How to Identify Resource Intensive SQL ("TOP SQL") (Doc ID 232443.1)

In this Document

APPLIES TO:

Oracle Database - Personal Edition - Version 9.2.0.1 and later
Oracle Database - Standard Edition - Version 9.2.0.1 and later
Oracle Database - Enterprise Edition - Version 9.2.0.1 and later
Information in this document applies to any platform.

GOAL

This article provides guidance on how to identify the most resource intensive SQL statements in a database for tuning purposes.

If there is a performance issue related to the performance of SQL, it may be that one or two statements are consuming the majority of the resources or it could be that a lot of SQL statements consume small amounts of resources adding up to a large amount. In terms of tuning it is easier to handle the first case with less statements to examine. There are a number of places that you can look to find the highest resource users and how you approach this depends on the area where you are seeing the problem. If the resource issue is high CPU usage then focus on the highest CPU users. If it is I/O then focus on the highest under that criteria.

The techniques described here can be used when initial diagnosis of a database performance problem suggests that further investigation needs to concentrate on finding and tuning the most resource intensive SQL statements according to specific criteria, e.g.

- Using the most CPU
- Performing the most disk I/O operations
- Having the most executions
- Taking the longest time to execute (elapsed time)

As with any performance issue, once you have identified and tuned a statement, re-test the criteria looking for improvements. If your goal for this statement/overall is met then stop! If your goal is not met then re-tune. If your goal for the statement is met but the overall performance is still not as desired, look for the highest resource user and tune that.

The article should be of use to Database Administrators, Support Engineers, Consultants and Database Performance Analysts.

SOLUTION

When to Look For Resource Intensive SQL Statements

In this section we discuss briefly the reasons that would lead us to look for the most resource intensive SQL statements while investigating a database performance problem.

- ***Response Time Analysis shows that heaviest time consumer is CPU-related (e.g. CPU Other or CPU Parse time) or an I/O-related Wait Event (e.g. db file sequential read or db file scattered read):***

  Tools such as AWR or statspack provide comprehensive information from a database that shows in detail what type of activities time is being spent upon.

Response time analysis is based on the following equation:

```
Response Time = Service Time + Wait Time
```

where Service Time is time spent on the CPU and Wait Time is the sum of time spent on Wait Events (i.e. non-idle time spent waiting for an event to complete or for a resource to become available).

Service Time

Service Time is comprised of time spent on the CPU for Parsing, Recursive CPU usage (for PLSQL and recursive SQL) and CPU used for execution of SQL statements (CPU Other). So

```
Service Time = CPU Parse + CPU Recursive + CPU Other
```

The components of Service Time can be found from the following statistics:

- Service Time - "CPU used by this session"
- CPU Parse - "parse time cpu"
- CPU Recursive -" recursive cpu usage"

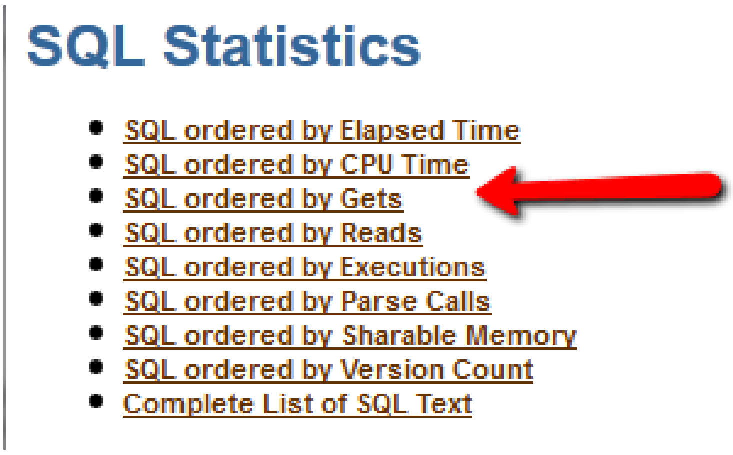
CPU Other can be calculated from these as follows:

```
CPU Other = CPU used by this session - parse time cpu -
recursive cpu usage
```

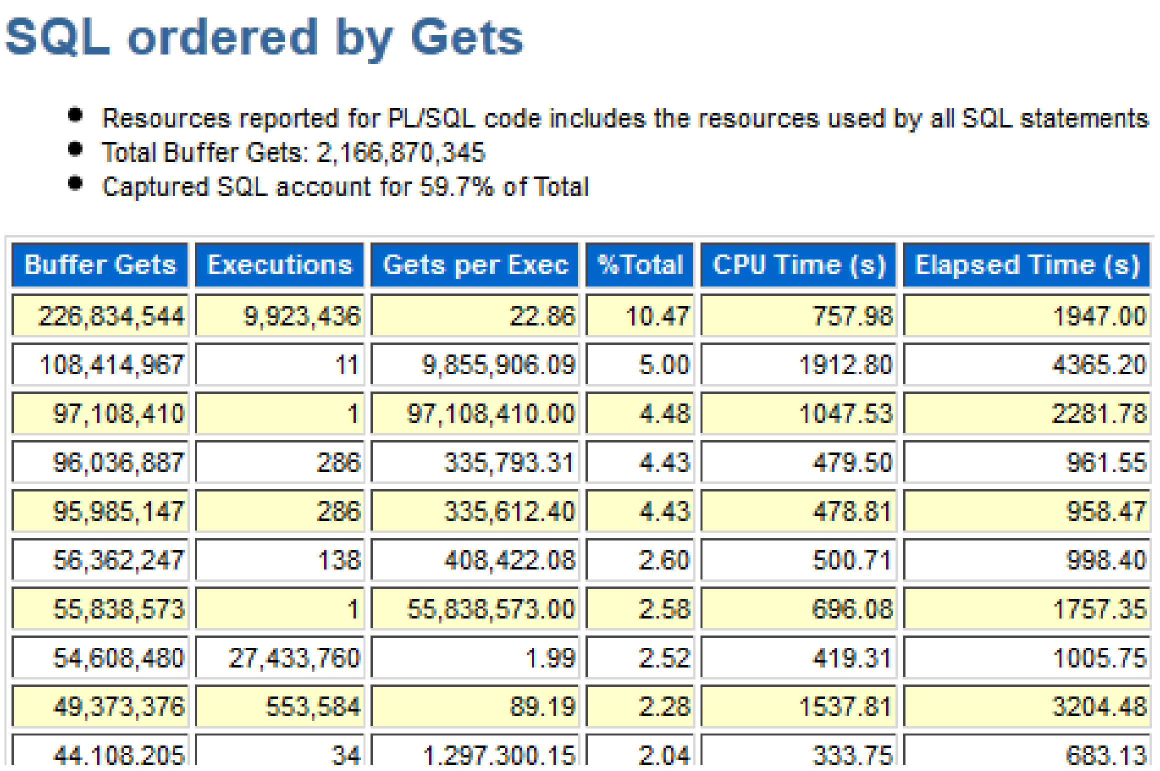
When "CPU Other" is a significant component of total Response Time:

