The Ultimate DBA SQL Performance Troubleshooting Guide

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

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.

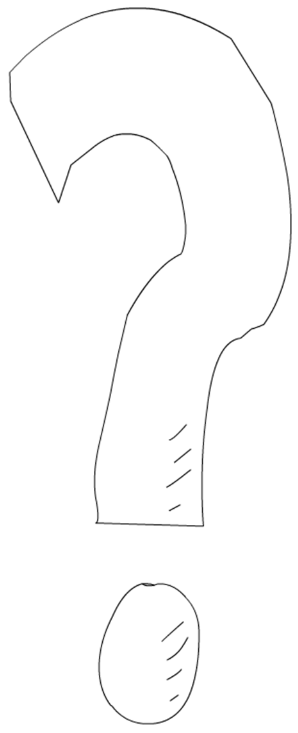
Why are DBAs, not all of them, afraid of performance problems? Because they don't have a process to follow, that would lead to the root cause of the problem. When I was in my junior DBA years, I felt intimidated by performance problems in the database. I didn't know how to approach them, I didn't understand 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?



WHO

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.

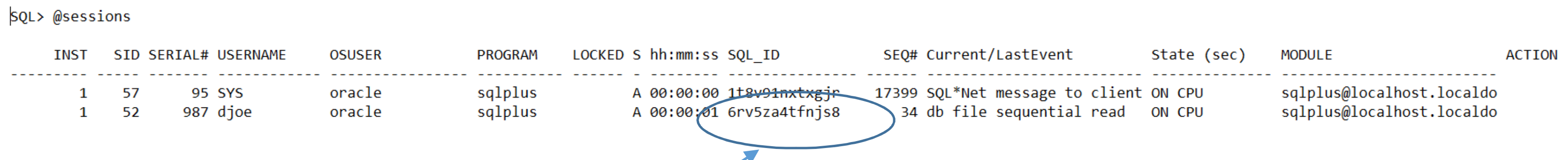
Log into sqlplus, or the tool of your choice that can execute sql statements and run the script:

```
SQL>@sessions.sql
```

sessions.sql

```
/*
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 processess
order by inst_id, status, last_call_et desc, sid
/
```

Sample Output

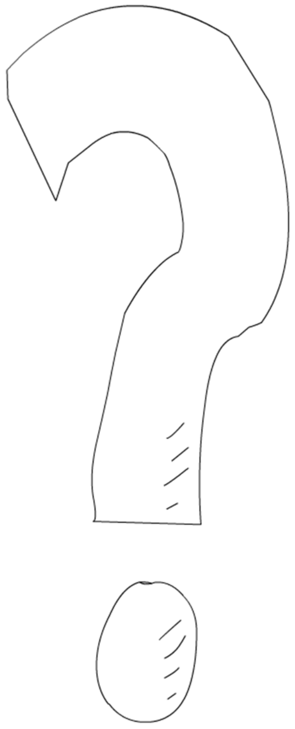


```
SQL> @sessions

     INST  SID SERIAL# USERNAME     OSUSER           PROGRAM    LOCKED S hh:mm:ss SQL_ID           SEQ# Current/LastEvent         State (sec)   MODULE                    ACTION
--------- ----- ------- ------------ ---------------- ---------- ------ - -------- ---------------- ------ ------------------------- ------------- -------------------------
        1    57      95 SYS          oracle           sqlplus           A 00:00:00 1t8v91nxtxgjr    17399 SQL*Net message to client ON CPU        sqlplus@localhost.localdo
        1    52     987 djoe         oracle           sqlplus           A 00:00:01 6rv5za4tfnjs8       34 db file sequential read   ON CPU        sqlplus@localhost.localdo
```

Problem SQL, run by DJOE user

DBA Paradise
The Place where DBAs grow

2. WHAT is the session running?



WHAT

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>@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>@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>@sqlplan sql_id
SQL>@sqlplan_hist sql_id
```

DBA Paradise
The Place where DBAs grow

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
```

