SQL Monitoring in Oracle Database 12c

Tanel Poder

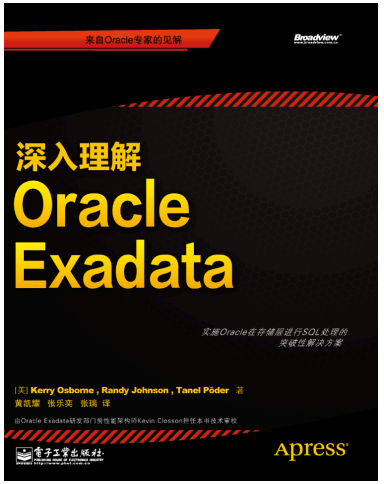
A long time computer performance geek

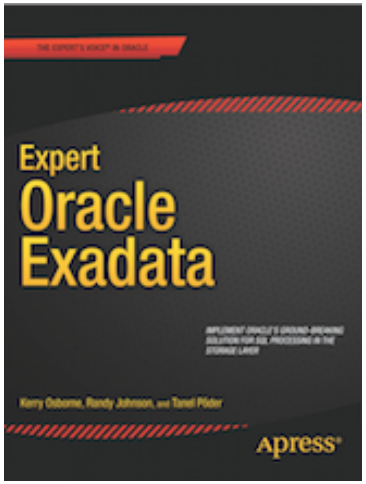
http://blog.tanelpoder.com

Intro: About me

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

Instant promotion



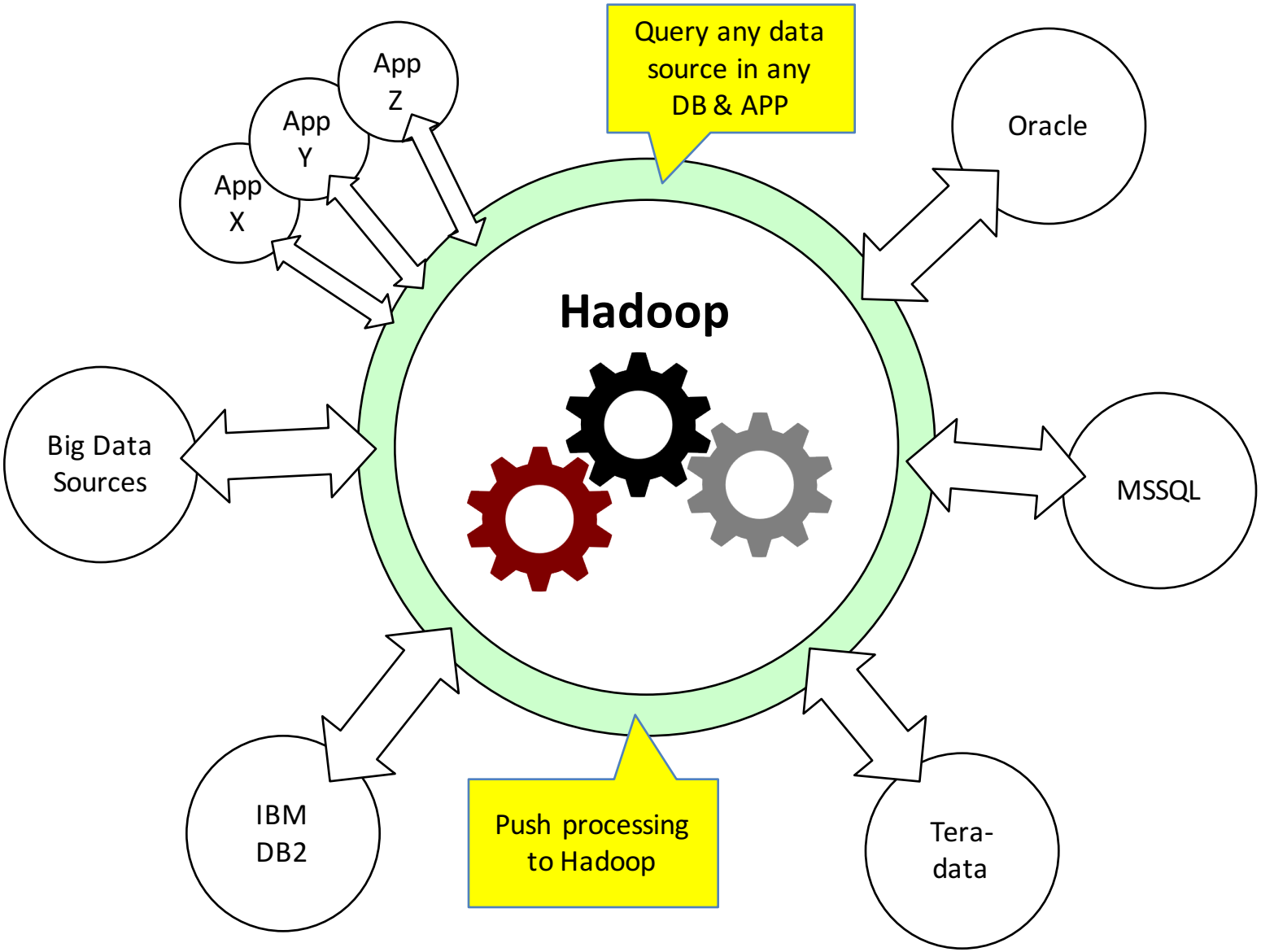


- CEO & co-founder:

gluent.

Expert Oracle Exadata book
(2nd edition is out now!)

Gluent: What we do



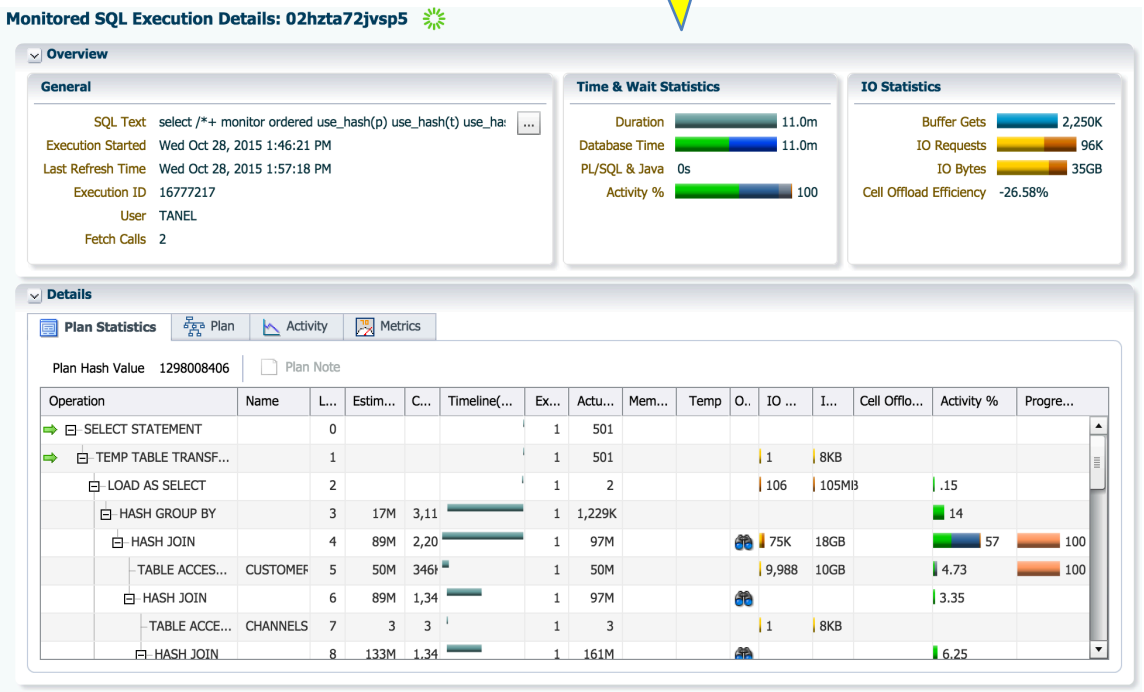
blog.tanelpoder.com

gluent.

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).