When "CPU Other" is a significant component of total Response Time, then it is likely that the time is being spent retrieving and manipulating Blocks and Buffers (Block accesses are also known as Buffer Gets and Logical I/Os). So the next step is to find the SQL statements that access the most blocks because these are likely to be responsible for the majority of this time.

AWR and Statspack list such SQL statements in sections such as "SQL ordered by Gets." For example:


SQL Statistics

- SQL ordered by Elapsed Time
- SQL ordered by CPU Time
- SQL ordered by Gets
- SQL ordered by Reads
- SQL ordered by Executions
- SQL ordered by Parse Calls
- SQL ordered by Sharable Memory
- SQL ordered by Version Count
- Complete List of SQL Text



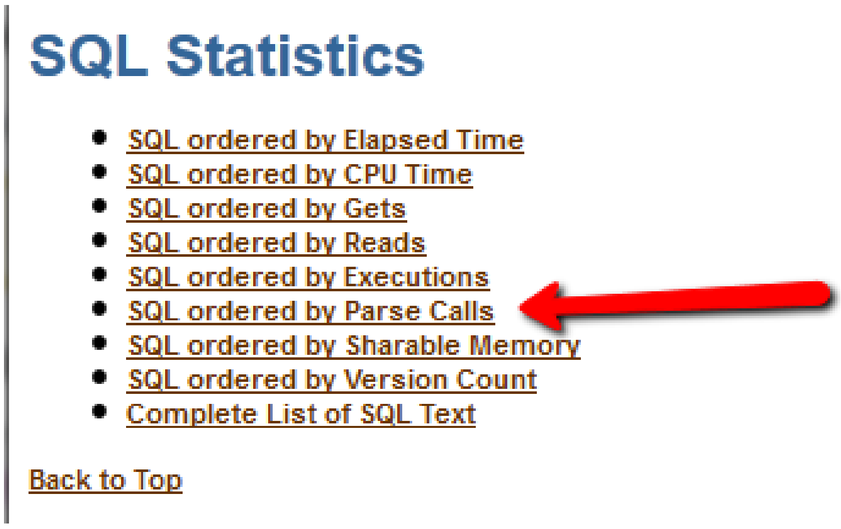
SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements
- Total Buffer Gets: 2,166,870,345
- Captured SQL account for 59.7% of Total

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)
226,834,544	9,923,436	22.86	10.47	757.98	1947.00
108,414,967	11	9,855,906.09	5.00	1912.80	4365.20
97,108,410	1	97,108,410.00	4.48	1047.53	2281.78
96,036,887	286	335,793.31	4.43	479.50	961.55
95,985,147	286	335,612.40	4.43	478.81	958.47
56,362,247	138	408,422.08	2.60	500.71	998.40
55,838,573	1	55,838,573.00	2.58	696.08	1757.35
54,608,480	27,433,760	1.99	2.52	419.31	1005.75
49,373,376	553,584	89.19	2.28	1537.81	3204.48
44,108,205	34	1,297,300.15	2.04	333.75	683.13

When "CPU Parse" is a significant component of total Response Time:

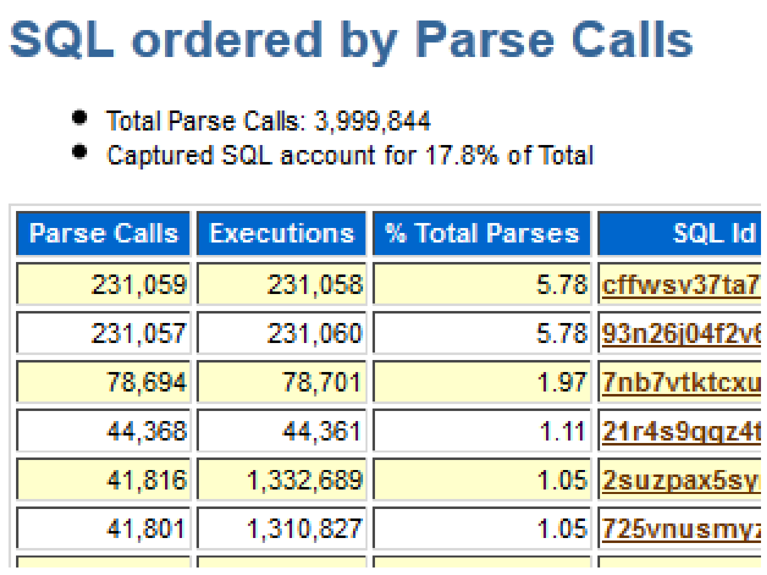
When "CPU Parse" is a significant component of total Response Time the next step is to find the SQL statements that have the most parses.
AWR and Statspack list such SQL statements in sections such as "SQL ordered by Parse Calls".



