

# SQL Monitoring in Oracle Database 12c

**Tanel Poder**

A long time computer performance geek

<http://blog.tanelpoder.com>

# Intro: About me



ORACLE  
ACE Director

ORACLE | CERTIFIED  
MASTER

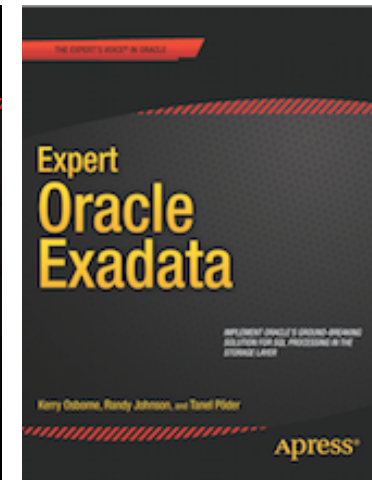
- Tanel Pöder

- Oracle Database Performance geek (18+ years)
- Exadata Performance geek
- Linux Performance geek
- Hadoop Performance geek



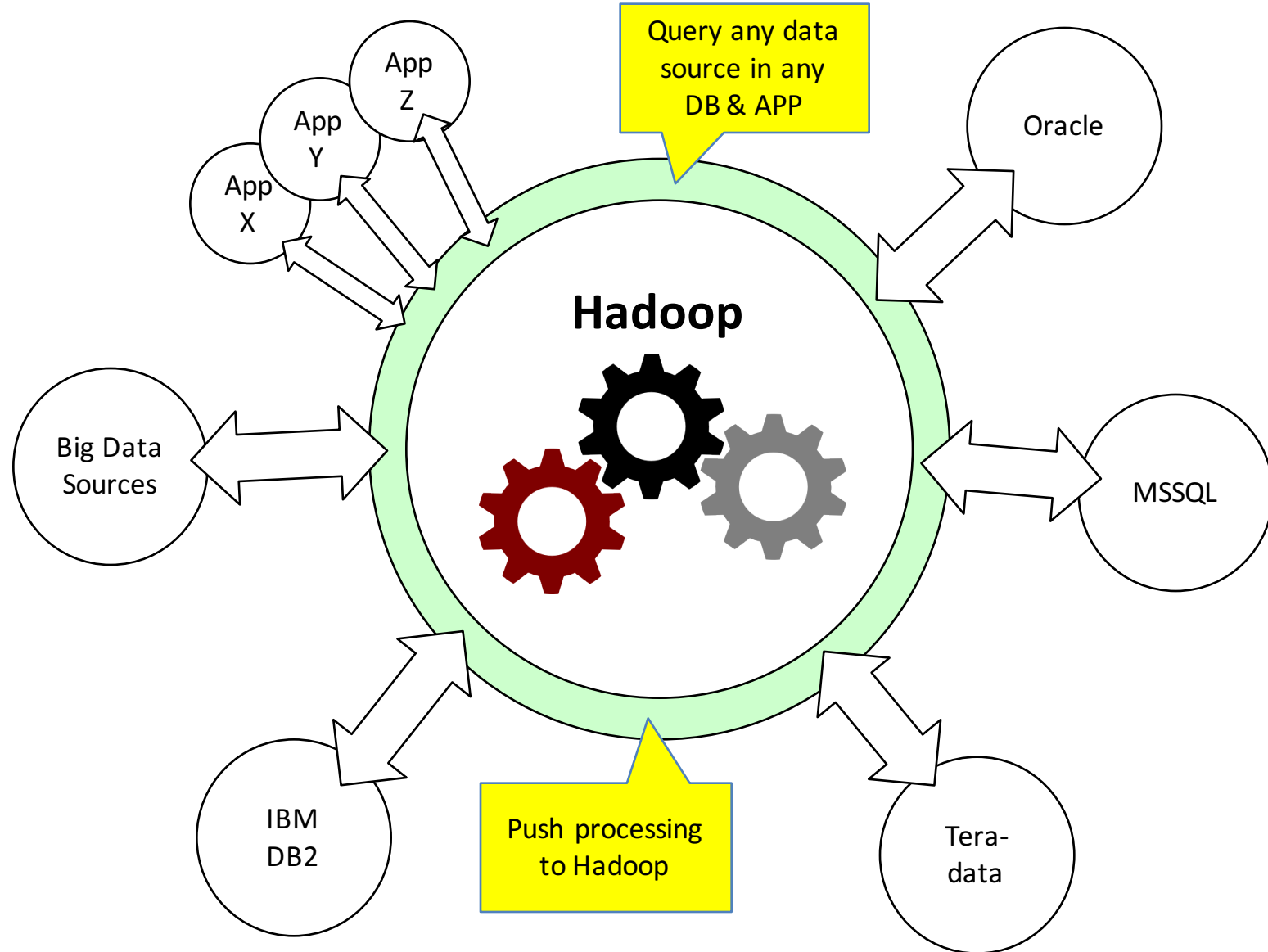
- CEO & co-founder:

gluent.



Expert Oracle Exadata  
book  
(2<sup>nd</sup> edition is out now!)

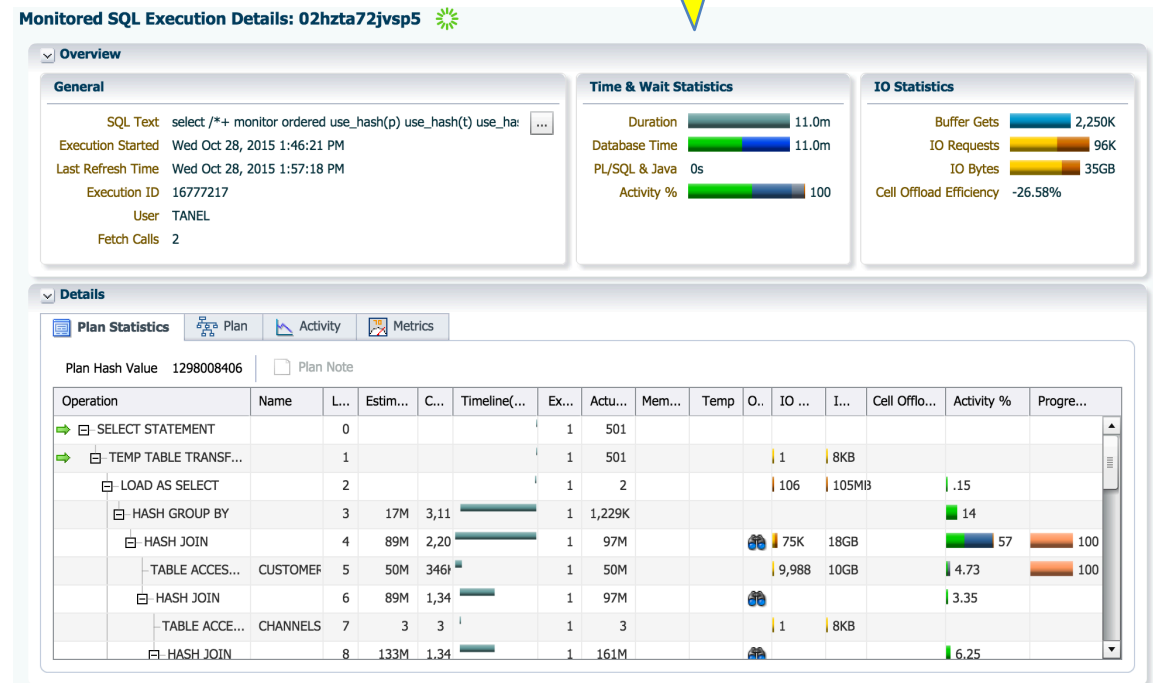
# Gluent: What we do



# Agenda

1. Getting SQL Monitoring Reports
2. Reading SQL Monitoring Reports
3. Oracle 12c-specific Improvements
4. Issues with SQL Monitoring
5. Under the Hood

