PRACTICAL USE OF ORACLE ACTIVE SESSION HISTORY

Prepared By David Kurtz, Go-Faster Consultancy Ltd.
A Monograph on ASH
Version 1.1
Monday 3 November 2014
(E-mail: david.kurtz@go-faster.co.uk, telephone +44-7771-760660)
File: Practical_ASH.docx, 3 November 2014

Contents

Introduction.................................................................................................................................. 3
Agenda......................................................................................................................................... 3
A Very Brief Overview of Active Session History ................................................................. 4
ASH in Oracle Enterprise Manager ......................................................................................... 5
What data does ASH retain? ................................................................................................. 6
Comparison with SQL Trace ................................................................................................. 10
Application Instrumentation ............................................................................................... 12
PeopleSoft Specific Instrumentation................................................................................... 12
Using SQL to Analyse ASH Data........................................................................................... 14
Statistical Analysis Approach ............................................................................................... 14
Objectives ............................................................................................................................... 15
PeopleSoft Specific ASH Queries.......................................................................................... 15
Batch Processes...................................................................................................................... 15
Application Engine from PeopleTools 8.52 ......................................................................... 16
On-Line Activity...................................................................................................................... 18
XML Report............................................................................................................................. 20
Other Techniques.................................................................................................................... 23
Monitoring Progress of Processes in Real Time ................................................................. 23
Developers not Using Bind Variables.................................................................................. 25
How Many Executions? .......................................................................................................... 29
Oracle 10g............................................................................................................................... 29
Oracle 11g............................................................................................................................... 29
How Many Transactions (in 10g)? ................................................................. 31
When Did the Transaction Start ................................................................. 32
Single Wait Event ......................................................................................... 34
What Kind of Single Block Read ............................................................... 38
Blocking Lock Analysis ................................................................................. 40
Resolving the Lock Chain to the Ultimate Blocking Session ................. 44
Which Tables Account for My I/O? ............................................................ 46
Who is using this index? ............................................................................. 51
Index Use from SQL Plans Captured by AWR ........................................... 51
Limitations of Method ................................................................................ 57
Did my Execution Plan Change? ............................................................... 58
What was the Effect of Plan Stability ......................................................... 59
Which line in the Execution Plan? ............................................................. 62
Recursive SQL ............................................................................................ 64
Temporary Space Overhead ........................................................................ 66
Things That Can Go Wrong ....................................................................... 67
DISPLAY_AWR reports old costs ............................................................... 67
Statement not in Library Cache ................................................................. 69
Only Some Statements are in the Library Cache ....................................... 70
Lots of Shortlived Non-Shareable SQL ..................................................... 72
Error ORA-06502 ..................................................................................... 75
Error ORA-01422 ..................................................................................... 75
Error ORA-44002 ..................................................................................... 76
Appendix ....................................................................................................... 77
Further reading ........................................................................................... 77
Introduction

This document started as preparation for a presentation

Agenda

- Briefly, what is ASH and what does it collect (see page 4)
  - Recent/Historical Activity
- OEM and ASH Report (see page 5)
- Compare and Contrast with SQL Trace (see page 10)
- Application Instrumentation (see page 12).
  - PeopleSoft specific example of adding your own instrumentation.
- Using SQL to Analyse
  - Top SQL
    - Monitoring progress of process in read time (see page 23).
    - Lock Analysis (see page 40)
      - Blocking Session Not Active.
  - Changing Execution Plans (see page 58)
  - Source of I/O (see page 46)
  - Temporary Tablespace Usage (see page 66)
  - Limitations (see page 67)
    - Cannot Obtain SQL (space 67)
    - Error Messages (see page 75)
A Very Brief Overview of Active Session History

Active Session History (ASH) was introduced in Oracle 10g. It samples the activity of each active database session every second. The data is held in a buffer in memory in the database. The design goal is to keep about an hour (your mileage will vary). If a session is not active it will not be sampled. The in-memory buffer is exposed via a view called \texttt{v$active\_session\_history}.

You could sort of simulate some of ASH by taking a snapshot of \texttt{v$session} for every session, but the overhead would be prohibitive. ASH is built into the Oracle kernel, so its overhead is minimal.

When an AWR snapshot is taken, 1 row in 10 from the ASH buffer is copied down into the AWR repository. It can also be flushed to disk between snapshots when the buffer reaches 66\% full, so there is no missed data. The data is stored in \texttt{WRHS\_ACTIVE\_SESSION\_HISTORY} and it is exposed via \texttt{DBA\_HIST\_ACTIVE\_SESS\_HISTORY}.

ASH is enabled by default, but before you rush off to use it, be aware that it is a licenced feature. It is part of the Diagnostic Pack, so you have to pay for it. I don’t like that either, but that’s how it is.

\footnote{I want to emphasise that if the session is not active it will not be sampled. You can actually set a parameter \texttt{_ash\_enable\_all = TRUE} to force all sessions, including idle sessions, to be sampled.}

But as Doug Burns points out in his blog posting (http://oracledoug.com/serendipity/index.php?archives/1395-ASH-and-the-psychology-of-Hidden-Parameters.html), these are undocumented, unsupported parameters, and they are set this way for a reason – you have been warned.
ASH in Oracle Enterprise Manager

Of course, OEM provides a way to run ASH reports, and here you see I have picked a particular time window, and I have specified a module name – in this case the main payroll calculation process.

And this is great. The report is easy to produce, and it tells you lots of things. Which SQL statements are consuming the most time, which objects have the most I/O.

You can see in this example I picked a module that was responsible for 86% of the total, and there were an average of 14.8 active sessions (I know there were 32 concurrent processes).

But, you don’t get execution plans, and for that you will need to dig deeper yourself, and learn to use the DBMS_XPLAN package.
What data does ASH retain?

Most of the columns on v$active_session_history are taken directly from column of the same name on v$session, some have different name, and there is some additional information that is not available elsewhere.

<table>
<thead>
<tr>
<th>Column on v$active_session_history</th>
<th>Correspondence to v$session</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLE_ID</td>
<td>ID of ASH Sample</td>
</tr>
<tr>
<td>SAMPLE_TIME</td>
<td>Time of ASH Sample</td>
</tr>
<tr>
<td>IS_AWR_SAMPLE</td>
<td>New in 11gR2</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>V$SESSION.SID</td>
</tr>
<tr>
<td>SESSION_SERIAL#</td>
<td>V$SESSION.SERIAL#</td>
</tr>
<tr>
<td>USER_ID</td>
<td>V$SESSION.USER#</td>
</tr>
<tr>
<td>SQL_ID</td>
<td>√</td>
</tr>
<tr>
<td>IS_SQL_ID_CURRENT</td>
<td>New in 11gR2</td>
</tr>
<tr>
<td>SQL_CHILD_NUMBER</td>
<td>√</td>
</tr>
<tr>
<td>FORCE_MATCHING_SIGNATURE</td>
<td>not on V$SESSION</td>
</tr>
<tr>
<td>SQL_OPCODE</td>
<td>√</td>
</tr>
<tr>
<td>TOP_LEVEL_SQL_ID</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>TOP_LEVEL_SQL_OPCODE</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>SQL_PLAN_HASH_VALUE</td>
<td>not on V$SESSION</td>
</tr>
<tr>
<td>SQL_PLAN_LINE_ID</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>SQL_PLAN_OPERATION</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>SQL_PLAN_OPTIONS</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>SQL_EXEC_ID</td>
<td>√ New in 11gR1</td>
</tr>
<tr>
<td>SQL_EXEC_START</td>
<td>√ New in 11gR1</td>
</tr>
<tr>
<td>PLSQL_ENTRY_OBJECT_ID</td>
<td>√</td>
</tr>
<tr>
<td>PLSQL_ENTRY_SUBPROGRAM_ID</td>
<td>√</td>
</tr>
<tr>
<td>PLSQL_OBJECT_ID</td>
<td>√</td>
</tr>
<tr>
<td>PLSQL_SUBPROGRAM_ID</td>
<td>√</td>
</tr>
<tr>
<td>SERVICE_HASH</td>
<td>V$ACTIVE_SERVICES.NAME_HASH</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>SESSION_TYPE</td>
<td>VSSESSION.TYPE</td>
</tr>
<tr>
<td>SESSION_STATE</td>
<td>Waiting/On-CPU</td>
</tr>
<tr>
<td>QC_SESSION_ID</td>
<td>Parallel query co-ordinator</td>
</tr>
<tr>
<td>QC_INSTANCE_ID</td>
<td>√</td>
</tr>
<tr>
<td>QC_SESSION_SERIAL#</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>BLOCKING_SESSION</td>
<td>√</td>
</tr>
<tr>
<td>BLOCKING_SESSION_STATUS</td>
<td>VALID – blocking session within the same instance</td>
</tr>
<tr>
<td></td>
<td>GLOBAL – blocking session in another instance.</td>
</tr>
<tr>
<td>BLOCKING_SESSION_SERIAL#</td>
<td>VSSESSION.SERIAL# of blocking session</td>
</tr>
<tr>
<td>EVENT</td>
<td>√</td>
</tr>
<tr>
<td>EVENT_ID</td>
<td>From V$EVENT_NAME</td>
</tr>
<tr>
<td>EVENT#</td>
<td>√</td>
</tr>
<tr>
<td>SEQ#</td>
<td>√</td>
</tr>
<tr>
<td>P1TEXT</td>
<td>√</td>
</tr>
<tr>
<td>P1</td>
<td>√</td>
</tr>
<tr>
<td>P2TEXT</td>
<td>√</td>
</tr>
<tr>
<td>P2</td>
<td>√</td>
</tr>
<tr>
<td>P3TEXT</td>
<td>√</td>
</tr>
<tr>
<td>P3</td>
<td>√</td>
</tr>
<tr>
<td>WAIT_CLASS</td>
<td>√</td>
</tr>
<tr>
<td>WAIT_CLASS_ID</td>
<td>√</td>
</tr>
<tr>
<td>WAIT_TIME</td>
<td>√</td>
</tr>
<tr>
<td>TIME_WAITED</td>
<td>√</td>
</tr>
<tr>
<td>XID</td>
<td>Not on V$SESSION</td>
</tr>
<tr>
<td>REMOTE_INSTANCE#</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>CURRENT_OBJ#</td>
<td>VSSESSION.ROW_WAIT_OBJ#</td>
</tr>
<tr>
<td>CURRENT_FILE#</td>
<td>VSSESSION.ROW_WAIT_FILE#</td>
</tr>
<tr>
<td>CURRENT_BLOCK#</td>
<td>VSSESSION.ROW_WAIT_BLOCK#</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>CURRENT_ROW#</td>
<td>√ New in 11gR1</td>
</tr>
<tr>
<td>CONSUMER_GROUP_ID</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>PROGRAM</td>
<td>√</td>
</tr>
<tr>
<td>MODULE</td>
<td>√</td>
</tr>
<tr>
<td>ACTION</td>
<td>√</td>
</tr>
<tr>
<td>CLIENT_ID</td>
<td>VSSESSION.CLIENT_IDENTIFIER</td>
</tr>
<tr>
<td>FLAGS</td>
<td>Undocumented</td>
</tr>
<tr>
<td>IN_CONNECTION_MGMT</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>IN_PARSE</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>IN_HARD_PARSE</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>IN_SQL_EXECUTION</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>IN_PLSQL_EXECUTION</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>IN_PLSQL_RPC</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>IN_PLSQL_COMPILATION</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>IN_JAVA_EXECUTION</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>IN_BIND</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>IN_CLOSE_CURSOR</td>
<td>New in 11gR1</td>
</tr>
<tr>
<td>IN_SEQUENCE_LOAD</td>
<td>New in 11gR2</td>
</tr>
<tr>
<td>CAPTURE_OVERHEAD</td>
<td>New in 11gR2</td>
</tr>
<tr>
<td>REPLAY_OVERHEAD</td>
<td>New in 11gR2</td>
</tr>
<tr>
<td>IS_CAPTURED</td>
<td>New in 11gR2</td>
</tr>
<tr>
<td>IS_REPLAYED</td>
<td>New in 11gR2</td>
</tr>
<tr>
<td>MACHINE</td>
<td>√ New in 11gR2</td>
</tr>
<tr>
<td>PORT</td>
<td>√ New in 11gR2</td>
</tr>
<tr>
<td>ECID</td>
<td>√ New in 11gR2</td>
</tr>
<tr>
<td>TM_DELTA_TIME</td>
<td>New in 11gR2</td>
</tr>
<tr>
<td>TM_DELTA_CPU_TIME</td>
<td>New in 11gR2</td>
</tr>
<tr>
<td>Metric</td>
<td>New in 11gR2</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>TM_DELTA_DB_TIME</td>
<td></td>
</tr>
<tr>
<td>DELTA_TIME</td>
<td></td>
</tr>
<tr>
<td>DELTA_READ_IO_REQUESTS</td>
<td></td>
</tr>
<tr>
<td>DELTA_WRITE_IO_REQUESTS</td>
<td></td>
</tr>
<tr>
<td>DELTA_READ_IO_BYTES</td>
<td></td>
</tr>
<tr>
<td>DELTA_WRITE_IO_BYTES</td>
<td></td>
</tr>
<tr>
<td>DELTA_INTERCONNECT_BYTES</td>
<td></td>
</tr>
<tr>
<td>PGA_ALLOCATED</td>
<td></td>
</tr>
<tr>
<td>TEMP_SPACE_ALLOCATED</td>
<td></td>
</tr>
</tbody>
</table>
Comparison with SQL Trace

ASH and SQL*Trace are not the same thing, but both are valuable tools for finding out about where processes spend time.

SQL*Trace (or event 10046 as we used to call it) has been my weapon of choice for solving performance issues for a very long time, and it is extremely effective, and there is still a place for it.

There are difficulties with using SQL trace, especially in a production environment.

- Firstly, it does have a run time overhead. You could afford to trace a single process, but you certainly couldn’t trace the entire database.
- You have to work with trace in a reactive way. You will probably not already be tracing a process when you experience a performance problem, so you need to run the process again and reproduce the poor performance with trace.
- Trace will tell you if a session is blocked waiting on a lock. However, it will not tell you who is blocking you. ASH will do this (although there are limitations).
- A trace file records everything that happens in a session, whereas ASH data samples the session every seconds. Short-lived events will be missed, so the data has to be handled statistically (see page 14).
- There are problems with both approaches if you have the kind of application where you have lots of different SQL statements because the application uses literal values rather than bind variables (and cursor sharing is EXACT).
- Oracle’s TKPROF trace file profiler cannot aggregate these statements, but I have found another called ORASRP (www.oracledba.ru/orasrp) that can. With ASH, you will see different SQL_IDs, but it can be effective to group statements with the same execution plan.
- You may have trouble finding the SQL text in the SGA (or via the DBMS_XPLAN package) because it has already been aged out of the library cache. You may have similar problems with historical ASH data because the statement had been aged out when the AWR snapshot was taken.
- A trace file, with STATISTICS_LEVEL set to ALL, will give you timings for each operation in the execution plan. So, you can see where in the execution plan the time was spent. ASH will only tell you how long the whole statement takes to execute, and how long was spent on which wait event.
Through the rest of this document you will see SQL_IDs. However, in a SQL trace the statements are identified by hash_value. Those hash values do not show up if you profile your trace file with tkprof, but they do if you use OraSRP. **SQL_ID is just a fancy representation of hash value**, so you can convert from a SQL_ID to a hash_value. Oracle supply function DBMS_UTILITY.SQLID_TO_SQLHASH(), but as the comment on the blog says Tanel’s script is much cooler².

You can’t get the whole of the SQL_ID back from the hash values (because it is trimmed off), but you can get the last 5 or 6 characters it help you find or match SQL statements³.


³ And I could never have written this without seeing Tanel’s code!
Application Instrumentation

Oracle has provided a package called DBMS_APPLICATION_INFO since at least Oracle 8. This allows you to set two attributes; MODULE and ACTION for a session. That value then appears in v$session, and can be very useful to help you identify what database sessions relate to what part of an application. These values are then also captured by ASH.

I cannot over-emphasise the importance of this instrumentation when analysing performance issues. Without sensible values in these columns all you have is the program name. You will probably struggle to identify ASH data for the sessions which are of interest.

These values are not set by default. Instead DBAs are dependent on developers to include them in their code. For example, Oracle E-Business Suite has built this into the application.

PeopleSoft Specific Instrumentation

However, other application vendors have not. For example, PeopleSoft (prior to PeopleTools 8.50) only write the name of the executable into the module4. This is really no help at all because the executable name is held in another column.

For batch processes, I have developed a trigger which is fired by batch processes as they start and which sets a meaningful process name, and puts the unique process instance number into the action.

```
CREATE OR REPLACE TRIGGER sysadm.psftapi_store_prcsinstance
BEFORE UPDATE OF runstatus ON sysadm.psprcsrqst FOR EACH ROW
WHEN (new.runstatus IN('3','7','8','9','10') OR old.runstatus IN('7','8'))
AND new.prctype != 'PSJob')
BEGIN
  ...
  psftapi.set_action(p_prcsinstance=>:new.prcsinstance
  ,p_runstatus=>:new.runstatus
  ,p_prcsname=>:new.prcsname);
  ...
EXCEPTION WHEN OTHERS THEN NULL; --exception deliberately coded to suppress all exceptions
END;
/
```

From PeopleTools 8.50, Oracle added instrumentation for the on-line part of the application.

In PeopleTools 8.52, further instrumentation was added for Application Engine. The Application Engine program name, section name, step name and step type are written to the ACTION. The PeopleSoft Operator ID is stored in CLIENT_ID
The results of this instrumentation are visible in Enterprise Manager.

Later, you will see the value of this instrumentation as I use it to join a combination of data in the application about batch processes with the ASH repository to identify where a given process spent time.
Using SQL to Analyse ASH Data

Statistical Analysis Approach

ASH data is a sample and so must be handled statistically. If something happens that lasts 10 seconds, then it will be sampled about 10 times.

However, not everything that happens is captured. If something happens that last less than a second, but it happens very frequently, some of them will be captured. For example, if something happens which lasts for 1/10th of a second, but happens 100 times then you would expect to capture it about 10 times. In all, the 100 occurrences lasted 10 times. So by counting each ASH row as worth 1 seconds of wait time you come out at the right answer. This is what I mean by taking a statistical approach.