SQL Statistics

- SQL ordered by Elapsed Time
- SQL ordered by CPU Time
- SQL ordered by Gets
- SQL ordered by Reads
- SQL ordered by Executions
- SQL ordered by Parse Calls
- SQL ordered by Sharable Memory
- SQL ordered by Version Count
- Complete List of SQL Text

Back to Top



SQL ordered by Parse Calls

- Total Parse Calls: 3,999,844
- Captured SQL account for 17.8% of Total

Parse Calls	Executions	% Total Parses	SQL Id
231,059	231,058	5.78	cffwsv37ta7
231,057	231,060	5.78	93n26j04f2v6
78,694	78,701	1.97	7nb7vtktcxu
44,368	44,361	1.11	21r4s9qqz4t
41,816	1,332,689	1.05	2suzpax5sy
41,801	1,310,827	1.05	725vnusmy

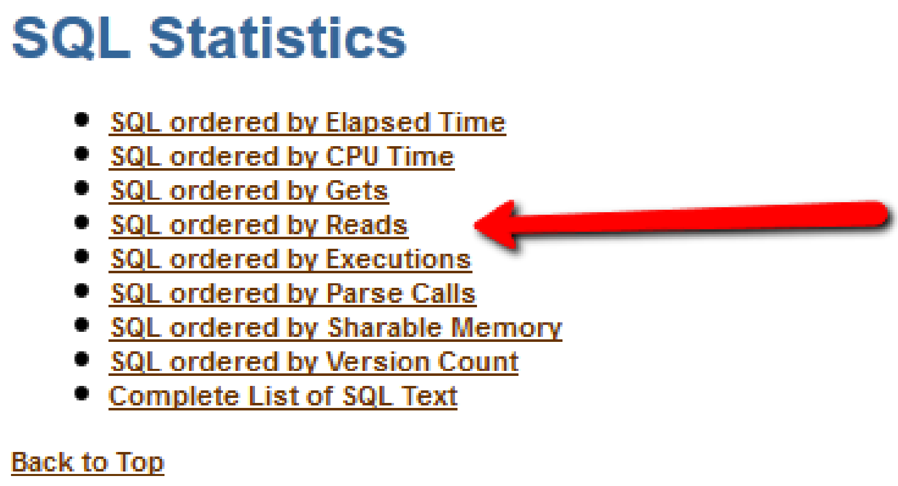
Wait Time

"Wait Time" is the sum of time waited for non-idle Wait Events.
These include I/O waits for reading blocks from disk as measured by the Wait Events 'db file sequential read' for single-block reads and 'db file scattered read' for multi-block reads.

When I/O Wait Events are found to be significant components of Response Time:

When such Wait Events are found to be significant components of Response Time, the next step is to find the SQL statements that read the most blocks from disk.

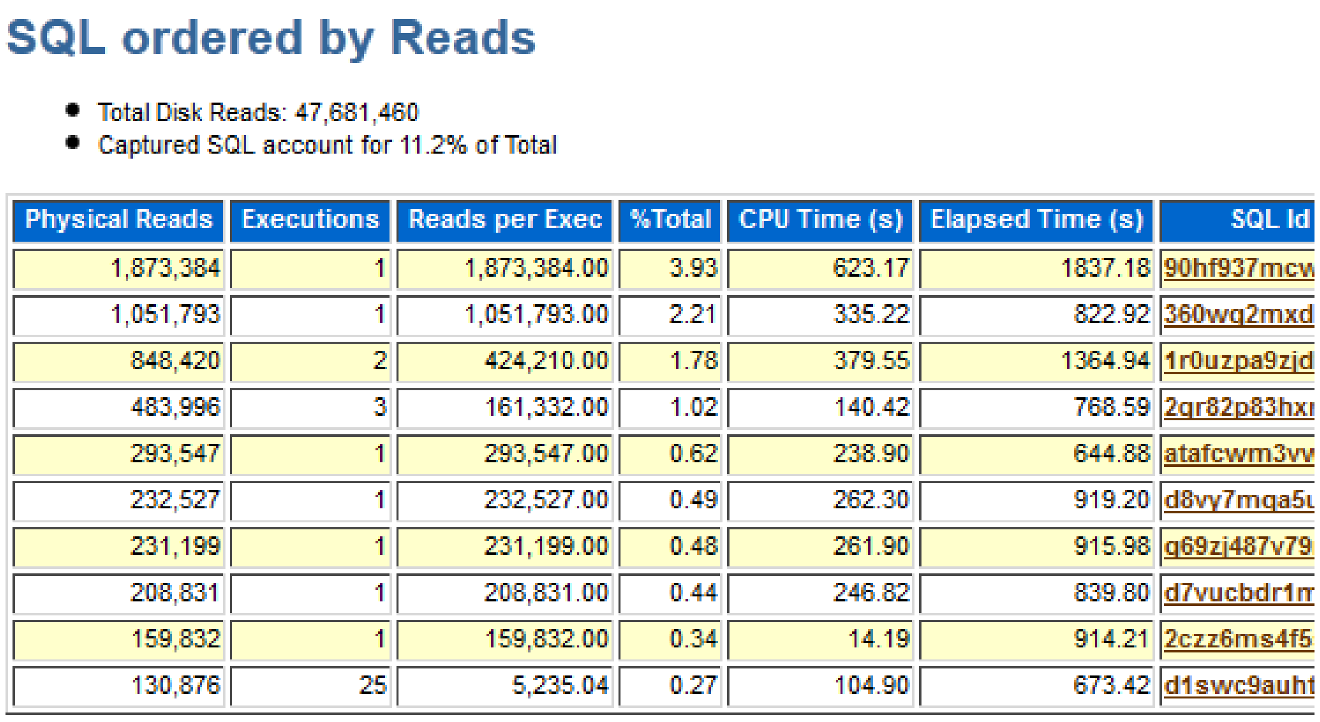
AWR and Statspack lists such SQL statements in sections such as "SQL ordered by Reads".



