Oracle DB-Tuning Essentials

Agenda

1. The DB server and the tuning environment
2. Objective, Tuning versus Troubleshooting, Cost Based Optimizer
3. Object statistics
4. Access paths I
5. Access paths II
6. Monitoring Performance I
7. Monitoring Performance II
8. Application setup

Monitoring Performance

- Database Time
- AWR, ASH, ADDM
- Statspack
- dbms_xplan
- SQL Monitoring
- Application instrumentation
- SQL Trace, tkprof

dbms_xplan
Execution plans

Reason:

- The CBO chooses the execution plan but we have to check it
- Understand bad choices (cardinality)
- Understand access path (predicates)
- Get optimizer information

dbms_xplan.display:

- Display from PLAN_TABLE (result from EXPLAIN PLAN)

dbms_xplan.display_cursor:

- Display from V$SQL_PLAN (already executed cursor child)

dbms_xplan.display_AWR:

- Display from DBA_HIST_SQL_PLAN (stored by AWR)

dbms_xplan
Display from PLAN_TABLE

```
SQL> EXPLAIN PLAN FOR
  2  select job,dname,count(*) from EMP join DEPT using(deptno)
  3  where deptno=:d group by job,dname;
Explained.

SQL> select * from table(dbms_xplan.display(format=>'basic +cost'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
Plan hash value: 3315835386

-----------------------------------------------------------------
| Id  | Operation                     | Name    | Cost (%CPU)|
-----------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |     5  (20)|
|   1 |  HASH GROUP BY                |         |     5  (20)|
|   2 |   NESTED LOOPS                |         |     4   (0)|
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPT    |     1   (0)|
|   4 |     INDEX UNIQUE SCAN         | PK_DEPT |     1   (0)|
|   5 |    TABLE ACCESS FULL          | EMP     |     3   (0)|
-----------------------------------------------------------------
```

dbms_xplan
Display from V$SQL_PLAN (+peeked_binds)

```
SQL> variable d number;
SQL> exec :d:=10;

PL/SQL procedure successfully completed.

SQL>
SQL> select job,dname,count(*) from EMP join DEPT using(deptno)
  2  where deptno=:d group by job,dname;

JOB       DNAME          COUNT(*)
--------- -------------- ----------
MANAGER   ACCOUNTING              1
PRESIDENT ACCOUNTING              1
CLERK     ACCOUNTING              1


SQL> select * from table(
 2          dbms_xplan.display_cursor(format=>'basic +peeked_binds')
 3 );
```

By default takes the latest executed cursor from the session

serveroutput must be off or the last statement is wrong

dbms_xplan
Display from V$SQL_PLAN (+peeked_binds)

```
EXPLAINED SQL STATEMENT:
------------------------
select job,dname,count(*) from EMP join DEPT using(deptno) where
deptno=:d group by job,dname

Plan hash value: 3315835386

-----------------------------------------------
| Id  | Operation                     | Name    |
-----------------------------------------------
|   0 | SELECT STATEMENT              |         |
|   1 |  HASH GROUP BY                |         |
|   2 |   NESTED LOOPS                |         |
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPT    |
|   4 |     INDEX UNIQUE SCAN         | PK_DEPT |
|   5 |    TABLE ACCESS FULL          | EMP     |
-----------------------------------------------

Peeked Binds (identified by position):
--------------------------------------

   1 - :D (NUMBER): 10
```

dbms_xplan
Display from V$SQL_PLAN (allstats)

```
SQL> alter session set statistics_level=all;

Session altered.

SQL> select job,dname,count(*) from EMP join DEPT using(deptno)
  2  where deptno=:d group by job,dname;

JOB       DNAME          COUNT(*)
--------- -------------- ----------
MANAGER   ACCOUNTING              1
PRESIDENT ACCOUNTING              1
CLERK     ACCOUNTING              1


SQL> select * from table(
  2          dbms_xplan.display_cursor(format=>'basic iostats last')
  3  );
```