Monitor long-running SQL statements while they are running (and after completion).



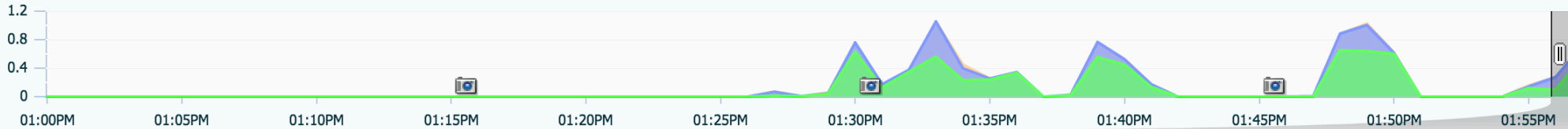
# Licensing

- SQL Monitoring requires both Diagnostics *and* Tuning Pack licenses
- Even when querying V\$SQL\_MONITOR or DBMS\_SQLTUNE manually

# Getting SQL Monitor Reports: Enterprise Manager

## Performance Hub: Real Time - Last Hour

Select Time Period Hide Time Picker PerfHub Report AWR Report Page Refreshed 2:00:18 PM GMT-0500 Auto Refresh 1 Mi

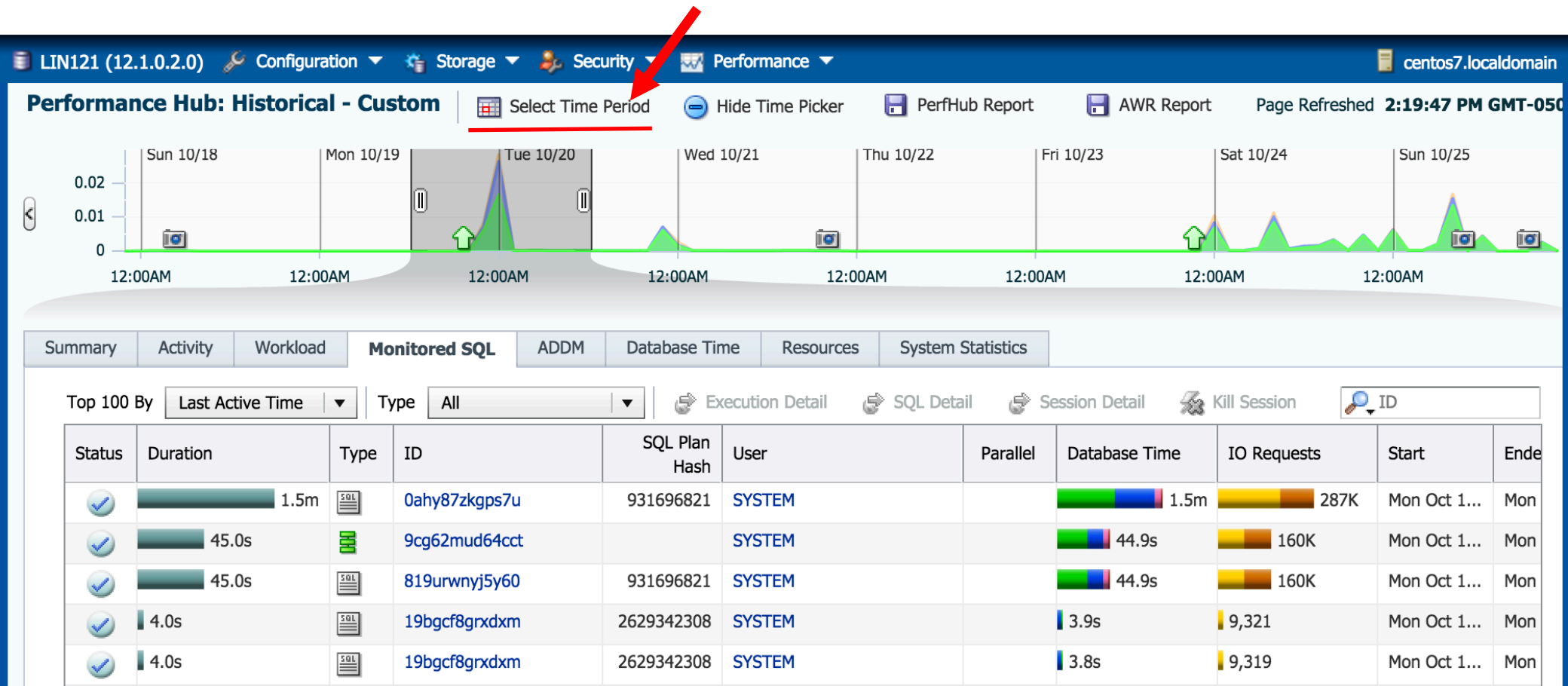


Summary Activity Workload **Monitored SQL** ADDM Current ADDM Findings

Top 100 By Last Active Time Type All Execution Detail SQL Detail Session Detail Kill Session ID

Status	Duration	Type	ID	SQL Plan Hash	User	Parallel	Database Time	IO Requests	Start	Ended	SQL Text
	6.0s		1mx9trdnmgt	3519235612	SYS		5.2s	3,117	2:00:12 PM		SELECT
	12.0s		1mx9trdntzngt	3519235612	SYS		12.0s	7,713	2:00:06 PM		SELECT
	11.0s		1mx9trdntzmg	3519235612	SYS		11.5s	10K	1:59:34 PM	1:59:45 PM	SELECT
	11.0s		1mx9trdntzmg	3519235612	SYS		11.6s	10K	1:59:20 PM	1:59:31 PM	SELECT
	42.0s		1p19kb988bbkc		SYS		43.7s	23K	1:57:11 PM	1:57:53 PM	BEGIN I
	42.0s		8d7wcfzbnkwt5	1315042658	SYS		42.9s	21K	1:57:11 PM	1:57:53 PM	INSERT
	20.0s		1p19kb988bbkc		SYS		20.4s	15K	1:56:09 PM	1:56:29 PM	BEGIN I
	19.0s		8d7wcfzbnkwt5	1315042658	SYS		19.3s	13K	1:56:09 PM	1:56:28 PM	INSERT
	12.0s		1p19kb988bbkc		SYS		12.2s	8,266	1:55:57 PM	1:56:09 PM	BEGIN I
	11.0s		8d7wcfzbnkwt5	1315042658	SYS		11.7s	7,000	1:55:57 PM	1:56:08 PM	INSERT