SQL Statistics

- SQL ordered by Elapsed Time
- SQL ordered by CPU Time
- SQL ordered by Gets
- SQL ordered by Reads
- SQL ordered by Executions
- SQL ordered by Parse Calls
- SQL ordered by Sharable Memory
- SQL ordered by Version Count
- Complete List of SQL Text

Back to Top



SQL ordered by Reads

- Total Disk Reads: 47,681,460
- Captured SQL account for 11.2% of Total

Physical Reads	Executions	Reads per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id
1,873,384	1	1,873,384.00	3.93	623.17	1837.18	90hf937mcw
1,051,793	1	1,051,793.00	2.21	335.22	822.92	360wq2mxd
848,420	2	424,210.00	1.78	379.55	1364.94	1r0uzpa9zjd
483,996	3	161,332.00	1.02	140.42	768.59	2qr82p83hxr
293,547	1	293,547.00	0.62	238.90	644.88	atafcwm3vv
232,527	1	232,527.00	0.49	262.30	919.20	d8vy7mqa5u
231,199	1	231,199.00	0.48	261.90	915.98	g69zj487v79
208,831	1	208,831.00	0.44	246.82	839.80	d7vucbdr1m
159,832	1	159,832.00	0.34	14.19	914.21	2czz6ms4f5
130,876	25	5,235.04	0.27	104.90	673.42	d1swc9auht

General Example from Statspack: (pre-Oracle9i Release 2)

Here is an example where CPU Other was found to be a significant component of total Response Time:

```
Top 5 Wait Events
~~~~~~~~~~~~~~~~~                                   Wait     % Total
Event                                 Waits  Time (cs)    Wt Time
---------------------------- ------------ ------------ -------
direct path read                      4,232        10,827   52.01
db file scattered read                6,105         6,264   30.09
direct path write                     1,992         3,268   15.70
control file parallel write             893           198     .95
db file parallel write                   40           131     .63
-------------------------------------------------------------

Statistic                                      Total   per Second    per Trans
--------------------------------- ---------------- ------------ ------------
CPU used by this session                     358,806        130.5     12,372.6
parse time cpu                                    38          0.0          1.3
recursive cpu usage                          186,636         67.9      6,435.7
```

From these figures we can obtain:

- *Wait Time = 10,827 x 100% / 52,01% = 20,817 cs*
- *Service Time = 358,806 cs*
- *Response Time = 358,806 + 20,817 = 379,623 cs*
- *CPU Other = 358,806 - 38 - 186,636 = 172,132 cs*

If we now calculate percentages for the top Response Time components:

- *CPU Other = 45.34%*
- *CPU Recursive = 49.16%*
- *direct path read = 2.85%*
- *etc. etc.*

"CPU Other" is a significant component of Response Time, so a possible next step is to look at the SQL ordered by Gets section.

General Example from Statspack: (Oracle9i Release 2 & above)

Starting with Oracle9i Release 2, Statspack presents Service Time (obtained from the statistic CPU used by this session) together with the top Wait Events in a section called Top 5 Timed Events, which replaces the section Top 5 Wait Events of previous releases.

Here is an example:

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~                                                    % Total
Event                                               Waits    Time (s) Ela Time
-------------------------------------------- ------------ ----------- --------
library cache lock                                    141         424    76.52
db file scattered read                              3,367          96    17.40
CPU time                                                           32     5.79
db file sequential read                               161           1      .18
control file parallel write                           245           0      .05
-------------------------------------------------------------------------------

Statistic                                      Total     per Second    per Trans
--------------------------------- ------------------ -------------- ------------
CPU used by this session                       3,211            4.3      1,605.5
parse time cpu                                    59            0.1         29.5
recursive cpu usage                              232            0.3        116.0
```

These figures give us directly the percentages of the Wait Events against the total Response Time so no further calculations are necessary to assess the impact of Wait Events. Service Time is presented as CPU time in this section and corresponds to the total CPU utilisation. We can drill down to the various components of Service Time as follows:

- *CPU Other = 3,211 - 59 - 232 = 2,920 cs*
- *CPU Other = 2,920 / 3,211 x 5.79% = 5.26%*
- *CPU Parse = 59 / 3,211 x 5.79% = 0.11%*
- *CPU Recursive = 232 / 3,211 x 5.79% = 0.42%*

In this example, the main performance problem was an issue related to the Library Cache. The second most important time consumer was waiting for physical I/O due to multiblock reads (db file scattered read).
In this case a possible approach would be to look at the "SQL ordered by Reads" section of Statspack.

- ***Operating System resource analysis shows that excessive CPU or Disk I/O usage is caused by one or more Oracle processes:***

If Operating System utilities show that most CPU usage is due to a small number of Oracle processes then typically they will be SQL Traced and have TKPROF reports generated from their tracefiles. Analysis of the TKPROF reports will lead to the most time consuming SQL statements which will be the subject of subsequent tuning.

If CPU usage is excessive but spread out fairly evenly among all the Oracle processes on the system then typically a tool such as Statspack will be used to perform Response Time analysis. The components of "Service Time" will be evaluated and if this shows CPU Other as being significant the next step will be to look at the SQL performing most block accesses in the SQL by Gets section of the Statspack report.

If Disk I/O is highly utilized on the system then a Response Time analysis using Statspack can be done to confirm that the Disk I/O usage is due to Oracle processes and I/O-related Wait Events can be expected to be significant components of overall "Response Time". SQL statements performing the most physical reads can then be found in the section"SQL ordered by Reads".

- ***Investigation of the performance of a batch job or other session which performs a number of SQL statements and has slow performance:***

  This will be done with SQL Trace and TKPROF as descrived above and the most important SQL statements in terms of time spent will be identified for further tuning.

- ***Examination of statistics stored for SQL statements in V$ dynamic views:***

  Part of the information stored with a SQL statement (Shared Cursor) in the Library Cache portion of the Shared Pool are a number of statistics related to its execution. These are available through the V$SQLAREA dynamic view and can be queried to monitor the most expensive SQL statements in each category.

  This is the approach used by the graphical tool SQL Analyze in the Oracle Enterprise Manager Tuning Pack.

  For more information on the topics discussed in this section please refer to:

  Document 190124.1 THE COE PERFORMANCE METHOD
  Document 228913.1 Systemwide Tuning using STATSPACK Reports:
  Document 223117.1 Tuning I/O-related waits

Top SQL Statements in AWR Reports

AWR reports (and statspack reports generated from snapshots at level 5 (default) and above) can contain SQL reporting sections for the following types of resource intensive SQL statements. The relevant reports are collected inthe SQL Statistics section:

SQL Statistics

Back to Top

Descriptions of some of the sections is below:

- ***SQL ordered by Gets***

  This section shows SQL statements with most buffer accesses.