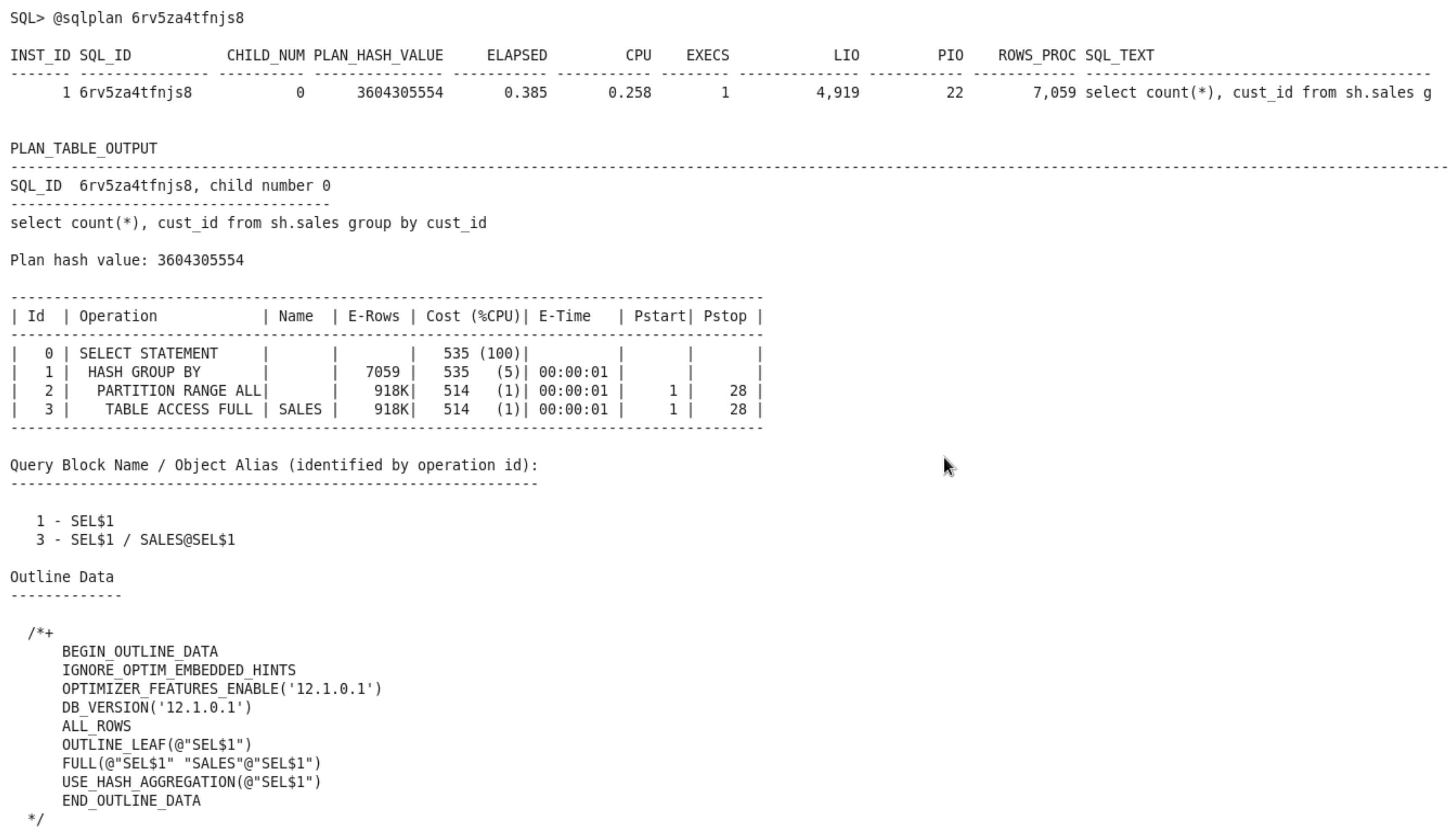
Sample Output

```
SQL> @findsql sales

SQL_ID          EXECUTIONS SQL_TEXT
--------------- ---------- --------------------------------------------------------------------------------
6rv5za4tfnjs8            2 select count(*), cust_id from sh.sales group by cust_id
```

sqlplan.sql

```
/*
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'));
```