# Historical SQL Monitoring Reports in 12c



Stored as CLOBs in DBA\_HIST\_REPORTS -> DBA\_HIST\_REPORTS\_DETAILS

- <http://mauro-pagano.com/2015/05/04/historical-sql-monitor-reports-in-12c/>

# Getting SQL Monitor Reports: SQL Developer

The screenshot shows the Oracle SQL Developer interface. The 'Tools' menu is open, with a red arrow pointing to 'Monitor SQL...'. Below it, the 'Real Time SQL Monitoring' window is visible, displaying a table of monitoring data. A second red arrow points to the 'Show SQL Details' option in the context menu over the table.

**Tools Menu:**

- Data Modeler
- Database Copy...
- Database Diff...
- Database Export...
- Migration
- Monitor SQL...
- Monitor Sessions...
- SQL Worksheet (F10)
- Unit Test
- Data Miner
- OLAP

**Real Time SQL Monitoring Table:**

STATUS	DURATION	SQL_ID	SESSION_ID	SESSION_SERIAL	INSTANCE_DOP	CPU_TIME
DONE (FIRST N ROWS)		77 7faqui05471xq	138	35707	1 0	76.71
DONE (FIRST N ROWS)		1xq	140	26550	1 0	76.51
DONE (ALL ROWS)		mgt	370	10572	1 0	3.08
DONE (ALL ROWS)		mgt	6	63856	1 0	2.96
DONE (ALL ROWS)		mgt	6	63856	1 0	2.71
DONE (ALL ROWS)		mgt	6	63856	1 0	2.66
DONE		bkc	6	63856	1 0	32.26
DONE		42 8d7wcfzmkwt5	6	63856	1 0	32.12



# Getting SQL Monitor Reports: SQL Developer

**Overview**

SQL Id: **7faguj054z1xq**  
 Execution Started: **10/28/2015 14:05:31**  
 Last Refresh time: **10/28/2015 14:06:48**  
 Execution Id: **16777217**  
 Session: **138**  
 Fetch Calls: **1**  
 Run Status: **DONE (FIRST N ROWS)**

**User Information:**

User Name: **SYSTEM**  
 OS User: **tanel**  
 Process: **22610**  
 Machine: **mac01**  
 Program: **SQL Developer**  
 Module: **SQL Developer**  
 Client Info:

```
select /*+
monitor
no_parallel
ordered
use_hash(p)
use_hash(t)
use_hash(ch)
use_hash(c)
```

OPERATION	NAME	ESTIMATED_RO...	COST	TIMELINE	EXECUTIONS	ACTUAL_ROWS	MEMORY/MEMO...	TEMP/TEMP(MAX)	CPU
SELECT STATEMENT					1	50	0 / 0	0 / 0	
TEMP TABLE TRAN					1	50	0 / 0	0 / 0	
LOAD AS SELE				=====	1	2	0 / 0	0 / 0	18.42
HASH JOIN		355512	6561	=====	1	27071744	0 / 4797440	0 / 0	26.32
TABLE	CUSTOMERS	55500	405		1	55500	0 / 0	0 / 0	
HASH J		358664	4269	=====	1	27071744	0 / 1007616	0 / 0	3.95
TA	CHANNELS	3	3		1	3	0 / 0	0 / 0	
HA		537995	4265	=====	1	37443072	0 / 1321984	0 / 0	5.26
	PRODUCTS	60	3		1	60	0 / 0	0 / 0	
		651020	4260	=====	1	43617664	0 / 1303552	0 / 0	3.95
	:BF0000	365	17		1	366	0 / 0	0 / 0	
	TIMES	365	17		1	366	0 / 0	0 / 0	
		2923527	4236	=====	1	43617664	0 / 0	0 / 0	
	SALES	2923527	4236	=====	12	43617664	0 / 0	0 / 0	3.95
MULTI-TABLE					1	1	0 / 0	0 / 0	
HASH (GRC		354155	10226		1	1984	0 / 11923456	0 / 0	
VIEW	VW_DAG_1	354155	10226		1	218726	0 / 0	0 / 0	
SO		354155	10226	=====	1	218726	0 / 29851648	0 / 0	32.89
	SYS_TEMP_OF...	355512	1443	=====	1	27071744	0 / 0	0 / 0	5.26
DIRECT LO	SYS_TEMP_OF...				1984	1	0 / 0	0 / 0	

# Getting SQL Monitor Reports: Command Line

- **DBMS\_SQLTUNE.REPORT\_SQL\_MONITOR**

- Queries V\$SQL\_MONITOR views + ASH and returns a report as text / XML etc

```
SQL> @desc dbms_sqltune
```

Argument Name	Type	In/Out	Default?
SQL_ID	VARCHAR2		
SESSION_ID	NUMBER		DEFAULT
SESSION_SERIAL	NUMBER		
SQL_EXEC_START	DATE		
SQL_EXEC_ID	NUMBER		
INST_ID	NUMBER		
START_TIME_FILTER	DATE	IN	DEFAULT
END_TIME_FILTER	DATE	IN	DEFAULT
INSTANCE_ID_FILTER	NUMBER	IN	DEFAULT
PARALLEL_FILTER	VARCHAR2	IN	DEFAULT
PLAN_LINE_FILTER	NUMBER	IN	DEFAULT
EVENT_DETAIL	VARCHAR2	IN	DEFAULT
BUCKET_MAX_COUNT	NUMBER	IN	DEFAULT
BUCKET_INTERVAL	NUMBER	IN	DEFAULT
BASE_PATH	VARCHAR2	IN	DEFAULT
LAST_REFRESH_TIME	DATE	IN	DEFAULT
REPORT_LEVEL	VARCHAR2	IN	DEFAULT
TYPE	VARCHAR2	IN	DEFAULT

Search for a specific monitored SQL\_ID

... or a session's (last) monitored SQL

Search for a specific monitored SQL\_ID

# Getting SQL Monitor Reports: Command Line

- **DBMS\_SQLTUNE.REPORT\_SQL\_MONITOR**

- Queries V\$SQL\_MONITOR views + ASH and returns a report as text / XML etc

```
SELECT
  DBMS_SQLTUNE.REPORT_SQL_MONITOR(
    session_id=> 1234,
    report_level=>'ALL',
    type => 'TEXT') as report
FROM dual
/
```