Need to set statistics_level to all or use gather_plan_statistics

dbms_xplan
Display from V$SQL_PLAN (allstats)

```
SQL_ID  c00n45rgsa03h, child number 1
-------------------------------------
select job,dname,count(*) from EMP join DEPT using(deptno) where
deptno=:d group by job,dname

Plan hash value: 3315835386
-----------------------------------------------------------------------------------------
| Id  | Operation     | Name    | Starts | E-Rows | A-Rows |   A-Time    | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEME|         |      1 |        |      3 |00:00:00.01 |       9 |
|   1 |  HASH GROUP BY|         |      1 |      3 |      3 |00:00:00.01 |       9 |
|   2 |   NESTED LOOPS|         |      1 |      3 |      3 |00:00:00.01 |       9 |
|   3 |    TABLE ACCES| DEPT    |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  4 |     INDEX UNIQ| PK_DEPT |      1 |      1 |      1 |00:00:00.01 |       1 |
|*  5 |    TABLE ACCES| EMP     |      1 |      3 |      3 |00:00:00.01 |       7 |
-----------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("DEPT"."DEPTNO"=:D)
   5 - filter("EMP"."DEPTNO"=:D)
Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

dbms_xplan
How to read an execution plan?

- Start with the first operation at the deepest level

```
-------------------------------------------------------------------
| Id  | Operation                     | Name    | Cost (%CPU)|
-------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |     5  (20)|
|   1 |  HASH GROUP BY                |         |     5  (20)|
|   2 |   NESTED LOOPS                |         |     4   (0)|
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPT    |     1   (0)|
|   4 |     INDEX UNIQUE SCAN         | PK DEPT |     1   (0)|
|   5 |    TABLE ACCESS FULL          | EMP     |     3   (0)|

Predicate Information (identified by operation id):
---------------------------------------------------
 4 - access("DEPT"."DEPTNO"=:D)
 5 - filter("EMP"."DEPTNO"=:D)
```

- Send resulted rows to the parent
- Process next child (NESTED LOOP: for each row for 1st execute the 2nd)
- Send rows to the parent
- Continue (NESTED LOOP)
- Send rows to the parent (GROUP BY is a blocking operation)

Monitoring Performance

- Database Time
- AWR, ASH, ADDM
- Statspack
- dbms_xplan
- SQL Monitoring
- Application instrumentation
- SQL Trace, tkprof

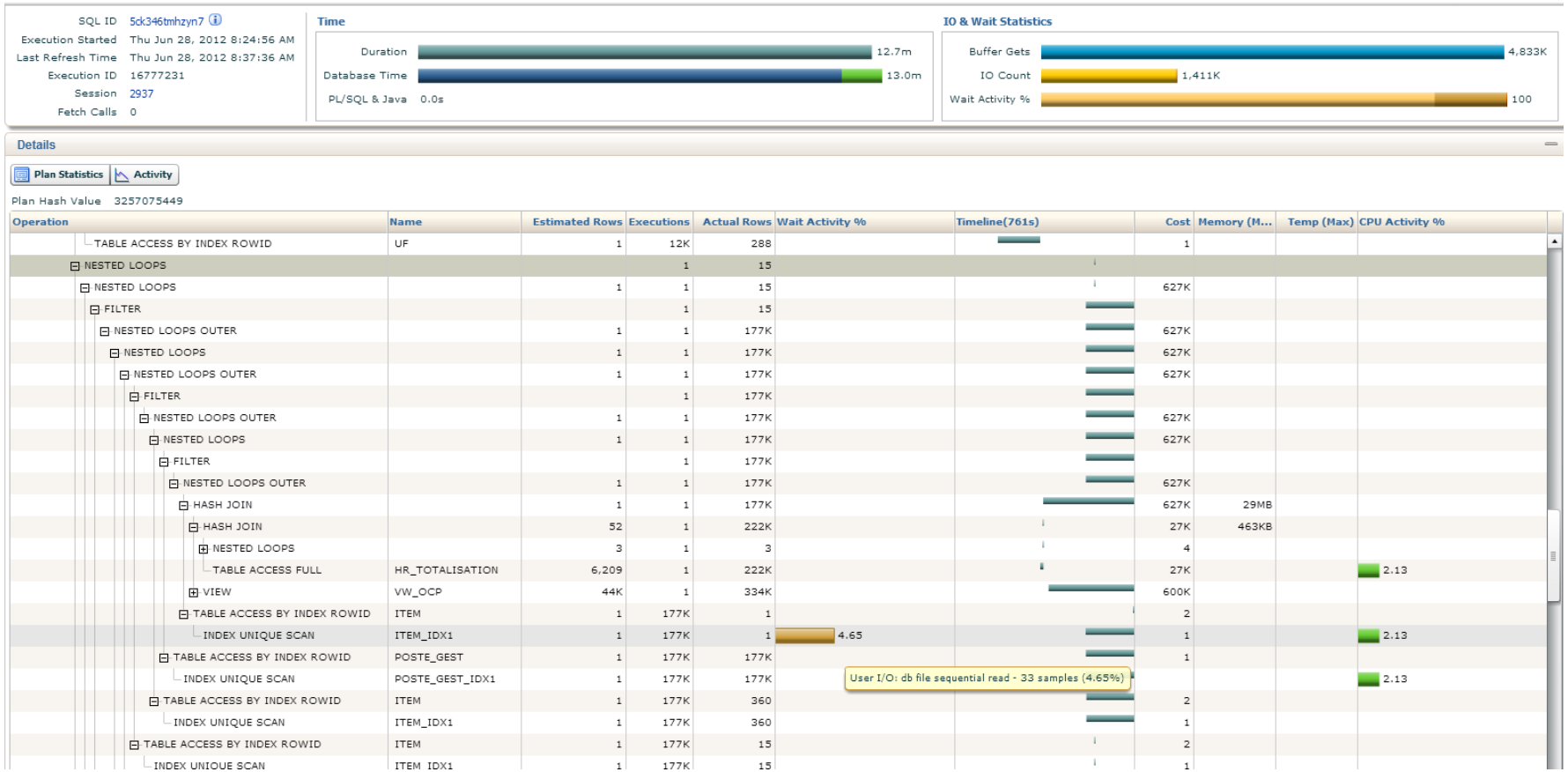
SQL Monitoring
Real-Time monitoring

SQL Monitoring is triggered for:

- All parallel queries
- Statements with more than a few seconds DB Time
- /*+ MONITOR */ hint
- It's real-time, even if the SQL is running or has failed
- Based on V$SQL_PLAN_STATISTICS + ASH
- Works for failed statements
- Easy with parallel query as well
- Integrated with EM or standalone

