The Ultimate DBA SQL Performance Troubleshooting Guide

3 STEP GUIDE TO BECOME A MORE CONFIDENT DBA, WHEN TROUBLESHOOTING SQL PERFORMANCE PROBLEMS

Copyright @ 2016 DBAPARADISE.COM
Have you ever felt so confident in your performance troubleshooting skills that you were looking forward to users contacting you with a performance issue in the database? Or that you were almost jumping in your seat when database performance problems occurred? Did you always knew what scripts to run and what the next step is in the process of troubleshooting? Or do you identify yourself with the DBA that secretly wishes that the guy next to him picks up the performance problem ticket?

You could be more confident in your performance troubleshooting skills, with the help of this report! This report is a step by step process that you can apply to any database performance problem, to determine the root cause. The scripts provided can be applied to Oracle databases, however the process and the steps would be the same for SQL Server, or any other database flavour.

Why are DBAs, not all of them, afraid of performance problems? Because they don’t have a process to follow, that would lead to the root cause of the problem. When I was in my junior DBA years, I felt intimidated by performance problems in the database. I didn’t know how to approach them, I didn’t understand them. Over the course of the years, with the help of my mentors and with experience, I developed the 3 Ws, the three important questions to ask, that would lead you to the root cause of the problem: WHO, WHAT, WHEN. Not always in this order.

This report is straight to the point. It will guide you through the exact steps to follow when a performance problem occurs, so you can feel confident that this time you can fix it! Scripts are included with sample output and instructions. You can copy paste the scripts into your own script file, so next time you have them handy.

Because a picture is worth a thousand words, the last page of this report, is an overview picture that I recommend you print out and have it handy all the time. This is a flashcard of the troubleshooting process!

So let's get this party started, and get down to business!
1. WHO is running the SQL that is causing performance problems?

Anytime there is a performance issue, you want to determine as much information as possible about the session that is running the report.

What specific information are we looking for? To be more exact: username, program, module, hostname, instance number, sql id of the sql that the session is running, wait events for the session.

I created a script, called `sessions.sql` that will provide you with this information. Log into sqlplus, or the tool of your choice that can execute sql statements and run the script:

```
SQL>@sessions.sql
```
/* 
sessions.sql 
Example: @sessions.sql 
Copyright @2016 dbaparadise.com 
*/ 
set linesize 200 
set pagesize 100 
clear columns 
col inst for 99999999 
col sid for 9990 
col serial# for 999990

col username for a12 
col osuser for a16 
col program for a10 trunc 
col Locked for a6 
col status for a1 trunc print 
col "hh:mm:ss" for a8 
col SQL_ID for a15 
col seq# for 99990 
col event heading 'Current/LastEvent' for a25 trunc 
col state head 'State (sec)' for a14 
select inst_id inst, sid, serial#, username 
    , ltrim(substr(osuser, greatest(instr(osuser, '\', -1, 1)+1,length(osuser)-14))) osuser 
    , substr(program, instr(program,'/','-1)+1,decode(instr(program,'@'),0,decode(instr(program,'.'),0,length(program),instr(program,'.'))-1),instr(program,'@')-1)) program, decode(lockwait,NULL,' ','L') locked, status, 
    to_char(to_date(mod(last_call_et,86400), 'sssss'), 'hh24:mi:ss') "hh:mm:ss" 
    , SQL_ID, seq#, event, 
    decode(state,'WAITING','WAITING '||lpad(to_char(mod(SECONDS_IN_WAIT,86400),'99990'),6) 
    , 'WAITED SHORT TIME','ON CPU','WAITED KNOWN TIME','ON CPU',state) state  
    , substr(module,1,25) module, substr(action,1,20) action 
from GV$SESSION 
where type = 'USER' 
and audsid != 0 -- to exclude internal processes 
order by inst_id, status, last_call_et desc, sid 
/
Sample Output

```sql
SQL> @sessions
```