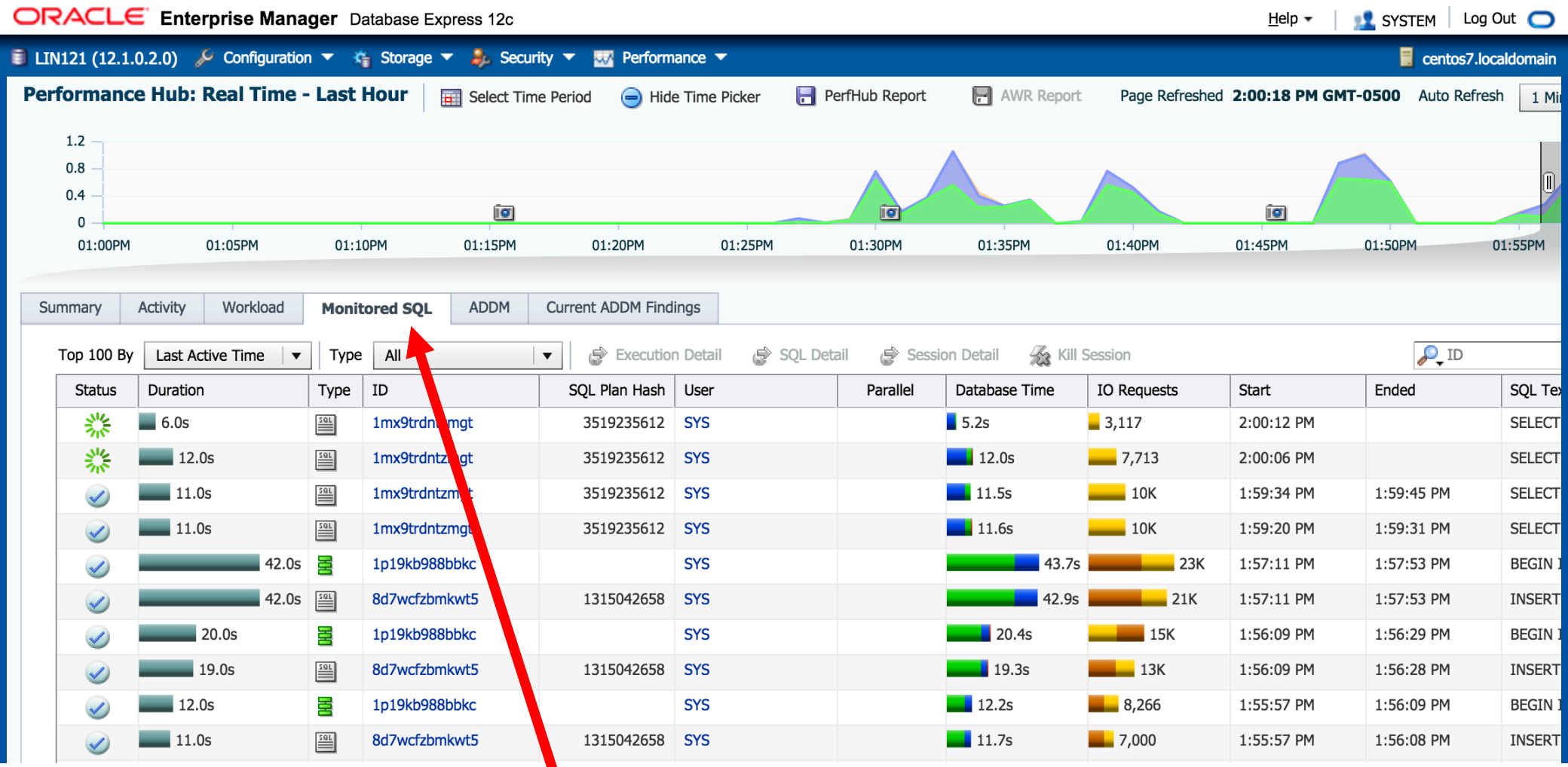
blog.tanelpoder.com

gluent.

Licensing

- SQL Monitoring requires both Diagnostics *and* Tuning Pack licenses
- Even when querying V$SQL_MONITOR or DBMS_SQLTUNE manually

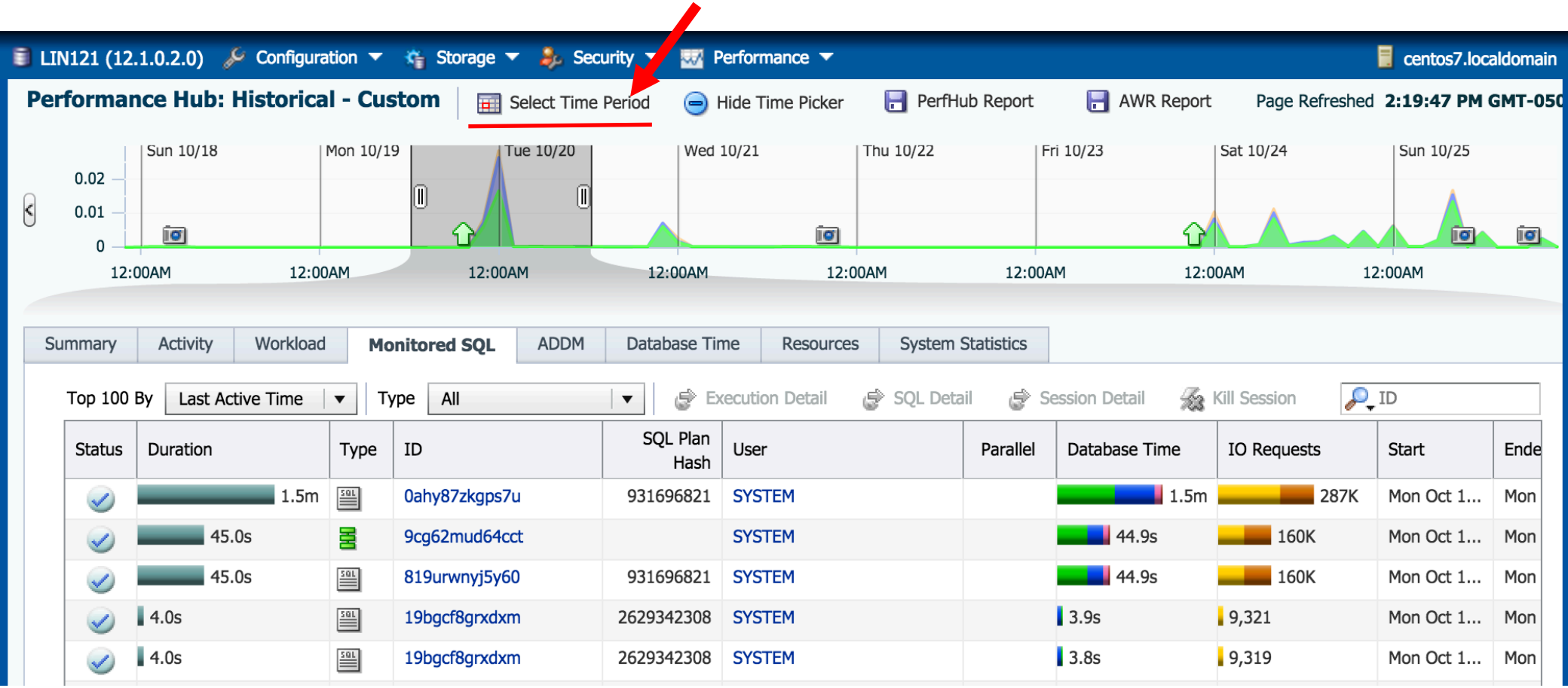
Getting SQL Monitor Reports: Enterprise Manager



blog.tanelpoder.com

gluent.