```
set pagesize 0 linesize 10000 trimspool on serveroutput off long 10000000
longc 10000000 echo off feedback off
alter session set events='emx_control compress_xml=none';
spool plan.htm
select dbms_sqltune.report_sql_monitor(report_level=>'all',type=>'active')
from dual;
spool off
```

SQL Monitoring
Example: Nested Loop



SQL ID	5ck346tmhzyn7
Execution Started	Thu Jun 28, 2012 8:24:56 AM
Last Refresh Time	Thu Jun 28, 2012 8:37:36 AM
Execution ID	16777231
Session	2937
Fetch Calls	0

Time

Duration	12.7m
Database Time	13.0m
PL/SQL & Java	0.0s

IO & Wait Statistics

Buffer Gets	4,833K
IO Count	1,411K
Wait Activity %	100

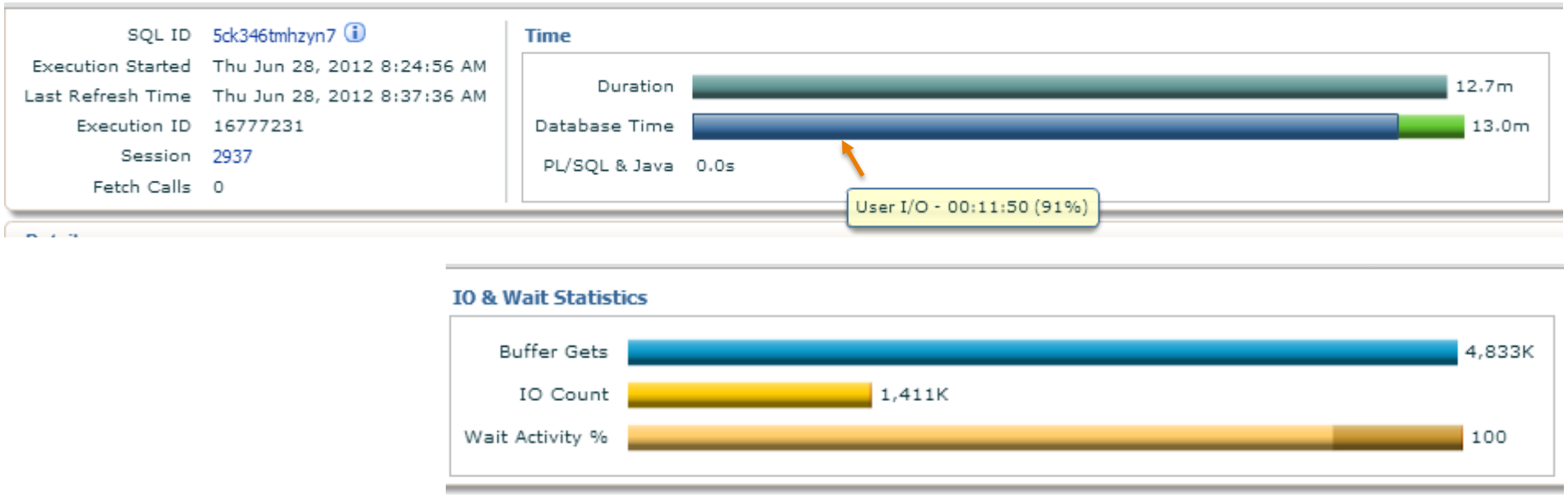
Details

Plan Statistics | Activity

Plan Hash Value 3257075449

Operation	Name	Estimated Rows	Executions	Actual Rows	Wait Activity %	Timeline(761s)	Cost	Memory (M...	Temp (Max)	CPU Activity %
TABLE ACCESS BY INDEX ROWID	UF	1	12K	288			1			
NESTED LOOPS			1	15						
NESTED LOOPS		1	1	15			627K			
FILTER			1	15						
NESTED LOOPS OUTER		1	1	177K			627K			
NESTED LOOPS		1	1	177K			627K			
NESTED LOOPS OUTER		1	1	177K			627K			
FILTER			1	177K						
NESTED LOOPS OUTER		1	1	177K			627K			
NESTED LOOPS		1	1	177K			627K			
FILTER			1	177K						
NESTED LOOPS OUTER		1	1	177K			627K			
HASH JOIN		1	1	177K			627K	29MB		