<table>
<thead>
<tr>
<th>INST</th>
<th>SID</th>
<th>SERIAL#</th>
<th>USERNAME</th>
<th>OSUSER</th>
<th>PROGRAM</th>
<th>LOCKED</th>
<th>hh:mm:ss</th>
<th>SQL_ID</th>
<th>SEQ#</th>
<th>Current/LastEvent</th>
<th>State (sec)</th>
<th>MODULE</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>57</td>
<td>95</td>
<td>sys</td>
<td>oracle</td>
<td>sqlplus</td>
<td>A</td>
<td>00:00:20</td>
<td>31807c</td>
<td>17936</td>
<td>SQL*Net message to client ON CPU</td>
<td><a href="mailto:sqlplus@localhost.localdo">sqlplus@localhost.localdo</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>52</td>
<td>987</td>
<td>djoel</td>
<td>oracle</td>
<td>sqlplus</td>
<td>A</td>
<td>00:00:34</td>
<td>e32502af4156251f4583</td>
<td>34</td>
<td>db file sequential read ON CPU</td>
<td><a href="mailto:sqlplus@localhost.localdo">sqlplus@localhost.localdo</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Problem SQL, run by DJOE user
2. **WHAT is the session running?**

What do I mean?

The previous section identified the session that was running the problematic report, and the sql_id associated with it. Next, we need to identify the sql text associated with the sql_id. So we can confirm with the user that indeed this is the problem report. Run `sqltext.sql` script.

```sql
SQL>@sqltext sql_id
```

In case we know the sql text or part of the text, that the user is running, or was running, we need to identify the sql_id of that report. Run `findsql.sql` script.

```sql
SQL>@findsql sql_text_string
```

Once the sql_id is identified, identify the execution plan(s). Run `sqlplan.sql` script, or `sqlplan_hist.sql` script, in case the plan is aged out from the cursor.

```sql
SQL>@sqlplan sql_id
SQL>@sqlplan_hist sql_id
```
sqltext.sql

/*
sqltext.sql
Example @sqltext.sql 6rv5za4tfnjs8
Copyright dbaparadise.com
*/

set define '&'
set verify off
define sqlid=&1

col sql_text for a80 word_wrapped
col inst_id for 9
break on inst_id
set linesize 150

select inst_id, sql_text
from gv$sqltext
where sql_id = '&sqlid'
order by inst_id,piece
/

Sample Output

SQL> @sqltext 6rv5za4tfnjs8

INST_ID   SQL_TEXT
---------- -----------------------------------------------
          1 select count(*), cust_id from sh.sales group by cust_id
findsql.sql

/*
findsql.sql
Example: @findsql.sql employee_contact
Copyright dbaparadise.com
*/

set define '&'
define sql_str=&1

col sql_id for A15
col sql_text for A150 word_wrapped
set linesize 170
set pagesize 300

SELECT /* findsql */ sql_id, executions, sql_text
FROM gv$sql
WHERE command_type IN (2,3,6,7,189)
AND UPPER(sql_text) LIKE UPPER('%&sql_str%')
AND UPPER(sql_text) NOT LIKE UPPER('%findsql%')
/

undef sql_str

Sample Output

SQL> @findsql sales

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>DEALINGS</th>
<th>SQL_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>6rv5za4tfnjsB</td>
<td>2</td>
<td>select count(*), cust_id from sh.sales group by cust_id</td>
</tr>
</tbody>
</table>
/*
sqlplan.sql
Copyright dbaparadise.com
Example: @sqlplan.sql 1t8v91nxtxgjr
*/

set define '&'
define sqlid=&1

col ELAPSED for 99,990.999
col CPU for 99,990.999
col ROWS_PROC for 999,999,990
col LIO for 9,999,999,990
col PIO for 99,999,990
col EXECS for 999,990
col sql_text for a40 trunc
set lines 200
set pages 300

select inst_id,sql_id,child_number child_num ,plan_hash_value,
    round(ELAPSED_TIME/1000000/greatest(EXECUTIONS,1),3) ELAPSED,
    round(CPU_TIME/1000000/greatest(EXECUTIONS,1),3) CPU,EXECUTIONS EXECS,
    BUFFER_GETS/greatest(EXECUTIONS,1) lio,
    DISK_READS/greatest(EXECUTIONS,1) pio,
    ROWS_PROCESSED/greatest(EXECUTIONS,1) ROWS_PROC, sql_text
from gv$sql
where sql_id = '&sqlid'
order by inst_id, sql_id, child_number
/

select plan_table_output from table(dbms_xplan.display_cursor('&sqlid',NULL,'ADVANCED -PROJECTION -BYTES RUNSTATS_LAST'));