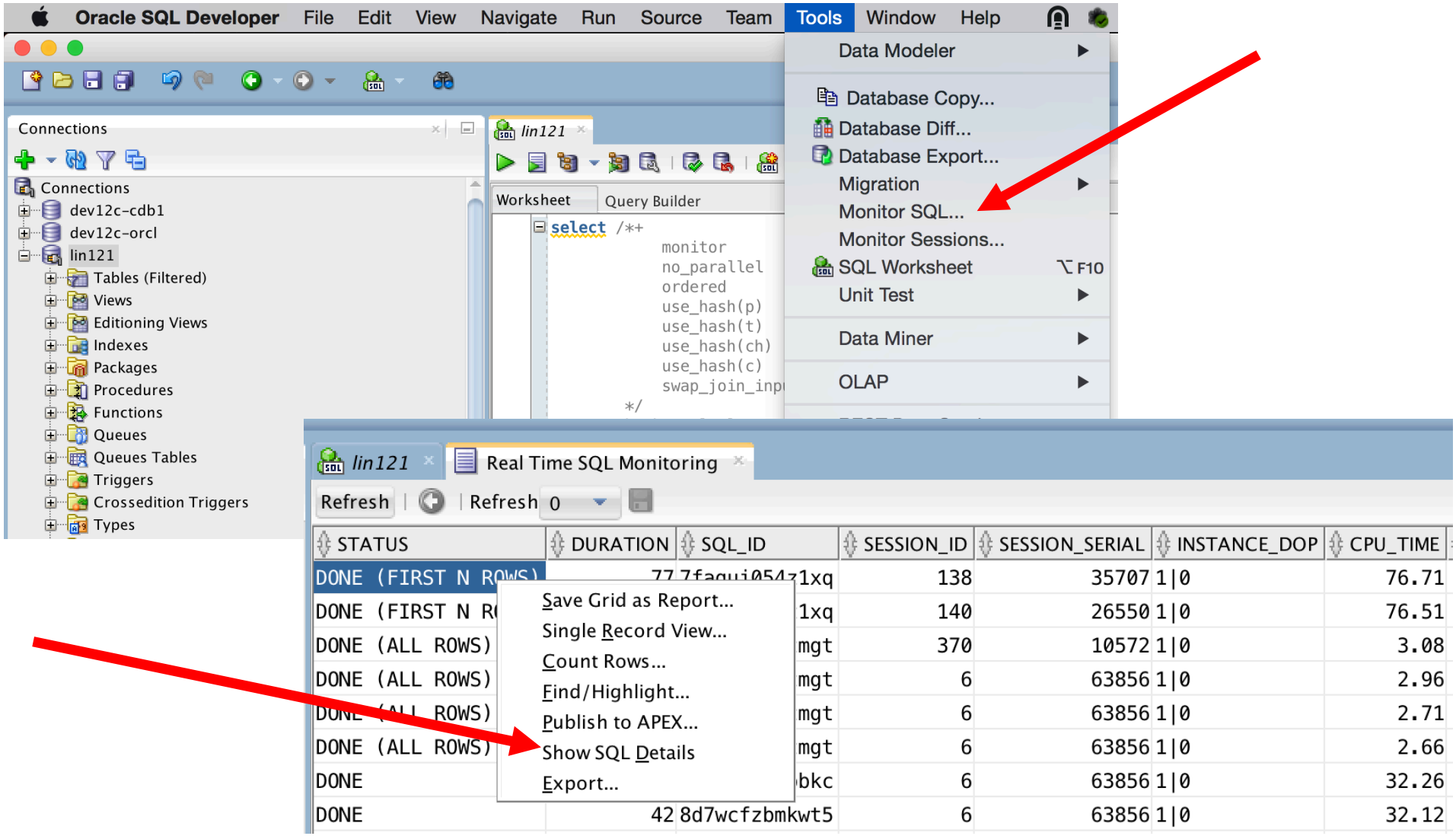
Historical SQL Monitoring Reports in 12c



Stored as CLOBs in DBA_HIST_REPORTS -> DBA_HIST_REPORTS_DETAILS

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

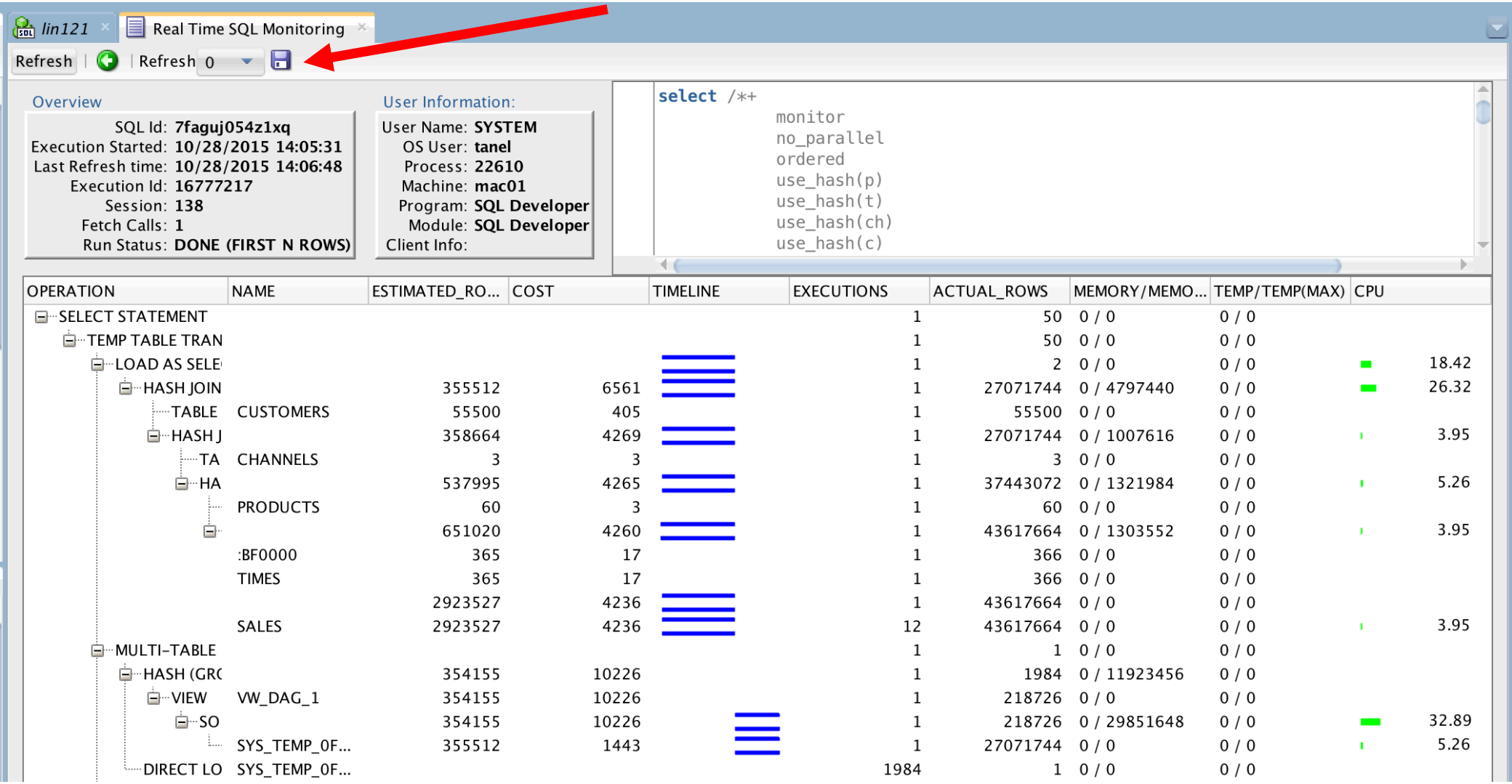
Getting SQL Monitor Reports: SQL Developer



blog.tanelpoder.com

gluent.

Getting SQL Monitor Reports: SQL Developer



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

FUNCTION REPORT_SQL_MONITOR RETURNS CLOB
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 SQL_ID                         VARCHAR2                IN     DEFAULT
 SESSION_ID                     NUMBER                  IN     DEFAULT
 SESSION_SERIAL                 NUMBER                  IN     DEFAULT
 SQL_EXEC_START                 DATE                    IN     DEFAULT
 SQL_EXEC_ID                    NUMBER                  IN     DEFAULT
 INST_ID                        NUMBER                  IN     DEFAULT
 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_8d7wcfzbmkwt5_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_fhpvstdhwf4kq_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_fhpvstdhwf4kq_16777216_151013_194532.html
-rw-------  1 tanel  staff  75964 Oct 13 17:41 sqlmon_2zfz7dzqf9azz_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?

