So..... now what?

A survey of built-in diagnostic tools

Part 0: A preliminary rant

(This is (sort-of) relevant, I promise)

Part 1: Dvoraky

(aka Just some made-up typing-practice game)

Rules

- To simulate real sentences, the app will generate.... Real sentences.

- To minimize boredom, the app will shuffle words, mad-libs-style

- To incentivize progress, the app will keep a High-Score history

The App

- Generate random sentences with replacements into templates

- Record typing accuracy and speed

- Save the results to a score history and all-time scoreboard.

Examples

<Exclamation>! <Integer> <Noun:[Creature,Plural]>
<Verb:[Intransitive,Past,Motion]> from the
<Noun:[Vehicle,Singular]> and presented
<Noun:[Historical-Figure,Singular]> with <Noun:[Any]>.

<Exclamation>! <Integer> <Noun:[Creature,Plural]>
<Verb:[Intransitive,Past,Motion]> from the
<Noun:[Vehicle,Singular]> and presented
<Noun:[Historical-Figure,Singular]> with <Noun:[Any]>.

Whoa! Twelve Hobbits sprinted from the monster truck and presented Abraham Lincoln with a scarf.

Behold! Nine Wookies strutted from the submarine and presented Stanley Kubrick with a riverdance.

Wow! Three Captain Americas galloped from the elevator and presented Charles Darwin with a teapot.

But....

Generating sentences is too slow.
Something is wrong.

So.... now what?

Part 2: Coarse Radar

First, a disclaimer

More disclaimers

(This is just one approach)

First stop: (G)V$SQL_MONITOR

V$SQL_MONITOR Features

- A repository of recent (and current!) long-running statements

- **Homing missiles** for the problem:
  - Elapsed-time, cpu-time, io stats, waits, parallel
  - Which app, which user, which session
  - The slow statement, in plain text !

Query is stuck right now

- V$SQL_MONITOR has your back.

Lets try it

- In the app, load the next practice sentence…
- While the app is loading…

```
SELECT
 ELAPSED_TIME AS TOT_TIME,
 CPU_TIME AS CPU_TIME,
 APPLICATION_WAIT_TIME AS APP_TIME,
 CLUSTER_WAIT_TIME AS CLU_TIME,
 USER_IO_WAIT_TIME AS IO_TIME,
 PLSQL_EXEC_TIME AS PL_TIME,
 SQL_TEXT AS SQL_TEXT
FROM GV$SQL_MONITOR
WHERE STATUS = 'EXECUTING'
ORDER BY ELAPSED_TIME DESC;
```

TOT_TIME	CPU_TIME	APP_TIME	CLU_TIME	IO_TIME	PL_TIME	SQL_TEXT
41463831	25307784	0	236	17045550	17099705	SELECT TYPING_PRACTICE.NEXT_TEMPLATE() FROM DUAL
41446431	25304094	0	0	17031026	17099673	SELECT TYPING_TEMPLATE_TEXT FROM TYPING_TEMPLATE ORDER BY DBMS_RANDOM.*VALUE*(**1**,**2123456789**) OFFSET **0** ROWS FETCH FIRST **1** ROWS ONLY;

```
SELECT TYPING_TEMPLATE_TEXT
FROM TYPING_TEMPLATE
ORDER BY DBMS_RANDOM.VALUE(1 , 2123456789)
OFFSET 0 ROWS FETCH FIRST 1 ROWS ONLY;
```

```
SELECT TYPING_TEMPLATE_TEXT
FROM TYPING_TEMPLATE
WHERE TYPING_TEMPLATE_ID = DBMS_RANDOM.VALUE(1, :MAX_TEMPLATE_ID);
```

** Gaps? Buggy approach? I know. Don’t worry.

But...