HASH JOIN		52	1	222K			27K	463KB		
NESTED LOOPS		3	1	3			4			
TABLE ACCESS FULL	HR_TOTALISATION	6,209	1	222K			27K			2.13
VIEW	VW_OCP	44K	1	334K			600K			
TABLE ACCESS BY INDEX ROWID	ITEM	1	177K	1			2			
INDEX UNIQUE SCAN	ITEM_IDX1	1	177K	1	4.65		1			2.13
TABLE ACCESS BY INDEX ROWID	POSTE_GEST	1	177K	177K			1			
INDEX UNIQUE SCAN	POSTE_GEST_IDX1	1	177K	177K		User I/O: db file sequential read - 33 samples (4.65%)				2.13
TABLE ACCESS BY INDEX ROWID	ITEM	1	177K	360			2			
INDEX UNIQUE SCAN	ITEM_IDX1	1	177K	360			1			
TABLE ACCESS BY INDEX ROWID	ITEM	1	177K	15			2			
INDEX UNIQUE SCAN	ITEM_IDX1	1	177K	15			1			

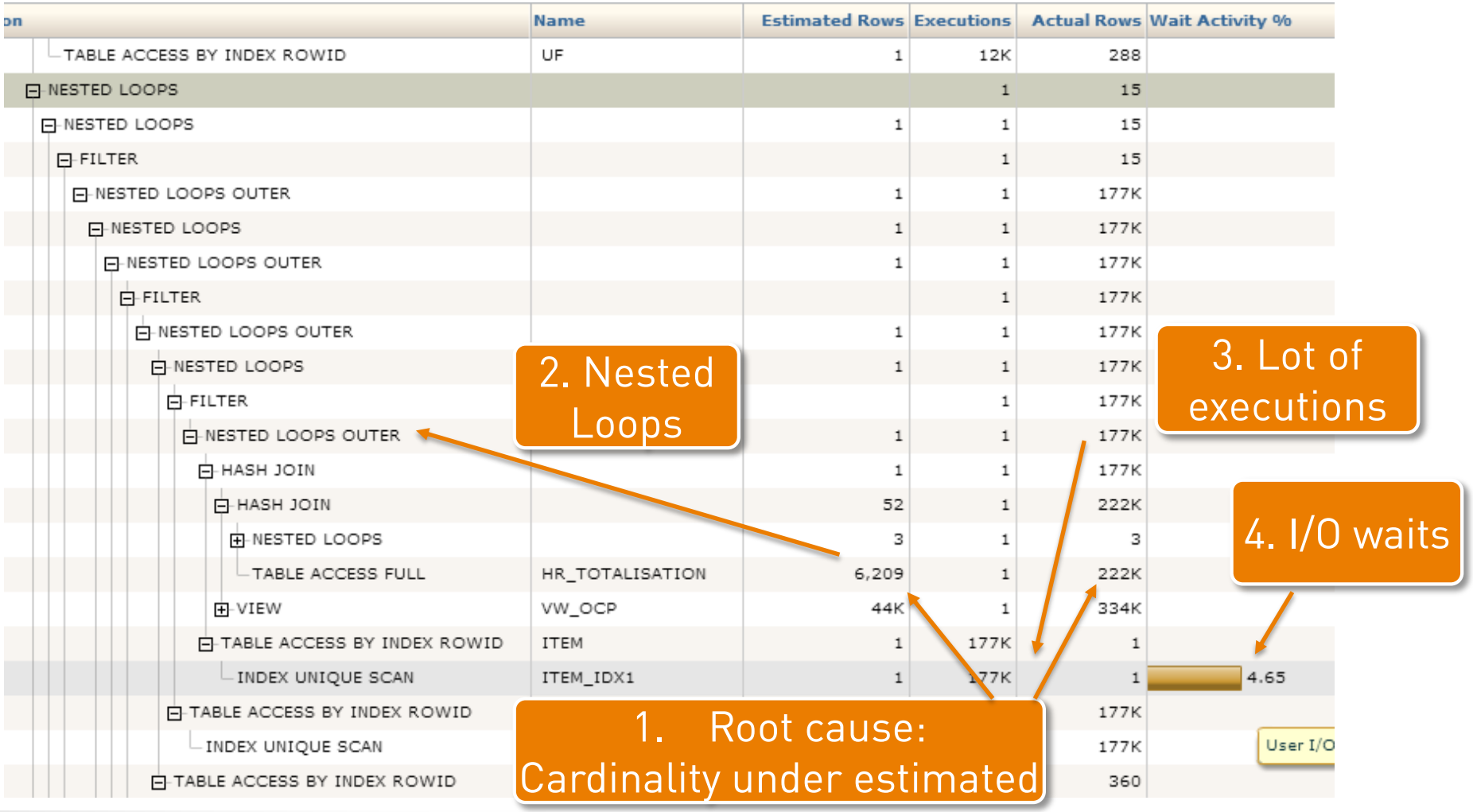
SQL Monitoring
Example: Nested Loop



Overview:

- Most of the time is I/O
- We need to improve the part of the plan that is doing lot of I/O

SQL Monitoring
Example: Nested Loop



SQL Monitoring
Example: Nested Loop

It's long because:

- An operation response time is too high or
- A short operation is called too many times

SQL Monitoring shows:

- Each operation with its time (samples) and number of calls (executions)
- Even when it is running
- Without any overhead
- Can be saved to interactive html

Requires Diagnostics + Tuning Pack

- Without it, need to use dbms_xplan.display_cursor with statistics_level=all

Monitoring Performance

- Database Time
- AWR, ASH, ADDM
- Statspack
- dbms_xplan
- SQL Monitoring
- Application instrumentation
- SQL Trace, tkprof

Application instrumentation
Why?

Identify who, what, which, and how

- Who is the end-user even if we use connection pool: CLIENT_ID
- What program: MODULE
- Which part of code: ACTION
- How it has been run: CLIENT_INFORMATION

Because:

- We want to debug, trace, get statistics, see who is locking, etc.

Example:

- User Smith is blocked when creating an order
- How much time is spent in the creation order
- Where is the code for that long running query

Application instrumentation
dbms_session and dbms_application_info

Using dbms_session package to identify client

- client_identifier
  - 64 bytes to identify the final user
  - Propagated through database links

Using dbms_application_info to identify tasks (e.g. New Order)

- module_name
  - 48 bytes to describe a unit task. Example: the java class
- action_name
  - 32 bytes to describe a task step. Example: the method
- client_info
  - 64 bytes to describe the context. Example: the use case and scenario

Also available from Java easily

- Connection.setClientInfo([OCSID.CLIENTID|MODULE|ACTION],"value")

Application instrumentation
dbms_session and application instrumentation

Available with SYS_CONTEXT function