So, if you are looking at a current or recent process you the raw ASH data, and the query that you have to construct when working with is something along these lines

```sql
SELECT ...
, SUM(1) ash_secs
FROM v$active_session_history
WHERE ...
GROUP BY ...
```

And if you are going further back in time then you have to work with the historical data, only 1 in 10 rows are kept, so now each row is worth 10 seconds

```sql
SELECT ...
, SUM(10) ash_secs
FROM dba_hist_active_sess_history
WHERE ...
GROUP BY ...
```

And of course, you won’t see recent data in this view until there is an AWR snapshot for the ASH buffer fills to 2/3 and flushes.
ASH History is exposed by the view DBA_HIST_ACTIVE_SESSION_HISTORY. It is stored in the table SYS.WRHS_ACTIVE_SESSION_HISTORY which is range partitioned on DBID and SNAP_ID. To make the SQL work efficiently you need to specify the snap ID, for that I use dba_hist_snapshotS to identify the range of snapshots that you want to use, and the partitions first so that you eliminate unwanted partitions. You may need the LEADING hint to force Oracle to start with the snapshot view, and then the USE NL hint to force it to work through each snapshot, which will guarantee a single partition access. Otherwise your queries could run for ever!

```
SELECT /*+LEADING(x) USE_NL(h)*/ _
  , SUM(10) ash_secs
FROM dba_hist_active_sess_history h
  , dba_hist_snapshot x
WHERE x.snap_id = h.snap_id
AND x.dbid = h.dbid
AND x.instance_number = h.instance_number
AND x.end_interval_time >= _
AND x.begin_interval_time <= _
AND ...
GROUP BY ...
```

**Objectives**

Ask yourself what you are trying to find out.

- Are you interested in a single database session, or a group of sessions, or the whole database?
- All ASH Data – v- One Wait Event
- Time Window

**PeopleSoft Specific ASH Queries**

To get the most out of ASH you need to know how to relate database session to processes. That starts with using DBMS_APPLICATION_INFO to register the process name and process instance of batch processes on the session (see page 12). But there is more.

**Batch Processes**

The start and end time of a batch process is recorded on the process request table, and you can use that to identify the snapshots, and thence the active session history.

```
SELECT /*+LEADING(r x h) USE_NL(h)*/ r.prcsinstance
  , h.sql_id
  , h.sql_child_number
  , h.sql_plan_hash_value
  , (CAST(r.enddtm AS DATE)-CAST(r.begindttm AS DATE))*86400 exec_secs
  , SUM(10) ash_secs
FROM dba_hist_snapshot x
```

Specify a hint to ensure good performance. Start with the process request table, then go to the snapshots, finally go to the ASH data and look it up with a nested loop join.
Application Engine from PeopleTools 8.52

From PeopleTools 8.52 there is additional instrumentation of the session in Application Engine processes.

- Module is now set to string composed of PSAE.<name of scheduled Application Engine program>.<session ID number>. The Application Engine name is as it appears in Process Monitor. The session ID number is the operation system process ID of the client process. It is recorded in PSPRCSQUE.SESSIONIDNUM.

- Action is set to the concatenation of the Application Engine program name, section name, step name and step type. The string can be truncated if it is too long.

Consequently a slightly different SQL query is required to analyse ASH data for these processes

```sql
FROM ( 
SELECT /*+leading(r q x h)
use_nl(h)*/
    r.prcsinstance, r.prcsname, r.begindttm, r.enddttm, h.sql_id, h.sql_plan_hash_value 
FROM dba_hist_active_sess_history h, sysadm.psprcsrqst
WHERE x.end_interval_time >= r.begindttm
   AND x.begin_interval_time <= r.enddttm
   AND h.sample_time BETWEEN r.begindttm AND r.enddttm
   AND h.snap_id = x.snap_id
   AND h.dbid = x.dbid
   AND h.instance_number = x.instance_number
   AND h.module = r.prcsname
   AND h.action LIKE 'PI='||r.prcsinstance||'%'  
   AND r.prcsinstance = 1956338
GROUP BY r.prcsinstance, r.prcsname, r.begindttm, r.enddttm, h.sql_id, h.sql_plan_hash_value
ORDER BY 1
)
```

6 This table described the process

7 Identify the AWR snapshots that coincide with the period that the process was running

8 Filter ASH data to exactly the period that the process was running.

9 Filter ASH data by Module which is the name of the process on the process request table

10 Filter ASH data by Action which includes the process instance number

11 Uniquely identify process

12 However, most of the examples in this document were written against PeopleTools 8.49.

13 Note that the LEADING hint has been changed to include PSPRCSQUE as the second table visited.
Now it is possible to include the step reference from the Action in the ASH profile. Of course it is likely, as in this example, that one step produces different SQL IDs on different executions either due to dynamically generated SQL, or different bind variables values in different executions being resolved to different literal values by Application Engine.

```
<table>
<thead>
<tr>
<th>PRCSINSTANCE</th>
<th>ACTION</th>
<th>SQL_ID</th>
<th>Hash Value</th>
<th>Secs</th>
<th>Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>10622159</td>
<td>TL_TIMEADMIN.END.STATS2.S</td>
<td>636f1tg06rjk</td>
<td>291563330</td>
<td>5901</td>
<td>320</td>
</tr>
<tr>
<td>10622159</td>
<td>TL_TIMEADMIN.END.STATS2.S</td>
<td>cbrj18vrf2qj</td>
<td>821036523</td>
<td>5901</td>
<td>320</td>
</tr>
<tr>
<td>10622159</td>
<td>FO_TL_OVR_RT.MAIN.Step03.S</td>
<td>4rgvvjm5jt1gn</td>
<td>286736047</td>
<td>5901</td>
<td>300</td>
</tr>
<tr>
<td>10622159</td>
<td>TL_TRPROFILE.TRPROFIL.End_Effd.</td>
<td>gbwayc9ac1jxu</td>
<td>3317352158</td>
<td>5901</td>
<td>300</td>
</tr>
<tr>
<td>10622159</td>
<td>FO_TL_OVR_RT.MAIN.Step05.S</td>
<td>2zyz4zr0js2j8</td>
<td>1281985392</td>
<td>5901</td>
<td>250</td>
</tr>
<tr>
<td>10622159</td>
<td>TL_TA001100.TA001120.Step09A.S</td>
<td>bcrxp3xps3466</td>
<td>537875261</td>
<td>5901</td>
<td>120</td>
</tr>
<tr>
<td>10622159</td>
<td>TL_TA000900.TA000960.Step130.S</td>
<td>9j6Dxwk6qct5</td>
<td>334959449</td>
<td>5901</td>
<td>90</td>
</tr>
<tr>
<td>10622159</td>
<td>TL_TA001000.TA001000.Step02.S</td>
<td>ammpdulsnL3y8h</td>
<td>2593883656</td>
<td>5901</td>
<td>80</td>
</tr>
<tr>
<td>10622159</td>
<td>FO_TL_OVR_RT.MAIN.Step01.S</td>
<td>Fkmbwcd00kgv8</td>
<td>2562066473</td>
<td>5901</td>
<td>70</td>
</tr>
</tbody>
</table>
```

14 PSPRCSQUE is also needed to obtain the session ID number and this can be joined to PSPRCSRQST by PRCSINSTANCE.

15 The combination of process name, session ID number and sample time is not guaranteed to be unique. It is possible that two instances of the same program with the same session ID number could run on different Process Schedulers on different servers concurrently, although this is not likely.
On-Line Activity

I have used the PeopleSoft Performance Monitor (PPM) to find a period in time when the system exhibits degraded performance.

With on-line activity it is not possible to add module and action instrumentation. At the moment the program name is copied to module, and that is no advantage at all because I already have program in the ASH data

Enhancement Request: PeopleSoft added instrumentation for Performance Monitor, the context information they there use there for a PIA transaction could also be set in DBMS_APPLICATION_INFO. Combine Component and Page to Module, and set Action as Action

So, all I can do is query ASH data relating to PSAPPSRV programs. If you have separate PSQRYSRV processes, you can analyse that separately too.

```
SELECT /*+LEADING(x h) USE_NL(h)*/
  h.sql_id,
  h.sql_plan_hash_value,
  SUM(10) ash_secs
FROM dba_hist_snapshot x
  , dba_hist_active_sess_history h
WHERE x.end_interval_time >= TO_DATE('201002010730','yyyymmddhh24mi')
AND x.begin_interval_time <= TO_DATE('201002010830','yyyymmddhh24mi')
AND h.sample_time BETWEEN TO_DATE('201002010730','yyyymmddhh24mi')
AND TO_DATE('201002010830','yyyymmddhh24mi')
AND h.snap_id = x.snap_id
AND h.dbid = x.dbid
AND h.instance_number = x.instance_number
AND h.module like 'PSAPPSRV'
GROUP BY h.sql_id, h.sql_plan_hash_value
ORDER BY ash_secs DESC;
```
At least most of the SQL in the on-line application uses bind variables (except for certain bits of dynamically generated code), so it does aggregate properly in the ASH data.

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>Hash Value</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>7hvaxp65s70qw</td>
<td>1051046890</td>
<td>1360</td>
</tr>
<tr>
<td>fdukyw87n6prc</td>
<td>313261966</td>
<td>760</td>
</tr>
<tr>
<td>8d56bz2qwx6y6j</td>
<td>2399544943</td>
<td>720</td>
</tr>
<tr>
<td>876mfmryd8yv7</td>
<td>156976114</td>
<td>710</td>
</tr>
<tr>
<td>bphpwrud1q83t</td>
<td>3575267335</td>
<td>690</td>
</tr>
</tbody>
</table>
XML Report

If you make use of XML reporting, usually to deliver PeopleSoft Queries then you find that they are all run through an Application Engine program called PSXPQRYRPT. You can use the PS_CDM_FILE_LIST table to work out the Report ID that was requested, and you can look at the report definition (PSXPRPTDEFN) to find the underlying query.

This query just reports run time for a report called XGF_WK_LATE. We haven’t added any ASH data yet.

```
SELECT r.prcsinstance, r.begindttm, d.report_defn_id, d.ds_type, d.ds_id, 
       (CAST(r.enddttm AS DATE) - CAST(r.begindttm AS DATE))*86400 secs
FROM sysadm.psprcsrqst r 
       , sysadm.ps_cdm_file_list f
       , sysadm.psxprptdefn d 
       , sysadm.psxpdatasrc s
WHERE r.prcsname = 'PSXPQRYRPT' 
    AND r.prcsinstance = f.prcsinstance 
    AND NOT f.cdm_file_type IN('AET','TRC','LOG') 
    AND d.report_defn_id = SUBSTR(f.filename,1,instr(f.filename,'.')-1) 
    AND d.report_defn_id = 'XGF_WK_LATE' 
    AND s.ds_type = d.ds_type 
    AND s.ds_id = d.ds_id 
    AND s.oprid = d.oprid 
    AND begindttm BETWEEN TO_DATE('201001200000','yyyymmddhh24mi') 
                  AND SYSDATE --TO_DATE('201001211600','yyyymmddhh24mi')
ORDER BY r.begindttm
```

<table>
<thead>
<tr>
<th>P.I. BEGINDTTM</th>
<th>Report ID</th>
<th>Type Data Source ID</th>
<th>SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1953197</td>
<td>19:56:56</td>
<td>20/01/2010 XGF_WK_LATE QRY XGF_WKLY_LATENESS_RPT</td>
<td>753</td>
</tr>
<tr>
<td>1956338</td>
<td>09:01:56</td>
<td>21/01/2010 XGF_WK_LATE QRY XGF_WKLY_LATENESS_RPT</td>
<td>19,283</td>
</tr>
<tr>
<td>1956805</td>
<td>09:50:08</td>
<td>21/01/2010 XGF_WK_LATE QRY XGF_WKLY_LATENESS_RPT</td>
<td>16,350</td>
</tr>
<tr>
<td>1956925</td>
<td>10:01:28</td>
<td>21/01/2010 XGF_WK_LATE QRY XGF_WKLY_LATENESS_RPT</td>
<td>15,654</td>
</tr>
</tbody>
</table>

Now I want to see what SQL Statements that were executed by those processes, and what were their execution plans.

```
SELECT /*+LEADING(r f d x h) USE_NL(h)*/
       r.prcsinstance 
       , h.sql_id 
       , h.sql_child_number 
       , h.sql_plan_hash_value 
       , (CAST(r.enddttm AS DATE) - CAST(r.begindttm AS DATE))*86400 exec_secs 
       , SUM(ash) ash_secs
FROM dba_hist_snapshot x 
       , dba_hist_active_sess_history h
       , sysadm.psprcsrqst r 
       , sysadm.ps_cdm_file_list f
       , sysadm.psxprptdefn d
WHERE s.end_interval_time between r.begindttm AND r.enddttm
```
One of the challenges of PeopleSoft Queries with Operator related row-level security is that a 
precate on the operator ID as added to the query, and the operator ID is a literal value not a 
bind variable. That means that if two different operators run the same query, they will 
generate different SQL_IDs.

```
SQL_ID djqf1zcypm5fm
----------------------
SELECT ...
FROM PS_TL_EXCEPTION A, PS_PERSONAL_DATA B, PS_PERALL_SEC_QRY B1,
     ...
WHERE B.EMPLID = B1.EMPLID AND B1.OPRID = '12345678'
```

This is rather perverse considering all the other parameters in a query are proper bind 
variables, so if a use runs the same query with different parameters that will usually have the 
same SQL_ID!
Most the SQL_IDs in this report are essentially the same query with different Operator IDs, and you can see that there are 4 different execution plans.

<table>
<thead>
<tr>
<th>P.I.</th>
<th>SQL_ID</th>
<th>SQL_PLAN_HASH_VALUE</th>
<th>EXEC_SECS</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>185</td>
<td>0uj7k70z1s76y</td>
<td>2239378934</td>
<td>619</td>
<td>210</td>
</tr>
<tr>
<td>185</td>
<td>0sd03jvun7us6</td>
<td>2239378934</td>
<td>336</td>
<td>20</td>
</tr>
<tr>
<td>185</td>
<td>22kn2sb7vttnp</td>
<td>2239378934</td>
<td>753</td>
<td>150</td>
</tr>
<tr>
<td>185</td>
<td>0kkjtywub2861</td>
<td>2602481067</td>
<td>19283</td>
<td>18550</td>
</tr>
<tr>
<td>185</td>
<td>998wf4g84dk8z</td>
<td>1041940423</td>
<td>19283</td>
<td>10</td>
</tr>
<tr>
<td>185</td>
<td>7c7dzavm70yku</td>
<td>2602481067</td>
<td>16350</td>
<td>15690</td>
</tr>
<tr>
<td>185</td>
<td>1knvx57drnz29</td>
<td>2602481067</td>
<td>15654</td>
<td>15010</td>
</tr>
<tr>
<td>185</td>
<td>a9mw8hjxfwczm</td>
<td>338220129</td>
<td>15654</td>
<td>10</td>
</tr>
<tr>
<td>185</td>
<td>9s2jct0jfwmfy</td>
<td>2602481067</td>
<td>15077</td>
<td>14430</td>
</tr>
<tr>
<td>185</td>
<td>9s2jct0jfwmfy</td>
<td>3265949623</td>
<td>15077</td>
<td>10</td>
</tr>
<tr>
<td>185</td>
<td>cwarnq7kv48d4</td>
<td>2602481067</td>
<td>14638</td>
<td>14000</td>
</tr>
<tr>
<td>185</td>
<td>9n93pl34xjbo</td>
<td>2602481067</td>
<td>13477</td>
<td>12980</td>
</tr>
<tr>
<td>185</td>
<td>9s2jct0jfwmfy</td>
<td>2602481067</td>
<td>9354</td>
<td>9140</td>
</tr>
<tr>
<td>185</td>
<td>1bd0fg0fvfsfyp</td>
<td>2602481067</td>
<td>9176</td>
<td>8950</td>
</tr>
<tr>
<td>185</td>
<td>1knvx57drnz29</td>
<td>2602481067</td>
<td>8903</td>
<td>8680</td>
</tr>
<tr>
<td>185</td>
<td>6uzhyw11wxxwn</td>
<td>2602481067</td>
<td>8025</td>
<td>7810</td>
</tr>
<tr>
<td>185</td>
<td>3ydv1rbx3yul</td>
<td>2602481067</td>
<td>7294</td>
<td>7100</td>
</tr>
<tr>
<td>185</td>
<td>bst3ytuxby0wm</td>
<td>481148914</td>
<td>7294</td>
<td>10</td>
</tr>
<tr>
<td>185</td>
<td>0yf3nx1tm4f18</td>
<td>2602481067</td>
<td>6084</td>
<td>5690</td>
</tr>
<tr>
<td>185</td>
<td>7gu27skrd5uuv</td>
<td>2602481067</td>
<td>5621</td>
<td>5230</td>
</tr>
<tr>
<td>185</td>
<td>6wxbk0rkgm08a</td>
<td>2602481067</td>
<td>5148</td>
<td>4550</td>
</tr>
<tr>
<td>185</td>
<td>c7btm765fcrjy</td>
<td>2602481067</td>
<td>4706</td>
<td>4100</td>
</tr>
<tr>
<td>185</td>
<td>ffjj75qcv9a3a</td>
<td>2602481067</td>
<td>4342</td>
<td>3740</td>
</tr>
<tr>
<td>185</td>
<td>5c2x8b7ur4hjz</td>
<td>2602481067</td>
<td>6361</td>
<td>5810</td>
</tr>
<tr>
<td>185</td>
<td>46smbgcfcrb8d</td>
<td>2602481067</td>
<td>5766</td>
<td>5210</td>
</tr>
</tbody>
</table>

This is one of those situations where it can be effective to just GROUP BY SQL_PLAN_HASH_VALUE and work out which execution plan has the most execution plan. That is might be an undesirable plan and you might want to work out why Oracle is choosing it, and consider what you are going to do about it.
Other Techniques

Monitoring Progress of Processes in Real Time

```sql
SELECT /*+LEADING(r)*/
  r.prcsinstance,
  h.sql_id,
  h.sql_child_number,
  h.sql_plan_hash_value,
  (NVL(r.enddttm,SYSDATE)-r.begindttm)*86400 exec_secs,
  SUM(1) ash_secs,
  max(sample_time) max_sample_time
FROM v$active_Session_history h,
sysadm.psprcsrqst r
WHERE h.sample_time BETWEEN r.begindttm AND NVL(r.enddttm,SYSDATE)
AND h.module = r.prcsname
AND h.action LIKE 'PI='||r.prcsinstance||'%' AND r.prcsinstance = 1561519
GROUP BY r.prcsinstance, r.prcsname, r.begindttm, r.enddttm, h.sql_id,
  h.sql_plan_hash_value, h.sql_child_number
ORDER BY max_sample_time desc
```

This was run on a fairly quiet database and the ASH buffer has held 5 hours of data.