TOT_TIME	CPU_TIME	APP_TIME	CLU_TIME	IO_TIME	PL_TIME	SQL_TEXT
85528677	23796525	0	1158	64228108	15184926	SELECT TYPING_PRACTICE.NEXT_TEMPLATE() FROM DUAL
64406690	21191075	0	0	44392520	15184852	SELECT TYPING_TEMPLATE_TEXT FROM TYPING_TEMPLATE ORDER BY DBMS_RANDOM.*VALUE*(1,2123456789) OFFSET 0 ROWS FETCH FIRST 1 ROWS ONLY;
21092675	2596472	0	0	19817460	0	SELECT *MAX*(TYPING_TEMPLATE_ID) FROM TYPING_TEMPLATE

Next Stop: DBMS_SQLTUNE

DBMS_SQLTUNE

Three main steps:

- Create a tuning task (based on the monitor's findings).

  DBMS_SQLTUNE.*CREATE_TUNING_TASK*

- Let the database work on it for a while.

  DBMS_SQLTUNE.*EXECUTE_TUNING_TASK*

- See what it figured out.

  DBMS_SQLTUNE.*REPORT_TUNING_TASK*

Next tool: DBMS_SQLTUNE

```
DECLARE
 V_SQL CLOB;
 V_TUNE_TASK VARCHAR2(64 BYTE);
BEGIN
 V_SQL := 'SELECT TYPING_TEMPLATE_TEXT FROM TYPING_TEMPLATE WHERE TYPING_TEMPLATE_ID = DBMS_RANDOM.VALUE(1,:B1 )';
 V_TUNE_TASK := DBMS_SQLTUNE.CREATE_TUNING_TASK(SQL_TEXT => V_SQL,
                        BIND_LIST => SQL_BINDS(ANYDATA.CONVERTNUMBER(5000000)),
                        USER_NAME => USER,
                        SCOPE => 'COMPREHENSIVE',
                        TIME_LIMIT => 300,
                        task_name => 'RANDOM_TYPING_TEMPLATE_TASK_1');
DBMS_SQLTUNE.EXECUTE_TUNING_TASK(TASK_NAME => V_TUNE_TASK, EXECUTION_NAME =>
'RANDOM_TYPING_TEMPLATE_EXEC_1');
END;
/
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK('RANDOM_TYPING_TEMPLATE_TASK_1') FROM DUAL;
```

Result:

```
SQL Text   : SELECT TYPING_TEMPLATE_TEXT
             FROM TYPING_TEMPLATE
             WHERE TYPING_TEMPLATE_ID = DBMS_RANDOM.VALUE(1,:B1 )

  Bind Variables :
  1 -  (NUMBER):5000000
  ---------------------------------------------------------------------------
  FINDINGS SECTION (1 finding)
  ---------------------------------------------------------------------------

  1- Index Finding (see explain plans section below)
  --------------------------------------------------
  The execution plan of this statement can be improved by creating one or more  indices.
```

Recommendation (estimated benefit: 99.98%)

```
  ----------------------------------------
  - Consider running the Access Advisor to improve the physical schema design  or creating the recommended index.
```

create index IDX$$_7EFE50001 on TYPING_TEMPLATE("TYPING_TEMPLATE_ID");

The app is stuck right now

(G)V$SQL_MONITOR features:

-- Filter by user, filter by module/action, filter by client-id

-- It straight-up includes the SQL-Text right-there

-- But it is coarse. Diagnostic. You'll need to go and debug the problem statement.

---- (Just Kidding! It even has optimizer info (**V$SQL_MONITOR_PLAN**)

What if the problem query already finished?

V$SQL_MONITOR has your back there too!

What if the problem query already finished?

```
SELECT * FROM V$SQL_MONITOR …

-- What user ran the statement?
… WHERE USERNAME = 'LOREM_IPSUM';
… WHERE CLIENT_IDENTIFIER = 'LOREM_IPSUM';
```

What if the problem query already finished?

```
SELECT * FROM V$SQL_MONITOR …

-- What application?
… WHERE MODULE = 'LOREM_IPSUM';
… WHERE ACTION = 'LOREM_IPSUM';
```

What if the problem query already finished?

```
SELECT * FROM V$SQL_MONITOR …

-- When?
… WHERE LAST_REFRESH_TIME > (SYSDATE - NUMTODSINTERVAL(10,'MINUTE'));
```

What if the problem query already finished?

```
SELECT * FROM V$SQL_MONITOR …

-- ANY OTHER Session-Info?
… WHERE SID = 123;
```

But… there’s nothing here

Monitor vision

The monitor only tracks statements that ***accumulate* 5s** CPU/IO ***execution-time***

- ***execution-time*** != ***clock-time***
- ***execution-time*** != ***wait-time*** ***

*** We'll come back to this

Monitor vision

- Waits/Sleeps
- Just-Fast-Enough
- Parallel...?