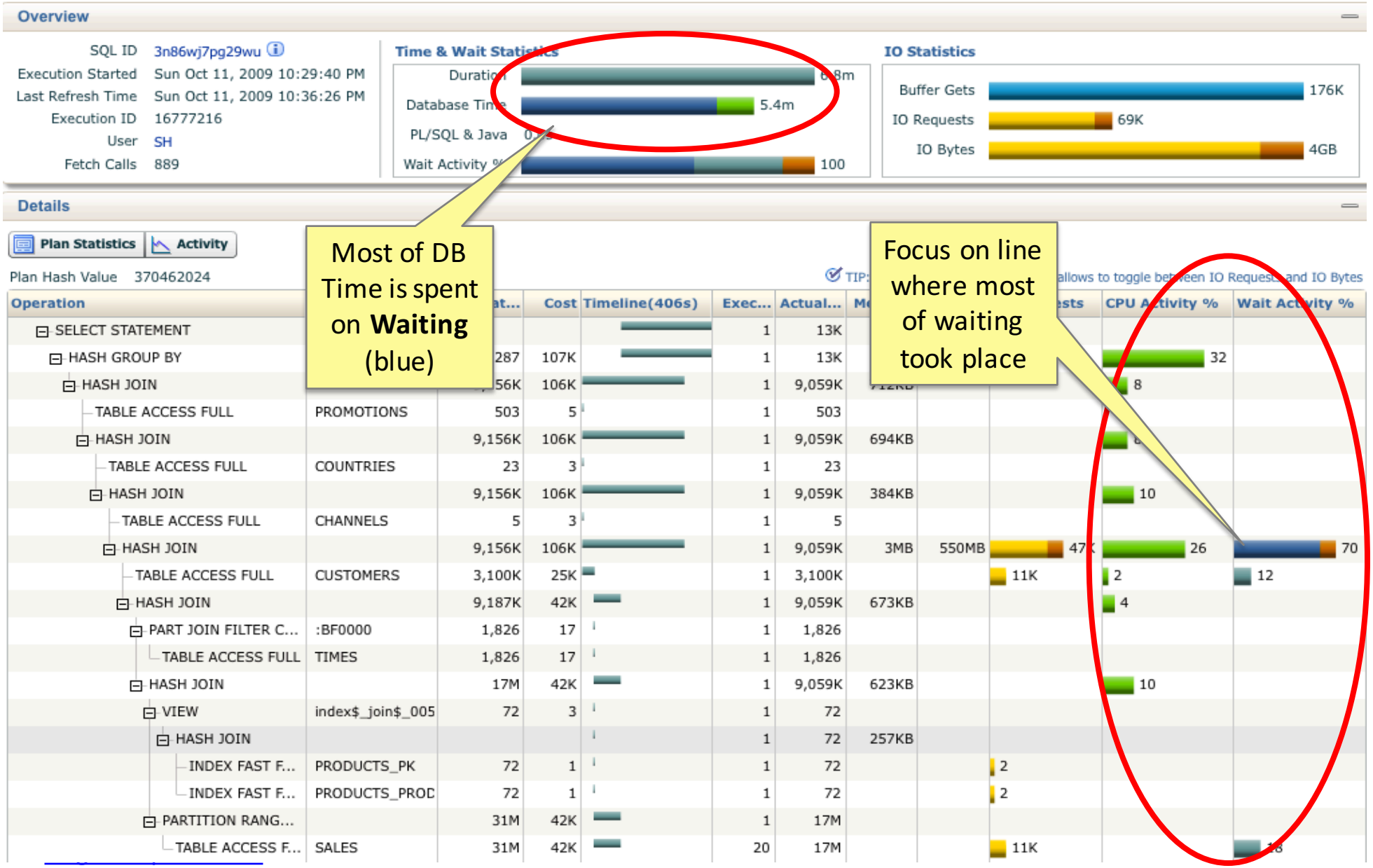
1. **Manually** and **immediately** when adding a MONITOR hint to SQL

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 suddently 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

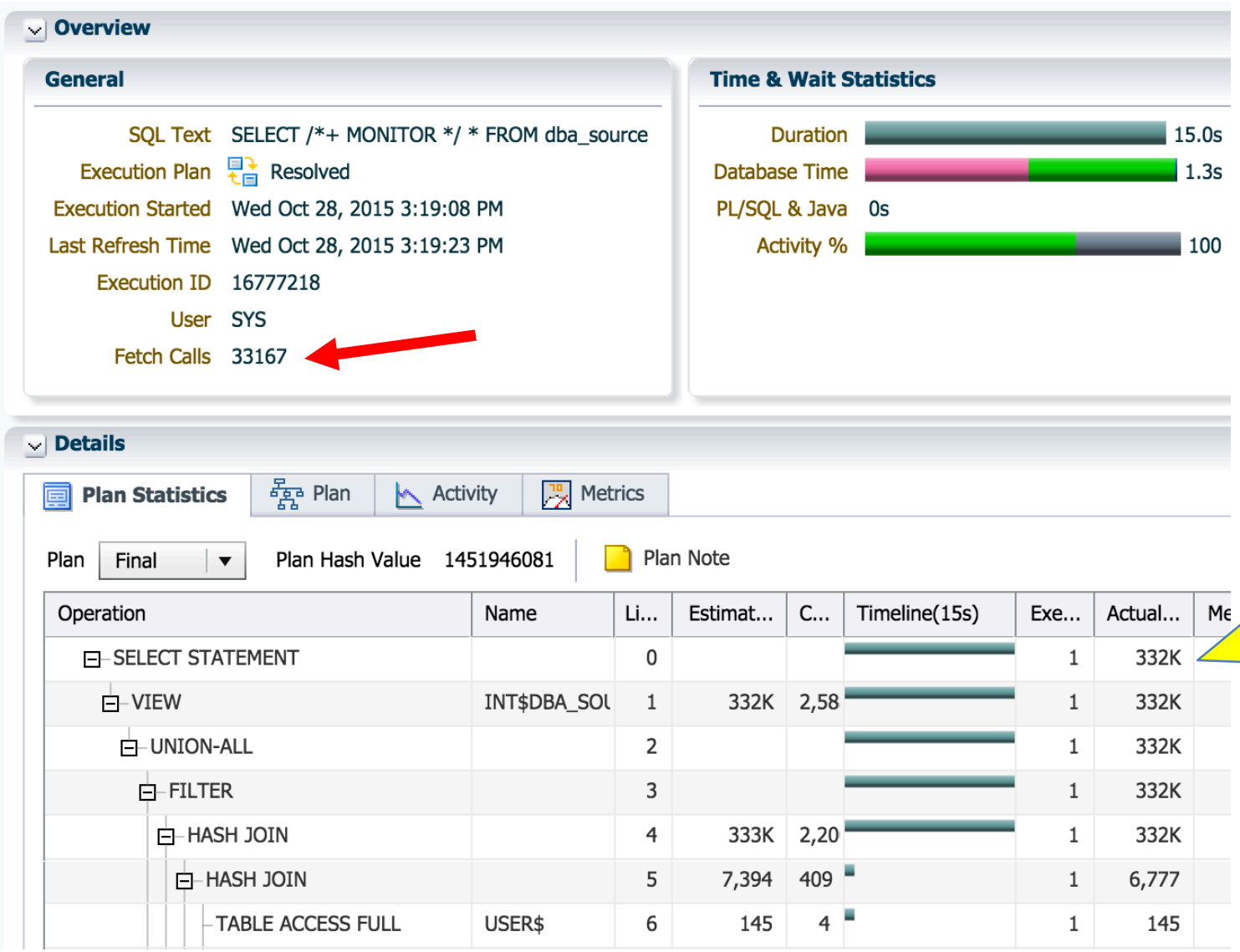
   - This tells you in which execution plan line(s) most of the time was spent

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**?!

