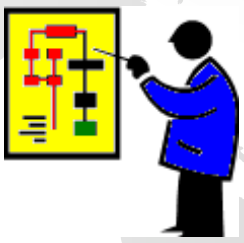
ORACLE TRACE DATA EVENTS 10046 AND 10053

Ric Van Dyke
Sr. DBA
Oracle Ace

AGENDA

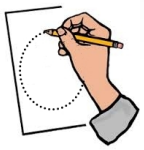
For both the 10046 and 10053 we will look at:

- Why Trace?
- How to trace?
- What is the trace?



THE TRACE EVENTS

- They are really "error messages"
  - **`ORA-10000`** to **`ORA-10999`**
  - Some are rather dangerous: **`ORA-10213: crash after control file write`**
- Only 2 are likely to be used
  - **`ORA-10046: enable SQL statement timing`**
  - **`ORA-10053: CBO Enable optimizer trace`**
- 10046 tracing
  - Excellent to see the "whole story" when something runs
- 10053 tracing
  - Can be insightful to understand "why" the optimizing picks a plan



10046 TRACE DATA

- Gives you the most information you can get in one place
  - An excellent tool for debugging and optimization
- Generally easy to read
  - It can be very long
  - It is repetitive, there are really only a few different types of lines
- Shows you "event by event" what happened
  - Contains timing for events (elapsed and CPU depending on the event)
  - Shows other statistics as appropriate about an event
  - Shows the execution plan with statistics
  - A draw back is it doesn't show predicate information

GETTING THE TRACE FILE

- Make sure you trace only the "thing" you want
  - Too much either way can give misleading information
- Best if you can turn tracing on and off within the code
- It's good to be able to read the raw trace data
  - A profile of the trace can be easier
- Classic way to get a trace:

```
alter session set events '10046 trace name context forever, level 12';
```

- Since version 10, this is what you should use:

```
EXEC DBMS_MONITOR.SESSION_TRACE_ENABLE (NULL, NULL, TRUE, TRUE, 'ALL_EXECUTIONS');
```


A FEW EXAMPLES OF DBMS_MONITOR

Tracing your own session:

```
EXEC DBMS_MONITOR.SESSION_TRACE_ENABLE (NULL, NULL, TRUE, TRUE,
'ALL_EXECUTIONS');
```

Tracing by client ID:

```
EXEC DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE ('RVD', TRUE, TRUE,
'ALL_EXECUTIONS');
```

Tracing by Service, Module and Action :

```
EXEC DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE ('MYSERICE', 'SALESPROCESSING'
,DBMS_MONITOR.ALL_ACTIONS,TRUE,TRUE);
```

Collecting stats on a particular Module and Action:

```
EXEC DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE (SERVICE_NAME => 'MYSERICE',
MODULE_NAME => 'LOAD PO TABLE', ACTION_NAME => 'DAILY');
```

For more details, see the Oracle *Database PL/SQL Packages and Types Reference*

OVERVIEW OF THE FILE

The file is made up of:

- The Preamble – Version of Oracle and the OS
- Session information – User and time trace was started
- Database calls – **PARSE/EXEC/FETCH** for each SQL
- Bind value information – Optional
- **WIAT** events – dispersed thru the file (optional)
- **STAT** lines – Execution plan with stats for each SQL
- **CLOSE** line – Shows when a cursor was closed

FOCUSING ON SQL

These lines below describe the phases or steps a SQL statement goes through. The **fetch** is the only one that should have multiples for a given execution.

This is the description of the cursor, not the actual parse:

```
PARSING IN CURSOR #335481272 len=199 dep=0 uid=148 oct=3 lid=148
tim=94359533757 hv=3249305567 ad='7ff090c8608' sqlid='c1y86dr0usxyz'
```

These are the actions: parsing, executing the SQL and fetching of the rows:

```
PARSE #335481272:
c=0,e=119,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=3626505148,tim=94359533756
```

```
EXEC #335481272:
c=0,e=57,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=3626505148,tim=94359533990
```

```
FETCH #335481272:
c=0,e=17546,p=32,cr=4,cu=0,mis=0,r=1,dep=0,og=1,plh=3626505148,tim=94359551688
```

THE WAIT EVENTS

- Which wait events show up depends on what is going on
- Some will nearly always show up
- The 3 parameters between the **ELA** and the **OBJ** will depend on the event
- As of 12.2 there are 1,809 events

```
WAIT #335481272: nam='SQL*Net message from client' ela= 283 driver
id=1111838976 #bytes=1 p3=0 obj#=348986 tim=94359552053
```