Part 3: Under the Radar

Monitor vision

- Waits/Sleeps
- Just-Fast-Enough
- **Parallel...?**

Parallel queries

- V$SQL_MONITOR ***still*** has your back.

- Execution-time rollup

- ***execution-time*** != ***clock-time***
- ***execution-time*** != ***wait-time***

Parallel queries

```
SELECT /*+ PARALLEL(8) */ COUNT(ALL *)
FROM TYPING_TEMPLATE;
```

Elapsed: 00:00:00.729

Parallel queries

```
SELECT TO_CHAR(FIRST_REFRESH_TIME,'MI:SS') AS FIRST_REFRESH_TIME,
    TO_CHAR(LAST_REFRESH_TIME,'MI:SS') AS LAST_REFRESH_TIME,
    REFRESH_COUNT,
    SQL_TEXT
FROM V$SQL_MONITOR
 WHERE V$SQL_MONITOR.SQL_TEXT = 'SELECT /*+ PARALLEL(8) */ COUNT(ALL *)
FROM TYPING_TEMPLATE';
```

FIRST_REFRESH_TIME	LAST_REFRESH_TIME	REFRESH_COUNT
54:51	54:52	4

Monitor vision

- Waits/Sleeps
- **Just-Fast-Enough**
- Parallel...?

Just-Fast-Enough

```
SELECT COUNT(*) FROM TYPING_TEMPLATE;
```

5000000

1 row selected.

Elapsed: 00:00:02.395

Just-Fast-Enough

- V$SQL_MONITOR ***STILL*** has your back.

- Four-Step process:
  - **Force-start** monitor
  - Do whatever
  - Stop the monitor
  - See what happened

Just-Fast-Enough

```
DECLARE
 V_MON_ID NUMBER;
BEGIN
 V_MON_ID := DBMS_SQL_MONITOR.BEGIN_OPERATION(
   DBOP_NAME => 'TYPING_RADAR',   DBOP_EID => 1,   FORCED_TRACKING => 'Y');  --DBOP_EID is OPTIONAL
END;
/

BEGIN
 DVORAKY.START_PRACTICE_RUN();
 -- etc. etc. Run the app .....
 DVORAKY.STOP_PRACTICE_RUN();
END;
/

BEGIN
 DBMS_SQL_MONITOR.END_OPERATION( DBOP_NAME => 'TYPING_RADAR', DBOP_EID => 1);
END;
/
```

Just-Fast-Enough

```
SELECT * FROM GV$SQL_MONITOR
WHERE GV$SQL_MONITOR.DBOP_NAME = 'TYPING_RADAR'
AND GV$SQL_MONITOR.DBOP_EXEC_ID = 1;
```

Monitor vision

- **Waits/Sleeps**
- Just-Fast-Enough
- Parallel...?

At last!

Monitor does not have your back.

It won’t see “all” the application waits.

New example

Generating the sentences is fixed, but saving is slow…

SQL_MONITOR doesn’t show anything

It is hard to reproduce -- testing seems ok, but customers are complaining

So.... Now what?

Network?

Concurrency?

Cluster?

Return to the monitor

- ***execution-time*** != ***clock-time***
- ***execution-time*** != ***wait-time***

Wait Classes

Oracle watches, categorizes every moment of session time

“Productive” time (Commit, I/O, Network traffic, Queueing, PL/SQL)

** Monitor tracks these **

“Application” time (Row Locks, Contention)

** Monitor (kind of) excludes these **

A little bad news about Application Wait time….

Wait Classes

A little bad news about Application Wait time....

https://unsplash.com/photos/aoN3HWLbhdI
Photo by Burst on Unsplash

Where did the time go?