Sample Output



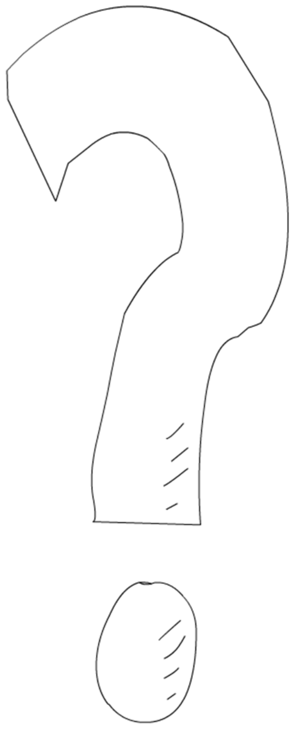
```
SQL> @sqlplan 6rv5za4tfnjs8

INST_ID SQL_ID          CHILD_NUM PLAN_HASH_VALUE     ELAPSED         CPU    EXECS            LIO         PIO    ROWS_PROC SQL_TEXT
------- -------------- ---------- --------------- ----------- ----------- -------- -------------- ----------- ------------ ----------------------------------------
      1 6rv5za4tfnjs8           0      3604305554       0.385       0.258        1          4,919          22        7,059 select count(*), cust_id from sh.sales g


PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
SQL_ID  6rv5za4tfnjs8, child number 0
-------------------------------------
select count(*), cust_id from sh.sales group by cust_id

Plan hash value: 3604305554

---------------------------------------------------------------------------------------
| Id  | Operation             | Name  | E-Rows | Cost (%CPU)| E-Time   | Pstart| Pstop |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |       |        |   535 (100)|          |       |       |
|   1 |  HASH GROUP BY        |       |   7059 |   535   (5)| 00:00:01 |       |       |
|   2 |   PARTITION RANGE ALL|       |    918K|   514   (1)| 00:00:01 |     1 |    28 |
|   3 |    TABLE ACCESS FULL | SALES |    918K|   514   (1)| 00:00:01 |     1 |    28 |
---------------------------------------------------------------------------------------

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! Diagnistic 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
------------------------------------------------------------------------------------------------------
------------
SQL_ID 6rv5za4tfnjs8
--------------------
select count(*), cust_id from sh.sales group by cust_id

Plan hash value: 2374739488

----------------------------------------------------------------------------------------------------
| Id  | Operation                       | Name             | Rows  | Bytes | Cost (%CPU)| Time       | Pstart| Pstop |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                |                  |       |       |   428 (100)|            |       |       |
|   1 |  HASH GROUP BY                  |                  |  7059 | 35295 |   428   (5)| 00:00:01   |       |       |
|   2 |   PARTITION RANGE ALL           |                  |   918K|  4486K|   407   (0)| 00:00:01   |     1 |    28 |
|   3 |    BITMAP CONVERSION COUNT      |                  |   918K|  4486K|   407   (0)| 00:00:01   |       |       |
|   4 |     BITMAP INDEX FAST FULL SCAN| SALES_CUST_BIX |       |       |            |            |     1 |    28 |
----------------------------------------------------------------------------------------------------

SQL_ID 6rv5za4tfnjs8
--------------------
select count(*), cust_id from sh.sales group by cust_id

Plan hash value: 3604305554

---------------------------------------------------------------------------------------------
| Id  | Operation            | Name    | Rows  | Bytes | Cost (%CPU)| Time       | Pstart| Pstop |
---------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |         |       |       |   535 (100)|            |       |       |
|   1 |  HASH GROUP BY       |         |  7059 | 35295 |   535   (5)| 00:00:01   |       |       |
|   2 |   PARTITION RANGE ALL|         |   918K|  4486K|   514   (1)| 00:00:01   |     1 |    28 |
|   3 |    TABLE ACCESS FULL | SALES   |   918K|  4486K|   514   (1)| 00:00:01   |     1 |    28 |
---------------------------------------------------------------------------------------------
```