Note that Statement 9yj020x2762a9 has clocked 17688 seconds at 4.24pm.

<table>
<thead>
<tr>
<th>Process</th>
<th>Child</th>
<th>SQL Plan</th>
<th>Exec</th>
<th>ASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance SQL_ID</td>
<td>No. Hash Value</td>
<td>Secs</td>
<td>Secs Last Running</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>9yj020x2762a9</td>
<td>18366</td>
<td>17688 19-FEB-10 04:24.41.392 PM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>9yj020x2762a9</td>
<td>18366</td>
<td>1 19-FEB-10 11:26.29.096 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>b5rHo0cxK2uh</td>
<td>18366</td>
<td>1 19-FEB-10 11:26.28.085 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>5vdhh2m8skh86</td>
<td>18366</td>
<td>1 19-FEB-10 11:26.27.075 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>gyuqlarbJ7ykx</td>
<td>18366</td>
<td>1 19-FEB-10 11:26.26.065 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>9yj020x2762a9</td>
<td>18366</td>
<td>1 19-FEB-10 11:26.25.055 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>Sjhknmxw7kJ2</td>
<td>18366</td>
<td>1 19-FEB-10 11:26.24.043 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>5pe2Gzn5Ohmkk</td>
<td>18366</td>
<td>1 19-FEB-10 11:26.23.033 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>5sog9fgf26wkb</td>
<td>18366</td>
<td>1 19-FEB-10 11:26.22.035 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>gpwr/38mg6ih</td>
<td>18366</td>
<td>422 10-FEB-10 11:26.21.014 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>gpwr/38mg6ih</td>
<td>18366</td>
<td>1 10-FEB-10 11:19.13.931 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>nbmbm51p0Sgq</td>
<td>18366</td>
<td>19-FEB-10 11:19.12.916 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>defwaHbog2m1j</td>
<td>18366</td>
<td>14 10-FEB-10 11:19.11.912 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>d0wud190ipbe4</td>
<td>18366</td>
<td>9 10-FEB-10 11:18.57.771 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>gpuzsubjw0sv54</td>
<td>18366</td>
<td>10 10-FEB-10 11:18.48.679 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>cbppam9ph5bu8</td>
<td>18366</td>
<td>1 10-FEB-10 11:18.38.571 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>cbppam9ph5bu8</td>
<td>18366</td>
<td>10 10-FEB-10 11:18.37.551 AM</td>
<td></td>
</tr>
<tr>
<td>1561509</td>
<td>3csn2zvbubjde3</td>
<td>18366</td>
<td>1 10-FEB-10 11:18.36.541 AM</td>
<td></td>
</tr>
</tbody>
</table>

But later not that the timings for statement 9yj020x2762a9, the timing has gone down. So part of the ASH data has been purged.

<table>
<thead>
<tr>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance SQL_ID</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>--------------</td>
</tr>
</tbody>
</table>

© Go-Faster Consultancy Ltd. • CONFIDENTIAL

PRACTICAL USE OF ORACLE ACTIVE SESSION HISTORY 23
And if I want to look at an execution plan

```sql
SELECT DISTINCT 'SELECT * FROM table(dbms_xplan.display_cursor(''|sql_id'||',''|sql_child_number||',''ADVANCED''));' FROM ( 
  
) 
To generate this command

```sql
SELECT * FROM table(dbms_xplan.display_cursor('9yj020v2762a9',0,'ADVANCED'));
```
Developers not Using Bind Variables

This is what happens when developers do not use Bind Variables. It happens in PeopleSoft Application Engine programs if developers do not use the ReUse statement option, which is not enabled by default. It can also happen when a process uses dynamically generated SQL.

I started with my standard query for analysing a named process.

```
SELECT /*+LEADING(r x h) USE_NL(h)*/
  r.prcsinstance,
  h.sql_id,
  h.sql_plan_hash_value,
  (CAST(r.enddttm AS DATE)-CAST(r.begindttm AS DATE))\*86400
  exec_secs,
  SUM(10) ash_secs
FROM dba_hist_snapshot x,
    dba_hist_active_sess_history h,
    sysadm.psprcsrqst r
WHERE x.end_interval_time >= r.enddttm
  And x.begin_interval_time <= r.enddttm
  AND h.sample_time BETWEEN r.begindttm AND r.enddttm
  AND h.snap_id = x.snap_id
  AND h.dbid = x.dbid
  AND h.instance_number = x.instance_number
  AND h.module = r.prcsname
  AND h.action LIKE 'PI'||r.prcsinstance||'%'
  AND r.prcsname = 'XXES036'
GROUP BY r.prcsinstance, r.prcsname, r.begindttm, r.enddttm
  , h.sql_id, h.sql_plan_hash_value
ORDER BY ash_secs DESC
```
I got lots of SQL statements with the same execution plan. That is going to happen when the statements are very similar, and/or when the only differences are the values of literals in the SQL.

SQL*Trace profiled TKPROF has the same problem. This is a challenge that I face very frequently, and ORASRP is a better profiling tool.

<table>
<thead>
<tr>
<th>PRCSINSTANCE</th>
<th>SQL_ID</th>
<th>SQL_PLAN_HASH_VALUE</th>
<th>EXEC_SECS</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>50002824</td>
<td>2ybtak62vmx58</td>
<td>2262951047</td>
<td>10306</td>
<td>50</td>
</tr>
<tr>
<td>50002824</td>
<td>ck3av6cnqwfc</td>
<td>2262951047</td>
<td>10306</td>
<td>20</td>
</tr>
<tr>
<td>50002824</td>
<td>gyys6kd9fqn7u</td>
<td>2262951047</td>
<td>10306</td>
<td>20</td>
</tr>
<tr>
<td>50002824</td>
<td>7ymcbn6q8utj8</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>9quad2n3qq7nzh</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>6pxvens97m1fu</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>5ngqj5zg8vbz8</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>9zp6ndtfn66b</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>15kfs3c3005xm</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>4qhvpge7cq2t</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>23yc8dcz9y4yj</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>bn8xcrzsv2hpr</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>9g6k9dnrjap08</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>1art8dhbvpwrt</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>6gqj337xnr5y4</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>77rx2cctnzwlgcf</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>5p5tvh4wfp1ur</td>
<td>2262951047</td>
<td>10306</td>
<td>10</td>
</tr>
</tbody>
</table>

So now, I will remove SQL ID FROM my query, and just GROUP BY SQL Plan Hash Value

```
SELECT /*+LEADING(r x h) USE_NL(h)*/
    r.prcsinstance
 , h.sql_plan_hash_value
 , (CAST(r.enddttm AS DATE) - CAST(r.begindttm AS DATE))*86400
     exec_secs
 , SUM(10) ash_secs
FROM dba_hist_snapshot x
 , dba_hist_active_sess_history h
 , sysadm.psprcsrqst r
WHERE x.end_interval_time >= r.enddttm
And x.begin_interval_time <= r.enddttm
AND h.sample_time BETWEEN r.begindttm AND r.enddttm
and h.snap_id = x.snap_id
AND h.dbid = x.dbid
AND h.instance_number = x.instance_number
AND h.module = r.prcsname
AND h.action LIKE 'PI='||r.prcsinstance||'%'
AND r.prcsname = 'XXES036'
GROUP BY r.prcsinstance, r.prcsname, r.begindttm, r.enddttm
 , h.sql_plan_hash_value
ORDER BY ash_secs DESC
```
Now, most of my time is in one execution plan.

<table>
<thead>
<tr>
<th>PRCINSTANCE</th>
<th>SQL_PLAN_HASH_VALUE</th>
<th>EXEC_SECS</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>50002824</td>
<td>2262951047</td>
<td>10306</td>
<td>2300</td>
</tr>
<tr>
<td>50002824</td>
<td>0</td>
<td>10306</td>
<td>60</td>
</tr>
<tr>
<td>50002824</td>
<td>3085938243</td>
<td>10306</td>
<td>20</td>
</tr>
<tr>
<td>50002824</td>
<td>563410926</td>
<td>10306</td>
<td>10</td>
</tr>
<tr>
<td>50002824</td>
<td>1068931976</td>
<td>10306</td>
<td>10</td>
</tr>
</tbody>
</table>

Now, I need to look at at least one of those SQL statements with that plan:

```sql
SELECT * FROM dba_xplan.display_awr('9vnan5kshla7', 2262951047, NULL, 'ADVANCED');
```
This query groups the SQL by SQL_ID and SQL PLAN hash plan, but reports the total amount of time for each plan in ASH, it ranks the statements within each plan by the amount of time recorded against statements captured by AWR.

```
SELECT 'SELECT * FROM table(dbms_xplan.display_awr('''||sql_id||''','||sql_plan_hash_value||',NULL,''ADVANCED''))/*'||tot_ash_secs||','||tot_awr_secs||'*/;' FROM (
SELECT  ROW_NUMBER()over (PARTITION BY x.sql_plan_hash_value order by x.awr_secs desc) as ranking ,
        x.sql_id, x.sql_plan_hash_value ,
        SUM(x.ash_secs) over (PARTITION BY x.sql_plan_hash_value) tot_ash_secs ,
        SUM(x.awr_secs) over (PARTITION BY x.sql_plan_hash_value) tot_awr_secs ,
        COUNT(distinct sql_id) over (PARTITION BY x.sql_plan_hash_value) sql_ids
FROM (
SELECT  h.sql_id ,
        h.sql_plan_hash_value ,
        SUM(10) ash_secs ,
        10*count(t.sql_id) awr_secs
FROM  DBA_HIST_SNAPSHOT x ,
      DBA_HIST_ACTIVE_SESS_HISTORY h
LEFT OUTER JOIN dba_hist_sqltext t16
ON t.sql_id = h.sql_id
WHERE  x.end_interval_time >= TRUNC(SYSDATE,'mm')
AND  x.begin_interval_time <= TRUNC(SYSDATE,'mm')+7
AND  h.sample_time BETWEEN TRUNC(SYSDATE,'mm') AND TRUNC(SYSDATE,'mm')+7
and  h.snap_id = x.snap_id
and  h.dbid = x.dbid
and  h.instance_number = x.instance_number
and  h.module = h.program
GROUP BY h.sql_id, h.sql_plan_hash_value ) x
) y
WHERE  y.ranking = 1
AND  tot_ash_secs > 900
ORDER BY tot_ash_secs desc, ranking
```

<table>
<thead>
<tr>
<th>RANKING</th>
<th>SQL_ID</th>
<th>SQL_PLAN_HASH_VALUE</th>
<th>TOT_ASH_SECS</th>
<th>TOT_AWR_SECS</th>
<th>SQL_IDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8mkvraydrxycn</td>
<td>0</td>
<td>38270</td>
<td>480</td>
<td>74</td>
</tr>
<tr>
<td>1</td>
<td>027qsfj7n71cy</td>
<td>1499159071</td>
<td>4230</td>
<td>4230</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>cxwz9m3auk4y</td>
<td>1898065720</td>
<td>4190</td>
<td>4190</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>95139bu2ruxc</td>
<td>2044891559</td>
<td>3590</td>
<td>3590</td>
<td>19</td>
</tr>
</tbody>
</table>

16 By outer joining the ASH data to DBA_HIST_SQLTEXT we can check whether the statement was captured by AWR.

17 The first statement is a special case. There is no plan – probably because it’s a PL/SQL function. There were 74 statements, but in reality they will all be totally different.

18 One SQL, one plan, this is a shareable SQL_ID, or it did just execute once.

19 This is many statements with the same plan, at least 198.
How Many Executions?

Oracle 10g

In 10g you cannot directly determine the number of executions from ASH data. Here is an example from OEM. This truncate statement is consuming a lot of time. But it isn’t a single execution. It is a huge number of small executions.

Oracle 11g

However, in 11g there is a new column sql_exec_id in the v$active_session_history and dba_hist_active_sess_history. Each execution of a statement gets a unique execution ID. Counting the number of distinct execution IDs determines the number of executions.
and h.dbid = x.dbid
and h.instance_number = x.instance_number
and h.user_id != 0 /*omit oracle shadow processes*/
group by h.program, h.sql_id, h.sql_plan_hash_value
order by ash_secs desc
/

So I can see that these statements burnt about 3020 and 320 seconds. This query has counted 297 and 32 executions respectively.

<table>
<thead>
<tr>
<th>SQL Plan</th>
<th>ASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM</td>
<td>SQL_ID</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>t_async.exe</td>
<td>7q90ra0vmd9xx</td>
</tr>
<tr>
<td>t_async.exe</td>
<td>6mw25bgbh1stj</td>
</tr>
</tbody>
</table>

However, remember that because this query was based on `dba_hist_active_sess_history` there is one sample per 10 seconds, so each row is counted as 10 seconds. The number of executions can never be calculated as being greater than the number of ASH records. So when the number of executions is close to or the same as the number of ASH records it is likely that there are actually many more executions that are recorded here.
How Many Transactions?

You cannot tell how many times a statement has executed in 10g. This becomes possible in 11g. However, you do have the transaction ID recorded in the ASH data, but only if the statement is a part of a transaction.

```
Column last_sample_time format a25
Column first_sample_time format a25
select /*+leading(r h) use_n(n0)*/
    r.prcsinstance
    --
    , h.sql_id
    --
    , h.sql_child_number
    , h.xid
    , h.sql_plan_hash_value
    , (NVL(r.enddttm,SYSDATE)-r.begindttm)*86400 exec_secs
    , SUM(1) ash_secs
    , min(sample_time) first_sample_time
    , max(sample_time) last_sample_time
from gv$active_session_history h
    , sysadm.psprcsrqst r
where h.sample_time between r.begindttm and NVL(r.enddttm,SYSDATE)
    and h.module = r.prcsname
    and h.action like 'PI'||r.prcsinstance||'%' 
    and r.prcsinstance = 10026580
    and h.sql_id = 'dungu07axr0z5'
group by r.prcsinstance, r.prcsname, r.begindttm, r.enddttm
    , h.sql_id, h.sql_plan_hash_value
    , h.sql_child_number
    , h.xid
    --
    , h.program
having sum(1) > (NVL(r.enddttm,SYSDATE)-r.begindttm)*86400/1000
order by last_sample_time, ash_secs desc
```

One statement executed 4 at least times in the same process, with the same process, but as a part of 3 different transactions. Note that the last entry is not part of any transaction.
When Did the Transaction Start

Here is the output for a very similar query at a different time. On these occasions the SQL starts without a transaction ID, and acquires one later.

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>Hash Value XID</th>
<th>ASH Exec</th>
<th>First Running</th>
<th>Last Running</th>
</tr>
</thead>
<tbody>
<tr>
<td>7uj72ad03k13k 3087414546</td>
<td>82</td>
<td>1124-28-APR-10 04:42.48.662 PM</td>
<td>28-APR-10 04:44.10.662 PM</td>
<td></td>
</tr>
<tr>
<td>7uj72ad03k13k 3087414546 000A014000446Cd</td>
<td>1</td>
<td>1124-28-APR-10 04:44.11.672 PM</td>
<td>28-APR-10 04:44.11</td>
<td></td>
</tr>
<tr>
<td>1ng9qkc0zspkh 3423396304</td>
<td>104</td>
<td>1124-28-APR-10 04:44.12.682 PM</td>
<td>28-APR-10 04:45.56.961 PM</td>
<td></td>
</tr>
</tbody>
</table>

The statements involved are monolithic deletes. My interpretation is that it takes a while for these queries to identify rows to be deleted, and it is not until the first row is deleted that a transaction is initiated. It is entirely plausible that, depending upon data, statements could run for a while before finding some data to delete.