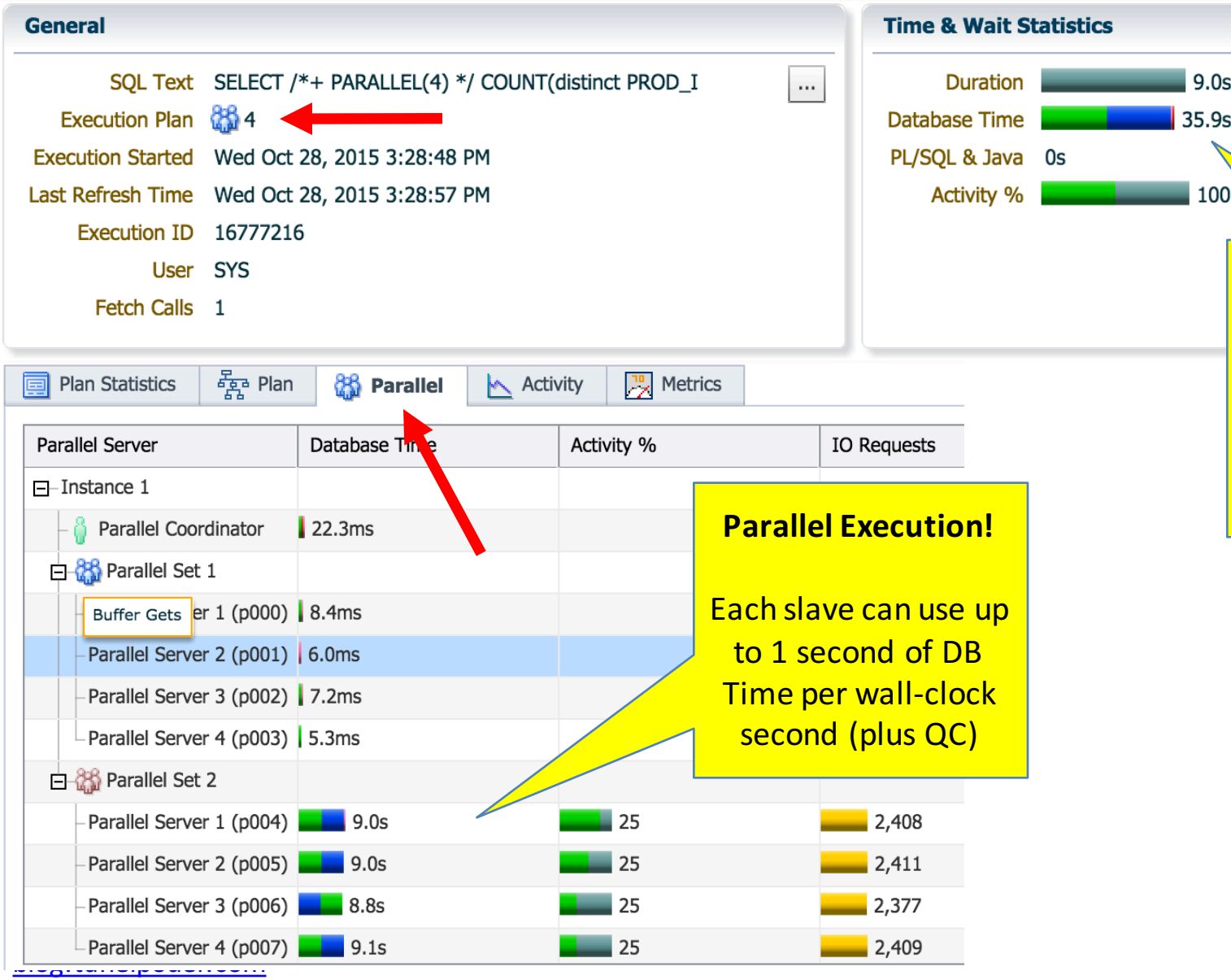


Duration = 15 seconds (wall-clock time)

DB Time = 1.3 seconds (time spent inside DB)

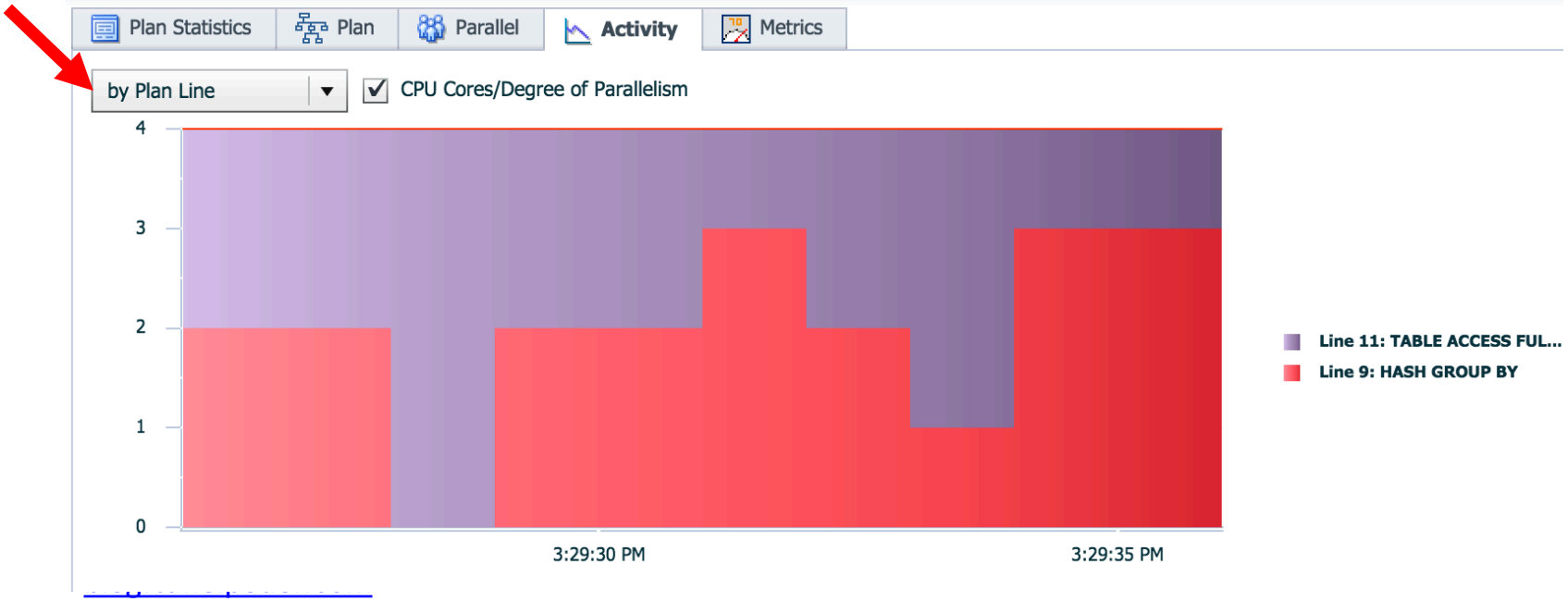
332 000 rows were returned, fetched 10 rows at a time (33167 fetches). **Most time spent on network roundtrips outside DB**

DB Time much bigger than **Duration**?