*type* can be TEXT,  
HTML, XML, ACTIVE

You can spool HTML  
into a file and open in  
browser

Tanel's Scripts (<http://blog.tanelpoder.com/files/>)

```
@xp.sql - explain with "profile" in TEXT mode
@xph.sql - explain with "profile" in HTML mode
@xpa.sql - explain with "profile" in ACTIVE mode (flash)
```

# Getting SQL Monitor Reports: Command Line

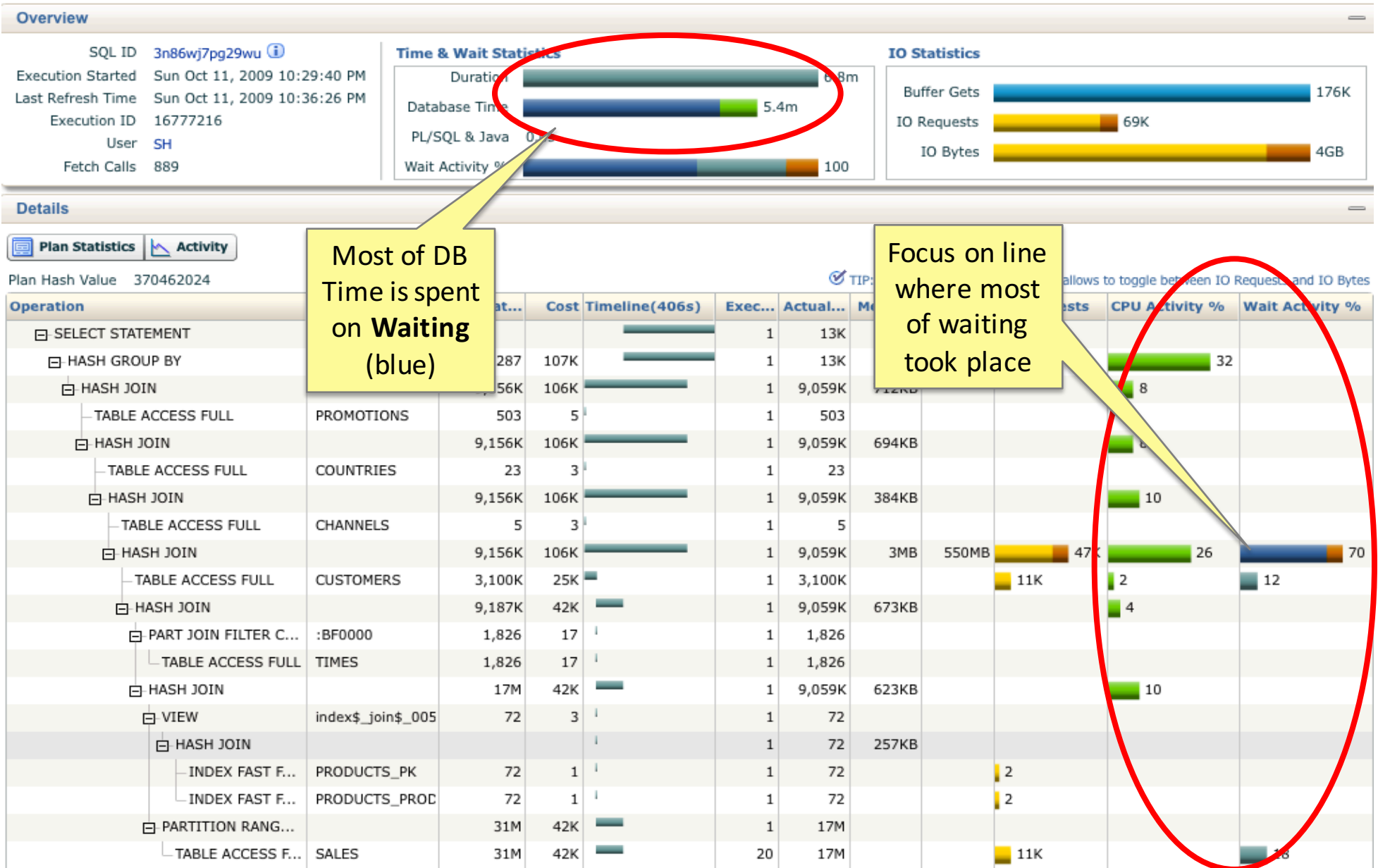
- **Sqlmon.sql (demo of a new script)**
  - Augment a SQL Monitoring report with additional metrics
  - Save all SQLMon reports into separate files (for search & history)
  - <http://blog.tanelpoder.com/files/scripts/sqlmon.sql>

```
tanel@mac01:~$ ls -lt sqlmon_*
-rw----- 1 tanel  staff  14620 Oct 28 11:38 sqlmon_8d7wcfzbnkwt5_16777217_151028_133803.html
-rw----- 1 tanel  staff   76029 Oct 14 02:53 sqlmon_ad18fzq7910zc_16777216_151014_045320.html
-rw----- 1 tanel  staff   29728 Oct 14 02:45 sqlmon_fuq5qhagj6wbb_16777219_151014_044552.html
-rw----- 1 tanel  staff   23900 Oct 14 02:41 sqlmon_fhqvstdhvf4kq_16777216_151014_044106.html
-rw----- 1 tanel  staff   30409 Oct 14 02:15 sqlmon_fuq5qhagj6wbb_16777218_151014_041507.html
-rw----- 1 tanel  staff   29714 Oct 13 17:49 sqlmon_fuq5qhagj6wbb_16777217_151013_194856.html
-rw----- 1 tanel  staff   23949 Oct 13 17:46 sqlmon_fhqvstdhvf4kq_16777216_151013_194532.html
-rw----- 1 tanel  staff   75964 Oct 13 17:41 sqlmon_2zfv7dzqf9azz_16777216_151013_194132.html
-rw----- 1 tanel  staff   80717 Oct 13 17:33 sqlmon_01c4gdrcmgzwb_16777216_151013_193249.html
-rw----- 1 tanel  staff   74601 Oct 13 17:29 sqlmon_2xfr5d9w4rg2s_16777216_151013_192938.html
-rw----- 1 tanel  staff   40975 Oct 13 17:27 sqlmon_7ua9dvrw9863m_16777216_151013_192737.html
-rw----- 1 tanel  staff   41272 Oct 13 17:26 sqlmon_5p3h1xt70hrvj_16777216_151013_192639.html
...
```

# When Does SQL Monitoring Kick in?

- 1. Manually** and **immediately** when adding a MONITOR hint to SQL
  - 2. Automatically** and **immediately** for all parallel statements
    - As PX statements are expected to be relatively long running ones
  - 3. Automatically** and **after 5 seconds** of CPU/IO time spent for serial queries
    - Undocumented parameter: `_sqlmon_threshold = 5`
    - *“CPU/IO time threshold before a statement is monitored. 0 is disabled”*
- Why 5 seconds of CPU/IO time not full DB Time or Duration?
    - This is to avoid suddenly monitoring all statements that are blocked by some lock or waiting for some (queue) event to happen