```
SQL_ID  7uj72ad03k13k, child number 0
-------------------------------------------------------------
Plan hash value: 3087414546
```

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DELETE STATEMENT</td>
<td>PS_GP_RSLT_ERND_DED</td>
<td></td>
<td></td>
<td>5 (100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DELETE</td>
<td>PS_GP_RSLT_ERND_DED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS SEMI</td>
<td></td>
<td>1</td>
<td>172</td>
<td>5 (20)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>HASH JOIN SEMI</td>
<td></td>
<td>1</td>
<td>131</td>
<td>5 (20)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PARTITION RANGE ITERATOR</td>
<td></td>
<td>2</td>
<td>164</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>*</td>
<td>INDEX RANGE SCAN</td>
<td>PS_GP_RSLT_ERND_DED</td>
<td>2</td>
<td>164</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>TABLE ACCESS FULL</td>
<td>PS_GP_PYE_RCLC_WRK</td>
<td>15</td>
<td>735</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>PARTITION RANGE ITERATOR</td>
<td></td>
<td>1</td>
<td>41</td>
<td>0 (0)</td>
<td></td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>*</td>
<td>INDEX RANGE SCAN</td>
<td>PS_GP_GRP_LIST_RUN</td>
<td>1</td>
<td>41</td>
<td>0 (0)</td>
<td></td>
<td>KEY</td>
<td>KEY</td>
</tr>
</tbody>
</table>
```
PLAN_TABLE_OUTPUT

SQL_ID ing9qkc0zspkh, child number 0


Plan hash value: 3423396304

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DELETE STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>5 (100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DELETE</td>
<td>PS_GP_RSLT_PIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS SEMI</td>
<td></td>
<td>1</td>
<td>170</td>
<td>5 (20)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 4</td>
<td>HASH JOIN SEMI</td>
<td></td>
<td>1</td>
<td>129</td>
<td>5 (20)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PARTITION RANGE ITERATOR</td>
<td></td>
<td>31</td>
<td>2480</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>* 6</td>
<td>PARTITION LIST SINGLE</td>
<td></td>
<td>31</td>
<td>2480</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>* 7</td>
<td>INDEX RANGE SCAN</td>
<td>PS_GP_RSLT_PIN</td>
<td>31</td>
<td>2480</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td></td>
</tr>
<tr>
<td>* 8</td>
<td>TABLE ACCESS FULL</td>
<td>PS_GP_PYE_RCLC_WRK</td>
<td>15</td>
<td>735</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>PARTITION RANGE ITERATOR</td>
<td></td>
<td>1</td>
<td>41</td>
<td>0 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>* 10</td>
<td>INDEX RANGE SCAN</td>
<td>PS_GP_GRP_LIST_RUN</td>
<td>1</td>
<td>41</td>
<td>0 (0)</td>
<td></td>
<td>KEY</td>
<td>KEY</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

4 - access("RW"."CAL_ID"="PS_GP_RSLT_PIN"."CAL_ID" AND "RW"."CAL_RUN_ID"="PS_GP_RSLT_PIN"."CAL_RUN_ID" AND "RW"."GP_PAYGROUP"="PS_GP_RSLT_PIN"."GP_PAYGROUP" AND "RW"."EMPLID"="PS_GP_RSLT_PIN"."EMPLID" AND "RW"."EMPL_RCD"="PS_GP_RSLT_PIN"."EMPL_RCD")
7 - access("EMPLID"=:1 AND "PS_GP_RSLT_PIN"."CAL_RUN_ID"=:8 AND "EMPLID"=:2)
filter(("CAL_RUN_ID"=:3 AND "PS_GP_RSLT_PIN"."CAL_RUN_ID"=:8 AND "PS_GP_RSLT_PIN"."EMPLID"=:
"PS_GP_RSLT_PIN"."EMPLID")
8 - filter(("RW"."CAL_RUN_ID"=:8 AND "RW"."CAL_RUN_ID"=:3 AND "RW"."EMPLID"=:6 AND "RW"."EMPLID" AND "RW"."EMPL_RCD"=:
1 AND "RW"."EMPL_RCD"=:
7))
10 - access("RUN_CNTL_ID"=4 AND "OPRID"=5 AND "EMPLID"="EMPLID")
filter(("EMPLID"=:1 AND "EMPLID"=:
and "EMPLID"=:
Note
-----
dynamic sampling used for this statement
Single Wait Event

Earlier we looked at an example of on-line activity, and I used the PeopleSoft Performance Monitor to identify a period when degradation in performance was noticed (see Application Engine from PeopleTools 8.52 on page 16). I want to look at the behaviour of the database in the same period.

Oracle Enterprise Manager will give you a graphical representation of the ASH data. I often graph wait event data collected by AWR in Excel\(^{20}\).

According to AWR, we have as many of 12 concurrent sessions waiting on this event.

<table>
<thead>
<tr>
<th>Time Waited</th>
<th>Event Name</th>
<th>Wait Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>db file sequential read</td>
<td>enq: TX - row lock contention</td>
</tr>
<tr>
<td>Snapshot Start Time</td>
<td>User I/O</td>
<td>Application</td>
</tr>
<tr>
<td>Mon 1.2.10 06:00</td>
<td>2,329.153</td>
<td>16,822</td>
</tr>
<tr>
<td>Mon 1.2.10 06:15</td>
<td>3,323.358</td>
<td>174,772</td>
</tr>
<tr>
<td>Mon 1.2.10 06:30</td>
<td>4,397.850</td>
<td>41,172</td>
</tr>
<tr>
<td>Mon 1.2.10 06:45</td>
<td>5,037.319</td>
<td>1,595</td>
</tr>
<tr>
<td>Mon 1.2.10 07:00</td>
<td>6,451.124</td>
<td>72,692</td>
</tr>
<tr>
<td>Mon 1.2.10 07:15</td>
<td>8,226.684</td>
<td>205,765</td>
</tr>
<tr>
<td>Mon 1.2.10 07:30</td>
<td>9,274.853</td>
<td>196,430</td>
</tr>
<tr>
<td>Mon 1.2.10 07:45</td>
<td>9,315.794</td>
<td>99,286</td>
</tr>
<tr>
<td>Mon 1.2.10 08:00</td>
<td>10,267.237</td>
<td>233,664</td>
</tr>
<tr>
<td>Mon 1.2.10 08:15</td>
<td>9,084.140</td>
<td>607,859</td>
</tr>
<tr>
<td>Mon 1.2.10 08:30</td>
<td>8,404.167</td>
<td>845,342</td>
</tr>
<tr>
<td>Mon 1.2.10 08:45</td>
<td>11,145.149</td>
<td>746,139</td>
</tr>
<tr>
<td>Mon 1.2.10 09:00</td>
<td>10,097.621</td>
<td>352,595</td>
</tr>
<tr>
<td>Mon 1.2.10 09:15</td>
<td>7,625.934</td>
<td>298,300</td>
</tr>
<tr>
<td>Mon 1.2.10 09:30</td>
<td>8,876.006</td>
<td>896,529</td>
</tr>
<tr>
<td>Grand Total</td>
<td>113,856.388</td>
<td>4,788,961</td>
</tr>
</tbody>
</table>

\(^{20}\) There are various advantages to this approach, see [http://blog.go-faster.co.uk/2008/12/graphing-awr-data-in-excel.html](http://blog.go-faster.co.uk/2008/12/graphing-awr-data-in-excel.html)
A simple variant on the usual query, and we can look for the statement with the highest I/O overhead.

```sql
SELECT /*+LEADING(x h) USE_NL(h)*/
  h.sql_id,
  h.sql_plan_hash_value,
  SUM(10) ash_secs
FROM dba_hist_snapshot x,
  dba_hist_active_sess_history h
WHERE x.end_interval_time <= TO_DATE('201002010830','yyyymmddhh24mi')
AND x.begin_interval_time >= TO_DATE('201002010730','yyyymmddhh24mi')
AND h.sample_time BETWEEN TO_DATE('201001261100','yyyymmddhh24mi')
  AND TO_DATE('201001261300','yyyymmddhh24mi')
AND h.snap_id = x.snap_id
AND h.dbid = x.dbid
AND h.instance_number = x.instance_number
AND h.event = 'db file sequential read'
GROUP BY h.sql_id, h.sql_plan_hash_value
ORDER BY ash_secs DESC
/```

So, here at the top statements

<table>
<thead>
<tr>
<th>SQL Plan</th>
<th>SQL_ID</th>
<th>Hash Value</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>--------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>90pp7bcnmz68r</td>
<td>2961772154</td>
<td>2490</td>
<td></td>
</tr>
<tr>
<td>81gz2rtaba8n</td>
<td>1919624473</td>
<td>2450</td>
<td></td>
</tr>
<tr>
<td>7hvaxp65s70qw</td>
<td>1051046890</td>
<td>1320</td>
<td></td>
</tr>
<tr>
<td>7fk8raq16ch0u</td>
<td>3950826368</td>
<td>890</td>
<td></td>
</tr>
<tr>
<td>9dzpwkff7zyccg</td>
<td>2020614776</td>
<td>840</td>
<td></td>
</tr>
</tbody>
</table>

And just for a laugh, this is the query

```sql
SELECT DISTINCT
  A.GP_PAYGROUP, M.XGF_REGION_NAME, M.XGF_AREA_NAME, A.LLOCATION, B.DESCRIPTION, D.DESCRIPTION, A.EMPLID,
  C.LAST_NAME, C.FIRST_NAME, TO_CHAR(A.TERMINATION_DT,'YYYY-MM-DD'), TO_CHAR(A.LAST_DATE_WORKED,'YYYY-MM-DD'),
  G.PIN_NET_VAL, B.SETID, B.LLOCATION, TO_CHAR(B.EFFDT,'YYYY-MM-DD'), D.SETID, D.DEPID, TO_CHAR(D.EFFDT,'YYYY-MM-DD')
FROM PS_JOB A,
  PS_XGF_JOB_QRY A1,
  PS_LOCATION_TBL B,
  PS_PERSONAL_DATA C,
  PS_PERALL_SEC_QRY C1,
  PS_DEPT_TBL D,
  PS_XGF_TREE_RPL_VW M,
  PS_GP_PYE_SEG_STATS G,
  PS_EMPLMT_SRCH_QRY C2,
  PS_GP_CAL_RUN_DTL F
```

Plan hash value: 2961772154

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%)</th>
<th>Time</th>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td>2139 (100)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT UNIQUE</td>
<td></td>
<td>1</td>
<td>578</td>
<td>2138 (2)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_SJ7_PERSON</td>
<td>2</td>
<td>72</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>578</td>
<td>2044 (1)</td>
<td>00:00:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>542</td>
<td>2040 (1)</td>
<td>00:00:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>509</td>
<td>2035 (1)</td>
<td>00:00:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>485</td>
<td>2034 (1)</td>
<td>00:00:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>429</td>
<td>2003 (1)</td>
<td>00:00:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>395</td>
<td>2001 (1)</td>
<td>00:00:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>365</td>
<td>1999 (1)</td>
<td>00:00:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>HASH JOIN</td>
<td></td>
<td>65</td>
<td>19045</td>
<td>1868 (1)</td>
<td>00:00:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>TABLE ACCESS FULL</td>
<td>PS_GP_CAL_RUN_DTL</td>
<td>48</td>
<td>3168</td>
<td>7 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>TABLE ACCESS BY LOCAL INDEX ROWID</td>
<td>PS_GP_PYE_SEG_STAT</td>
<td>18</td>
<td>900</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td>8376</td>
<td>1856K</td>
<td>1859 (1)</td>
<td>00:00:03</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td>474</td>
<td>8398</td>
<td>1107 (1)</td>
<td>00:00:02</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td>479</td>
<td>67539</td>
<td>35 (0)</td>
<td>00:00:01</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td>6</td>
<td>588</td>
<td>11 (0)</td>
<td>00:00:01</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td>1</td>
<td>72</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td>1</td>
<td>48</td>
<td>3 (0)</td>
<td>00:00:01</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PSOPRDEFN</td>
<td>1</td>
<td>24</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>INDEX UNIQUE SCAN</td>
<td>PSOPRDEFN</td>
<td>1</td>
<td>1</td>
<td>0 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PSOPRDEFN</td>
<td>1</td>
<td>24</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>INDEX UNIQUE SCAN</td>
<td>PSOPRDEFN</td>
<td>1</td>
<td>0 (0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PSOPRDEFN</td>
<td>1</td>
<td>24</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>INDEX UNIQUE SCAN</td>
<td>PSOPRDEFN</td>
<td>1</td>
<td>0 (0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_SJT_DPR_CLS</td>
<td>6</td>
<td>156</td>
<td>7 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>INDEX RANGE SCAN</td>
<td>PS_SJT_DPR_CLS</td>
<td>6</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>PARTITION LIST SINGLE</td>
<td></td>
<td>83</td>
<td>3569</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>INDEX RANGE SCAN</td>
<td>PS_SJT_CLASS_ALL</td>
<td>83</td>
<td>3569</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_SJT_PERSON</td>
<td>1</td>
<td>36</td>
<td>3 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>INDEX RANGE SCAN</td>
<td>PS_SJT_PERSON</td>
<td>1</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>PARTITION RANGE ITERATOR</td>
<td></td>
<td>31</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>INDEX RANGE SCAN</td>
<td>PS_GP_PYE_SEG_STAT</td>
<td>11</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>PARTITION RANGE ITERATOR</td>
<td></td>
<td>1</td>
<td>72</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>35</td>
<td>TABLE ACCESS BY LOCAL INDEX ROWID</td>
<td>PS_JOB</td>
<td>1</td>
<td>72</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>36</td>
<td>INDEX RANGE SCAN</td>
<td>PSAJOB</td>
<td>1</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>PARTITION RANGE SINGLE</td>
<td></td>
<td>1</td>
<td>20</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>39</td>
<td>INDEX RANGE SCAN</td>
<td>PSAJOB</td>
<td>1</td>
<td>20</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>40</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>PARTITION RANGE SINGLE</td>
<td></td>
<td>1</td>
<td>23</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>42</td>
<td>INDEX RANGE SCAN</td>
<td>PSAJOB</td>
<td>1</td>
<td>23</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>43</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_LOCATION_TBL</td>
<td>1</td>
<td>30</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>INDEX RANGE SCAN</td>
<td>PS_LOCATION_TBL</td>
<td>1</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>INDEX RANGE SCAN</td>
<td>PS_LOCATION_TBL</td>
<td>1</td>
<td>19</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_DEPT_TBL</td>
<td>1</td>
<td>34</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>INDEX RANGE SCAN</td>
<td>PS_DEPT_TBL</td>
<td>1</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>INDEX RANGE SCAN</td>
<td>PS_DEPT_TBL</td>
<td>1</td>
<td>21</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_KGF_TREE</td>
<td>1</td>
<td>56</td>
<td>31 (4)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>INDEX RANGE SCAN</td>
<td>PS_KGF_TREE</td>
<td>1</td>
<td>30 (4)</td>
<td>00:01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>INDEX RANGE SCAN</td>
<td>PS_KGF_TREE</td>
<td>4150</td>
<td>113K</td>
<td>33 (10)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_PERSONAL_DATA</td>
<td>1</td>
<td>24</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>INDEX UNIQUE SCAN</td>
<td>PS_PERSONAL_DATA</td>
<td>1</td>
<td>0 (0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_SYT_PERSON</td>
<td>5</td>
<td>165</td>
<td>5 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>INDEX RANGE SCAN</td>
<td>PSA57_PERSON</td>
<td>5</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>INDEX RANGE SCAN</td>
<td>PSA57_PERSON</td>
<td>3</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>TABLE ACCESS FULL</td>
<td>PS_GP_CAL_RUN_DTL</td>
<td>14</td>
<td>280</td>
<td>7 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>TABLE ACCESS FULL</td>
<td>PS_GP_CAL_RUN_DTL</td>
<td>16</td>
<td>304</td>
<td>7 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>TABLE ACCESS FULL</td>
<td>PS_GP_CAL_RUN_DTL</td>
<td>14</td>
<td>280</td>
<td>7 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>69</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>PARTITION LIST SINGLE</td>
<td></td>
<td>1</td>
<td>43</td>
<td>3 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>69</td>
<td>INDEX RANGE SCAN</td>
<td>PSA57_CLASS_ALL</td>
<td>1</td>
<td>43</td>
<td>3 (0)</td>
<td>00:00:01</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>70</td>
<td>INDEX RANGE SCAN</td>
<td>PSA57FromClass</td>
<td>1</td>
<td>26</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>60</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>PARTITION LIST SINGLE</td>
<td></td>
<td>1</td>
<td>34</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>73</td>
<td>INDEX RANGE SCAN</td>
<td>PSA57_CLASS_ALL</td>
<td>1</td>
<td>34</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>74</td>
<td>INDEX RANGE SCAN</td>
<td>PSA57FromClass</td>
<td>1</td>
<td>26</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>COUNT STOPKEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>69</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>PARTITION LIST SINGLE</td>
<td></td>
<td>1</td>
<td>43</td>
<td>3 (0)</td>
<td>00:00:01</td>
<td>KEY</td>
<td>KEY</td>
</tr>
<tr>
<td>79</td>
<td>INDEX RANGE SCAN</td>
<td>PSA57_CLASS_ALL</td>
<td>1</td>
<td>43</td>
<td>3 (0)</td>
<td>00:00:01</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>80</td>
<td>INDEX RANGE SCAN</td>
<td>PSA57FromClass</td>
<td>1</td>
<td>26</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

©Go-Faster Consultancy Ltd. • Confidential

Practical Use of Oracle Active Session History 37
What Kind of Single Block Read

I created a temporary working storage table with a classification for each tablespace. Here my classification is by object type in the tablespace. This is relatively easy if you have a reasonable tablespace naming convention.