WHEN

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
/
```

Sample Output:

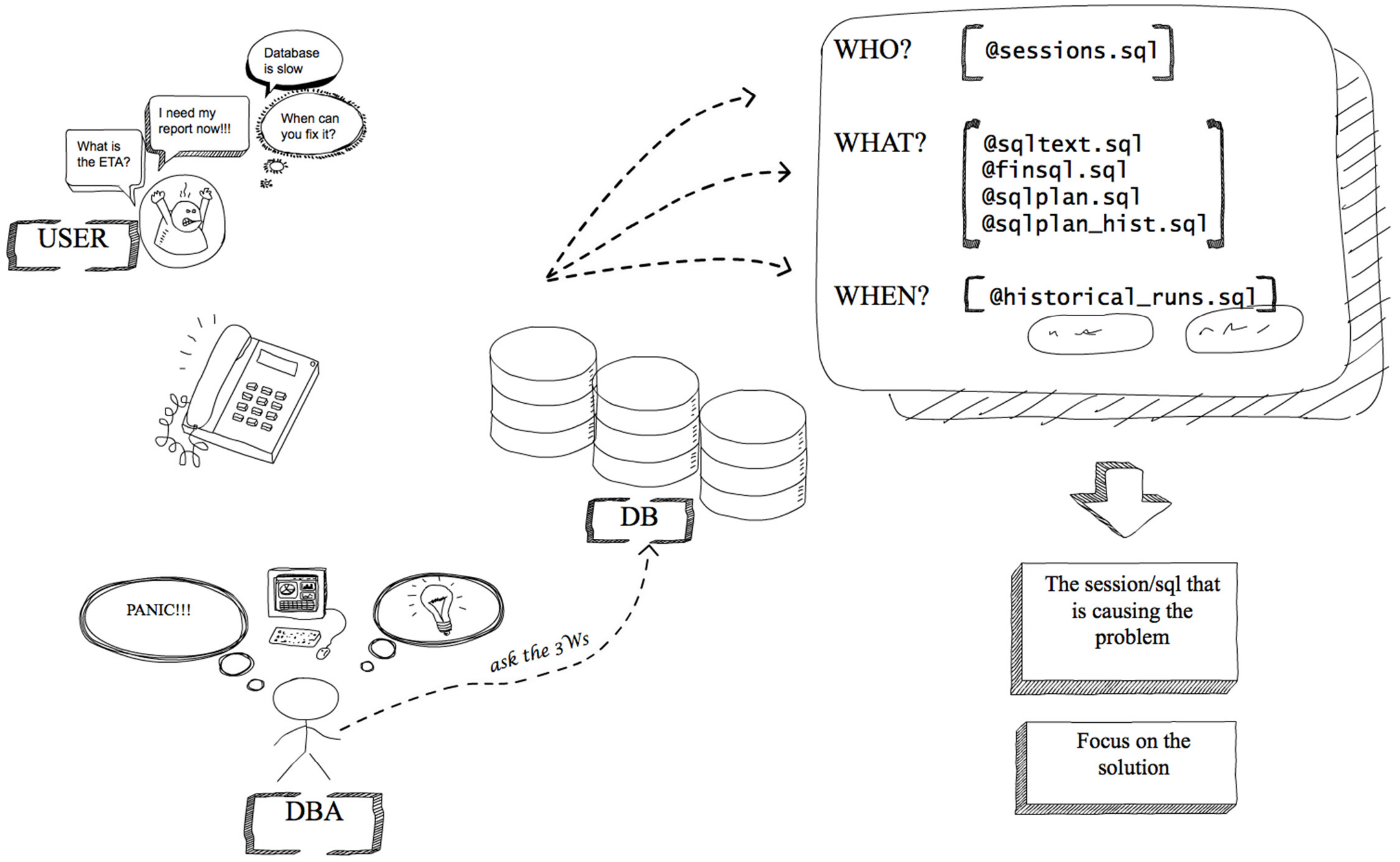
```
 SQL> @historical_runs 6rv5za4tfnjs8

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

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.

DBA Paradise
The Place where DBAs grow

Find the Problem SQL



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