First let’s find ***generally*** where the time is

Then, home-in on the issue

V$ACTIVE_SESSION_HISTORY

Not sure which session you want? (*customers* are complaining, not you)

Not sure what you're looking for?

ASH has your back.

V$ACTIVE_SESSION_HISTORY

Get a base-line:

```
SELECT WAIT_CLASS, COUNT(*) AS SAMPLE_COUNT
FROM V$ACTIVE_SESSION_HISTORY
WHERE SAMPLE_TIME > (SYSDATE - 1/48)
GROUP BY WAIT_CLASS
ORDER BY 2 DESC;
```

V$ACTIVE_SESSION_HISTORY

WAIT_CLASS	SAMPLE_COUNT
User I/O	7820
null	3540
System I/O	2722
Other	795
Commit	380
Cluster	185
Application	103
Network	7

V$ACTIVE_SESSION_HISTORY

Check your app:

```
SELECT WAIT_CLASS, COUNT(*) SAMPLE_COUNT
FROM V$ACTIVE_SESSION_HISTORY
WHERE SAMPLE_TIME > (SYSDATE - 1/48)
-- filter by app, user, client, machine, etc. etc. etc!
  AND (CLIENT_ID = 'MY_ID'
 OR MODULE = 'MY_MODULE'
 OR MACHINE = 'MY_MACHINE'
 OR PROGRAM = 'MY_PROGRAM')
GROUP BY V$ACTIVE_SESSION_HISTORY.WAIT_CLASS
ORDER BY 2 DESC;
```

V$ACTIVE_SESSION_HISTORY

WAIT_CLASS	SAMPLE_COUNT
Application	95
User I/O	11
Commit	5
null	3
Network	3

Let’s compare those

V$ACTIVE_SESSION_HISTORY

That’s where the time went!

This app is not like the rest of the DB…

V$ACTIVE_SESSION_HISTORY

So, what’s the bottleneck?

You could check the code… or let ASH do it for you!

V$ACTIVE_SESSION_HISTORY

```
SELECT
 SQL_ID,
 USER_ID,
 SQL_OPNAME,
 BLOCKING_SESSION AS BLOCKER
FROM V$ACTIVE_SESSION_HISTORY
WHERE SAMPLE_TIME > (SYSDATE - 1/24)
 AND MODULE = 'DVORAKY'
 AND WAIT_CLASS = 'Application'
ORDER BY SAMPLE_TIME ASC;
```

V$ACTIVE_SESSION_HISTORY

SQL_ID	USER_ID	SQL_OPNAME	BLOCKER
7kv24vvdvtpqu	898	UPDATE	772
7kv24vvdvtpqu	898	UPDATE	772
7kv24vvdvtpqu	898	UPDATE	772
7kv24vvdvtpqu	772	UPDATE	755
7kv24vvdvtpqu	772	UPDATE	755

Tuning

https://unsplash.com/photos/SiDXdjE-T_k
Photo by Olav Ahrens Røtne on Unsplash

V$ACTIVE_SESSION_HISTORY

7kv24vvdvtpqu (**don’t worry to read this complex gross thing, there’s a pattern**)

```
UPDATE DVORAK_SCORE_RANKINGS
SET (DVORAK_TYPING_SCORE , DVORAK_TYPING_RANK) = (
 SELECT DVORAK_TYPING_SCORE, DVORAK_TYPING_RANK
 FROM (
   SELECT DVORAK_TYPIST_NAME,
     MAX_SCORE               AS DVORAK_TYPING_SCORE,
     ROW_NUMBER() OVER (ORDER BY MAX_SCORE DESC,
       DVORAK_TYPIST_NAME ASC) AS DVORAK_TYPING_RANK
  FROM (SELECT DVORAK_TYPIST_NAME,
          MAX(DVORAK_TYPING_SCORE) AS MAX_SCORE
        FROM DVORAK_TYPIST_SCORE
        GROUP BY DVORAK_TYPIST_NAME) TYPIST_BEST_SCORE) RANKED_SCORE
 WHERE RANKED_SCORE.DVORAK_TYPIST_NAME = DVORAK_SCORE_RANKINGS.DVORAK_TYPIST_NAME);
```

V$ACTIVE_SESSION_HISTORY

7kv24vvdvtpqu (Simplified)

```
UPDATE TARGET_TABLE
SET (TARGET_DATA) = (REPLACEMENT_DATA);
```

Where's the **WHERE** ?