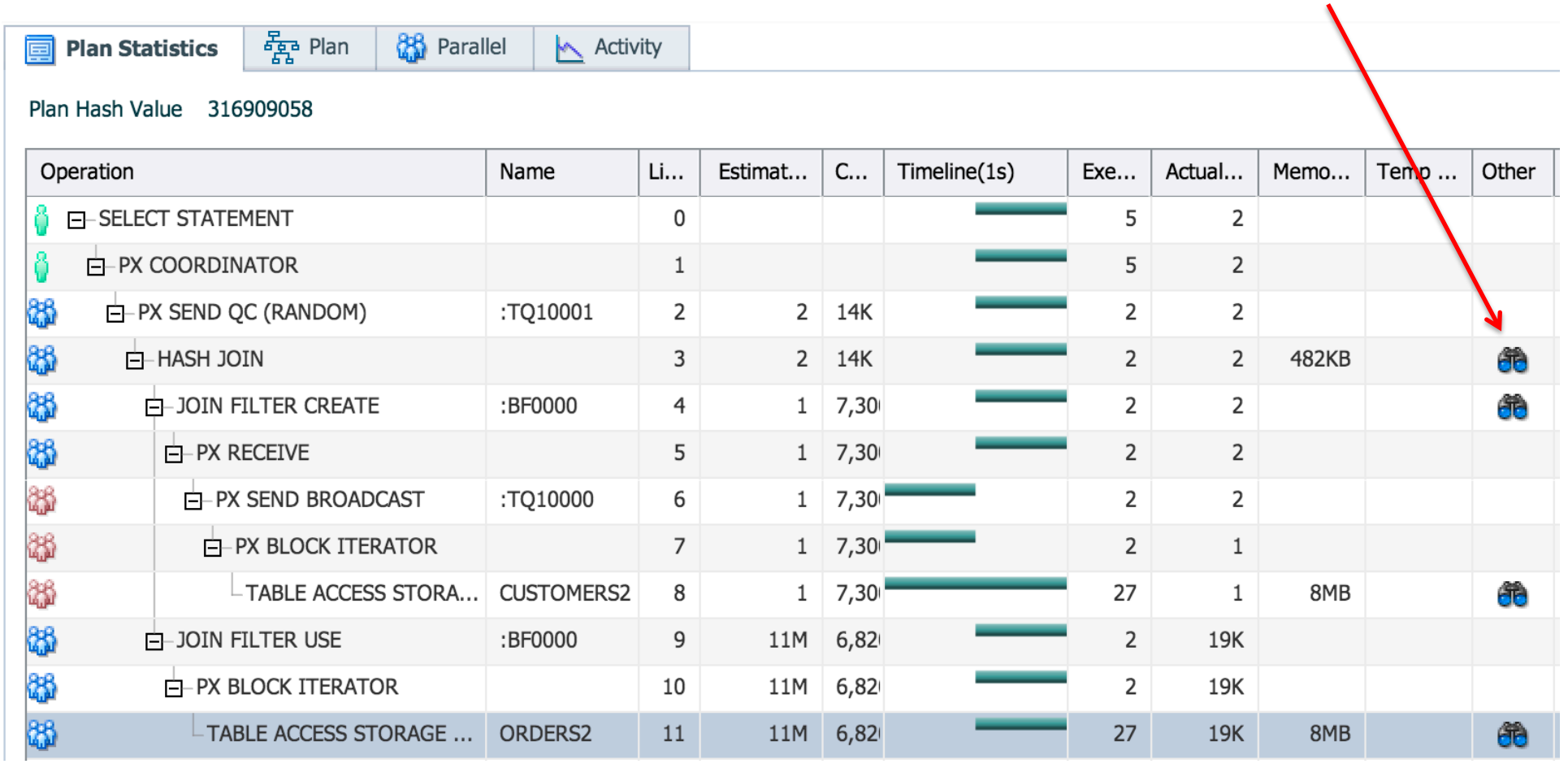
DB Time much bigger than **Duration**?

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!!!