```
SQL> select
     sys_context('userenv','client_identifier') AS client_identifier,
     sys_context('userenv','client_info') AS client_info,
     sys_context('userenv','module') AS module_name,
     sys_context('userenv','action') AS action_name
from dual;
```

Used/View in several places

- Performance views (V$SESSION)
- Use with dbms_monitor
  - To set sql_trace for specific part of the code
  - To gather some statistics for specific part of the code

It's at a lower level than SERVICE_NAME

Monitoring Performance

- Database Time
- AWR, ASH, ADDM
- Statspack
- dbms_xplan
- SQL Monitoring
- Application instrumentation
- SQL Trace, tkprof

SQL Trace, tkprof
Create tracefile using oradebug

Session information

```
select s.username,p.spid,s.program,s.terminal
from  v$session s, v$process p
where s.username ='<username>'
and s.paddr=p.addr and lower(s.program) like '%sqlplus%'
/
USERNAME                       SPID      PROGRAM
------------------------------ --------  --------
<username>                     5296268   sqlplus@srvdbitest1
```

Enable trace mode for this session

```
SQL> oradebug setospid 5296268
SQL> oradebug event 10046 trace name context forever, level 12
```

Disable trace mode for this session

```
SQL> oradebug event 10046 trace name context off
```

Find out where the tracefile is located

```
SQL> oradebug TRACEFILE_NAME
/u01/app/oracle/diag/rdbms/testdb/TESTDB/trace/TESTDB1_ora_5296268.trc
```

SQL Trace, tkprof
Create tracefile using dbms_system.set_ev

How to retrieve session information

```
select username,sid,serial#,program
     from v$session
     where username='<Username>' and lower(program) like '%sqlplus%'

USERNAME     SID    SERIAL#    PROGRAM
------------ ------ ---------- -------------- ----------------
<Username>   187    46136      extract@chbsslargus01(TNS V1-V3
<Username>   213    56816      sqlplus@<Hostname>(TNS V1-V3
```

Enable trace mode for this session

```
SQL> EXECUTE dbms_system.set_ev(213,56816,10046,12,'');
```

Disable trace mode for this session

```
SQL> EXECUTE dbms_system.set_ev(213,56816,10046,0,'');
```

Tracefile location

```
with 10g:${ORA_ADMIN_SID}/udump
with 11g:_${ORACLE_DIAG}/diag/rdbms/…/<SID>/trace/<SID>_ora_<ID>.trc
```

SQL Trace, tkprof
Create tracefile using dbms_monitor

```
select username,sid,serial#,program
     from v$session
     where username='<Username>' and lower(program) like '%sqlplus%'

USERNAME     SID    SERIAL#    PROGRAM
------------ ------ ---------- -------------- ----------------
<Username>   187    46136      extract@chbsslargus01(TNS V1-V3
<Username>   213    56816      sqlplus@<Hostname>(TNS V1-V3
```

Enable trace mode for this session

```
SQL> EXECUTE dbms_monitor.session_trace_enable(213,56816,WAITS=>TRUE,-
BINDS=>TRUE,PLAN_STAT=>'ALL_EXECUTIONS');
```

Disable trace mode for this session