```sql
drop table dmk_data_files
/
create table dmk_data_files as
SELECT tablespace_name, file_id,
CASE
  WHEN f.tablespace_name LIKE 'SYS%' THEN 'SYSTEM'
  WHEN f.tablespace_name LIKE 'UNDO%' THEN 'UNDO'
  WHEN f.tablespace_name LIKE '%IX%' THEN 'INDEX'
  ELSE 'TABLE'
END as tablespace_type
FROM dba_data_files f
ORDER BY tablespace_name
/
create unique index dmk_data_files on dmk_data_files(file_id)
/
```

I recommend that you do not work directly with DBA_DATA_FILES, because the resulting query will be slow. Instead, build a working storage table.
When ASH reports a wait on file I/O it also logs the object, file and block numbers. Although, beware, because the values may not have been cleared out FROM the previous sample.

So you know which database, and hence which tablespaces was accessed.

It’s a simple matter work out how much time was spent writing to which type of tablespace

```sql
SELECT /*+LEADING(x h) USE_NL(h f)*/
    f.tablespace_type
,    SUM(10) ash_secs
FROM   dba_hist_snapshot x
,       dba_hist_active_sess_history h
,       dmk_data_files f
WHERE  x.end_interval_time <= TO_DATE('201002161300','yyyymmddhh24mi')
AND    x.begin_interval_time >= TO_DATE('201002161100','yyyymmddhh24mi')
AND    h.sample_time BETWEEN TO_DATE('201001261100','yyyymmddhh24mi')
     AND TO_DATE('201001261300','yyyymmddhh24mi')
and    h.snap_id = x.snap_id
AND    h.dbid = x.dbid
AND    h.instance_number = x.instance_number
AND    h.event LIKE 'db file%'
AND    h.p1text = 'file#'
and    h.p2text = 'block#'
AND    h.event IS NOT NULL
AND    f.file_id = h.p1
GROUP BY f.tablespace_type
ORDER BY ash_secs DESC
/
```

Here, we can see we are spending more time on index reads that table reads, and very little on the undo tablespace, so there is not too much work to maintain read consistency occurring.

<table>
<thead>
<tr>
<th>TABLES</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX</td>
<td>30860</td>
</tr>
<tr>
<td>TABLE</td>
<td>26970</td>
</tr>
<tr>
<td>UNDO</td>
<td>1370</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>490</td>
</tr>
</tbody>
</table>

Of course, you could classify your tablespaces differently. You might have different applications all in one database. You might want to know how much of the load comes FROM which application.

I suppose you could look go down to each individual object being accessed, but that will be more involved, and I haven’t tried that.
Blocking Lock Analysis

This graph is derived from AWR data\(^{21}\), and it shows a period of time when a system exhibited a lot of time lost to row level wait. We lost 13 hours of user time in the two-hour period from 11am to 1pm.

\[
\begin{align*}
\text{SELECT} & \quad /*+LEADING(x h) USE_NL(h)*/ \\
& \quad h.sql_id \\
& \quad h.sql_plan_hash_value \\
& \quad SUM(10) ash_secs \\
\text{FROM} & \quad \text{dba_hist_snapshot x} \\
& \quad \text{dba_hist_active_sess_history h} \\
\text{WHERE} & \quad x.end_interval_time <= TO_DATE('201001261300','yyyymmddhh24mi') \\
& \quad \text{AND} \quad x.begin_interval_time >= TO_DATE('201001261100','yyyymmddhh24mi') \\
& \quad \text{AND} \quad h.sample_time BETWEEN TO_DATE('201001261100','yyyymmddhh24mi') \\
& \quad \text{AND} \quad \text{TO_DATE('201001261300','yyyymmddhh24mi')} \\
& \quad \text{AND} \quad \text{h.snap_id = x.snap_id} \\
& \quad \text{AND} \quad \text{h.dbid = x.dbid} \\
& \quad \text{AND} \quad \text{h.instance_number = x.instance_number} \\
& \quad \text{AND} \quad \text{h.event = 'enq: TX - row lock contention'} \\
\text{GROUP} & \quad \text{BY h.sql_id, h.sql_plan_hash_value} \\
\text{ORDER} & \quad \text{BY ash_secs DESC} \\
\end{align*}
\]

Lets take a look at the historical ASH data in the AWR snapshots, and see where we lost time to row level locking in that period across the whole database.

\[21\] This blog extra explains how to produce such a graph: [http://blog.go-faster.co.uk/2008/12/graphing-awr-data-in-excel.html](http://blog.go-faster.co.uk/2008/12/graphing-awr-data-in-excel.html)
And rather reassuringly the ASH total agrees quite well with AWR. The top statement alone is costing us nearly 5 hours.

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>Hash Value</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>7qxdrwc4yzhh</td>
<td>3723363341</td>
<td>26030</td>
</tr>
<tr>
<td>652mx4tfq415</td>
<td>1888029394</td>
<td>11230</td>
</tr>
<tr>
<td>c9jjtvk0qf649</td>
<td>3605988889</td>
<td>6090</td>
</tr>
<tr>
<td>artqpxug4zf01</td>
<td>8450529</td>
<td>240</td>
</tr>
<tr>
<td>gtj7zuzy2b4g6</td>
<td>2565837323</td>
<td>100</td>
</tr>
</tbody>
</table>

Let’s look at the statements involved. They all come FROM the PeopleSoft Publish and Subscribe Servers.

The first statement shows a homemade sequence. PeopleSoft is a platform agnostic development, so it doesn’t use Oracle sequence objects. The other two statements show an update to a queue management table.

```
SQL_ID 7qxdrwc4yzhh
---------------------
UPDATE PSIBQUEUEINST SET QUEUESEQID=QUEUESEQID+:1 WHERE QUEUENAME=:2
```

```
SQL_ID 652mx4tfq415
---------------------
UPDATE PSAPMSGPUBSYNC SET LASTUPDDTTM=SYSDATE WHERE QUEUENAME=:1
```

```
SQL_ID c9jjtvk0qf649
---------------------
UPDATE PSAPMSGSUBCSYNC SET LASTUPDDTTM=SYSDATE WHERE QUEUENAME=:1
```

There is nothing I can do about any of these because the code is deep inside PeopleTools and cannot be changed. This is the way that the Integration Broker works.

I cannot find the statement that is blocking these statements. Oracle doesn’t hold that information. It is probably another instance of the same statement, but that it isn’t the question. The real question is ‘what is the session that is holding the lock doing while it is holding the lock, and can I do something about that?’

The ASH data has three columns that help me to identify the blocking session.

- **BLOCKING_SESSION_STATUS** – this column has the value VALID if the blocking session is within the same instance, but GLOBAL if is in another instance.
- **BLOCKING_SESSION** – this is the session ID of the blocking session if the session is within the same instance, otherwise it is null.
- **BLOCKING_SESSION_SERIAL#** - this is the serial number of the blocking session if the session is within the same instance, otherwise it is null.
For cross-instance locking I cannot use ASH in 10g to find the exact session that is holding the lock. All I know is that I am locked by a session connected to another instance. The 11g ASH data does contain this information. So this technique only works for locking within a single instance on 10g.

The queries that I need to write don’t perform well on the ASH views, so I am going to extract them to a temporary working storage table.

```sql
DROP TABLE my_ash /

CREATE TABLE my_ash AS
SELECT /*+LEADING(x) USE_NL(h)^/ h.*
FROM dba_hist_snapshot x ,
     dba_hist_active_sess_history h
WHERE x.end_interval_time >= TO_DATE('201001261100','yyyymmddhh24mi')
     AND x.begin_interval_time <= TO_DATE('201001261300','yyyymmddhh24mi')
     AND h.sample_time BETWEEN TO_DATE('201001261100','yyyymmddhh24mi')
          AND TO_DATE('201001261300','yyyymmddhh24mi')
     AND h.snap_id = x.snap_id
     AND h.dbid = x.dbid
     AND h.instance_number = x.instance_number /

CREATE INDEX my_ash ON my_ash (dbid, instance_number, snap_id, sample_id, sample_time) COMPRESS 3 /
CREATE INDEX my_ash2 ON my_ash (event, dbid, instance_number, snap_id) COMPRESS 3 /

I now want to look for statements running in the sessions that are blocking the sessions that are waiting on TX enqueue.

```sql
SELECT /*+LEADING(x w) USE_NL(h w)^/
     h.sql_id
 ,
     h.sql_plan_hash_value
 ,
     SUM(10) ash_secs
FROM my_ash w
     left outer join my_ash h
     on h.snap_id = w.snap_id
     AND h.dbid = w.dbid
     AND h.instance_number = w.instance_number
     AND h.sample_id = w.sample_id
     AND h.sample_time = w.sample_time
     AND h.session_id = w.blocking_session
     AND h.session_serial# = w.blocking_session_serial#
WHERE w.event = 'enq: TX - row lock contention'
GROUP BY h.sql_id, h.sql_plan_hash_value
ORDER BY ash_secs DESC
```
This is the top of list of statements.

Note that two of the statements that appear in this list were the original SQL_IDs that we started with. I'll come back to this below.

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>SQL_PLAN_HASH_VALUE</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5st3un4a2y92</td>
<td>2494504609</td>
<td>10670</td>
</tr>
<tr>
<td>652mx4tffq415</td>
<td>1888029394</td>
<td>7030</td>
</tr>
<tr>
<td>artqgxug4z0f1</td>
<td>8450529</td>
<td>580</td>
</tr>
<tr>
<td>7qxdrwc4yzhh</td>
<td>3723363341</td>
<td>270</td>
</tr>
</tbody>
</table>

The first line in the report is blank because there is no ASH data for the session holding the lock because it is not active on the database. This indicates that the client process is busy, or waiting on something else outside the database. This is where the majority of the time is spent, and there is nothing that can be done within the database to address this. It is a matter of looking at the client process.

However the line in the report says that a statement blocks other sessions for 10670 seconds. We can look at that.

```
SELECT * FROM table(dbms_xplan.display_awr('5st32un4a2y92',2494504609,NULL,'ADVANCED'));
```

Note also that this is the execution plan when the query was first seen. The cost is the cost then, not now. The value of the bind variable was the value then not now!
If I run a fresh execution plan on this statement, the cost is now 3178. This reflects how the table has grown over time.

```sql
explain plan for select 'X' from PS_CDM_LIST where CONTENTID = :1
/

Explained.

Plan hash value: 2494504609
```

### Resolving the Lock Chain to the Ultimate Blocking Session

The second longest running blocking statement is one of the statements that we found in the first place, so this shows that we have a chain of locks, and we need to resolve that back to the blocking statement that is not itself blocked.

```sql
SELECT * FROM table(dbms_xplan.display_awr('652mx4tffq415',1888029394,NULL,'ADVANCED'))
```

**SQL_ID 652mx4tffq415**

```sql
UPDATE PSAPMSGPUBSYNC_SET_LASTUPDDTTM=SYSDATE WHERE QUEUENAME=:1
```

If one session is held by a second session which is itself blocked by a third session, I am more interested in what the third session is doing. The following SQL updates the blocking session data recorded in the first session that indicates the session to point to the third session. I don’t need to find the ASH data for the third session. It might not exist because the third session might not be active on the database (because the user or client process is busy with non-database activity) while it continues to hold the lock.

If I run the SQL repeatedly until no more rows are updated, I will be able to associate the time spent waiting on a lock with the session that is ultimately responsible for the lock.
So this moves the emphasis further onto the query of PS_CDM_LIST.

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>Hash Value</th>
<th>ASH SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5st32un4a2y92</td>
<td>2494504609</td>
<td>12840 (was 10670)</td>
</tr>
<tr>
<td>652mx4tffq415</td>
<td>1888029394</td>
<td>5030 (was 7030)</td>
</tr>
<tr>
<td>7qxdrcwc4nyzh</td>
<td>3723363341</td>
<td>320 (was 270)</td>
</tr>
</tbody>
</table>
Which Tables Account for My I/O?

ASH holds object number data. But I want to work in terms of tables. So, I am going to produce my own version of DBA_OBJECTS. I want to be able to easily group all the objects in a table, its indexes, their partitions and sub-partitions.

```sql
CREATE TABLE DMK_OBJECTS
(   OBJECT_ID NUMBER NOT NULL,
    OWNER VARCHAR2(30) NOT NULL,
    OBJECT_NAME VARCHAR2(128) NOT NULL,
    SUBOBJECT_NAME VARCHAR2(30),
    PRIMARY KEY (OBJECT_ID)
)
/
insert into dmk_objects
SELECT object_id, owner, object_name, subobject_name
FROM dba_objects
where object_type like 'TABLE%'
union all
SELECT o.object_id, i.table_owner, i.table_name, o.subobject_name
FROM dba_objects o, dba_indexes i
where o.object_type like 'INDEX%'
and i.owner = o.owner
and i.index_name = o.object_name
/
```
So, for a single process identified by process instance number, I want to take the ash entries for that process that relate to the db file wait events, and I want to see which tables they relate to.

```sql
SELECT /*+LEADING(r x h) USE_NL(h)*/
  r.prcsinstance,
  o.owner, o.object_name,
  (CAST(r.enddttm AS DATE) - CAST(r.begindttm AS DATE)) * 86400 exec_secs,
  SUM(10) ash_secs
FROM dba_hist_snapshot x,
  dba_hist_active_sess_history h,
  sysadm.psprcsrqst r,
  dmk_objects o
WHERE x.end_interval_time >= r.begindttm
  AND x.begin_interval_time <= r.enddttm
  AND h.sample_time BETWEEN r.begindttm AND r.enddttm
  AND h.snap_id = x.snap_id
  AND h.dbid = x.dbid
  AND h.instance_number = x.instance_number
  AND h.module = r.prcsname
  AND h.action LIKE 'PI='||r.prcsinstance||'%'
  AND h.event LIKE 'db file%'
  AND r.prcsinstance = 2256605
  AND h.current_obj# = o.object_id
GROUP BY r.prcsinstance, r.prcsname, r.begindttm, r.enddttm,
  o.owner, o.object_name
having SUM(10) >= 60
```

This process spends a lot of time reading GP_RSLT_ACUM.

<table>
<thead>
<tr>
<th>Process Instance</th>
<th>Owner</th>
<th>Object Name</th>
<th>Exec</th>
<th>ASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>2256605</td>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>5469</td>
<td>590</td>
</tr>
<tr>
<td>2256605</td>
<td>SYSADM</td>
<td>PS_GP_RSLT_PIN</td>
<td>5469</td>
<td>310</td>
</tr>
<tr>
<td>2256605</td>
<td>SYSADM</td>
<td>PS_GP_PYEO_PRCT_STAT</td>
<td>5469</td>
<td>170</td>
</tr>
<tr>
<td>2256605</td>
<td>SYSADM</td>
<td>PS_JOB</td>
<td>5469</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Sum</strong></td>
<td></td>
<td><strong>1100</strong></td>
</tr>
</tbody>
</table>

We can then get the execution plans for the individual statements:

```sql
SELECT /*+LEADING(r x h) USE_NL(h)*/
  r.prcsinstance,
  o.owner, o.object_name,
  h.sql_id, h.sql_plan_hash_value,
  (CAST(r.enddttm AS DATE) - CAST(r.begindttm AS DATE)) * 86400 exec_secs,
  SUM(10) ash_secs
FROM dba_hist_snapshot x,
  dba_hist_active_sess_history h
```
This is the beginning of the top statement

```
-- having sum(10) >= 60
ORDER BY ash_secs DESC
) x
ORDER BY ash_secs DESC
/
```

```
SELECT * FROM table(dbms_xplan.display_awr('5n5tu62039ak2',843197476,NULL,'ADVANCED'));
SELECT * FROM table(dbms_xplan.display_awr('ggwkkzmw1wmfs',3417552465,NULL,'ADVANCED'));
SELECT * FROM table(dbms_xplan.display_awr('g1yupgb61zndq',3420404643,NULL,'ADVANCED'));
```

```
INSERT INTO ...
WHERE B.PIN_NUM = C.PIN_NUM AND A.PROCESS_INSTANCE =2256605 AND P.EMPLID = A.EMPLID AND
P.EMPL_RCD = A.PERIOD_BEGIN_DT AND B.USER_KEY1 > ' ',
AND B.USER_KEY1 =to_char(G.HIRE_DT,'YYYY-MM-DD')
AND C.PIN_NM IN ('AE PHO_TAKE', 'AE PHO B_TAKE')
```
Across an entire system, for the last week which tables are the cause of the most I/O?

```
SELECT /*+LEADING(x h) USE_NL(h)*/
  o.owner, o.object_name
,  SUM(10) ash_secs
FROM  dba_hist_snapshot x
 ,  dba_hist_active_sess_history h
 ,  dmk_objects o
WHERE  x.end_interval_time  >= SYSDATE-7
  AND  x.begin_interval_time  <= SYSDATE
  AND  h.sample_time          >= SYSDATE-7
  AND  h.sample_time          <= SYSDATE
  AND  h.snap_id = x.snap_id
  AND  h.dbid = x.dbid
  AND  h.instance_number = x.instance_number
  AND  h.event LIKE 'db file%'
  AND  h.current_obj# = o.object_id
GROUP  by o.owner, o.object_name
HAVING  SUM(10) >= 3600
ORDER  by ash_secs desc
```

This is just to put things into context. I am going to look at GP_RSLT_ACUM, because I know it is the output of the payroll calc process, and it may be a case for doing a selective extract into a reporting table.

<table>
<thead>
<tr>
<th>ASHOWNER</th>
<th>OBJECT_NAME</th>
<th>Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSADM</td>
<td>PS_TL_RPTD_TIME</td>
<td>800510</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_TL_PAYABLE_TIME</td>
<td>327280</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>287870</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_SCH_DEFN_DTL</td>
<td>161690</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_SCH_DEFN_TBL</td>
<td>128070</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_PIN</td>
<td>124560</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_PYR_PRC_STAT</td>
<td>92410</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_SCH_ADHOC_DTL</td>
<td>88810</td>
</tr>
</tbody>
</table>
Which processes hit this table?

```
SELECT /*+LEADING(x) USE_NL(h)*/
    o.owner, o.object_name
  , h.module
  -- , h.sql_id, h.sql_plan_hash_value
  , SUM(10) ash_secs
FROM dba_hist_snapshot x
  , dba_hist_active_sess_history h
  , dmk_objects o
WHERE x.end_interval_time >= SYSDATE-7
  AND x.begin_interval_time <= SYSDATE
  AND h.sample_time         >= SYSDATE-7
  AND h.sample_time         <= SYSDATE
  AND h.snap_id = x.snap_id
  AND h.dbid = x.dbid
  AND h.instance_number = x.instance_number
  AND h.event LIKE 'db file%'
  AND h.current_obj# = o.object_id
  AND o.object_name = 'PS_GP_RSLT_ACUM'
  -- AND h.module != 'GPPDPRUN'
  -- AND h.module = 'DBMS_SCHEDULER'
GROUP BY o.owner, o.object_name
  , h.module
  -- , h.sql_id, h.sql_plan_hash_value
having SUM(10) >= 900
ORDER BY ash_secs DESC
```

So these processes spend this long reading the accumulator table and its index

<table>
<thead>
<tr>
<th>OWNER</th>
<th>OBJECT_NAME</th>
<th>MODULE</th>
<th>Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>XGF_HOL_MGMT</td>
<td>79680</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>DBMS_SCHEDULER</td>
<td>37810</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>SQL*Plus</td>
<td>37060</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>GPGBHLE</td>
<td>30710</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>GPPDPRUN</td>
<td>27440</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>XGF_AE_AB007</td>
<td>21440</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>SQL Developer</td>
<td>11210</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>GPGBEPTD</td>
<td>7240</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>XGF_CAPITA</td>
<td>5850</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>GPGB_PSLIP_X</td>
<td>5030</td>
</tr>
<tr>
<td>SYSADM</td>
<td>PS_GP_RSLT_ACUM</td>
<td>GPGB_EDI</td>
<td>4880</td>
</tr>
</tbody>
</table>
Who is using this index?

Or, to put it another way, I want to change or drop this index, who and what will I impact?

The challenge is is certainly not exclusive to PeopleSoft, but in PeopleSoft, the Application Designer tool makes it very easy for developers to add indexes to tables. Sometimes, too easy! I often find tables with far more indexes than are good for them.

There are several concerns:

- Indexes are maintained during data modification. The more indexes you have, the greater the overhead.
- If you have too many indexes, Oracle might choose to use the wrong one, resulting in poorer performance.
- There is of course also a space overhead for each index, but this is often of less concern.

If you can get rid of an index, Oracle doesn't store, maintain or use it.