Re-setting the ***whole*** ranking ***every*** time ***anyone*** saves.

V$ACTIVE_SESSION_HISTORY

- Tracks where sessions actually spend their time
- Includes APPLICATION WAITs missing from V$SQL_MONITOR
- Gives the problem SQL right there!
- Shows what is happening right now
- Gives a (little) history of the system

Side-Note

- For longer history on the top-offenders, see AWR
- DBA_HIST_*

ASH Questions

- Only one snapshot per second --- Where’s the rest of every second?
- What about memory leaks? Cursor leaks?
- Are there higher-res data available?
- Are there breakouts by operation? By time? By resources?

Yes

V$SESSION_EVENT

- Convenient. Already-aggregated!
- Accumulated total wait, mean, max, by type
  Example: How much IO waiting is my app doing?

```
SELECT SID, V$SESSION_EVENT.EVENT, TOTAL_WAITS, TIME_WAITED, AVERAGE_WAIT, MAX_WAIT
FROM V$SESSION_EVENT
    INNER JOIN V$SESSION
USING (SID)
WHERE MODULE = 'DVORAKY'
  AND V$SESSION_EVENT.WAIT_CLASS = 'User I/O';
```

V$SESSTAT, V$MYSTAT

- Convenient. Already-aggregated!
- Accumulated total operation-counts, time-spent, resources-used
- Includes some max values

Example: What is the current and max memory for my app's sessions? (or whatever filter)

```
SELECT SID,
 DISPLAY_NAME,
 VALUE
FROM V$SESSTAT
     NATURAL JOIN V$STATNAME
     NATURAL JOIN V$SESSION
WHERE DISPLAY_NAME LIKE 'session%memory%'
    AND MODULE = 'DVORAKY';
```

V$SESS_TIME_MODEL, V$SESS_IO

Convenient targeted summaries

Part 4: High-Res

So....

- Super-Slow queries tuned!
- Concurrency bugs fixed!
- Memory checked!
- Nothing stands out in the monitor
- Nothing stands out in the ASH

- But….After a while, things get slower...

So.... Now What?

- What to tune?

- Disclaimer (well enough alone...)

- If monitor/ASH are too low-res...

PROFILING

Dvoraky sprocs

Even in this tiny example, there are several moving pieces:

- Cycling through the template sentences

- Randomly-selecting a suitable replacement for each target

- Substituting the replacements

- Can we just get a report of where the time goes, line-by-line?

DBMS_PROFILER

Steps

- 0. create the profiler tables (if not present)
  - PROFTAB.SQL ?
  - Or just create them yourself...
- 1. Start the profiler
- 2. Run whatever you want, as long as you want.
- 3. Flush the profile
- 4. Stop the profiler

Profiler Tables

PLSQL_PROFILER_RUNS:

- Comment/Key on the Run
- Total Time for the profiling
- High-Level Summary

Profiler Tables

PLSQL_PROFILER_UNITS:

For each run:

- Object-Type
- Object-Owner
- Object-Name

Profiler Tables

PLSQL_PROFILER_DATA:

For each Run-Unit:

- Line Number in the object
- Total Occurrences of that line
- Total Time on that line

Let’s Try It

```
CREATE OR REPLACE PACKAGE DVORAKY AUTHID DEFINER
AS
 PROCEDURE START_PRACTICE_RUN;

 PROCEDURE STOP_PRACTICE_RUN;

 FUNCTION  LOAD_NEXT_PRACTICE_SENTENCE RETURN CHARACTER VARYING;

 PROCEDURE SCORE_PRACTICE_SENCENCE (P_SENTENCE IN CHARACTER VARYING);

END DVORAKY;
/
```

Let’s Try It

```
BEGIN
 DVORAKY.START_PRACTICE_RUN();
END;
/
```

```
BEGIN
 DBMS_PROFILER.START_PROFILER(RUN_COMMENT => 'VOLTRON');
END;
/
```