  Statspack information:
  The default threshold value is 10,000 buffer gets.
  Columns displayed are:
  Buffer Gets, Executions, Gets per Exec, % Total, Hash Value (8.1.7)
  Buffer Gets, Executions, Gets per Exec, %Total, CPU Time (s), Elapsed Time (s), Hash Value (9.0.1 & 9.2)

- ***SQL ordered by Reads***

  containing the SQL statements with most read I/Os from disk.

  Statspack information:
  The default threshold value is 1,000 disk reads.
  Columns displayed are:
  Physical Reads, Executions, Reads per Exec, % Total, Hash Value (8.1.7)
  Physical Reads, Executions, Reads per Exec, %Total, CPU Time (s), Elapsed Time (s), Hash Value (9.0.1 & 9.2)

- ***SQL ordered by Executions***

  containing the SQL statements executed the most times.

  Statspack information:
  The default threshold value is 100 executions.
  Columns displayed are:
  Executions, Rows Processed, Rows per Exec, Hash Value (8.1.7)
  Executions, Rows Processed, Rows per Exec, CPU per Exec (s), Elapsed per Exec (s), Hash Value (9.0.1 & 9.2)

- ***SQL ordered by Parse Calls (Oracle9i and above)*** containing the SQL statements with most soft parse calls.

Statspack information:
The default threshold value is 1,000 parse calls.
Columns displayed are:
Parse Calls, Executions, % Total Parses, Hash Value

- ***SQL ordered by Sharable Memory***

  containing the SQL statements occupying the most Library Cache memory.

  Statspack information:
  The default threshold value is 1Mb (1,048,576 bytes).
  Columns displayed are:
  Sharable Mem (b), Executions, % Total, Hash Value (8.1.7, 9.0.1 & 9.2)

- ***SQL ordered by Version Count***

  containing the SQL statements with most versions (children cursors).

  Statspack information:
  The default threshold value is 20 versions.
  Columns displayed are:
  Version Count, Executions, Hash Value (8.1.7, 9.0.1 & 9.2)

  In the next few sections we look at examples of each type of Statspack SQL reporting section.

Top SQL Statements in Statspack Reports

Finding SQL with High CPU Other Time in Statspack

Here are a few examples of "SQL ordered by Gets" sections from Statspack.

```
SQL ordered by Gets for DB: PROD  Instance: prod  Snaps: 8 -9
-> End Buffer Gets Threshold:   10000
-> Note that resources reported for PL/SQL includes the resources used by
   all SQL statements called within the PL/SQL code.  As individual SQL
   statements are also reported, it is possible and valid for the summed
   total % to exceed 100

  Buffer Gets    Executions  Gets per Exec  % Total  Hash Value
--------------- ------------ -------------- -------- ------------
     91,938,671        4,249       21,637.7    24.1   3503723562
SELECT  "TEKSTI_IK", "U_VERSION", "ASIAKIR_IK", "KONTAKT_IK", "L
OMAKE_IK", "PVM",  "TIEDON_LKA", "TEKSTI_VER", "TEKST_TILA", "VA
LK_AUKKO", "SUOR_PA_IK", "SUOR_PA_ID", "RESURSS_IK",  "RESURSS_I

     39,196,483        4,257        9,207.5    10.3    576408779
SELECT  "KONTAKT_IK", "ASIAKAS_IK", "KAYNTIPVM", "KLO_JNRO", "KT
_PIKASEL", "RESURSS_ID", "SUOR_PA_IK",  "SUOR_PA_ID", "TEKSTI_IK
", "KT_TEKSTI", "KONT_LAJI" FROM "TEI1000_VIEW" WHERE (kontakt_i

     31,870,113        4,262        7,477.7     8.3   3583640853
```

```
SELECT  "LAAKE_T_IK", "U_VERSION", "TAPAHTU_IK", "ASIAKAS_IK", "
TOT_DATIM", "LAAKE_IK", "KAUPPANIMI",  "VALM_MUOTO", "VAHVUUS",
"PAKK_KOKO", "ANNOS", "INDIKAATIO", "PYSYVAIS", "VOIM_OLEVA", "S

     30,567,449        4,259        7,177.1     8.0   1901268379
SELECT  "LAB_TUL_IK", "U_VERSION", "TAPAHTU_IK", "ASIAKAS_IK", "
TOT_DATE", "TOT_TIME", "PALVELU_IK",  "PALV_LYHEN", "SL_KOODI",
"OSAVAS_NRO", "OSAVAS_HOP", "VAST_TYYPP", "VAST_ARVO", "VAST_SIJ
```

Here the first SQL statement (with hash value 3503723562) alone accounts for 24.1% of all buffer gets in the instance.
The next 3 statements account for 10.3%, 8.3% and 8.0%.
All 4 statements are executed approximately the same number of times (around 4,250 executions).
The first statement has more total Gets because it fetches more buffers each time it runs (Gets/Exec is 21,637.7 compared to 9,207.5, 7,477.7 and 7,177.1).
So it is a first candidate for tuning as it has greater impact on CPU Other time than the other 3 statements.
A better execution plan for this statement resulting in fewer Gets/Exec will reduce its CPU consumption.

Here is another example:

```
  Buffer Gets    Executions  Gets per Exec  % Total  Hash Value
--------------- ------------ -------------- ------- ------------
      3,200,593            1    3,200,593.0    52.2    397365298
select c.contract_no||'/'||c.contract_version,        c.owner_ag
ency_id,        a.agency_name,        TERRITORYPACK.getsalescont
act(a.agency_id,'SALES','N'),        c.start_date,        LEAST(

        404,024       88,481            4.6     6.6    985453413
select cv_dist_flag from applicant
   where applicant_id = :p1

        178,600        3,028           59.0     2.9   3013728279
select privilege#,level from sysauth$ connect by grantee#=prior
privilege# and privilege#>0 start with (grantee#=:1 or grantee#=
```

The first statement (hash value 397365298) generated 52.2% of the buffer gets in the instance with just 1 execution.
It has a high number of Gets/Exec 3,200,593.0 when compared to the others.
If this statement is not just a one-time query then it is a good candidate for tuning before it starts getting used more often.