# SQL Monitoring: Example



# Reading a SQL Monitoring report – in 3 steps

## 1. Database Time breakdown

- The total time actively consumed by *this* execution of the query
- Activity time of QC + all PX slaves if parallel execution
- **Q: Mostly on CPU or mostly waiting?**

## 2. Activity % breakdown

- If DB Time was spent mostly on CPU then look into **CPU Activity %** column
- If DB Time was spent mostly on waiting then **Wait Activity %** column
- This tells you in which execution plan line(s) most of the time was spent

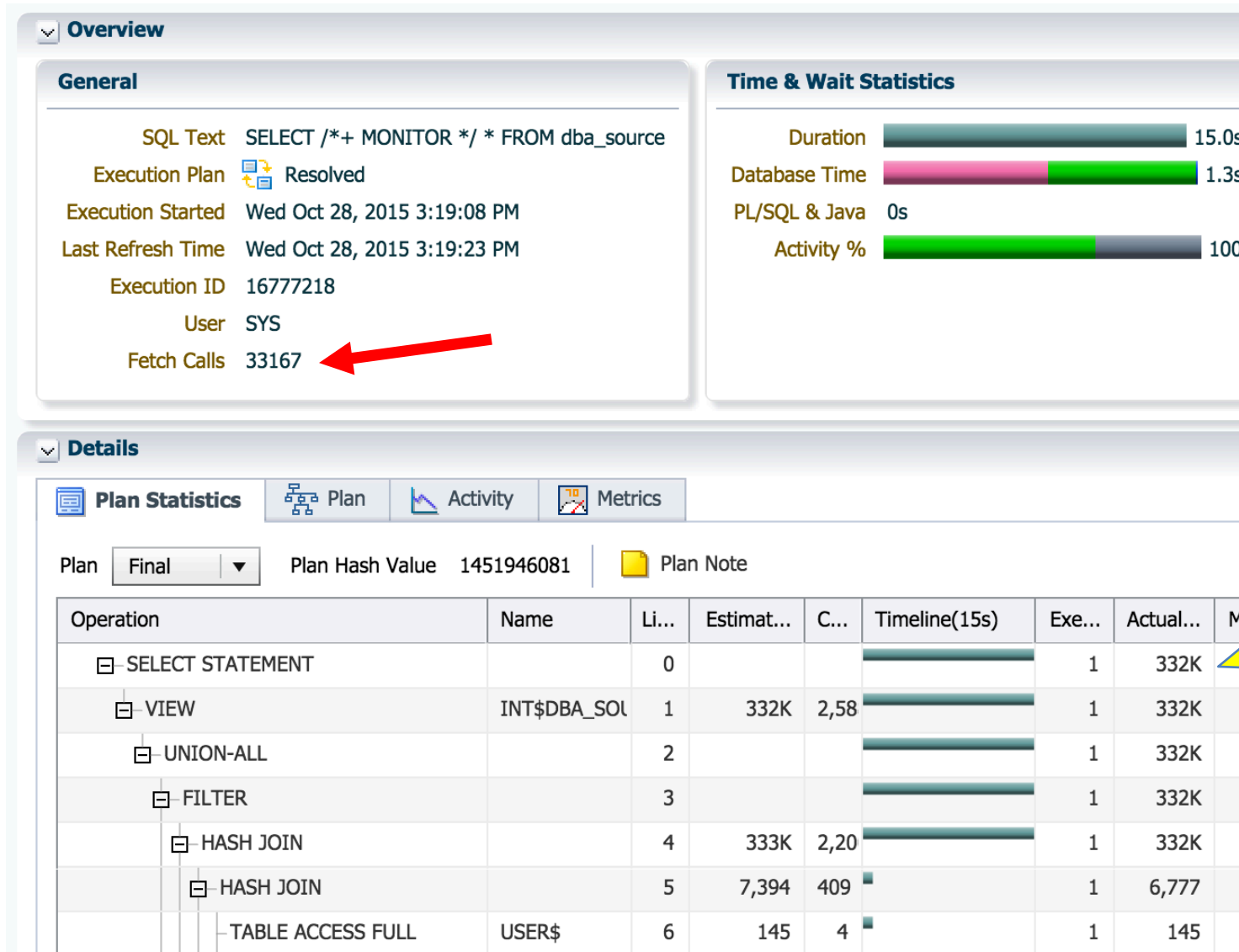
## 1. Executions, IO Bytes/Requests, Actual Rows returned metrics tell you how much work the row sources had to do

# SQL Monitoring – Duration vs DB Time?

- **Duration** is the amount of wall-clock time from the *execution start*, up to *closing the cursor* (or reaching end of data)
  - If you open a cursor and don't fetch for a while, the Duration still keeps increasing
- **DB Time** is the amount of database time (activity inside the DB) your session (and all its PX slaves, if any) spent executing this SQL
  - So if you run an 1-hour DML statement with 8 parallel slaves, you may see DB Time up to 9hours (up to 1h for the QC, up to 8h for all PX slaves)



# DB Time much smaller than Duration?!



**Duration = 15 seconds**  
(wall-clock time)

**DB Time = 1.3 seconds**  
(time spent inside DB)

332 000 rows were returned, fetched 10 rows at a time (33167 fetches). **Most time spent on network roundtrips outside DB**

# DB Time much bigger than Duration?

**General**

SQL Text `SELECT /*+ PARALLEL(4) */ COUNT(distinct PROD_I` ...

Execution Plan 4

Execution Started Wed Oct 28, 2015 3:28:48 PM

Last Refresh Time Wed Oct 28, 2015 3:28:57 PM

Execution ID 16777216

User SYS

Fetch Calls 1

**Time & Wait Statistics**

Duration 9.0s

Database Time 35.9s

PL/SQL & Java 0s

Activity % 100

**Duration = 9 seconds**  
(wall-clock time)

**DB Time = ~36**  
seconds  
(time spent inside DB)

Plan Statistics | Plan | **Parallel** | Activity | Metrics

Parallel Server	Database Time	Activity %	IO Requests
[-] Instance 1			
Parallel Coordinator	22.3ms		
[-] Parallel Set 1			
Buffer Gets er 1 (p000)	8.4ms		
Parallel Server 2 (p001)	6.0ms		
Parallel Server 3 (p002)	7.2ms		
Parallel Server 4 (p003)	5.3ms		
[-] Parallel Set 2			
Parallel Server 1 (p004)	9.0s	25	2,408
Parallel Server 2 (p005)	9.0s	25	2,411
Parallel Server 3 (p006)	8.8s	25	2,377
Parallel Server 4 (p007)	9.1s	25	2,409