Let’s Try It

```
SELECT PLSQL_PROFILER_DATA.LINE#,
 PLSQL_PROFILER_DATA.TOTAL_TIME,
 PLSQL_PROFILER_DATA.TOTAL_OCCUR
FROM PLSQL_PROFILER_RUNS
    INNER JOIN PLSQL_PROFILER_UNITS
ON PLSQL_PROFILER_RUNS.RUNID = PLSQL_PROFILER_UNITS.RUNID
    INNER JOIN PLSQL_PROFILER_DATA
ON PLSQL_PROFILER_UNITS.RUNID = PLSQL_PROFILER_DATA.RUNID
 AND PLSQL_PROFILER_UNITS.UNIT_NUMBER = PLSQL_PROFILER_DATA.UNIT_NUMBER
WHERE RUN_COMMENT = 'VOLTRON'
AND UNIT_NAME = 'DVORAKY'
AND PLPROF_LINE_EXEC_COUNT > 0
ORDER BY LINE# ASC;
```

Let’s Try It

Nothing so far...

LINE#	TOTAL_TIME	TOTAL_OCCUR

Let’s Try It

```
DECLARE
 V_SENCENCE CHARACTER VARYING(4000);
BEGIN
 V_SENCENCE := DVORAKY.LOAD_NEXT_PRACTICE_SENTENCE();
END;
/



BEGIN
 DBMS_PROFILER.FLUSH_DATA();
END;
/
```

Let’s Try It

After one sentence:

LINE#	TOTAL_TIME	TOTAL_OCCUR
38	999	1
40	0	1
41	0	1
43	**942989**	1
45	**195821780**	1
48	999	1
49	999	1

Let’s Try It

But that’s only one execution…

Let’s Try It

After **10x**:

LINE#	TOTAL_TIME	TOTAL_OCCUR
38	1999	10
40	0	**16**
41	0	**16**
43	**5584964**	**16**
45	**205370705**	**16**
48	3999	10
49	1999	10

Let’s Try It

Weird.

Let’s Try It

After **100x**:

LINE#	TOTAL_TIME	TOTAL_OCCUR
38	8999	100
40	0	**326**
41	0	**326**
43	**6438959**	**326**
45	**349259811**	**326**
48	20999	100
49	8999	100

Let’s Try It

Weird!

Tuning

https://unsplash.com/photos/Q4Honp3Pyqs
Photo by Clint Bustrillos on Unsplash

Let’s Try It

```
SELECT LINE, TEXT
FROM USER_SOURCE
WHERE NAME = 'DVORAKY'
AND LINE BETWEEN 38 AND 56;
```

```
38  BEGIN
39    -- There are tons of templates, Just grab one randomly and make sure it isn't in last 10 cache.
40  LOOP
41      BEGIN
42        --TODO: set the final value here
43        V_RANDOM_TEMPLATE := FLOOR(DBMS_RANDOM.VALUE(1, 15));
44
45      SELECT TYPING_TEMPLATE_ID  INTO V_SELECTED_TEMPLATE
46      FROM TYPING_TEMPLATE  WHERE TYPING_TEMPLATE_ID = V_RANDOM_TEMPLATE
47        AND TYPING_TEMPLATE_ID NOT IN (SELECT COLUMN_VALUE FROM
TABLE(V_RECENT_TEMPLATES));
48  RECORD_USED_TEMPLATE(V_SELECTED_TEMPLATE);
49  RETURN V_SELECTED_TEMPLATE;
```

```
…
BEGIN
 -- There are tons of templates, Just grab one randomly and make sure it isn't in last 10 cache.
 LOOP
   BEGIN
     --TODO: set the final value here
     V_RANDOM_TEMPLATE := ROUND(DBMS_RANDOM.VALUE(1 , 15));

     SELECT TYPING_TEMPLATE_ID  INTO V_SELECTED_TEMPLATE
     FROM TYPING_TEMPLATE  WHERE TYPING_TEMPLATE_ID = V_RANDOM_TEMPLATE
        AND TYPING_TEMPLATE_ID NOT IN (SELECT COLUMN_VALUE FROM TABLE(V_RECENT_TEMPLATES));

     RECORD_USED_TEMPLATE(V_SELECTED_TEMPLATE);
     RETURN V_SELECTED_TEMPLATE;

   EXCEPTION WHEN NO_DATA_FOUND THEN NULL;   END;
 END LOOP;
...
```

```
38  BEGIN
39      -- There are tons of templates, Just grab one randomly and make sure it isn't in last 10 cache.
40  LOOP
41      BEGIN
42          --TODO: set the final value here
43          V_RANDOM_TEMPLATE := FLOOR(DBMS_RANDOM.VALUE(1, 15));
44
45      SELECT TYPING_TEMPLATE_ID  INTO V_SELECTED_TEMPLATE
46      FROM TYPING_TEMPLATE  WHERE TYPING_TEMPLATE_ID = V_RANDOM_TEMPLATE
47          AND TYPING_TEMPLATE_ID NOT IN (SELECT COLUMN_VALUE FROM
    TABLE(V_RECENT_TEMPLATES));
48  RECORD_USED_TEMPLATE(V_SELECTED_TEMPLATE);
49  RETURN V_SELECTED_TEMPLATE;
```