```
SQL> EXECUTE dbms_monitor.session_trace_disable(213,56816);
```

SQL Trace, tkprof
Create tracefile for the current session

Enable trace for the current session

```
-- alter session set sql_trace=true; → Trace Level 1.
alter session set events '10046 trace name context forever, level 16';
-- or
alter session set events 'sql_trace wait=true, bind=true,
plan_stat=all_executions';
```

Disable trace for the current session

```
alter session set events '10046 trace name context off';
-- or
alter session set events 'sql_trace off';
```

SQL Trace, tkprof
10046 event level description

- 0: The debugging event is disabled.
- 1: The debugging event is enabled.
- 4: As in level 1, with additional information about bind variables.
- 8: As in level 1, plus detailed information about wait time.
- 16: As in level 1, plus the execution plans information is written to the trace file for each execution. Available as of 11.1 only.
- 32: As in level 1, but without the execution plans information. Available as of 11.1 only.
- 64: As in level 1, plus the execution plans information might be written for executions following the first one. The condition is that, since the last write of execution plans information, a particular cursor consumed at least one additional minute of DB time.

- In addition to the levels described above, you can also combine the levels 4 and 8 with every other level greater than 1. For example:
  - Level 12 (4 + 8): simultaneously enable level 4 and level 8.
  - Level 28 (4 + 8 + 16): simultaneously enable level 4, level 8 and level 16.
  - Level 68 (4 + 64): simultaneously enable level 4 and level 64.

SQL Trace, tkprof
Format tracefile using TKPROF

Tkprof allow us to read a tracefile into an human readable format

```
$ tkprof trace_file.trc formated_file.tkprof [options]
```

Most important tkprof options parameters

- **SORT**
  - Allows to sort the tkprof file content using different criterias
  - To sort by elapsed time **sort=prsela,exeela,fchela**
- **SYS**
  - Enables or disables the SYS executed statements
  - In some situation it can be interesting to activate it, but mostly of the time we use **sys=no**
- **EXPLAIN**
  - Execute the explain plan for specified user
  - Can also be used in some case if plan is not displayed **explain=user/passwd**

SQL Trace, tkprof
Example of tkprof output

```
$ tkprof tracefile.trc output.tkprof sort=prsela,exeela,fchela sys=no
$ tail output.tkprof

SELECT emp.ename, dept.dname
FROM emp, dept
  WHERE emp.deptno = dept.deptno

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.11       0.13          2          0          1           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          2          2          4          14
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.11       0.13          4          2          5          14

Misses in library cache during parse: 1
Optimizer goal: ALL_ROWS
Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
     14   MERGE JOIN
      4    SORT (JOIN)
      4     TABLE ACCESS (FULL) OF 'DEPT'
     14    SORT (JOIN)
     14     TABLE ACCESS (FULL) OF 'EMP'
```

SQL Trace, tkprof
Exercise: Trace service and module

Start trace for service APP and module 'New Order'

```
SQL> exec
sys.dbms_monitor.serv_mod_act_trace_enable(service_name=>'APP',module_
name =>'New Order',waits=>TRUE);
```

Run some activity

```
/home/oracle/swingbench/bin/charbench -cs //192.168.22.10x:1521/APP -u
soe -p soe -uc 10 -min 5 -max 50 -a -v -rt 00:01
```

Stop trace

```
SQL> exec
sys.dbms_monitor.serv_mod_act_trace_disable(service_name=>'APP',module
_name =>'New Order')
SQL> select * from v$diag_info where name='Diag Trace';
```

Go to trace directory and use tkprof on one .trc

- You can also group several .trc with trcsess

```
trcsess service=APP module="New Order" $(find -name "DB1_ora_*.trc" -
ctime -1) > ALL.TRC
tkprof ALL.TRC ALL.TXT sort='(exeela,fchela)'
```

Monitoring performance
Core Message

Oracle is very well instrumented

- Tune from DB Time down to more detail
- Focus only on what can improve performance significantly
- Add your own information (action/module/client_info)

AWR / Statspack

- If you don't have Diagnostics Pack install Statspack
- It's mandatory to analyze what happened in the past

Understand and test

- Check execution plan cardinality estimations
- Try other plans to see where the problem is

Any questions? Please do ask.