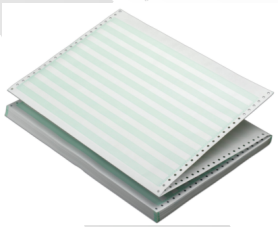
```
WAIT #335481272: nam='db file scattered read' ela= 11734 file#=6 block#=1157000
blocks=8 obj#=348985 tim=94359545948
```



```
set serveroutput on feedback on termout on heading on
/* flush the buffer pool to get some file io */
ALTER SYSTEM FLUSH BUFFER_CACHE;
declare
  /* dynamicly setting tracefile idenitifier with time and user name */
  t_trc_file varchar2(256):= 'alter session set tracefile_identifier='
      ||'DEMO99_'||to_char(sysdate, 'hh24miss')||'_'||user;
begin
    execute immediate t_trc_file;  -- Set tracefile identifier
end;
/
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(NULL, NULL, TRUE, TRUE,
'ALL_EXECUTIONS' );
SELECT
 emp.empno empno, emp.ename ename, emp.job job, emp.mgr mgr, emp.deptno deptno,
 dept.dname dname,  dept.loc loc
FROM
 emp
 JOIN dept ON emp.deptno = dept.deptno
Order by emp.ename;
EXECUTE DBMS_MONITOR.SESSION_TRACE_DISABLE(NULL, NULL);
SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Default Trace File';
```

EXAMPLE TRACE FILE

orcl_ora_10368_DEMO99_111518_RIC.trc



THE **TKPROF** UTILITY

- Installed by default
- Doesn't create a true profile
  - It's more of a summarization for each SQL within the trace
- Useful for basic summary info
  - Helpful to figure out where the issue likely is
- Many other profilers for 10046 trace files exist
  - Google something like "10046 trace file profiler"
  - Some are free some are not
- The full name is : **T**ransient **K**ernel **PRO**filer
  - The TK doesn't stand for Tom Kyte. ☺
- For more on **TKPROF** go to the Oracle *Database SQL Tuning Guide*

EXAMPLE OUTPUT TKPROF

```
********************************************************************************
SELECT
    emp.empno empno, emp.ename ename, emp.job job, emp.mgr mgr, emp.deptno deptno,
    dept.dname dname,  dept.loc loc
FROM
    emp
    JOIN dept ON emp.deptno = dept.deptno
Order by emp.ename
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.01         32          4          0          13
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.00       0.01         32          4          0          13
Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 148
Number of plan statistics captured: 1
```


EXAMPLE OUTPUT TKPROF (CONTINUED)

```
Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
        13         13         13  SORT ORDER BY (cr=4 pr=32 pw=0 time=17563 us starts=1 cost=6 size=585 card=13)
        13         13         13   MERGE JOIN  (cr=4 pr=32 pw=0 time=17536 us starts=1 cost=5 size=585 card=13)
         4          4          4    TABLE ACCESS BY INDEX ROWID DEPT (cr=2 pr=16 pw=0 time=12667 us starts=1 cost=2
                                                                      size=80 card=4)
         4          4          4     INDEX FULL SCAN DEPT_DEPTNO_PK (cr=1 pr=8 pw=0 time=11928 us starts=1 cost=1
                                                                     size=0 card=4)(object id 348985)
        13         13         13    SORT JOIN (cr=2 pr=16 pw=0 time=4853 us starts=4 cost=3 size=325 card=13)
        13         13         13     TABLE ACCESS BY INDEX ROWID BATCHED EMP (cr=2 pr=16 pw=0 time=4819 us starts=1
                                                                           cost=2 size=325 card=13)
        13         13         13      INDEX FULL SCAN EMP_DEPT_IDX (cr=1 pr=8 pw=0 time=3988 us starts=1 cost=1
                                                                    size=0 card=13)(object id 348991)

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                       2        0.00          0.00
  db file scattered read                          4        0.01          0.01
  SQL*Net message from client                     2        0.01          0.01
********************************************************************************
```

OPTIMIZER TRACING – EVENT 10053

- This traces the computations done by the optimizer to create the execution plan
- It is not easy to read and can be confusing
- Unlike the 10046 it is not a repetitive set of lines
- Best to have an idea of what you are looking for before you start
- Can help answer "why did the optimizer do that?"
- Best to use **`dbms_sqldiag.dump_trace`** to get the trace file:

```
dbms_sqldiag.dump_trace(p_sql_id=>'c1y86dr0usxyz', p_child_number=>0,
p_component=>'Optimizer', p_file_id=>'MY_TRACE');
```