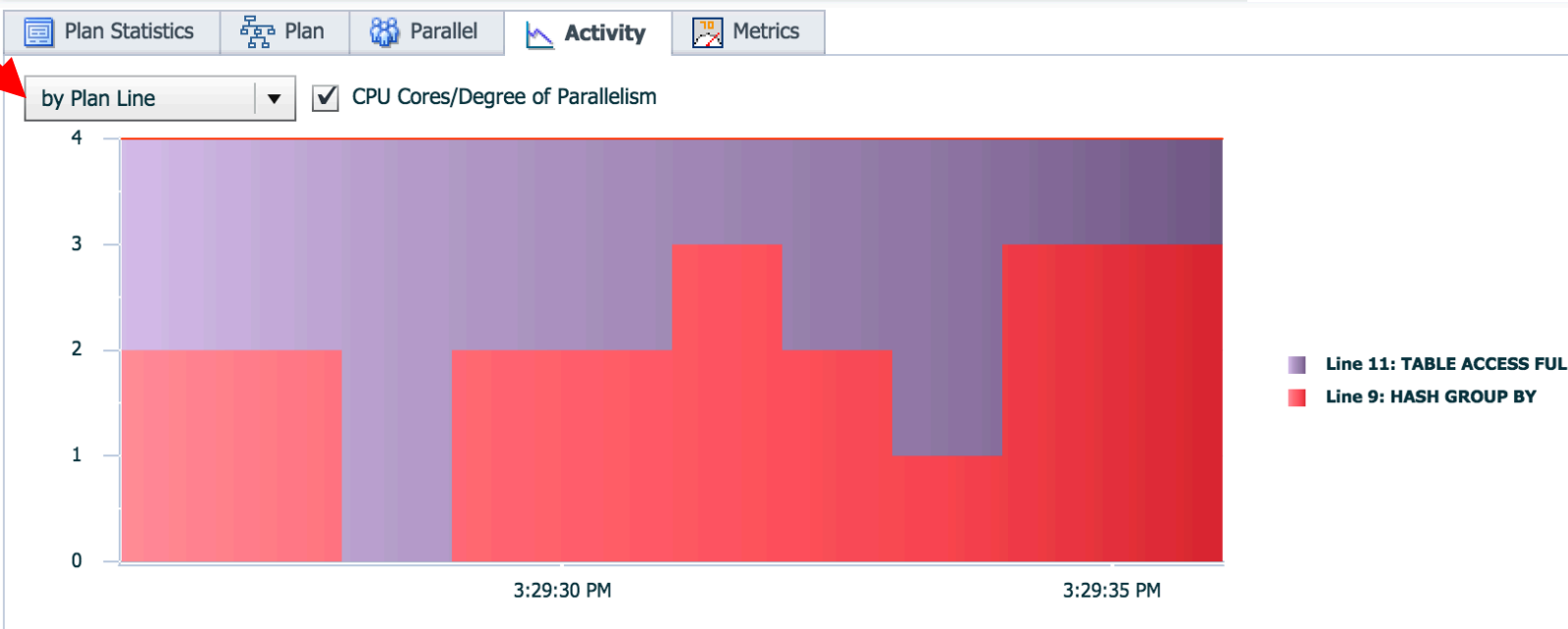
**Parallel Execution!**

Each slave can use up to 1 second of DB Time per wall-clock second (plus QC)

# DB Time much bigger than Duration?



Break down this SQL execution activity (across all PX slaves) by wait event or plan line



# SQL Monitoring Improvements in Oracle 12c

- Execution **plan line** level additional metrics in the ***Other*** column!!!

Plan Statistics Plan Parallel Activity

Plan Hash Value 316909058

Operation	Name	Li...	Estimat...	C...	Timeline(1s)	Exe...	Actual...	Memo...	Temp ...	Other
SELECT STATEMENT		0				5	2			
PX COORDINATOR		1				5	2			
PX SEND QC (RANDOM)	:TQ10001	2	2	14K		2	2			
HASH JOIN		3	2	14K		2	2	482KB		
JOIN FILTER CREATE	:BF0000	4	1	7,30		2	2			
PX RECEIVE		5	1	7,30		2	2			
PX SEND BROADCAST	:TQ10000	6	1	7,30		2	2			
PX BLOCK ITERATOR		7	1	7,30		2	1			
TABLE ACCESS STORA...	CUSTOMERS2	8	1	7,30		27	1	8MB		
JOIN FILTER USE	:BF0000	9	11M	6,82		2	19K			
PX BLOCK ITERATOR		10	11M	6,82		2	19K			
TABLE ACCESS STORAGE ...	ORDERS2	11	11M	6,82		27	19K	8MB		

# SQL Monitoring Improvements in Oracle 12c

**Other Plan Line Statistics**

Build Size	120MB
Build Row Count	1
Fan-out	8
Slot Size	123K
Total Build Partitions	16
Total Cached Partitions	16

HASH JOIN  
row source

**Other Plan Line Statistics**

Bitmap Size	8,192
Folded Bitmap Size	8,192
Bits Set	1
Total Creator Rows	1

OK

JOIN FILTER  
CREATE  
row source

Only one row was  
used for building  
the filter, resulting  
in one bit set

# SQL Monitoring Improvements in Oracle 12c

**Other Plan Line Statistics**

Slow metadata bytes	3,064
Eligible bytes	760M
Filtered bytes	115K
Flash cache bytes	760M

OK

Only 115kB was returned out of 760MB of smart IO issued. All IO from Flash Cache

TABLE ACCESS FULL on *customers*

**Other Plan Line Statistics**

Slow metadata bytes	11K
Eligible bytes	709M
Filtered bytes	1,113K
SI saved bytes	424M
Flash cache bytes	285M

OK

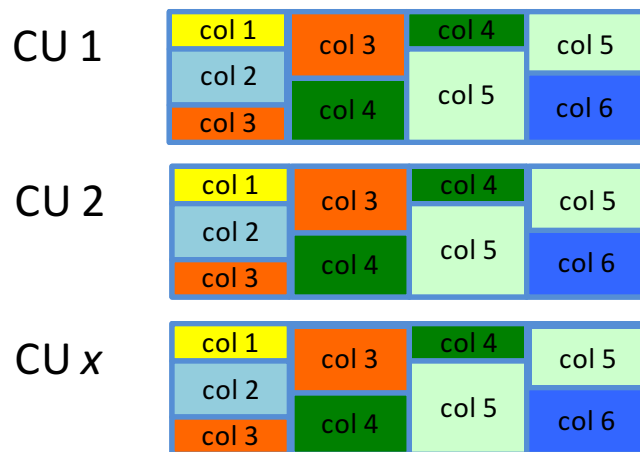
TABLE ACCESS FULL on *orders*

Storage Indexes helped to *skip* 424 MB of IOs

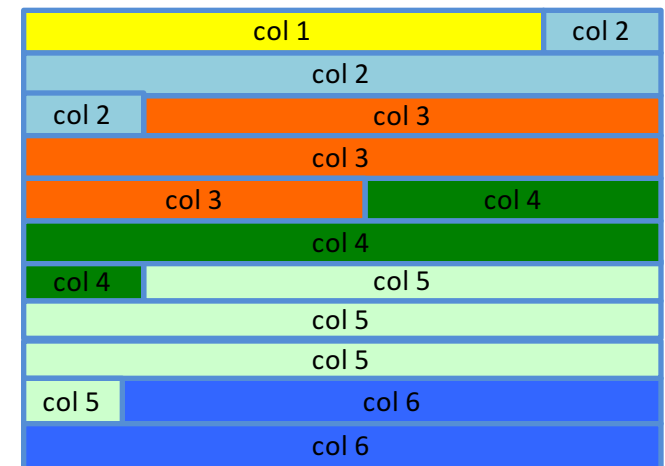
# Example: Exadata Columnar Flash Cache

- Cellsrv **12.1.2.1.0** (January 2015)
  - Independent dual format caching in storage cell flash
  - Smart Scans use Columnar Flash Cache (if all required columns are cached)
  - Block IO uses traditional Flash Cache
  - Conceptual overview traditional vs. columnar Flash Cache