In some cases, I have wanted to remove unnecessary indexes, and in others to adjust indexes. However, this immediately raises the question of where are these indexes used, and who will be impacted by the change. Naturally, I turn to the Active Session History (ASH) to help me find the answers.

As we have already discussed ASH reports the object number, file number, block number and (from 11g) row number being accessed by physical file operations. These values are not reliable for other events because they are merely left over from the previous file event that set them. So, we can profile the amount of time spent on physical I/O on different indexes, but not other forms of DB Time, such as CPU time, spent accessing the blocks in the buffer cache.

However, if you want to find where an index is used, then this query will also identify SQL_IDs where the index is either used in the query or maintained by DML. If I am interested in looking for places where changing or deleting an index could have an impact then I am only interested in SQL query activity. ASH samples which relate to index maintenance are a false positive. Yet, I cannot simply eliminate ASH samples where the SQL_OPNAME is not SELECT because the index may be used in a query within the DML statement.

Index Use from SQL Plans Captured by AWR

During an AWR snapshot the top-n SQL statements by each SQL criteria in the AWR report (Elapsed Time, CPU Time, Parse Calls, Shareable Memory, Version Count) , see dbms_workload_repository. The SQL plans are exposed by the view DBA_HIST_SQL_PLAN.

On PeopleSoft systems, I generally recommend decreasing the snapshot interval from the default of 60 minutes to 15. The main reason is that SQL gets aged out of the library cache very quickly in PeopleSoft systems. They generate lots of dynamic code, often with literal values rather than bind variables. Cursor sharing is not recommended for PeopleSoft, so different bind variables result in different SQL_IDs. The dynamic code also results in different SQL IDs even with cursor sharing (see http://blog.psftdba.com/2014/08/to-hint-or-not-to-hint-application.html). Therefore, increasing the snapshot frequency means that will capture more SQL statements and plans. This will increase total volume of the AWR
repository simply because there are more snapshots. However, the overall volume of ASH data captured does not change, it just gets copied to the repository earlier.

On DBA_HIST_SQL_PLAN the object ID, owner, type and name are recorded, so I can find the plans which referenced a particular object. I am going to take an example from a PeopleSoft Financials system, and look at indexes on the PS_PROJ_RESOURCE table. These are some of the indexes on PS_PROJ_RESOURCE. We have 4 indexes that all lead on PROCESS_INSTANCE. I suspect that not all are essential, but I need to work out what is using them.

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>Pos</th>
<th>COLUMN_NAME</th>
<th>COLUMN_EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSJPROJ_RESOURCE</td>
<td>1</td>
<td>PROCESS_INSTANCE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>BUSINESS_UNIT_GL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>BUSINESS_UNIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>PROJECT_ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>ACTIVITY_ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>CUST_ID</td>
<td></td>
</tr>
<tr>
<td>PSLPROJ_RESOURCE</td>
<td>1</td>
<td>PROCESS_INSTANCE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>EMPLID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>EMPL_RCD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>TRANS_DT</td>
<td></td>
</tr>
<tr>
<td>PSMPROJ_RESOURCE</td>
<td>1</td>
<td>PROCESS_INSTANCE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>BUSINESS_UNIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>PROJECT_ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>ACTIVITY_ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>RESOURCE_ID</td>
<td></td>
</tr>
<tr>
<td>PSNPROJ_RESOURCE</td>
<td>1</td>
<td>PROCESS_INSTANCE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>BUSINESS_UNIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>TIME_RPTG_CD</td>
<td></td>
</tr>
</tbody>
</table>

I find it easier to extract the ASH data to my own working storage table. For each index on PS_PROJ_RESOURCE, I am going to extract a distinct list of plan hash values. I will then extract all ASH data for those plans.
Note, that I have not joined the SQL_ID on DBA_HIST_SQL_PLAN. That is because different SQL_IDs can produce the same execution plan. The plan is equally valid for all SQL_IDs that produce the plan, not just the one where the SQL_ID also matches. Although, of course, costs may vary.

```
DROP TABLE my_ash purge
/
CREATE TABLE my_ash COMPRESS AS
WITH p AS (
    SELECT DISTINCT p.plan_hash_value, p.object#, p.object_owner, p.object_type, p.object_name
    FROM      dba_hist_sql_plan p
    WHERE     p.object_name like 'PS_PROJ_RESOURCE'
    AND       p.object_type LIKE 'INDEX%'
    AND       p.object_owner = 'SYSADM'
)
SELECT p.object# object_id, p.object_owner, p.object_type, p.object_name
 ,      h.*
FROM   dba_hist_active_sess_history h
 ,      p
WHERE  h.sql_plan_hash_value = p.plan_hash_value
/
```

I am fortunate that PeopleSoft is a well instrumented application. Module and Action are set to fairly sensible values that will tell me whereabouts in the application the ASH sample relates.

In the following query I have omitted any ASH data generated by SQL*Plus, Toad, or SQL Developer, and also any generated by Oracle processes to prevent statistics collection being included.

```
Set pages 999 lines 150 trimspool on
break on object_name skip 1
compute sum of ash_secs on object_name
column ash_secs heading 'ASH|Secs' format 9999999
column module format a20
column action format a32
column object_name format a18
column max_sample_time format a19 heading 'Last|Sample'
column sql_plans heading 'SQL|Plans' format 9999
column sql_execs heading 'SQL|Execs' format 99999
WITH h AS (
    SELECT
        object_name, module, action
 ,      sum(10) ash_secs
 ,      COUNT(DISTINCT sql_plan_hash_value) sql_plans
 ,      COUNT(DISTINCT sql_id||sql_plan_hash_value||sql_exec_id) sql_execs
 ,      MAX(sample_time) max_sample_time
FROM h
WHERE NOT lower(module) IN('oracle','toad','sqlplus','sqlplusw')
AND       NOT lower(module) LIKE 'sql%'
GROUP BY object_name, module, action
ORDER BY SUBSTR(object_name,4), object_name, ash_Secs desc
/
Spool off
```
I now have a profile of how much each index is used. In this particular case, I found something using every index. It is possible that you will not find anything that uses some indexes.

<table>
<thead>
<tr>
<th>OBJECT_NAME</th>
<th>MODULE</th>
<th>ACTION</th>
<th>Ash</th>
<th>SQL</th>
<th>SQL Last</th>
<th>Secs</th>
<th>Plans</th>
<th>Execs</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSJPROJ_RESOURCE</td>
<td>PC_TL_TO_PC</td>
<td>GF_PRINT_AE.CallmeA.Step24.S</td>
<td>7300</td>
<td>1</td>
<td>66 06:32:57 27/08/2014</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC_PRICING</td>
<td></td>
<td>GF_PRINT_AE.CallmeA.Step24.S</td>
<td>40</td>
<td>1</td>
<td>2 08:38:57 22/08/2014</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7340</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1220</td>
<td></td>
<td>53 06:33:17 27/08/2014</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The next stage is to look at individual SQL statements.

This query looks for which SQL statement is using a particular index on PROJ_RESOURCE. If I can't find the SQL which cost the most time, then just choose another SQL with the same plan.

- I have found that sometimes a plan is captured by AWR, but the SQL statement is not. Personally, I think that is a bug. Working around it has made the following query quite complicated.
WITH h AS (  
  SELECT  h.object_name  ,  
      CASE WHEN h.module IS NULL THEN REGEXP_SUBSTR(h.program,'[^@.]+',1,1)  
      WHEN h.module LIKE 'PSAE.%' THEN REGEXP_SUBSTR(h.module,'[^.]+',1,2)  
      ELSE REGEXP_SUBSTR(h.program,'[^@.]+',1,1)  
  END as module  ,  
      CASE WHEN h.action LIKE 'PT-%' THEN NULL  
      ELSE h.action  
  END as action  ,  
      h.sql_id, h.sql_plan_hash_value  ,  
      t.command_type  
  --not null if plan and statement captured  
  FROM    my_ash  
  LEFT OUTER JOIN (  
      SELECT t1.*  
      FROM dba_hist_sqltext t1  
      ,  
      dba_hist_sql_plan pi  
      WHERE t1.sql_id = pi.sql_id  
      AND pi.id = 1  
      ) t  
  ON   t.sql_id = h.sql_id  
  AND  t.dbid = h.dbid  
  WHERE   h.object_name IN('PSMPROJRESOURCE')  
  AND     h.object_type = 'INDEX'  
  AND     h.object_owner = 'SYSADM'  
  AND     NOT lower(module) IN('oracle','toad','sqlplus','sqlplusw')  
  AND     NOT lower(module) LIKE 'sql%'  
  )
, x AS (  
  --aggregate DB time by object and statement  
  SELECT    object_name, sql_id, sql_plan_hash_value  ,  
      sum(10) ash_secs  ,  
      10*COUNT(command_type) sql_secs  
  --DB time for captured statements only  
  FROM      h  
  WHERE NOT lower(module) IN('oracle','toad','sqlplus','sqlplusw')  
  AND       NOT lower(module) LIKE 'sql%'  
  GROUP BY  object_name  ,  
      sql_id, sql_plan_hash_value  
  ), y AS (  
  --rank DB time per object and plan  
  SELECT  object_name, sql_id, sql_plan_hash_value  ,  
      ash_secs  ,  
      SUM(ash_secs) over (partition by object_name, sql_plan_hash_value) plan_ash_secs  ,  
      row_number() over (partition by object_name, sql_plan_hash_value ORDER BY sql_Secs DESC) ranking  
  FROM  x  
  )
, z AS (  
  SELECT object_name  ,  
      CASE WHEN t.sql_text IS NOT NULL THEN y.sql_id  
      ELSE (SELECT t1.sql_id  
      FROM dba_hist_sqltext t1  
      ,  
      dba_hist_sql_plan pi  
      WHERE  t1.sql_id = pi.sql_id  
      AND  pi.plan_hash_value = y.sql_plan_hash_value  
      AND  rownum = 1)  
      --if still cannot find statement just pick any one  
  END AS sql_id  
  ,  
      y.sql_plan_hash_value, y.plan_ash_secs  ,  
      CASE WHEN t.sql_text IS NOT NULL THEN t.sql_text  
      ELSE (SELECT t1.sql_text  
      FROM dba_hist_sqltext t1  
      )
  )

SELECT object_name, sql_id, sql_plan_hash_value  ,  
      ash_secs  ,  
      SUM(ash_secs) over (partition by object_name, sql_plan_hash_value) plan_ash_secs  ,  
      row_number() over (partition by object_name, sql_plan_hash_value ORDER BY sql_Secs DESC) ranking  
FROM  x  
WHERE NOT lower(module) IN('oracle','toad','sqlplus','sqlplusw')  
AND       NOT lower(module) LIKE 'sql%'  
GROUP BY  object_name, sql_id, sql_plan_hash_value  
)
So now I can see the individual SQL statements.

Ultimately, I have needed to look through the SQL plans that do use an index to decide whether I need to keep that index, or to decide whether the statement would perform adequately using another index. In this case, on this particular system, I think the index PSMPROJ_RESOURCE would be adequate for this statement, and I would consider dropping PSLPROJ_RESOURCE.

The decision also requires some background knowledge about the system. I carried on with examination of SQL and execution plan to determine whether each index is really needed or another index (or even no index at all) would do as well.
Getting Rid of Indexes

So, I am going to jump forward to the point where I have decided that I want drop the J, L and N indexes on PROJ RESOURCE and just keep M. Obviously this needs to be tested carefully in all the places that reference the index.

- If all the testing is successful and you decide to go ahead and drop the index in production, you might prefer to make it invisible first for a while. It is likely that the indexes you choose to examine are large and will take time to rebuild. An invisible index will not be used by the Optimizer, but it will continue to be maintained during DML. If there are any unfortunate consequences, you can immediately make the index visible without having to rebuild it.

Limitations of Method

- AWR does not capture all SQLs, nor all SQL plans. First the SQL has to be in the library cache and then it must be one of the top-n. A SQL that is efficient because it uses an appropriate index may not be captured, and will not be detected by this approach.

- ASH data is purged after a period of time, by default 31 days. If an index is only used by a process that has not run within the retention period, then it will not be detected by this approach.\(^{22}\) This is another reason to retain ASH and AWR in a repository for a longer period. I have heard 400 days suggested, so that you have ASH for a year and a month.
  - However, this also causes the SYSAUX tablespace to be become very large, so I would suggest regularly moving the data to a separate database. I know one customer who has built a central AWR repository for all their production and test databases and automated transfer of data. This repository has been of immense diagnostic value.

\(^{22}\) However, if you only need an index during an annual process, perhaps it would be better to build it for that process and drop it again afterwards, rather than have it in place for the whole year?
Did my Execution Plan Change?

We were experiencing a problem with a query in a particular report. We fixed it by adding a hint. I wanted to prove that when the hint was put into production, the execution plan changed. This query is very similar to the one described in Batch Processes (see page 15), but here I want to list all the queries run by all instances of a named report, and see if the execution plan changed.

```
SELECT /*+LEADING(r f d x h) USE_NL(h)*/
    r.prcsinstance,
    r.begindttm,
    h.sql_id,
    h.sql_child_number,
    h.sql_plan_hash_value,
    (CAST(r.enddttm AS DATE)-CAST(r.begindttm AS DATE))*86400 exec_secs,
    SUM(10) ash_secs
FROM dba_hist_snapshot x,
     dba_hist_active_sess_history h,
     sysadm.ppsrcsrqst r,
     sysadm.pscdml_file_list f,
     sysadm.psxprptdefn d
WHERE x.end_interval_time >= r.begindttm
  AND x.begin_interval_time < r.enddttm
  AND h.sample_time BETWEEN r.begindttm AND r.enddttm
  AND h.snap_id = x.snap_id
  AND h.dbid = x.dbid
  AND h.instance_number = x.instance_number
  AND h.module = r.prcsname
  AND h.action LIKE 'PI'||r.prcsinstance||'%'
  AND r.prcsinstance = f.prcsinstance
  AND NOT f.cdm_file_type IN('AET','TRC','LOG')
  AND d.report_defn_id = SUBSTR(f.filename,1,instr(f.filename,'.')-1)
  AND d.report_defn_id = 'XGF_WK_LATE'
  AND r.prcsname = 'PSXPQRYRPT'
  AND r.begindttm >= TRUNC(SYSDATE)
ORDER BY begindttm
```

And we can see that after the fix was applied and the users were told they could start to run this report again, the execution plan changed and the run time was much better.

<table>
<thead>
<tr>
<th>PRCSINSTANCE</th>
<th>BEGINDTTM</th>
<th>SQL_ID</th>
<th>SQL_PLAN_HASH_VALUE</th>
<th>EXEC_SECS</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1964975</td>
<td>08:30:52</td>
<td>22/01/2010</td>
<td>460bqpcrcr8bd</td>
<td>20379</td>
<td>20080</td>
</tr>
<tr>
<td>1965250</td>
<td>09:08:51</td>
<td>22/01/2010</td>
<td>460bqpcrcr8bd</td>
<td>20983</td>
<td>20690</td>
</tr>
<tr>
<td>1968443</td>
<td>16:42:51</td>
<td>22/01/2010</td>
<td>3398716340</td>
<td>105</td>
<td>80</td>
</tr>
<tr>
<td>1968469</td>
<td>16:47:21</td>
<td>22/01/2010</td>
<td>3398716340</td>
<td>90</td>
<td>70</td>
</tr>
<tr>
<td>1968485</td>
<td>16:50:19</td>
<td>22/01/2010</td>
<td>3398716340</td>
<td>62</td>
<td>40</td>
</tr>
<tr>
<td>1968698</td>
<td>17:40:01</td>
<td>22/01/2010</td>
<td>3398716340</td>
<td>76</td>
<td>50</td>
</tr>
<tr>
<td>1968884</td>
<td>18:19:19</td>
<td>22/01/2010</td>
<td>3398716340</td>
<td>119</td>
<td>120</td>
</tr>
<tr>
<td>1969066</td>
<td>18:34:24</td>
<td>22/01/2010</td>
<td>3398716340</td>
<td>187</td>
<td>170</td>
</tr>
</tbody>
</table>

So, not only have I diagnosed a problem with ASH, I have also proven that the fix, when applied to production has successfully resolved the issue.
What was the Effect of Plan Stability

I have experienced unstable execution plans with processing of Payroll calculations. The performance of the larger pay group is fine, but some of the execution plans for the smaller paygroups are different, and performance can be poor.

A set of stored outlines were created for a full payroll identification and calculation process for the larger payroll, and applied to all subsequent payrolls. Now, I want to prove not only that the outlines were used, but that they have a beneficial effect.

I have three test scenarios.

1. A large streamed payroll calculation was run. It ran without using outlines for 2h 42m, which can considered to be good performance (in fact I used this process to collect the stored outlines).

2. A small non-streamed payroll calculation without outlines. This ran for over 8 hours before it was cancelled. Hence, I don’t have data for all statements for this scenario.

3. A small non-streamed payroll calculation again, but this time with outlines enabled. It ran for 2h5m. Not great, considering it has a lot fewer payees than a single stream of the large payroll, but better than scenario 2.

I can use the ASH data to see whether the execution plan changed, and what effect that had on performance.