Copyright@2016 dbaparadise.com
Sample Output

SQL> @sqlplan 6rv52a4tfnjs8

<table>
<thead>
<tr>
<th>INST_ID</th>
<th>SOL_ID</th>
<th>CHILD_NUM</th>
<th>PLAN_HASH_VALUE</th>
<th>ELAPSED</th>
<th>CPU</th>
<th>EXEC</th>
<th>LIO</th>
<th>PIO</th>
<th>ROWS</th>
<th>PROC</th>
<th>SOL_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6rv52a4tfnjs8</td>
<td>0</td>
<td>3604305554</td>
<td>0.385</td>
<td>0.258</td>
<td>1</td>
<td>4,919</td>
<td>22</td>
<td>7,059</td>
<td>select count(*), cust_id from sh.sales group by cust_id</td>
<td></td>
</tr>
</tbody>
</table>

PLAN_TABLE_OUTPUT

SQL_ID 6rv52a4tfnjs8, child number 0

select count(*), cust_id from sh.sales group by cust_id

Plan hash value: 3604305554

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>E-Rows</th>
<th>Cost (%CPU)</th>
<th>E-Time</th>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>7059</td>
<td>535 (100)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>918K</td>
<td>514 (1)</td>
<td>00:00:01</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>918K</td>
<td>514 (1)</td>
<td>00:00:01</td>
<td>1</td>
<td>28</td>
</tr>
</tbody>
</table>

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

1 - SEL$1
3 - SEL$1 / SALES@SEL$1

Outline Data

/*
   BEGIN OUTLINE_DATA
   IGNORE OPTIM EMBEDDED HINTS
   OPTIMIZER FEATURES ENABLE('12.1.0.1')
   DB_VERSION('12.1.0.1')
   ALL_ROWS
   OUTLINE LEAF(@"SEL$1")
   FULL(@"SEL$1" "SALES@SEL$1")
   USE_HASH_AGGREGATION(@"SEL$1")
   END_OUTLINE_DATA
*/
sqlplan_hist.sql

/*
sqlplan_hist.sql
Copyright dbaparadise.com
Example: @sqlplan_hist.sql 1t8v91nxtxgjr

WARNING! Diagnostic and Tuning Pack licensing is required to run this script!!!
*/

set linesize 200
set pagesize 200
set verify off
set define '&'
define sqlid=&1

select plan_table_output from table(dbms_xplan.display_awr('&sqlid'));
Sample Output:

```
SQL> @sqlplan_hist.sql 6rv5za4tfnjs8

PLAN_TABLE_OUTPUT

<table>
<thead>
<tr>
<th>SQL_ID 6rv5za4tfnjs8</th>
</tr>
</thead>
<tbody>
<tr>
<td>select count(*), cust_id from sh.sales group by cust_id</td>
</tr>
<tr>
<td>Plan hash value: 2374739488</td>
</tr>
</tbody>
</table>

<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>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>428 (100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>7059</td>
<td>35295</td>
<td>428 (5)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>918K</td>
<td>4486K</td>
<td>407 (0)</td>
<td>00:00:01</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>BITMAP CONVERSION COUNT</td>
<td></td>
<td>918K</td>
<td>4486K</td>
<td>407 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>BITMAP INDEX FAST FULL SCAN</td>
<td>SALES_CUST_BIX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SQL_ID 6rv5za4tfnjs8

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>select count(*), cust_id from sh.sales group by cust_id</td>
</tr>
<tr>
<td>Plan hash value: 3604305554</td>
</tr>
</tbody>
</table>

<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>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>535 (100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>7059</td>
<td>35295</td>
<td>535 (5)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>918K</td>
<td>4486K</td>
<td>514 (1)</td>
<td>00:00:01</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>918K</td>
<td>4486K</td>
<td>514 (1)</td>
<td>00:00:01</td>
<td>1</td>
<td>28</td>
</tr>
</tbody>
</table>
```
3. WHEN was the last time this report ran successfully and in optimal time?

