function calls

Subtree	Ind%	Function	Ind%	Descendants	Ind%	Calls	Ind%	Function Name	SQL ID	SQL TEXT
360092	100%	136544	37.9%	223548	62.1%	100001	50.0%	__plsql_vm		
223513	62.1%	223513	62.1%	0	0.0%	100000	50.0%	SCOTT.F_CHANGE_TX.F_CHANGE_TX (Line 1)		
35	0.0%	35	0.0%	0	0.0%	1	0.0%	__anonymous_block		
0	0.0%	0	0.0%	0	0.0%	1	0.0%	SYS.DBMS_HPROF.STOP_PROFILING (Line 453)		

No SQL IDs

100k Calls

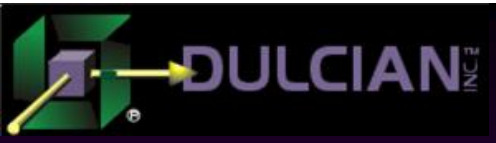


True Story #2: Unexpected Usage

DULCIAN INC.

Background

- Third-party module code is slow
  - Functionality: Take some tables and columns /return formatted CLOB
  - The code is wrapped
  - Original developers don’t want to accept the blame.
- Action:
  - Gather as many statistics about the module as you can
  - Wrap suspicious call in HProf start/stop



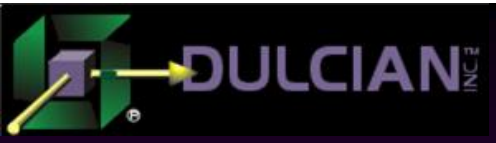
Statistics (1)

```
SQL> exec runstats_pkg.rs_start;
SQL> DECLARE
2    v_CL CLOB;
3 BEGIN
4    v_cl :=wrapped_pkg.f_getdata_cl('ename','emp');
5    dbms_output.put_line('length:'||LENGTH(v_cl));
6 END;
7 /
length:84
SQL> exec runstats_pkg.rs_middle;
SQL> DECLARE
2    v_CL CLOB;
3 BEGIN
4    v_cl :=wrapped_pkg.f_getdata_cl('object_name','test_tab');
5    dbms_output.put_line('length:'||LENGTH(v_cl));
6 END;
7 /
length:1247887
```

Wrapped module

14 rows

50000 rows



Statistics (2)

```
SQL> exec runstats_pkg.rs_stop;
Run1 ran in 0 cpu hsecs
Run2 ran in 3195 cpu hsecs
run 1 ran in 0% of the time
```

Name	Run1	Run2	Diff
...			
STAT...physical reads direct (lob)	13	49,991	49,978
STAT...physical reads direct temporary tablespace	13	49,991	49,978
STAT...lob writes	14	50,000	49,986
STAT...physical writes direct temporary tablespace	14	50,145	50,131
STAT...physical writes direct (lob)	14	50,145	50,131

Direct Temp I/O?!?!

Profile for the Slow Case

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

57671407 microsecs (elapsed time) & 100010 function calls

Subtree	Ind%	Function	Ind%	Descendants	Ind%	Calls	Ind%	Function Name
57671288	100%	1304042	2.3%	56367246	97.7%	1	0.0%	SCOTT.WRAPPED_PKG.F_GETDATA_CL (Line 3)
50800744	88.1%	50800744	88.1%	0	0.0%	50000	50.0%	SYS.DBMS_LOB.WRITEAPPEND (Line 1142)
5565739	9.7%	5565739	9.7%	0	0.0%	50001	50.0%	SCOTT.WRAPPED_PKG.__sql_fetch_line14 (Line 14)
478	0.0%	478	0.0%	0	0.0%	1	0.0%	SCOTT.WRAPPED_PKG.__dyn_sql_exec_line10 (Line 10)
190	0.0%	190	0.0%	0	0.0%	2	0.0%	SYS.DBMS_ASSERT.SIMPLE_SQL_NAME (Line 153)
119	0.0%	12	0.0%	107	0.0%	1	0.0%	SYS.DBMS_OUTPUT.PUT_LINE (Line 109)
103	0.0%	103	0.0%	0	0.0%	1	0.0%	SYS.DBMS_OUTPUT.PUT (Line 77)
95	0.0%	95	0.0%	0	0.0%	1	0.0%	SYS.DBMS_LOB.CREATETEMPORARY (Line 720)
4	0.0%	4	0.0%	0	0.0%	1	0.0%	SYS.DBMS_OUTPUT.NEW_LINE (Line 117)
0	0.0%	0	0.0%	0	0.0%	1	0.0%	SYS.DBMS_HPROF.STOP_PROFILING (Line 59)

50k calls

Explicit "create temp"

<Show files>

DULCIAN INC.

Analysis

- Problem #1: Direct IO for all temporary LOB operations
  - Could happen only if LOB variable is initiated as NOCACHE via *DBMS_LOB.createTemporary*
- Problem #2: IO operation for every row in conjunction with fetch for every row
  - Could happen only if *DBMS_LOB.writeAppend* is called within the loop

Unwrapped code (FYI)

```
FUNCTION f_getData_cl(i_column_tx VARCHAR2, i_table_tx VARCHAR2) RETURN CLOB IS
    v_cl CLOB;
    v_tx VARCHAR2(32767);
    v_cur SYS_REFCURSOR;
BEGIN
    dbms_lob.createTemporary(v_cl,false,dbms_lob.call);
    OPEN v_cur FOR 'SELECT '||
                   dbms_assert.simple_sql_name(i_column_tx)||' field_tx'||
                   ' FROM '||dbms_assert.simple_sql_name(i_table_tx);
    LOOP
        FETCH v_cur into v_tx;
        EXIT WHEN v_cur%notfound;
        dbms_lob.writeAppend(v_cl,length(v_tx)+1,v_tx||'|');
    END LOOP;
    CLOSE v_cur;
    RETURN v_cl;
END;
```

Issue #1: no cache

Issue #2: no buffer

<Show fixed code if time permits>

Summary

- End users only care that their requests come back quickly … and not about CPU/Memory/IO utilization
- Yes, sometimes it IS the database ☹
  - … but 90% of time it isn't ☺

- PL/SQL Hierarchical profiler lets you see the system from the end-user angle and find real performance issues
  - …i.e. request-driven (with drill-down option)
- PL/SQL Hierarchical profiler is constantly improving
  - .. i.e. don't forget to read "New Features" guide!

Contact Information

- Michael Rosenblum – mrosenblum@dulcian.com
- Dulcian, Inc. website - www.dulcian.com
- Blog: wonderingmisha.blogspot.com