The SQL to perform the comparison looks horrendous, but it is effectively the usual query for each test scenario in in-line views that are then joined together.

```sql
SET PAGES 40
COLUMN sql_plan_hash_value Heading 'sql_plan_hash_value' FORMAT 999999999999
COLUMN sql_plan_hash_value2 Heading 'sql_plan_hash_value' FORMAT A12
SELECT /*+ LEADING(q1 r1q1 x1q1 h1q1) USE_NL(h1q1)
   LEADING(q2 r2q2 x2q2 h2q2) USE_NL(h2q2)
   LEADING(q3 r3q3 x3q3 h3q3) USE_NL(h3q3) */
   q1.sql_id,
   q1.sql_plan_hash_value,
   q1.ash_secs,
   DECODE(q1.sql_plan_hash_value,q2.sql_plan_hash_value,'**SAME**',q2.sql_plan_hash_value) sql_plan_hash_value2,
   q2.ash_secs,
   DECODE(q1.sql_plan_hash_value,q3.sql_plan_hash_value,'**SAME**',q3.sql_plan_hash_value) sql_plan_hash_value2,
   q3.ash_secs
FROM
   (SELECT /*+qb_name(q1)*/
    h1.sql_id,
    h1.sql_plan_hash_value,
    (NVL(r1.enddttm,SYSDATE)-r1.begindttm)*86400 exec_secs,
    SUM(10) ash_secs
    FROM dba_hist_snapshot x1,
    dba_hist_active_sess_history h1,
    sysadm.pspcrsreqst r1
    WHERE x1.end_interval_time >= r1.begindttm
    AND x1.begin_interval_time <= NVL(r1.enddttm,SYSDATE)
    AND h1.sample_time BETWEEN r1.begindttm AND NVL(r1.enddttm,SYSDATE)
    AND h1.snap_id = x1.snap_id
    GROUP BY h1.sql_id,h1.sql_plan_hash_value)
```
AND h1.dbid = x1.dbid
AND h1.instance_number = x1.instance_number
AND h1.module like r1.prcsname
AND h1.action LIKE 'PI='||r1.prcsinstance||'%'  
AND r1.prcsname = 'GPPDPRUN'
AND r1.prcsinstance = 2524397
GROUP BY r1.prcsname, r1.beginidttm, r1.endidttm, h1.sql_id, h1.sql_plan_hash_value 
) q1
INNER JOIN ( 
SELECT /*+qb_name(q2)*/
    h2.sql_id,
    h2.sql_plan_hash_value,
    (NVL(r2.endidttm,SYSDATE)-r2.beginidttm)*86400 exec_secs,
    SUM(10) ash_secs
FROM dba_hist_snapshot x2,
    dba_hist_active_sess_history h2,
    sysadm.pspcrsrqst r2
WHERE x2.end_interval_time >= r2.beginidttm
AND x2.begin_interval_time <= NVL(r2.endidttm,SYSDATE)
AND h2.sample_time BETWEEN r2.beginidttm AND NVL(r2.endidttm,SYSDATE)
AND h2.Snap_id = x2.Snap_id
AND h2.dbid = x2.dbid
AND h2.instance_number = x2.instance_number
AND h2.module like r2.prcsname
AND h2.action LIKE 'PI='||r2.prcsinstance||'%'  
AND r2.prcsname = 'GPPDPRUN'
AND r2.prcsinstance = 2524456
GROUP BY r2.prcsname, r2.beginidttm, r2.endidttm, h2.sql_id, h2.sql_plan_hash_value 
) q2
ON q1.sql_id = q2.sql_id
INNER JOIN ( 
SELECT /*+qb_name(q3)*/
    h3.sql_id,
    h3.sql_plan_hash_value,
    (NVL(r3.endidttm,SYSDATE)-r3.beginidttm)*86400 exec_secs,
    SUM(1) ash_secs
FROM v$active_session_history h3,
    sysadm.pspcrsrqst r3
WHERE h3.sample_time BETWEEN r3.beginidttm AND NVL(r3.endidttm,SYSDATE)
AND h3.module like r3.prcsname
AND h3.action LIKE 'PI='||r3.prcsinstance||'%'  
AND r3.prcsname = 'GPPDPRUN'
AND r3.prcsinstance = 2524456
GROUP BY r3.prcsname, r3.beginidttm, r3.endidttm, h3.sql_id, h3.sql_plan_hash_value 
) q3
ON q1.sql_id = q3.sql_id
order by q3.ash_secs desc, q1.sql_id
/

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>SCENARIO 1</th>
<th>ASH_SECS SCENARIO 2</th>
<th>ASH_SECS SCENARIO 3</th>
<th>ASH_SECS</th>
</tr>
</thead>
</table>

This query was run soon after test scenario 3 was run so it uses v$active_session_history.
<table>
<thead>
<tr>
<th>User ID</th>
<th>Consumed CPU</th>
<th>Consumed GPU</th>
<th>Result</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>4uzmzh74rdnrnz</td>
<td>2514155560</td>
<td>280 3829487612</td>
<td>28750 5023</td>
<td><strong>SAME</strong></td>
</tr>
<tr>
<td>4n482cm7r9qyn</td>
<td>1595742310</td>
<td>680 869376931</td>
<td>140 889</td>
<td><strong>SAME</strong></td>
</tr>
<tr>
<td>2f66y2u54ru1v</td>
<td>1145975676</td>
<td>630</td>
<td><strong>SAME</strong></td>
<td>531</td>
</tr>
<tr>
<td>1n2d7fb3jn7m</td>
<td>1293172177</td>
<td>150</td>
<td><strong>SAME</strong></td>
<td>150</td>
</tr>
<tr>
<td>652y9682bqvp</td>
<td>3325291917</td>
<td>30</td>
<td><strong>SAME</strong></td>
<td>110</td>
</tr>
<tr>
<td>d8gxmqop2zydta</td>
<td>1716202706</td>
<td>10 678016679</td>
<td>10 <strong>SAME</strong></td>
<td>32</td>
</tr>
<tr>
<td>2np47twhd5nga</td>
<td>3496258537</td>
<td>10</td>
<td><strong>SAME</strong></td>
<td>27</td>
</tr>
<tr>
<td>4ru0618dszw3y</td>
<td>2621940820</td>
<td>10</td>
<td>539127764</td>
<td>22</td>
</tr>
<tr>
<td>4ru0618dszw3y</td>
<td>539127764</td>
<td>100</td>
<td><strong>SAME</strong></td>
<td>22</td>
</tr>
<tr>
<td>4ru0618dszw3y</td>
<td>3325291917</td>
<td>10</td>
<td>539127764</td>
<td>22</td>
</tr>
<tr>
<td>4ru0618dszw3y</td>
<td>1403673054</td>
<td>110</td>
<td>539127764</td>
<td>22</td>
</tr>
<tr>
<td>gnnu2hfkjm2y</td>
<td>1559323680</td>
<td>80</td>
<td><strong>SAME</strong></td>
<td>19</td>
</tr>
<tr>
<td>fxz4z38pybu3x</td>
<td>1478655624</td>
<td>30</td>
<td>4036143672</td>
<td>18</td>
</tr>
<tr>
<td>2xkjjwvm9f9c</td>
<td>1393004311</td>
<td>20</td>
<td><strong>SAME</strong></td>
<td>18</td>
</tr>
<tr>
<td>a05wrd51zy3kj</td>
<td>2641254321</td>
<td>10</td>
<td><strong>SAME</strong></td>
<td>15</td>
</tr>
</tbody>
</table>

24 On the small payroll calculation, without outlines, this statement move than 100 times longer. It had not completed by this stage – the process was cancelled. With outlines enabled this statement used the same execution plan as in scenario 1. It didn’t perform that well compared to the large payroll calculation; clearly more work is required for this statement. However, at least it did complete and it did result in improved performance for the small payroll.

25 This is an example of a statement that performed better on the small payroll without an outline. So, sometimes it is better to let the optimiser change the plan!

26 This statement executed with 4 different execution plans during the large payroll, but once the outline was applied only one was used, and this seems to be
Which line in the Execution Plan?

Again from 11g, the line in the execution plan is recorded in the ASH data in SQL_PLAN_LINE_ID. I can also group the ASH data by this column and determine not just which statement consumes the most time, but which operation in the execution plan for that statement is consuming the time. I usually do this for one SQL statement at a time.

```
select /*+leading(r x h) use_nl(h)*/
  r.prcsinstance, H.SQL_plan_hash_value, h.sql_plan_line_id,
  sum(10) ash_secs
from DBA_HIST_SNAPSHOT x, DBA_HIST_ACTIVE_SESS_HISTORY h,
  sysadm.psprcsrqst r
WHERE X.END_INTERVAL_TIME >= r.begindttm
  AND X.BEGIN_INTERVAL_TIME <= NVL(r.enddttm,SYSDATE)
  AND h.sample_time between r.begindttm AND NVL(r.enddttm,SYSDATE)
  AND h.SNAP_id = X.SNAP_id
  AND h.dbid = x.dbid
  AND h.instance_number = x.instance_number
  AND h.module  = r.prcsname
  AND h.action LIKE 'PI='||r.prcsinstance||'%' 
  AND r.begindttm >= TRUNC(SYSDATE)
  AND r.prcsname = 'CM_CSTACCTG'
  AND h.sql_id = 'a47fb0x1b23jn'
group by H.SQL_plan_hash_value, r.prcsinstance, h.sql_plan_line_id
ORDER BY prcsinstance, ASH_SECS DESC
```

I now have a profile of a single SQL statement by plan line number.

```
<table>
<thead>
<tr>
<th>PRCSINSTANCE</th>
<th>SQL_PLAN_HASH_VALUE</th>
<th>SQL_PLAN_LINE_ID</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4945802</td>
<td>483167840</td>
<td>25</td>
<td>2410</td>
</tr>
<tr>
<td>483167840</td>
<td>24 1190</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483167840</td>
<td>26 210</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483167840</td>
<td>20 190</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483167840</td>
<td>21 30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483167840</td>
<td>16 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483167840</td>
<td>23 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483167840</td>
<td>22 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483167840</td>
<td>18 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483167840</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483167840</td>
<td>7 10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The plan line IDs can be related back to the execution plan.

```
Plan hash value: 483167840

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>TQ</th>
<th>IN-Out</th>
<th>PQ Distrib</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>HASH JOIN SEMI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>PX RECEIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>PX SEND HASH</td>
<td>:TQ10003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>PX BLOCK ITERATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>TABLE ACCESS FULL</td>
<td>PS_CM_DEPLETE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>BUFFER SORT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>PX RECEIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>PX SEND HASH</td>
<td>:TQ10001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>INDEX FULL SCAN</td>
<td>PS_CM_DEPLETE_COST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>INDEX UNIQUE SCAN</td>
<td>PS_TRANSACTION_INV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_TRANSACTION_INV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Recursive SQL

Sometimes a SQL statement causes another SQL statement to run behind the scenes. During SQL parse, Oracle may issue SQL to retrieve information from the catalogue that is usually referred to as ‘recursive SQL’. Other examples include SQL that is executed within a trigger, or within a PL/SQL procedure.

From Oracle 11gR2, there is a new column in the ASH data; TOP_LEVEL_SQL_ID. This is the ID of the SQL statement that spawned the recursive SQL.

```sql
Select * From ( 
select /*+leading(r x h) use_n1(h)*/ 
r.prcsinstance , h.top_level_sql_id , h.sql_id, h.sql_plan_hash_value , (r.enddttm-r.beginndttm)*86400 exec_secs , COUNT(DISTINCT sql_exec_id) num_execs , SUM(10) ash_secs , 10*COUNT(DISTINCT sample_id) elap_secs , COUNT(DISTINCT r.prcsinstance) PIs 
from DBA_HIST_SNAPSHOT x , DBA_HIST_ACTIVE_SESS_HISTORY h , sysadm.psprcsrqst r 
WHERE X.END_INTERVAL_TIME >= r.beginndttm AND X.BEGIN_INTERVAL_TIME <= NVL(r.enddttm,SYSDATE) And h.sample_time between r.beginndttm AND NVL(r.enddttm,SYSDATE) and h.SNAP_id = X.SNAP_id and h.dbid = x.dbid and h.instance_number = x.instance_number and h.module = r.prcsname and h.action LIKE 'PI='||r.prcsinstance||'%' and r.prcsinstance = 4604485 and h.top_level_sql_id = 'bvnq31hbmpzzy' group by r.prcsinstance, r.prcsname, r.beginndttm, r.enddttm , h.top_level_sql_id , h.sql_id, h.sql_plan_hash_value ) order by ash_secs desc ) order by ash_secs desc / 
```

Here we can see that two recursive statements were spawned by `bvnq31hbmpzzy`, and most of the time was spent in them.
In this example `35cpmm408n5qj` is an insert statement that is issued by a PL/SQL block. I can tell that because the bind variable numbr is prefixed with a ‘B’.

```
select sql_id, sql_text
from dba_hist_sqltext
where sql_id = '35cpmm408n5qj'
```

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>SQL_TEXT</th>
</tr>
</thead>
</table>

In fact, the insert statement comes from a standard PeopleSoft auditing trigger that is executed for each row processed on the original table. We can only count 79 executions because there are only 79 rows of data, the Application Engine trace shows that over 100,000 rows were updated on the table with the trigger.

Top SQL ID can also simply refer to the originating PL/SQL call.
Temporary Space Overhead

From 11gR2, ASH data includes information about memory utilisation in a column called TEMP_SPACE_ALLOCATED. Let me give you a real life practical example.

A Financials customer runs four concurrent instances of the cost accounting process. Two of them complete successfully, but two fail regularly with ORA-1652: Unable to extend temp segment … but complete successfully when run in isolation. The question is what is consuming the temporary tablespace and why.

```
Select * From (  
  select /*+leading(r x h) use_nl(h)*/  
  r.prcsinstance  
  , h.sql_id, h.sql_plan_hash_value  
  , (r.enddttm-r.begindttm)*86400 exec_secs  
  , count(distinct sql_exec_id) num_execs  
  , sum(10) ash_secs  
  , 10*count(distinct sample_id) elap_secs  
  , round(max(temp_space_Allocated)/1024/1024,0) tempMb  
  from DBA_HIST_SNAPSHOT x  
  , DBA_HIST_ACTIVE_SESS_HISTORY h  
  , sysadm.pspcrsrgt r  
  WHERE X.END_INTERVAL_TIME >= r.begindttm  
  AND X.BEGIN_INTERVAL_TIME <= NVL(r.enddttm,SYSDATE)  
  And h.sample_time between r.begindttm AND NVL(r.enddttm,SYSDATE)  
  and h.SNAP_id = X.SNAP_id  
  and h.dbid = x.dbid  
  and h.instance_number = x.instance_number  
  and h.module  = r.prcsname  
  and h.action LIKE 'PI='||r.prcsinstance||'%'  
  And r.begindttm >= TRUNC(SYSDATE)  
  and r.prcsname  = 'CM_CSTACCTG'  
  group by r.prcsinstance, r.prcsname, r.begindttm, r.enddttm  
  , h.sql_id, h.sql_plan_hash_value  
  having sum(10) > (NVL(r.enddttm,SYSDATE)-r.begindttm)*86400/100*5 --%  
  ORDER BY ASH_SECS DESC  
) order by ash_secs desc  

This report shows the maximum temporary segment consumption of each SQL statement in each process. With a temporary tablespace of 300Gb it is easy to see why 2 processes doing this is enough to cause trouble.

```

<table>
<thead>
<tr>
<th>PRCSINSTANCE</th>
<th>SQL_ID</th>
<th>SQL_PLAN_HASH_VALUE</th>
<th>EXEC_SECS</th>
<th>NUM_EXECS</th>
<th>ASH_SECS</th>
<th>ELAP_SECS</th>
<th>TEMPMB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4945802</td>
<td>a4758nx1a2j3jn</td>
<td>483167840</td>
<td>2</td>
<td>3909</td>
<td>3800</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>4945803</td>
<td>a4758nx1a2j3jn</td>
<td>3805982318</td>
<td>2</td>
<td>3440</td>
<td>3424</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>4945803</td>
<td>5107xy4xwbl</td>
<td>3926646197</td>
<td>1</td>
<td>3330</td>
<td>3330</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>4945802</td>
<td>5107xy4xwbl</td>
<td>3926646197</td>
<td>1</td>
<td>1140</td>
<td>1140</td>
<td>1140</td>
<td></td>
</tr>
<tr>
<td>4945803</td>
<td>6a8vfc0e8sz</td>
<td>1628923514</td>
<td>1</td>
<td>680</td>
<td>680</td>
<td>680</td>
<td></td>
</tr>
<tr>
<td>4945802</td>
<td>6a8vfc0e8sz</td>
<td>1628923514</td>
<td>1</td>
<td>490</td>
<td>490</td>
<td>490</td>
<td></td>
</tr>
<tr>
<td>4945803</td>
<td>68d6ycz6y6pq</td>
<td>2555729951</td>
<td>1</td>
<td>470</td>
<td>470</td>
<td>470</td>
<td></td>
</tr>
<tr>
<td>4945803</td>
<td>68d6ycz6y6pq</td>
<td>2555729951</td>
<td>1</td>
<td>480</td>
<td>480</td>
<td>480</td>
<td></td>
</tr>
</tbody>
</table>

There are two execution plans in play for the same problem statement in different instances of the process. I could also have profiled this by line number of plan to identify exactly which operation in the plan was consuming memory.
Things That Can Go Wrong

**DISPLAY_AWR reports old costs**

This is not really something that goes wrong, but it is a word of warning.