compact data – less flash IO for a column scan



CU 1 .. x



# Columnar Flash Cache metrics – Session level

```
SQL> @snapper all 5 1 1234
```

TYPE, STATISTIC		DELTA,	HDELTA/SEC
STAT, physical read IO requests	,	6416,	1.26k
STAT, <b>physical read bytes</b>	,	<b>6717366272,</b>	<b>1.32G</b>
STAT, cell physical IO bytes eligible for predicate offload	,	6717366272,	1.32G
STAT, cell physical IO interconnect bytes returned by smart scan,		445177024,	87.68M
STAT, <b>cell physical IO bytes saved by columnar cache</b>	,	<b>5128519680,</b>	<b>1.01G</b>
STAT, cell blocks processed by cache layer	,	250366,	49.31k
STAT, cell blocks processed by txn layer	,	250366,	49.31k
STAT, cell blocks processed by data layer	,	50327,	9.91k
STAT, cell flash cache read hits	,	6416,	1.26k
WAIT, enq: KO - fast object checkpoint	,	747,	147.12us
WAIT, cell smart table scan	,	315077,	62.05ms
WAIT, SQL*Net message to client	,	8,	1.58us
WAIT, SQL*Net message from client	,	2668093,	525.48ms
WAIT, events in waitclass Other	,	1136,	223.73us

```
Stats snap 1, end=2015-02-11 10:57:34, seconds=5.1
```



# Columnar Flash Cache metrics – Plan Line Level

**General**

SQL Text: SELECT /\*+ NO\_F

Execution Started: Tue Feb 10, 2015

Last Refresh Time: Tue Feb 10, 2015

Execution ID: 16777220

User: TANEL

Fetch Calls: 1

**Other Plan Line Statistics**

Eligible bytes: 6,717M

Filtered bytes: 108M

Flash cache bytes: 6,717M

Columnar cache saved bytes: 5,123M

OK

**Statistics**

Buffer Gets: 820K

IO Requests: 6,416

IO Bytes: 6GB

Offload Efficiency: 99%

**Details**

Plan Hash Value: 1201118828 | Plan Note

Operation	Name	L...	Estim...	C..	Timeline(1s)	Ex...	Actu...	Me...	Tem...	O	IO ...	I...	Cell Offlo...	Activity %
SELECT STATEMENT		0				1	1							
SORT AGGREGATE		1	1			1	1							
TABLE ACCESS STORAGE ...	CUSTOMER	2	3,482K	11k		1	15M	3MB			6,416	6GB	99	

# Full list of SQL Monitoring stats: v\$sql\_monitor\_statname

```
SQL> @sqlmon_sn %
```

NAME	DESCRIPTION
Build Size	Size of the build input in bytes
Build Row Count	Number of rows for the build
Fan-out	Number of partitions used to split both inputs
Slot Size	Size of an in-memory hash-join slot
Total Build Partitions	Total number of build partitions
Total Cached Partitions	Total number of build partitions left in-memory before probing
Multi-pass Partition Pairs	Total number of partition pairs processed multi-pass
Total Spilled Probe Rows	Total number of rows from the probe spilled to disk (excluding buffering)
Bitmap Size	Size of the bitmap in bytes
...	
Folded Bitmap Size	Folded size of the bitmap in bytes
Bits Set	Number of bits set in the bitmap
Total Creator Rows	Total number of rows from the creator side
Total user Rows	Total number of rows from the user side
Total Filtered Rows	Total number of rows filtered by the bloom filter
Times hash func called	Number of times hash func is called
Slow metadata bytes	Size of slow metadata in bytes
Eligible bytes	Total bytes eligible for offload
Filtered bytes	Total bytes returned after offload
SI saved bytes	Total bytes saved by storage index
Flash cache bytes	Total bytes fetched from flash cache
Partial flash cache and disk bytes	Total bytes where IO was partially fetched from flash cache and disk
Cell passthru IO bytes	Total cell passthru IO bytes
Block IO bytes	Total block IO bytes
Slow metadata bytes	Size of slow metadata in bytes

This is just a glance, the script produces more output of available plan line level metrics

# Getting Bind Variable Values of a long-running SQL

Overview

General

SQL Text: SELECT /\*+ MONITOR \*/ COUNT(\*) FROM dba\_source

Execution Plan: Resolved

Execution Started: Wed Oct 28, 2015 3:55:03 PM

Last Refresh Time: Wed

Execution ID: 16777

User: SYS

Fetch Calls: 1

Time & Wait Statistics

Duration: 0.5s

Database Time: 0.5s

PL/SQL & Java: 0s

SQL Text

SELECT /\*+ MONITOR \*/ COUNT(\*) FROM dba\_source

WHERE line < :x AND text != :y

Show SQL Binds Save OK

SQL Binds

Name

Position ▲	Name	Value	Type
1	:X	123456	NUMBER
2	:Y	TANEL TESTING	VARCHAR2(2000)

Save OK

# Getting Bind Variable Values of a long-running SQL

```
SQL> @xp &mysid
eXplain with Profile: Running DBMS_SQLTUNE.REPORT_SQL_MONITOR for SID 6...

SQL Text
-----
SELECT /*+ MONITOR */ COUNT(*) FROM dba_source WHERE line < :x AND text != :y
```

## Global Information

```
-----
Status                : DONE (ALL ROWS)
Instance ID           : 1
Session               : SYS (6:45275)
SQL ID                : 1ujxpp90w537r
SQL Execution ID      : 16777216
Execution Started     : 10/28/2015 15:55:03
First Refresh Time    : 10/28/2015 15:55:03
Last Refresh Time     : 10/28/2015 15:55:03
Duration              : .511703s
Module/Action         : SQL*Plus/-
Program               : sqlplus@mac01 (TNS V1-V3)
Fetch Calls           : 1
```

## Binds

Name	Position	Type	Value
:X	1	NUMBER	123456
:Y	2	VARCHAR2(2000)	TANEL TESTING

You can also query  
**V\$SQL\_MONITOR.BINDS\_XML**  
directly

<http://blog.tanelpoder.com/2010/10/18/read-currently-running-sql-statements-bind-variable-values/>

# What about monitoring every single statement?

- What about adding a MONITOR hint to all my OLTP statements?
  - **BAD IDEA!**
- SQL Monitoring needs to copy runtime metrics to SGA
  - ... **separate structure for every monitored execution!**
  - V\$SQL\_MONITOR view data lives in Shared Pool
  - Latch contention (Real-time plan statistics latch in 11g, gone in 12c)
  - Shared Pool Memory usage
  - Extra CPU usage
- ASH data to the rescue!