The key takeaway from this question is the optimal time. This is a crucial question, it can help you tremendously, especially if you have Diagnostic and Tuning license for the database. You want to know the answer to this question, so you can actually compare the current results with previous result, and see the differences.

I have a script that will show you historical runs of the sql statement that were captured by AWR, with plan changes and average elapsed times. You can correlate the “good” time with the good execution plan and the “bad” time with the bad execution plan.
If you can still find the good execution plan, then basically you have your solution right in front of your eyes.

SQL>@historical_runs sql_id
historical_runs.sql

/*
historical_runs.sql
Copyright dbaparadise.com
Example @historical_runs.sql sql_id
*/
set linesize 200
set pagesize 200
set verify off
set define '&'
define sqlid=&1
col execs for 999,999,999
col avg_etime for 999,999.999
col avg_lio for 999,999,999.9
col begin_interval_time for a30
col node for 9999
break on plan_hash_value skip 1

select sh.snap_id, sh.instance_number inst, sh.begin_interval_time, s.sql_id, s.plan_hash_value,
nvl(s.executions_delta,0) execs,
(s.elapsed_time_delta/decode(nvl(s.executions_delta,0),0,1,s.executions_delta))/1000000 avg_etime,
(s.buffer_gets_delta/decode(nvl(s.buffer_gets_delta,0),0,1,s.executions_delta)) avg_lio
from DBA_HIST_SQLSTAT S, DBA_HIST_SNAPSHOT SH
where s.sql_id = '&sqlid'
and sh.snap_id = s.snap_id
and sh.instance_number = s.instance_number
order by 1, 2, 3
/

Copyright@2016 dbaparadise.com
Sample Output:

```
SQL> @historical_runs 6rv5za4tfnjs8

<table>
<thead>
<tr>
<th>SNAP_ID</th>
<th>INST</th>
<th>BEGIN_INTERVAL_TIME</th>
<th>SQL_ID</th>
<th>PLAN_HASH_VALUE</th>
<th>EXECS</th>
<th>AVG_ETIME</th>
<th>AVG_LIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>07-APR-16 02.05.34.469 PM</td>
<td>6rv5za4tfnjs8</td>
<td>2374739488</td>
<td>8</td>
<td>.116</td>
<td>1,471.9</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>07-APR-16 02.29.04.577 PM</td>
<td>6rv5za4tfnjs8</td>
<td></td>
<td>3</td>
<td>.176</td>
<td>1,419.3</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>10-APR-16 02.22.49.751 PM</td>
<td>6rv5za4tfnjs8</td>
<td>3604305554</td>
<td>1</td>
<td>.183</td>
<td>1,419.0</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>10-APR-16 02.48.03.572 PM</td>
<td>6rv5za4tfnjs8</td>
<td></td>
<td>2</td>
<td>.595</td>
<td>2,951.5</td>
</tr>
</tbody>
</table>
```

Execution Plan has changed. Why?

At this point you determined, there is a good execution plan, and a bad execution plan. Verify the 2 plans (sqlplan_hist.sql), and notice that one is using an index, and the other one is using a full table scan. The root cause of this problem: **the execution plan has changed**, the new plan is worse than the initial one. Further, you need to determine the reason why the plan has changed. It could be that the index got dropped or invalidate, data changed, the optimizer decided the new plan is better, stale statistics. The important part is that you figured out the problem, now the solution is just one step away.
Find the Problem SQL

WHO? [@sessions.sql]
WHAT? [@sqltext.sql]
   @finsql.sql
   @sqlplan.sql
   @sqlplan_hist.sql
WHEN? [@historical_runs.sql]

The session/sql that is causing the problem
Focus on the solution