Here is an output from `display_awr`. Note the cost.

```sql
SELECT AWPATH_ID, AWTHREAD_ID
FROM PS_SAC_AW_STEPINST
WHERE AWRCS_ID = :1 AND SETID = :2
AND EFFDT = TO_DATE(:3,'YYYY-MM-DD') AND STAGE_NBR = :4 AND AWRSTEP_STATUS <> :5 AND
AWTHREAD_ID IN (SELECT AWTHREAD_ID FROM PS_PV_REQ_AW WHERE PARENT_THREAD = 601330)
GROUP BY AWTHREAD_ID, AWPATH_ID
ORDER BY AWTHREAD_ID, AWPATH_ID

Plan hash value: 1898065720

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>1165 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT GROUP BY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_PV_REQ_AW</td>
<td>1</td>
<td>10</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td>1164 (2)</td>
<td>00:00:14</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>PS_SAC_AW_STEPINST</td>
<td>167</td>
<td>10354</td>
<td>663 (4)</td>
<td>00:00:08</td>
</tr>
<tr>
<td>5</td>
<td>INDEX RANGE SCAN</td>
<td>PS_PV_REQ_AW</td>
<td>1</td>
<td></td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Query Block Name / Object Alias (identified by operation id):

```
1 - SEL$5DA710D3
2 - SEL$5DA710D3 / PS_PV_REQ_AW@SEL$2
4 - SEL$5DA710D3 / PS_SAC_AW_STEPINST@SEL$1
5 - SEL$5DA710D3 / PS_PV_REQ_AW@SEL$2
```

Outline Data

```
/*+ BEGIN_OUTLINE_DATA
  IGNORE_OPTIM_EMITTED_HINTS
  OPTIMIZER_FEATURES_ENABLE('10.2.0.4')
  OPT_PARAM('_b_tree_bitmap_plans' 'false')
  OPT_PARAM('_complex_view_merging' 'false')
  OPT_PARAM('_unnest_subquery' 'false')
  OPT_PARAM('optimizer_dynamic_sampling' 4)
  ALL_ROWS
  OUTLINE_LEAF('@SEL$5DA710D3')
  UNNEST('@SEL$2')
  OUTLINE('@SEL$1')
  OUTLINE('@SEL$2')
  FULL('@SEL$5DA710D3' "PS_SAC_AW_STEPINST" '@SEL$1')
  INDEX('@SEL$5DA710D3' "PS_PV_REQ_AW" '@SEL$2' ("PS_PV_REQ_AW"."AWTHREAD_ID")
```
```
"PS_PV_REQ_AW"."AWPRCS_ID")
LEADING (@"SEL$5DA710D3" "PS_SAC_AW_STEPINST" @"SEL$1" "PS_PV_REQ_AW" @"SEL$2")
USE_NL (@"SEL$5DA710D3" "PS_PV_REQ_AW" @"SEL$2")
END_OUTLINE_DATA
*/

Peeked Binds (identified by position):
--------------------------------------
1 - :1 (VARCHAR2(30), CSID=31): 'Requisition'
2 - :2 (VARCHAR2(30), CSID=31): 'SHARE'
3 - :3 (VARCHAR2(30), CSID=31): '1901-01-01'
4 - :4 (VARCHAR2(30), CSID=31): '5'
5 - :5 (VARCHAR2(30), CSID=31): 'F'

Note
-----
- dynamic sampling used for this statement
This is a plan I collected with EXPLAIN PLAN FOR and dbms_xplan.display. Same plan, but different cost. The cost in the plan produced by DISPLAY_AWR is the cost when the statement was first captured by AWR.

```
Plan hash value: 1898065720

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT STATEMENT</td>
<td></td>
<td>3</td>
<td>216</td>
<td>136K (1)</td>
<td>00:27:16</td>
</tr>
<tr>
<td>1</td>
<td>SORT GROUP BY</td>
<td></td>
<td>3</td>
<td>216</td>
<td>136K (1)</td>
<td>00:27:16</td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PS_PV_REQ_AW</td>
<td>1</td>
<td>10</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td>3</td>
<td>216</td>
<td>136K (1)</td>
<td>00:27:16</td>
</tr>
<tr>
<td>* 4</td>
<td>TABLE ACCESS FULL</td>
<td>PS_SAC_AW_STEPINST</td>
<td>45158</td>
<td>2734K</td>
<td>667 (4)</td>
<td>00:00:09</td>
</tr>
<tr>
<td>* 5</td>
<td>INDEX RANGE SCAN</td>
<td>PS_PV_REQ_AW</td>
<td>1</td>
<td>2</td>
<td>0 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - filter("PARENT_THREAD"=601330)
4 - filter("STAGE_NBR"=TO_NUMBER(:4) AND "AWSTEP_STATUS"<>:5 AND "AWPRCS_ID"=:1 AND "SETID"=:2 AND "EFFDT"=TO_DATE(:3,'YYYY-MM-DD'))
5 - access("AWTHREAD_ID"="AWTHREAD_ID")
```

Sometimes, when I use explain plan for I don’t get the same plan. That is a bit of an alarm bell, but I can force the same plan by using the profile of hints in the plan produced by DISPLAY_AWR

**Statement not in Library Cache**

In an active system, especially one that routinely doesn’t use bind variables, statements will get aged out of the library cache.

```
SELECT * FROM table(dbms_xplan.display_cursor('gpdwr389mg61h',0,'ADVANCED'));
```

Try looking in AWR with the dbms_xplan.display_awr function. You may still not find it because it had already been aged out at the time of the AWR snapshot. If you do find it remember that the costs could be old.
Only Some Statements are in the Library Cache

You’ve seen examples where literal values mean that each statement is different. So we aggregate by sql_plan_hash_value. This is a different variant on the theme. The innermost query sums time by SQL_ID and SQL_PLAN_HASH_VALUE, but we also outer join to DBA_HIST_SQLTEXT to see if we have captured the SQL text and plan.

Then I use an analytic function to find the rank statement within each execution plan, but notice I am ranking by time for statements in the AWR repository.

I still want the plans which have the most time.

```
Select *
FROM
  (SELECT
   ROW_NUMBER() OVER (PARTITION BY x.sql_plan_hash_value ORDER BY x.awr_secs DESC) as ranking,
   x.sql_id, x.sql_plan_hash_value,
   SUM(x.ash_secs) OVER (PARTITION BY x.sql_plan_hash_value) tot_ash_secs,
   SUM(x.awr_secs) OVER (PARTITION BY x.sql_plan_hash_value) tot_awr_secs,
   COUNT(DISTINCT sql_id) OVER (PARTITION BY x.sql_plan_hash_value) sql_ids
  FROM
    (SELECT
     h.sql_id, h.sql_plan_hash_value,
     SUM(10) AS ash_secs,
     10*COUNT(t.sql_id) AS awr_secs
    FROM dba_hist_snapshot x
    LEFT OUTER JOIN dba_hist_active_sess_history h
    ON t.sql_id = h.sql_id
    WHERE x.end_interval_time >= TO_DATE('201003080830','yyyymmddhh24mi')
    AND x.begin_interval_time <= TO_DATE('201003081200','yyyymmddhh24mi')
    AND h.sample_time >= TO_DATE('201003080830','yyyymmddhh24mi')
    AND h.sample_time <= TO_DATE('201003081200','yyyymmddhh24mi')
    AND h.snap_id = x.snap_id
    AND h.dbid = x.dbid
    AND h.instance_number = x.instance_number
    AND h.module = 'WMS_RUN_TADM'
    GROUP BY h.sql_id, h.sql_plan_hash_value
  ) x
  ) y
WHERE y.ranking = 1
ORDER BY tot_ash_secs DESC, ranking
/
```

27 I am using ROW_NUMBER not rank because I want an arbitrary ranked first statement, not all the equally first statements.

28 So here I am counting time for statement in the ASH repository.

29 Here I am counting time for statements all found in the AWR repository.
So now, I know that I can get plans for the SQL IDs with non-zero AWR time. There are still some statements for which I can get neither the SQL nor the execution plan.

<table>
<thead>
<tr>
<th>RANKING</th>
<th>SQL_ID</th>
<th>Hash Value</th>
<th>TOT_ASH_SECS</th>
<th>TOT_AWR_SECS</th>
<th>SQL_IDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1wfhpn9k2x3h0</td>
<td>0</td>
<td>7960</td>
<td>4600</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>2wsan9j1pk3j2</td>
<td>1061502179</td>
<td>4230</td>
<td>4230</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>bnxddum0rryv9h</td>
<td>918066299</td>
<td>2640</td>
<td>1200</td>
<td>179</td>
</tr>
<tr>
<td>4</td>
<td>02cyzmnyt4mdh</td>
<td>508527075</td>
<td>2070</td>
<td>0</td>
<td>45 30</td>
</tr>
<tr>
<td>5</td>
<td>5m0xbf7vn8490</td>
<td>2783301143</td>
<td>1700</td>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td>6</td>
<td>0jfp0g054cb3n</td>
<td>4135405048</td>
<td>1500</td>
<td>0</td>
<td>47</td>
</tr>
<tr>
<td>7</td>
<td>11bygm2nykh0s</td>
<td>3700906241</td>
<td>1370</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>02cymzmyt4mdh</td>
<td>508527075</td>
<td>2070</td>
<td>0</td>
<td>45</td>
</tr>
</tbody>
</table>

I can do the usual trick of generating the commands to get the SQL

```sql
SELECT 'SELECT /*' || tot_awr_secs || ',' || tot_ash_secs || '*/;' FROM
((SELECT ROW_NUMBER() OVER (PARTITION BY x.sql_plan_hash_value ORDER BY x.awr_secs DESC) AS ranking,
  x.sql_id, x.sql_plan_hash_value,
  SUM(x.ash_secs) OVER (PARTITION BY x.sql_plan_hash_value) tot_ash_secs,
  SUM(x.awr_secs) OVER (PARTITION BY x.sql_plan_hash_value) tot_awr_secs,
  COUNT(DISTINCT sql_id) OVER (PARTITION BY x.sql_plan_hash_value) sql_ids
  FROM
  (SELECT h.sql_id,
       h.sql_plan_hash_value,
       SUM(10) ash_secs,
       10*count(t.sql_id) awr_Secs
    FROM dba_hist_snapshot x
    LEFT OUTER JOIN dba_hist_active_sess_history h ON t.sql_id = h.sql_id
    WHERE x.end_interval_time >= TO_DATE('201003080830','yyyymmddhh24mi')
    AND x.begin_interval_time <= TO_DATE('201003081200','yyyymmddhh24mi')
    AND h.sample_time >= TO_DATE('201003080830','yyyymmddhh24mi')
    AND h.sample_time <= TO_DATE('201003081200','yyyymmddhh24mi')
    AND h.snap_id = x.snap_id
    AND h.dbid = x.dbid
    AND h.instance_number = x.instance_number
    AND h.module = 'WMS_RUN TADM'
    GROUP BY h.sql_id, h.sql_plan_hash_value ) y)
WHERE y.ranking = 1
```

30 So we had 207 samples, representing 2070 seconds of SQL for statement with this execution plan. There are 45 distinct SQL_IDs, we don’t know how many executions we are talking about, it is probably one per SQL_ID, but I don’t know that until 11g.
Lots of Shortlived Non-Shareable SQL

I have done the usual query to sum the time by SQL_ID, and I get one row per SQL ID, so instead I will GROUP BY plan hash value. So the SQL is different every time, but quite similar because they share plan hash values.

We are working from AWR history, so one sample every 10 seconds. We get one sample for each SQL_ID. So clearly I have lots of similar but different statements that don’t take very long. I imagine a loop with literal values instead of bind variables!

<table>
<thead>
<tr>
<th>PRCSINSTANCE</th>
<th>NUM_SQL_ID</th>
<th>SQL_PLAN_HASH_VALUE</th>
<th>EXEC_SECS</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>50007687</td>
<td>169</td>
<td>953836181</td>
<td>3170</td>
<td>1690</td>
</tr>
<tr>
<td>50007687</td>
<td>50</td>
<td>807301148</td>
<td>3170</td>
<td>500</td>
</tr>
<tr>
<td>50007687</td>
<td>22</td>
<td>4034059499</td>
<td>3170</td>
<td>220</td>
</tr>
<tr>
<td>50007687</td>
<td>14</td>
<td>2504475139</td>
<td>3170</td>
<td>140</td>
</tr>
<tr>
<td>50007687</td>
<td>2</td>
<td>0</td>
<td>3170</td>
<td>70</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>1309703960</td>
<td>3170</td>
<td>20</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>3230852326</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>3257716453</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>3852975016</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>3205663729</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>2791534567</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>2098696903</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>1880529843</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>1173536273</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>1089066969</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>301402716</td>
<td>3170</td>
<td>10</td>
</tr>
</tbody>
</table>
Actually, I can get the execution plan for any of these statements in the AWR history, so in
this variant of the query I have joined to DBA_HIST_SQLTEXT to see which SQL_IDs I do
have information for (I can switch that to a left outer join to get back to the usual behaviour).

```
SELECT /*+LEADING(r x h) USE_NL(h)*/  
   r.prcsinstance  
   , COUNT(DISTINCT h.sql_id) num_sql_id  
   , h.sql_plan_hash_value  
   , (CAST(r.enddttm AS DATE)-CAST(r.begindttm AS DATE))*86400 exec_secs  
   , SUM(10) ash_secs  
FROM dba_hist_snapshot x  
   , dba_hist_active_sess_history h  
   INNER /*LEFT OUTER*/ JOIN DBA_HIST_SQLTEXT q  
   ON q.dbid = h.dbid AND q.sql_id = h.sql_id  
   , sysadm.pspcrsrqst r  
WHERE x.end_interval_time >= r.begindttm  
   AND x.begin_interval_time <= r.enddttm  
   AND h.sample_time BETWEEN r.begindttm AND r.enddttm  
   AND h.snap_id = x.snap_id  
   AND h.dbid = x.dbid  
   AND h.instance_number = x.instance_number  
   AND h.module = r.prcsname  
   AND h.action LIKE 'PI'||r.prcsinstance||'%'  
   AND r.prcsinstance = 50007687  
GROUP BY r.prcsinstance, r.prcsname, r.begindttm, r.enddttm  
   , h.sql_plan_hash_value  
ORDER BY ash_secs DESC
```

So the few that I have a plan for, are not very significant.

<table>
<thead>
<tr>
<th>PRCSINSTANCE</th>
<th>NUM_SQL_ID</th>
<th>SQL_PLAN_HASH_VALUE</th>
<th>EXEC_SECS</th>
<th>ASH_SECS</th>
</tr>
</thead>
<tbody>
<tr>
<td>50007687</td>
<td>1</td>
<td>0</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>3205663729</td>
<td>3170</td>
<td>10</td>
</tr>
<tr>
<td>50007687</td>
<td>1</td>
<td>2791534567</td>
<td>3170</td>
<td>10</td>
</tr>
</tbody>
</table>
This is the Application Engine batch timings report for the same process. ASH suggests that the top execution plan had 169 executions, but remember that is a sample every 10 seconds.

The truth is much worse. The batch timings say there is a step that is executed 64224 times. It took 2566 seconds, so that is only 40ms per execution. So I am only sampling 1 in 250 executions, so no wonder I don’t have many of them in the AWR repository. They are getting aged out too quickly.

It was also compiled 64224 times, and that tells me that this step does not have reuse statement, possible because there is dynamic SQL in play.

<table>
<thead>
<tr>
<th>Batch Timings - Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
</tr>
<tr>
<td>Instance: 50007687</td>
</tr>
<tr>
<td>Name: AR_CNDMON</td>
</tr>
<tr>
<td>Type: Application Engine</td>
</tr>
<tr>
<td>Description: Receivables Condition Monitor</td>
</tr>
<tr>
<td>Time (in milliseconds)</td>
</tr>
<tr>
<td>Elapsed: 3164110</td>
</tr>
<tr>
<td>In PeopleCode: 30600</td>
</tr>
<tr>
<td>In SQL: 2940990</td>
</tr>
<tr>
<td>Trace Level</td>
</tr>
<tr>
<td>Application Engine: 1159</td>
</tr>
<tr>
<td>SQL &amp; PeopleCode: 128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program</th>
<th>Compile Count</th>
<th>Compile Time</th>
<th>Execute Count</th>
<th>Execute Time</th>
<th>Fetch Count</th>
<th>Fetch Time</th>
<th>PC Count</th>
<th>PC Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR_CNDMON CHK_USER.INSPRC2.B</td>
<td>64224</td>
<td>39660</td>
<td>84224</td>
<td>2566340</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AR_CNDMON CHK_USER.LDSQLS</td>
<td>64224</td>
<td>9230</td>
<td>84224</td>
<td>230220</td>
<td>84224</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AR_CNDMON CANCLACT.CANBLIST3.B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>18010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AR_CNDMON ABRULES.LOADRULES.B</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>15820</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AR_CNDMON ABRULES.DELMACC.B</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>2710</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AR_CNDMON DEPACTIVE.INSTNR.B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>7600</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

I could criticise the kind of programming that leads to this, but it also shows a scenario where ASH will be of limited benefit.

This is a situation where I might want to use SQL trace to see what is going on in these statements. On the other hand, 40ms isn’t bad for a SQL statement, how much faster can I make it.
### Error ORA-06502

I have no idea why `display_awr` produces ORA-6502, but sometimes it does. It seems to be something to do with very large SQL statements. But you still get the execution plan.

```sql
SELECT * FROM table(dbms_xplan.display_awr('9vnan5kqsh1aq', 2262951047,NULL,'ADVANCED'));
```

**SQL_ID 9vnan5kqsh1aq**

An uncaught error happened in `prepare_sql_statement`: ORA-06502: PL/SQL: numeric or value error

Plan hash value: 2262951047

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td></td>
<td>164</td>
<td>1 (100)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

The text is there, so you can go get it FROM the AWR cache yourself.

```sql
SELECT sql_text FROM dba_hist_sqltext where sql_id = '9vnan5kqsh1aq'
```

### Error ORA-01422

Sometimes, `dbms_xplan` fails because there are two SQL statements with the same SQL_ID.

An uncaught error happened in `prepare_sql_statement`: ORA-01422: exact fetch returns more than requested number of rows

This usually happens because the database has been cloned (from Production) and renamed, and then the same SQL statement has been captured by an AWR snapshot. The answer is to delete at least the duplicate rows from `sys.wrh$sqltext`.

```sql
delete from sys.wrh$_sqltext t1
where t1.dbid != (select d.dbid from v$database d)
and exists(select 'x'
  from sys.wrh$_sqltext t2
  where t2.dbid = (select d.dbid from v$database d)
  and t2.sql_id = t1.sql_id)
```
Error ORA-44002

I have seen this with Global Temporary Tables and with direct path mode (the APPEND hint).

```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------
ERROR: cannot get definition for table 'BZTNCMUX31XP5'
ORA-44002: invalid object name
```
Appendix

Further reading

- **Sifting through the ASHes**, Graham Wood, Oracle (http://www.oracle.com/technology/products/manageability/database/pdf/twp03/PPT_active_session_history.pdf)

  - And you can watch the video of Graham giving this presentation at MOW2009 on the Oracle Table Website
  - [http://www.oaktable.net/media/mow2010-graham-wood-ashes-time-part1](http://www.oaktable.net/media/mow2010-graham-wood-ashes-time-part1)
  - [http://www.oaktable.net/media/mow2010-graham-wood-ashes-time-part-2](http://www.oaktable.net/media/mow2010-graham-wood-ashes-time-part-2)

- Doug Burns has written some excellent material many subjects including ASH on his Oracle Blog (http://oraclerouge.com/serendipity/index.php?/plugin/tag/ASH).

- Introduction to DBMS_XPLAN (http://www.go-faster.co.uk/Intro_DBMS_XPLAN.ppt), UKOUG2008
  - With acknowledgements to 10g/11g DBMS_XPLAN, Carol Dacko, Collaborate 08