The second statement (hash value 985453413) fetches on average less than 5 buffers per execution but appears high in the list because it is executed very frequently.
Initially it is not significant enough to warrant further investigation.
If after tuning the first statement, "CPU Other" is still a significant component of overall "Response Time" and a new Statspack report shows the second statement still high on the list, then it could be looked at more closely.

Here is a similar case, this time it is from Oracle9i and we can see the new CPU & Elapsed Time columns:

```
                                                            CPU      Elapsd
  Buffer Gets    Executions  Gets per Exec  %Total Time (s)  Time (s) Hash Valu
--------------- ------------ -------------- ------ -------- --------- --------
--
     16,177,286            1   16,177,286.0   12.1   209.00    297.91  34258831
select   sf.satz_id, f1.feld, f2.feld feld20,   substr(sf.fehler
,instr(sf.fehler,'geschrieben:')+13) feld30 --merkwï¿½rdigerweise
wird ab 31 Byte in eine neue Zeile geschrieben from   d98_ditr_s

      8,177,886      375,622           21.8    6.1   214.09    302.26 354492189
SELECT /*+ RULE */ *   from d98_schemaeintraege_view   where pro
f_id = :b1   order by sortierung
```

The Statspack report does not always show the full text of the SQL statement. The Hash Value can be used to get this using the following query, provided the SQL statement is still in the Library Cache at the time the query is run:

```
SELECT sql_text
FROM v$sql_text
WHERE hash_value = '&hash_value_of_SQL'
ORDER BY piece;
```

Finding SQL Statements With High CPU Parse Time in Statspack

If "CPU Parse time" is a significant component of Response Time, it can be because cursors are repeatedly opened and closed every time they are executed instead of being opened once, kept open for multiple executions and only closed when they are no longer required.

The "SQL ordered by Parse Calls" can help find such cursors, here is an example:

```
SQL ordered by Parse Calls for DB: DWI1  Instance: DWI1  Snaps: 1 -4
-> End Parse Calls Threshold:      1000

                           % Total
 Parse Calls  Executions   Parses  Hash Value
------------ ------------ -------- ----------
  13,632,745   13,632,745    98.90 3980186470
SELECT distinct TS002.C_JOB_DEP, TS002.C_JOB       FROM TS002_JO
B_DEP TS002, TS001_JOB TS001      WHERE TS001.C_JOB = TS002.C_JO
B_DEP         AND TS002.C_JOB = :b1        AND TS001.C_TIP_JOB !=

      11,701   27,255,840     0.08 3615375148

COMMIT

       8,192        8,192     0.06  238087931
select t.schema, t.name, t.flags, q.name from system.aq$_queue_t
ables t, sys.aq$_queue_table_affinities aft,      system.aq$_que
ues q where aft.table_objno = t.objno and aft.owner_instance = :
1 and         q.table_objno = t.objno and q.usage = 0 and       b
itand(t.flags, 4+16+32+64+128+256) = 0 for update of t.name, aft

       8,192        8,192     0.06 2780709284
      8,192        8,192     0.06 2780709284
select q_name, state, delay, expiration, rowid, msgid,   dequeue
_msgid, chain_no, local_order_no, enq_time, enq_tid, step_no,
priority, exception_qschema, exception_queue, retry_count, corri
d,   time_manager_info, sender_name, sender_address, sender_prot
ocol   from SYS.AQ_SRVNTFN_TABLE   where time_manager_info <= :1
```

The first SQL statement (hash value 3980186470) has had the most parses issued against it (98.90% of all parses in the instance). It is parsed every time it is executed (Parse Calls = Executions). Due to its frequency it is a prime candidate for reducing parse calls as described above.

Note: in excessive parsing situations, it is likely that there will be symptoms such as `latch free waits` on the Library Cache latches and possibly the Shared Pool latch, in addition to CPU Parse time.

Finding SQL Statements With High Disk I/O Waits in Statspack

Identifying SQL statements responsible for most physical reads from the Statspack section "SQL ordered by Reads" has similar concepts as for "SQL ordered by Gets".
% Total can be used to evaluate the impact of each statement.
Reads per Exec together with Executions can be used as a hint of whether the statement has a suboptimal execution plan causing many physical reads or if it is there simply because it is executed often.

Possible reasons for high Reads per Exec are use of unselective indexes require large numbers of blocks to be fetched where such blocks are not cached well in the buffer cache, index fragmentation, large Clustering Factor in index etc.

Here is an example:

```
SQL ordered by Reads for DB: PROD  Instance: prod  Snaps: 14 -16
-> End Disk Reads Threshold:    1000

 Physical Reads  Executions  Reads per Exec % Total  Hash Value

--------------- ------------ -------------- -------- ------------
      3,633,017           48       75,687.9    28.0   3954888792
SELECT  "VAR_RES_IK", "RESURSS_IK", "PVM", "ALKAEN_KLO", "PAATT_
KLO", "AJ_STATUS", "ILMOITT", "HETU", "AS_NIMI", "KASITELTY", "T
OIMIPI_IK", "SUOR_PA_IK", "SUOR_PA_ID", "AIKATYY_IK", "AIKATYY_I

      1,511,867           26       58,148.7    11.6    394819206
SELECT  "VAR_RES_IK", "RESURSS_IK", "PVM", "ALKAEN_KLO", "PAATT_
KLO", "AJ_STATUS", "ILMOITT", "HETU", "AS_NIMI", "KASITELTY", "T
OIMIPI_IK", "SUOR_PA_IK", "SUOR_PA_ID", "AIKATYY_IK", "AIKATYY_I

        762,101            6      127,016.8     5.9   4274178025
SELECT  "LAB_TUL_IK", "PALV_LYHEN", "PALV_TILA", "OSAVAS_HOP", "
VAST_ARVO", "VAST_SIJ", "MITTAYKS", "POIKKEAVA", "PALVELU_IK", "
ASIAKAS_IK", "VERIRYHMA", "VV_MAARPVM", "TOT_DATE", "TOT_TIME",

        512,142            3      170,714.0     3.9   1591034069
SELECT  "LAB_TUL_IK", "PALV_LYHEN", "PALV_TILA", "OSAVAS_HOP", "
SELECT  "LAB_TUL_IK", "PALV_LYHEN", "PALV_TILA", "OSAVAS_HOP", "
VAST_ARVO", "VAST_SIJ", "MITTAYKS", "POIKKEAVA", "PALVELU_IK", "
ASIAKAS_IK", "VERIRYHMA", "VV_MAARPVM", "TOT_DATE", "TOT_TIME",
```

The first two SQL statements are both executed more often than the others and cause more blocks to be read in from disk each time.
Together they account for almost 40% of read I/O. They both are prime candicates for further SQL tuning.