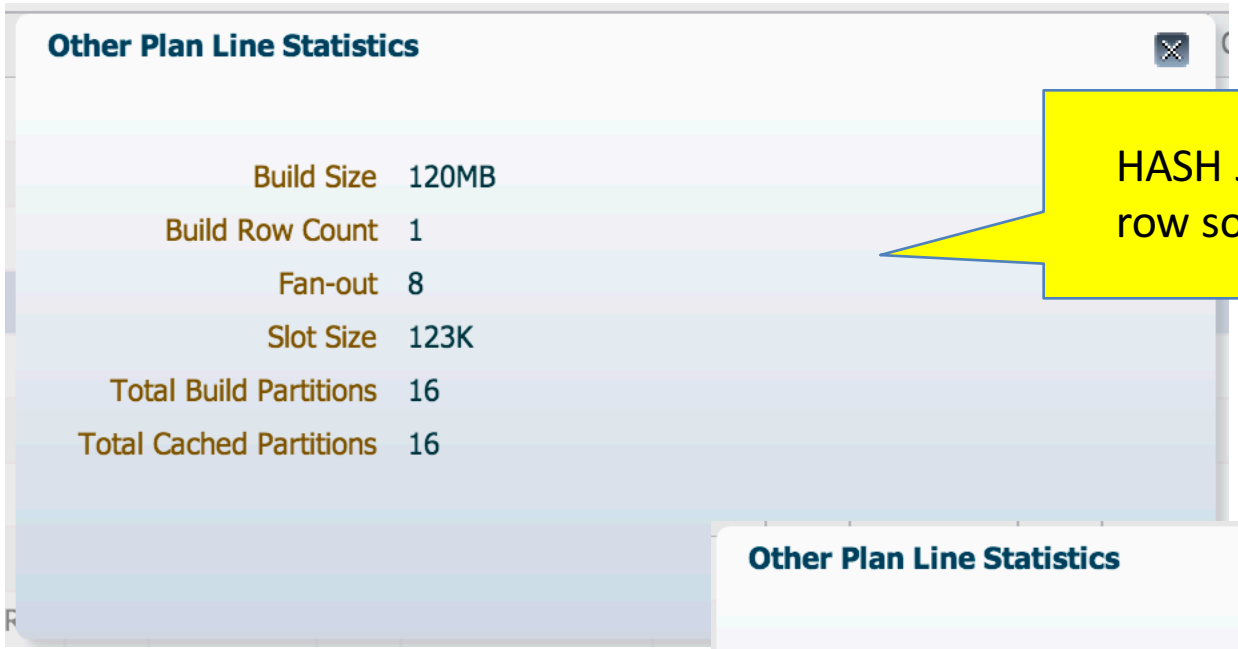


Plan Statistics | Plan | Parallel | Activity

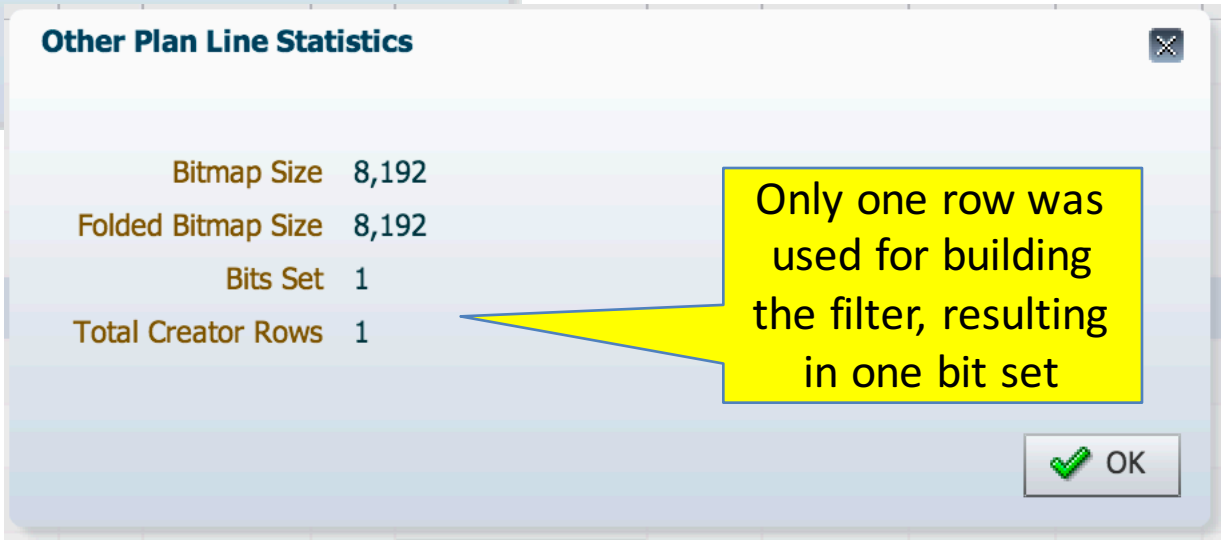
Plan Hash Value 316909058

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



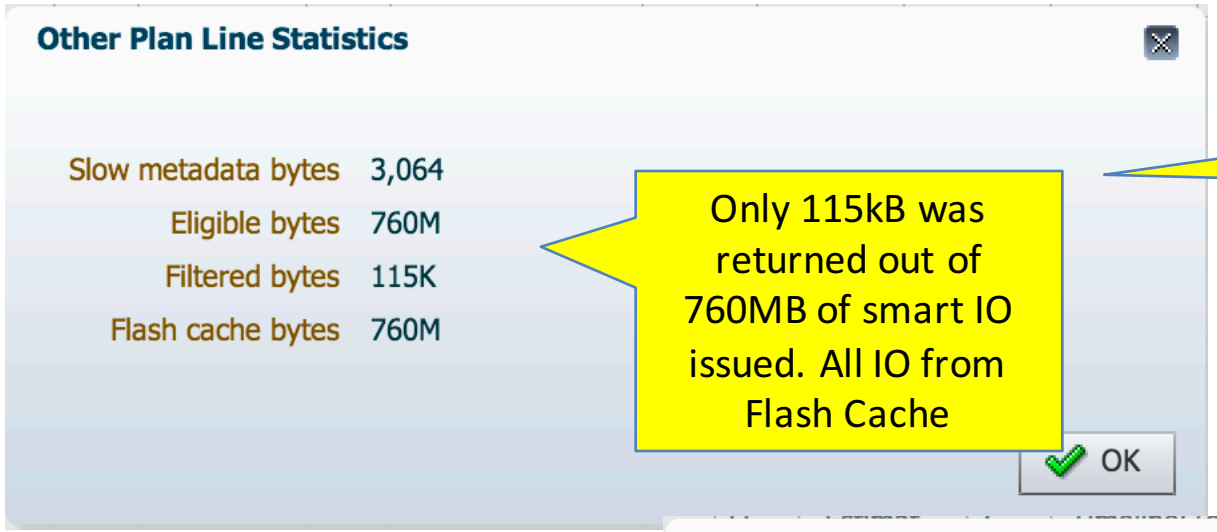
HASH JOIN
row source



JOIN FILTER
CREATE
row source

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

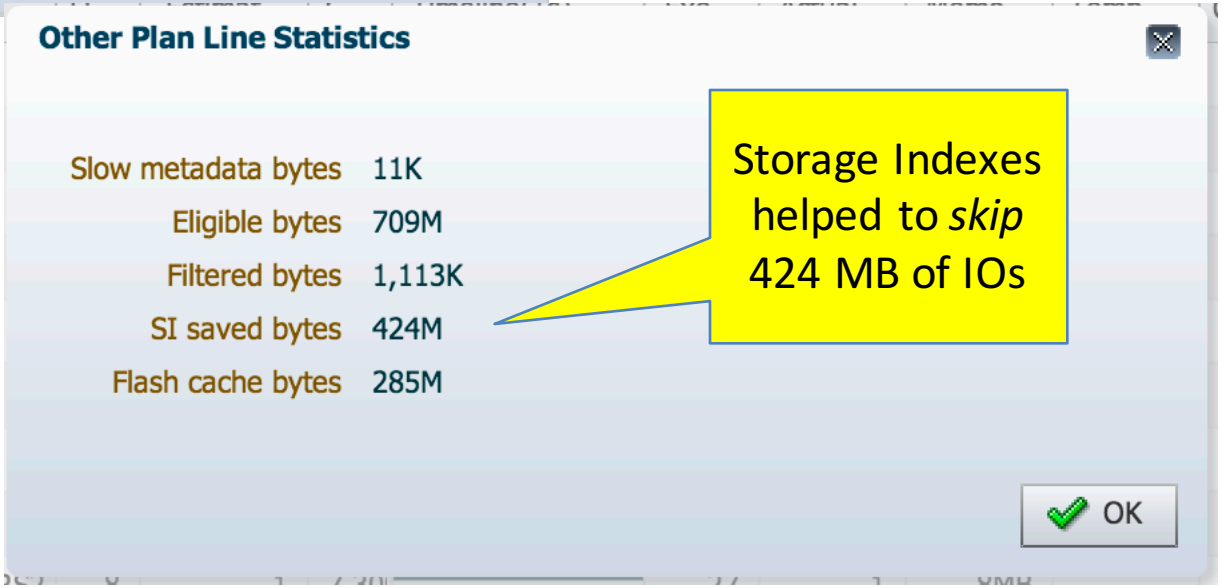
SQL Monitoring Improvements in Oracle 12c