# ASH plan line level breakdown in Oracle 11.1+

```
SQL> @desc v$active_session_history
Name                                         Null?    Type
-----
1      SAMPLE_ID                                 NUMBER
2      SAMPLE_TIME                              TIMESTAMP(3)
3      IS_AWR_SAMPLE                             VARCHAR2(1)
4      SESSION_ID                               NUMBER
5      SESSION_SERIAL#                          NUMBER
6      SESSION_TYPE                             VARCHAR2(10)
7      FLAGS                                     NUMBER
8      USER_ID                                  NUMBER
9      SQL_ID                                    VARCHAR2(13)
10     IS_SQLID_CURRENT                          VARCHAR2(1)
11     SQL_CHILD_NUMBER                         NUMBER
12     SQL_OPCODE                              NUMBER
13     SQL_OPNAME                               VARCHAR2(64)
14     FORCE_MATCHING_SIGNATURE                 NUMBER
15     TOP_LEVEL_SQL_ID                       VARCHAR2(13)
16     TOP_LEVEL_SQL_OPCODE                   NUMBER
17     SQL_ADAPTIVE_PLAN_RESOLVED              NUMBER
18     SQL_FULL_PLAN_HASH_VALUE                NUMBER
19     SQL_PLAN_HASH_VALUE                     NUMBER
20     SQL_PLAN_LINE_ID                       NUMBER
21     SQL_PLAN_OPERATION                      VARCHAR2(30)
22     SQL_PLAN_OPTIONS                        VARCHAR2(30)
23     SQL_EXEC_ID                             NUMBER
...

```

Drill down into SQL  
Execution Plan **LINE**  
level!

# ASH-based “SQL monitoring” report: asqlmon.sql

- Deliberately wide output – highlight a row and scroll left/right:

```
SQL> @ash/asqlmon 8dq0v1mjngj7t 1
```

SEC	Activity	Visual	ID	OPERATION	STATE	EVENT
6	.3 %		0	SELECT STATEMENT	ON CPU	
	%		1	COUNT STOPKEY		
1257	<b>58.4 %</b>	<b> #####</b>	2	<b>TABLE ACCESS BY INDEX ROWID [CUSTOMERS]</b>	<b>WAITING</b>	<b>db file sequential read</b>
2	.1 %		2		ON CPU	
11	.5 %		2		WAITING	db file scattered read
2	.1 %		3	INDEX UNIQUE SCAN [CUSTOMERS_PK]	ON CPU	
28	1.3 %		3		WAITING	db file scattered read
845	<b>39.3 %</b>	<b> ####</b>	3		<b>WAITING</b>	<b>db file sequential read</b>

OBJ_ALIAS_QBC_NAME	ASQLMON_PREDICATES	PROJECTION
[SEL\$1]	[F:]ROWNUM<:B1	"CUSTOMER_ID"[NUMBER,22], "CUST_FIRST_NAME"[VARCHAR2,30],
CUSTOMERS@SEL\$1 [SEL\$1]		"CUSTOMER_ID"[NUMBER,22], "CUST_FIRST_NAME"[VARCHAR2,30],
CUSTOMERS@SEL\$1 [SEL\$1]		"CUSTOMER_ID"[NUMBER,22], "CUST_FIRST_NAME"[VARCHAR2,30],
CUSTOMERS@SEL\$1 [SEL\$1]		"CUSTOMER_ID"[NUMBER,22], "CUST_FIRST_NAME"[VARCHAR2,30],
CUSTOMERS@SEL\$1 [SEL\$1]	[A:] "CUSTOMER_ID"=:B2	"CUSTOMERS".ROWID[ROWID,10], "CUSTOMER_ID"[NUMBER,22]
CUSTOMERS@SEL\$1 [SEL\$1]	[A:] "CUSTOMER_ID"=:B2	"CUSTOMERS".ROWID[ROWID,10], "CUSTOMER_ID"[NUMBER,22]

# More ASH Geekery ☺

- Understand which rowsource types consume the most time in your DB!

```
SQL> @ashtop sql_opname,sql_plan_operation,sql_plan_options session_type='FOREGROUND' \
        SYSDATE-1/24 SYSDATE
```

Total Seconds	AAS	%This	SQL_OPNAME	SQL_PLAN_OPERATION	SQL_PLAN_OPTIONS
374	.1	31%	SELECT	TABLE ACCESS	FULL
225	.1	18%	SELECT	OPTIMIZER STATISTICS GATHERING	
164	.0	13%	PL/SQL EXECUTE		
75	.0	6%	SELECT	SORT	AGGREGATE
75	.0	6%	SELECT	TABLE ACCESS	SAMPLE
56	.0	5%	SELECT		
37	.0	3%	SELECT	HASH	GROUP BY
37	.0	3%	SELECT	INDEX	FULL SCAN
25	.0	2%	SELECT	SELECT STATEMENT	
20	.0	2%	INSERT		
19	.0	2%	INSERT	LOAD TABLE CONVENTIONAL	
17	.0	1%			
12	.0	1%	SELECT	SORT	GROUP BY
12	.0	1%	SELECT	TABLE ACCESS	BY USER ROWID
8	.0	1%	SELECT	INDEX	RANGE SCAN
7	.0	1%	DELETE	DELETE	
7	.0	1%	INSERT	TABLE ACCESS	SAMPLE
6	.0	0%	SELECT	TABLE ACCESS	CLUSTER
5	.0	0%	DELETE	DELETE STATEMENT	
5	.0	0%	SELECT	INDEX	FAST FULL SCAN



# Issues

- SQL Monitoring reports not persisted (11g)
  - Fixed in 12c by regularly saving reports for longest running SQL
  - Stored in DBA\_HIST\_REPORTS / DBA\_HIST\_REPORTS\_DETAILS
  - An 11g Workaround is to write your own PL/SQL proc for persisting V\$ data
- SQL Monitoring doesn't monitor very large plans
  - Supposedly to save memory
  - `_sqlmon_max_planlines` = 300 by default
  - Plans with more than 300 lines not monitored!
  - See MOS Note:
    - **How to Monitor SQL Statements with Large Plans Using Real-Time SQL Monitoring? (Doc ID 1613163.1)**

**Thank You!**

Questions?

[tanel@tanelpoder.com](mailto:tanel@tanelpoder.com)

[blog.tanelpoder.com](http://blog.tanelpoder.com)

@tanelpoder

We are hiring! 😊

<http://gluent.com>