Evaluating SQL Statements With Most Executions in Statspack

Identifying those SQL statements that execute most often in a database and tuning them can improve performance even when such statements do not consume many resources in each execution.

This is because of two reasons:

1. The overall resource consumption of these statements across all their executions may be significant.
2. Such frequently executed statements are often part of OLTP-style short transactions. Tuning them can improve the performance of the database as experienced by users entering such transactions into the system.

Here is an example of Statspack "SQL ordered by Executions":

```
SQL ordered by Executions for DB: DIN  Instance: DIN  Snaps: 263 -264
-> End Executions Threshold:     100

                                                    CPU per    Elap per
 Executions   Rows Processed   Rows per Exec    Exec (s)   Exec (s)  Hash Value
------------ ---------------- ---------------- ---------- ---------- ----------
-
     404,871         133,781              0.3       0.00        0.00 3592950473
SELECT nvl(NVL(EIGTWX.WERT,EIGTW.WERT),eig.wert) wert,
  1 sortier     FROM     D8_EIGENTYPWERTE_TEXT EIGTWX     ,D98_
EIGENTYPWERTE EIGTW     ,D98_EIGENSCHAFTEN EIG     WHERE EIG.ANH

     324,014         324,014              1.0       0.00        0.00  293077222
SELECT /*+ INDEX (D98_SCHEMAFORMATE SCHFMT_FORM_FKT_UNTER) */
        upper(funktionsname), unterformat        FROM
D98_SCHEMAFORMATE        WHERE          formatierung = :b1

     183,276         183,276              1.0       0.00        0.00   66213032
INSERT INTO D98_Batch_Variablenwerte                (ausf_id, b
atch_id, var_nr, wert)             VALUES                (:b4,
 :b3, :b2, :b1)

     114,224           8,936              0.1       0.00        0.00 1604001664
SELECT termin_ist     FROM        d98_termine     WHERE         a
nhang_id=:b2        AND terminart = :b1        AND aktiv = 'J'
   order by termin_ist desc
```

It will frequently be the case that the timing columns in this section will show 0.00 for CPU and Elapsed time, as the most frequently executing SQL statements are likely to be quite fast.

Finding SQL Statements With High Shared Pool Consumption in Statspack

This can help with Shared Pool and Library Cache/Shared Pool latch tuning.

Statements with many versions (multiple child cursors with the same parent cursor i.e. identical SQL text but different properties such as owning schema of objects, optimizer session settings, types & lengths of bind variables etc.) are unsharable.

This means they can consume excessive memory resources in the Shared Pool and cause performance problems related to parsing e.g. Library Cache and Shared Pool latch contention or lookup time e.g. Library Cache latch contention.

Statspack has 2 sections to help find such unsharable statements, "SQL ordered by Sharable Memory" and "SQL ordered by Version Count":

```
SQL ordered by Sharable Memory for DB: DIN  Instance: DIN  Snaps: 263 -264
-> End Sharable Memory Threshold:   1048576

Sharable Mem (b)  Executions  % Total  Hash Value
---------------- ------------ ------- ------------
       3,445,680            1     0.1   2317124142
select /*+rule*/   decode(UPPER(:P_TITEL_SORT), 'NAME', fin_sort_ansp_name,
'FIRMA', fin_sort_ans_firma, 'KURZBEZ. FIRMA', fin_sort_ans_name,

       3,386,429           76     0.1   3470710979
SELECT ID,ANHANGSTYP,OBJEKT_ID,PROF_ID,ANHANGSART,SPRACHE,AKTIV,
ANFANG,ENDE,SEITEN,ARCH_BEMERKUNG,DESKRIPTOR,EIGTW_ID,EIG_WERT,T

       2,836,746          447     0.1   2274525714
SELECT ID,ANHANGSTYP,OBJEKT_ID,PROF_ID,ANHANGSART,SPRACHE,AKTIV,
ANFANG,ENDE,SEITEN,ARCH_BEMERKUNG,DESKRIPTOR,EIGTW_ID,EIG_WERT,T


SQL ordered by Version Count for DB: P97  Instance: P97  Snaps: 177 -180
-> End Version Count Threshold:        20

 Version
   Count  Executions   Hash Value
-------- ------------ ------------
      26       36,228   3957083415
SELECT 1   FROM NK245_RECHTE A  WHERE (NK245_NK211_ID BETWEEN :b
1 AND :b2 ) AND NK245_GEHEIMSTUFE >= :b3  AND NK245_NK209_ID = :
b4  AND NK245_NKXX_ID = :b5  AND NK245_ANZAHL > 0  AND (:b6 BETW

      25          606   2916558383
UPDATE KVS.NK21_DOKUMENTE SET NK21_NK41_PDA_KUERZEL=:b1,NK21_NK4
1_PDA_NAME=:b2,NK21_STR_LEVEL=:b3,NK21_STR_NR=:b4,NK21_STR_TYP=:
b5,NK21_STRL1_NR=:b6,NK21_STRL1_TYP=:b7,NK21_STRL2_NR=:b8,NK21_S

      24        1,602   1157590177
INSERT INTO NK297_NACHRICHTEN ( NK297_ID,NK297_TIMESTAMP,NK297_T
IMESTAMP_E,NK297_NK210_ABSENDER,NK297_NK270_USERNAME_ABS,NK297_T
ITEL,NK297_NTEXT,NK297_NK248_ID,NK297_NK244_ID,NK297_NK213_ID,NK
```

For more information on tuning the Shared Pool please refer to the article:

Document 62143.1 Understanding and Tuning the Shared Pool

Top SQL Statements in V$SQLAREA and V$SQL (without using AWR or STATSPACK)

The Oracle Server provides 3 dynamic views for querying execution statistics of all SQL statements currently cached in the Library Cache of the Shared Pool.
They are V$SQL, V$SQLAREA and V$SQLXS.