Aha!

First execution… (0 cached) 0% collision

6th execution… (5 cached) 33% collision

11th execution… (10 cached) 67% collision

Aha!

We have **5M** templates! Not **15**.

We could bump that **15** up…

Or just rethink the approach (no looping needed)

```
SELECT COALESCE(MIN(TYPING_TEMPLATE_ID),V_FAILOVER)
    INTO V_SELECTED_TEMPLATE
FROM TYPING_TEMPLATE
WHERETYPING_TEMPLATE_ID >= V_RANDOM_TEMPLATE
   AND TYPING_TEMPLATE_ID NOT IN (SELECT COLUMN_VALUE FROM (V_RECENT_TEMPLATES));
```

Let’s Try It

After **100x**:

LINE#	TOTAL_TIME	TOTAL_OCCUR
38	0	100
41	36424762	100
42	5741593	100
44	322533315	100
50	24002	100
51	6000	100

DBMS_PROFILER

Profiling revealed where to tune!

But... that is very flat data....

What if some of the count/time are from *DIFFERENT* routines?

DBMS_HPROF

For a 3-D profile, DBMS_HPROF has your back!

It has the exact same four(plus setup)-step process as DBMS_PROFILER

DBMS_HPROF

Turn the flat DBMS_PROFILER line-counts into:

DEPTH	PROCEDURE	SUBTREE_TIME	FUNCTION_TIME	CALLS
1	DVORAKY.LOAD_NEXT_SENTENCE	832362044	11799	10
2	DVORAKY.FETCH_UNUSED_TEMPLATE	205370705	1507955	10
3	DVORAKY.FETCH_UNUSED_TEMPLATE. __static_sql_exec_line13	203862750	203862750	**16**
2	DVORAKY.APPLY_REPLACEMENTS	626979540	8317650	10
3	DVORAKY.APPLY_REPLACEMENTS. __static_sql_exec_line9	618661890	618661890	73

Even Higher-Res?

Summary

Summary

Often we don’t know what we’re looking for

Diagnosis comes before Tuning

Summary: Real-Time

V$SQL_MONITOR

- Casts a wide, awesome, customizable, net
- Doesn’t detect client bugs (application waits)

V$ACTIVE_SESSION_HISTORY

- Shows where sessions are actually spending their time
- Detects application waits

Summary: Retrospective

V$SQL_MONITOR and V$ACTIVE_SESSION_HISTORY are retrospective too!

V$SESSION_EVENT/V$SESS_TIME_MODEL/V$SESSTAT

- Summary of session history by time
- Summary of session history by operation
- Memory leaks? Cursor Leaks?

DBMS_PROFILER / DBMS_HPROF

- Reveal hot spots in a call-stack
- Reveal anomalies line-by-line

Summary: Low-Level/Prospective

DBMS_SQLTUNE

- After homing-in on statement(s)
- Automatically tries to optimize the problem

DBMS_XPLAN/TRACE

- “Traditional” statement-by-statement analysis