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

TABLE ACCESS FULL on *customers*

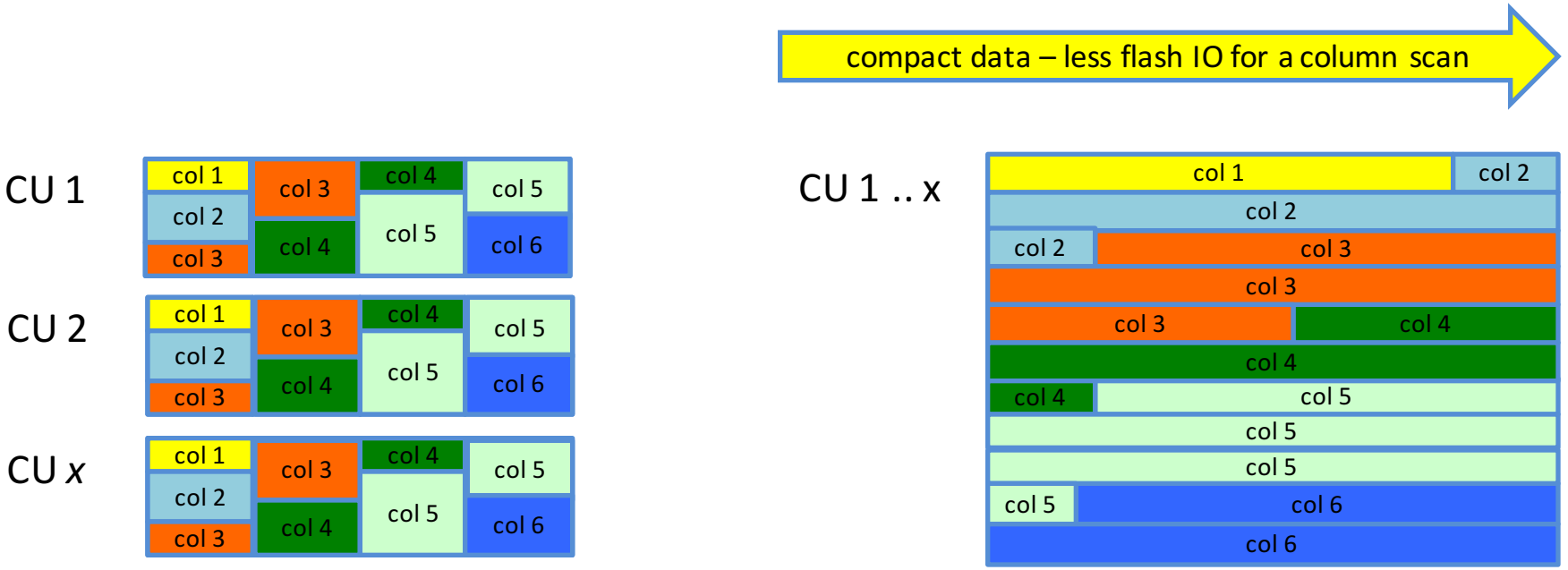
TABLE ACCESS FULL on *orders*



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



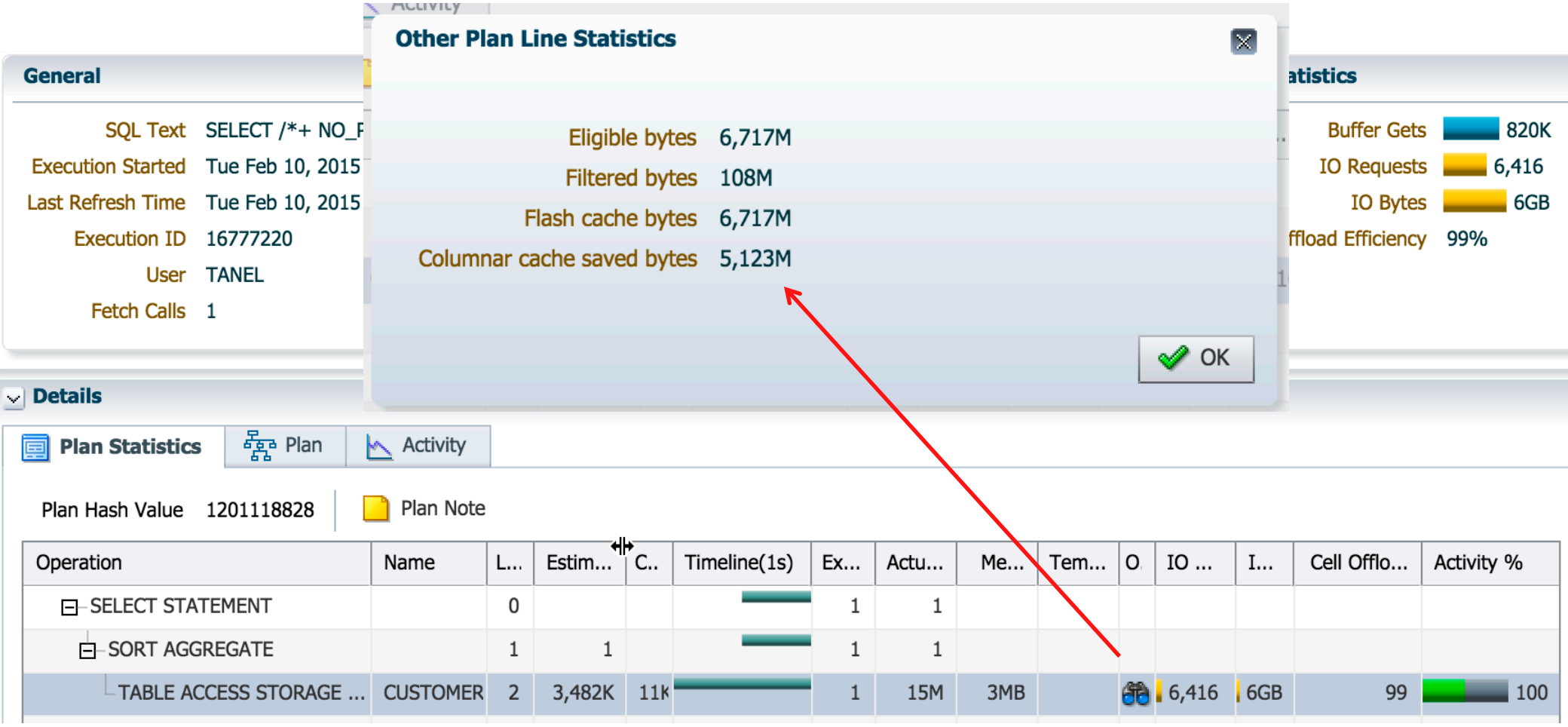
Columnar Flash Cache metrics – Session level

```
SQL> @snapper all 5 1 1234

-------------------------------------------------------------------------------
 TYPE, STATISTIC                                                ,         DELTA, HDELTA/SEC
-------------------------------------------------------------------------------
 STAT, physical read IO requests                                ,          6416,      1.26k
 STAT, physical read bytes                                      ,    6717366272,      1.32G
 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, cell physical IO bytes saved by columnar cache           ,    5128519680,      1.01G
 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



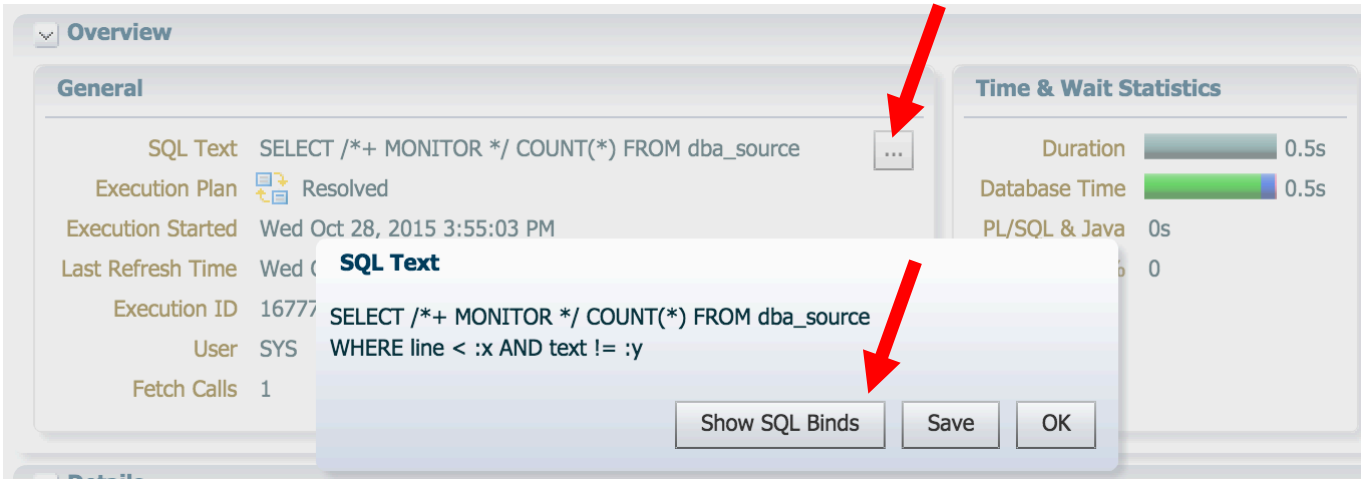
Full list of SQL Monitoring stats: v$sql_monitor_statname

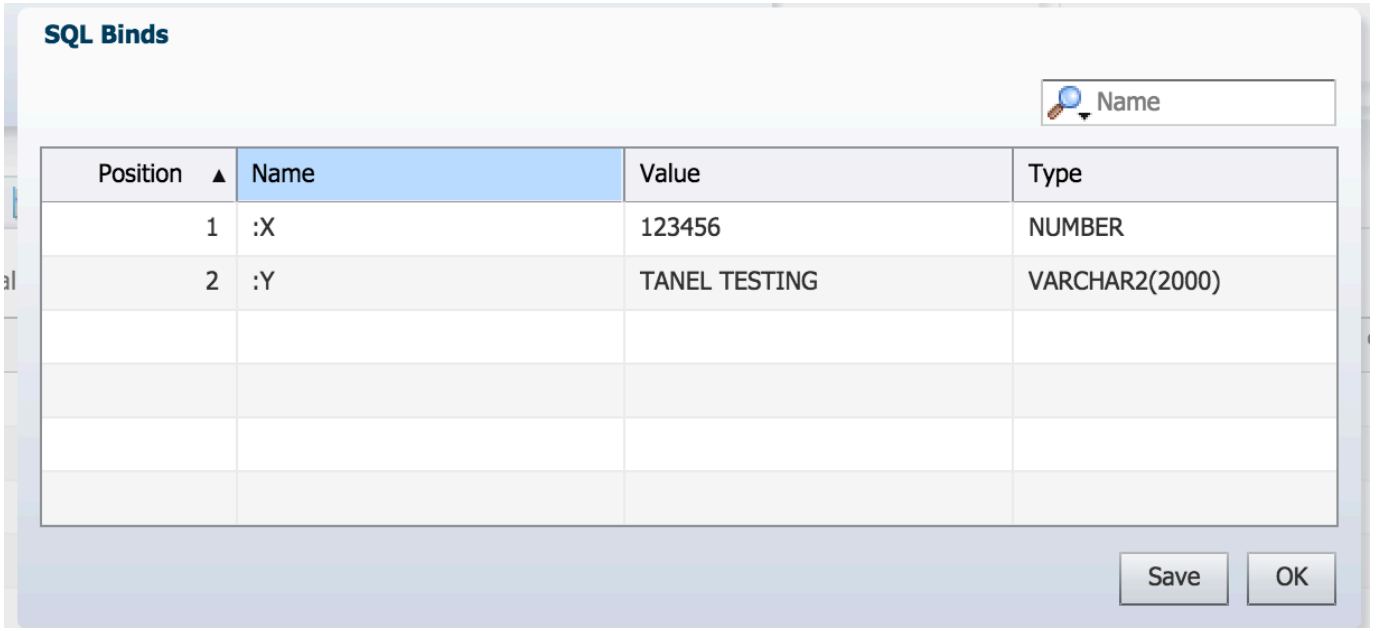
```
SQL> @sqlmon_sn %

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





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   58.4 % |######     |   2   TABLE ACCESS BY INDEX ROWID [CUSTOMERS] WAITING db file sequential read
   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   39.3 % |####       |   3                                           WAITING db file sequential read
```

```
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

blog.tanelpoder.com

@tanelpoder

We are hiring! ☺

http://gluent.com