V$SQL has 1 row for each different version of a SQL statement.
This means that each child cursor has its own execution statistics.

V$SQLAREA has 1 row for each different SQL string i.e. each parent cursor.
This means that the statistics for all child cursors i.e. different versions of this cursor are grouped together. It is not based on V$SQL.

V$SQLXS is a simpler version of V$SQLAREA.
It is used mainly by Statspack for generating the SQL reporting sections.
It queries V$SQL with a GROUP BY.
It is defined in ORACLE_HOME/rdbms/admin/catsnmp.sql.

V$SQLAREA or V$SQLXS can be used most often to find the top few SQL statements for a specific category of statistic.
Once these are identified, V$SQL can be used to drill down to see whether different versions of each statement exhibit similar statistics or whether some particular versions stand out.

V$SQL is less resource intensive than V$SQLAREA as it avoids the GROUP BY operation and causes less Library Cache latch contention.

Here is a general form of a query on any of these views:

```
SELECT * FROM
  (SELECT hash_value,address,substr(sql_text,1,40) sql,
          [list of columns], [list of derived values]
     FROM [V$SQL or V$SQLXS or V$SQLAREA]
    WHERE [list of threshold conditions for columns]
    ORDER BY [list of ordering columns] DESC)
WHERE rownum <= [number of top SQL statements];
```

Here is an example:

```
SELECT * FROM
  (SELECT hash_value,address,substr(sql_text,1,40) sql,
          buffer_gets, executions, buffer_gets/executions "Gets/Exec"
     FROM V$SQLAREA
    WHERE buffer_gets > 100000 AND executions > 10
   ORDER BY buffer_gets DESC)
WHERE rownum <= 10;
```

- [list of columns] = buffer_gets, executions
- [list of derived values] = buffer_gets/executions
- [list of threshold conditions for columns] = buffer_gets > 100000, executions > 10
- [list of ordering columns] = buffer_gets
- [number of top SQL statements] = 10

The following article contains a selection of ready to run Top-10 queries:

Document 235146.1 Example "Top SQL" queries from V$SQLAREA

A list of columns on which to base similar queries are:

- buffer_gets
- disk_reads
- executions
- rows_processed
- sorts
- parse_calls
- sharable_mem
- version_count
- invalidations

All of these are available in all three views VSQL, VSQLAREA & V$SQLXS.
There are a number of other columns for queries of this kind , not so frequently used, which can be found by inspecting V$SQL and V$SQLAREA.

Document 43761.1 VIEW: "V$SQLAREA" Reference Note
Document 43762.1 VIEW: "V$SQL" Reference Note

Top SQL Statements in TKPROF Reports

TKPROF is a tool for producing formatted reports from SQL Trace files (also known as event 10046 tracefiles).

Each SQL (and PL/SQL) statement appearing in the tracefile has its information summarized and collected in one place in the TKPROF report.
This information includes: number of parse, execution & fetch operations, total cpu & elapsed times, buffers read in consistent (query) and current mode, blocks read from disk and row counts, Row Source operations, execution plans, library cache misses, parsing user id & optimizer mode and with TKPROF version 9.0.1 and above, summary of wait events for tracefiles generated with event 10046 at levels 8 or 12.

A powerful feature available in TKPROF is the **sort** option which allows for ordering the SQL statement in the report according to a number of criteria.
This enables the easy identification of the most resource-intensive SQL statements and helps target efficiently the SQL tuning process.
Here are the options available for sorting, they may be combined:

```
prscnt number of times parse was called
prscpu cpu time parsing
prsela elapsed time parsing
prsdsk number of disk reads during parse
prsqry number of buffers for consistent read during parse
prscu number of buffers for current read during parse
prsmis number of misses in library cache during parse
execnt number of execute was called
execpu cpu time spent executing
exeela elapsed time executing
exedsk number of disk reads during execute
exeqry number of buffers for consistent read during execute
execu number of buffers for current read during execute
exerow number of rows processed during execute
exemis number of library cache misses during execute
fchcnt number of times fetch was called
fchcpu cpu time spent fetching
fchela elapsed time fetching
fchdsk number of disk reads during fetch
fchqry number of buffers for consistent read during fetch
fchcu number of buffers for current read during fetch
fchrow number of rows fetched
```

The most commonly used sort options are the ones ordering according to elapsed times consumed for the execute and fetch phases: we are usually interested in tuning statements that take more time to run as time is the most important property in the tuning process.

```
exeela elapsed time executing
fchela fchela elapsed time fetching
```

It should be clear however that it is as simple to order the SQL in order to find which ones cause the most disk i/o during fetch

```
fchdsk number of disk reads during fetch
```

or according to which ones access the most buffers in consistent mode e.g. for queries this would be in the fetch phase

```
fchqry number of buffers for consistent read during fetch
```

or for DML it would be in the execute phase

```
exeqry number of buffers for consistent read during execute
```

For more information on working with TKPROF please refer to Document 32951.1 Tkprof Interpretation

REFERENCES

NOTE:190124.1 - The COE Performance Method
NOTE:223117.1 - Troubleshooting I/O-related waits
NOTE:228913.1 - Systemwide Tuning using STATSPACK Reports

NOTE:235146.1 - Example "Top SQL" Queries from V$SQLAREA
NOTE:32951.1 - TKProf Interpretation (9i and below)
NOTE:43761.1 - VIEW: "V$SQLAREA" Reference Note
NOTE:43762.1 - VIEW: "V$SQL" Reference Note
NOTE:62143.1 - Troubleshooting: Tuning the Shared Pool and Tuning Library Cache Latch Contention