SECTIONS OF THE 10053 TRACE

- Header
- Query block signature
- Optimizer information
- Parameters used by the optimizer
- Bug fix control environment
- Parameters in **`opt_param`** hint
- Column usage monitoring
- Query transformations
- Peeked values of the binds in SQL statement
- Query block text
- Query block signature
- System statistics information
- Base statistical information
- Access path analysis for tables
- Optimizer statistics and computations
- General plans
- SQL Dump
- Plan Table Output
- Other XML Data
- Outline Data
- Optimizer state dump
- Footer

Each section is different from the others, and the length of some will depend on the SQL being parsed.

THE GOOD NEWS

- Of all those sections only a few will likely be of interest
- This is where the plan is built:

- Final query after transformations
- Base statistical information
- Access path analysis for tables
- General plans

FINAL QUERY AFTER TRANSFORMATIONS

This is the only way to see the query that is really optimized. The final query is one line, all upper case, and fully qualified.

```
SELECT
    emp.empno empno, emp.ename ename, emp.job job, emp.mgr mgr, emp.deptno deptno,
    dept.dname dname,  dept.loc loc
FROM
    emp
    JOIN dept ON emp.deptno = dept.deptno
Order by emp.ename;
```

```
Final query after transformations:******* UNPARSED QUERY IS *******

SELECT "EMP"."EMPNO" "EMPNO","EMP"."ENAME" "ENAME","EMP"."JOB" "JOB","EMP"."MGR"
"MGR","EMP"."DEPTNO" "DEPTNO","DEPT"."DNAME" "DNAME","DEPT"."LOC" "LOC" FROM
"RIC"."EMP" "EMP","RIC"."DEPT" "DEPT" WHERE "EMP"."DEPTNO"="DEPT"."DEPTNO" ORDER BY
"EMP"."ENAME"
```

BASE STATISTICAL INFORMATION

This shows the stats on each table and it's indexes

```
**************************************
BASE STATISTICAL INFORMATION
**********************
Table Stats::
  Table: DEPT  Alias: DEPT
  #Rows: 4  SSZ: 0  LGR: 0  #Blks:  5  AvgRowLen:  20.00  NEB: 0  ChainCnt:  0.00  ScanRate:  0.00
SPC: 0  RFL: 0  RNF: 0  CBK: 0  CHR: 0  KQDFLG: 1
  #IMCUs: 0  IMCRowCnt: 0  IMCJournalRowCnt: 0  #IMCBlocks: 0  IMCQuotient: 0.000000
  Column (#1): DEPTNO(NUMBER)
    AvgLen: 3 NDV: 4 Nulls: 0 Density: 0.250000 Min: 0.000000 Max: 10.000000
Index Stats::
  Index: DEPT_DEPTNO_PK  Col#: 1
  LVLS: 0  #LB: 1  #DK: 4  LB/K: 1.00  DB/K: 1.00  CLUF: 1.00  NRW: 4.00 SSZ: 0.00 LGR: 0.00 CBK:
0.00 GQL: 0.00 CHR: 0.00 KQDFLG: 0 BSZ: 8192
  KKEISFLG: 1
**********************
(All the other tables in the SQL and indexes on the tables would follow this)
```


ACCESS PATH ANALYSIS FOR TABLES

This cost out each table independently for access

```
Access path analysis for EMP
***************************************
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for EMP[EMP]
  SPD: Return code in qosdDSDirSetup: NOCTX, estType = TABLE
 kkecdn: Single Table Predicate:"EMP"."DEPTNO" IS NOT NULL
  Table: EMP  Alias: EMP
    Card: Original: 14.000000  Rounded: 13  Computed: 13.000000  Non Adjusted: 13.000000
  Scan IO  Cost (Disk) =   3.000000
  Scan CPU Cost (Disk) =   39947.200000
  Cost of predicates:
    io = NOCOST, cpu = 20.000000, sel = 0.928571 flag = 2048  ("EMP"."DEPTNO" IS NOT NULL)
  Total Scan IO  Cost  =   3.000000 (scan (Disk))
                         + 0.000000 (io filter eval) (= 0.000000 (per row) * 14.000000 (#rows))
                       =   3.000000
. . . (Other access paths here, indexes that could be used with every possible way an index can
be used)
 Best:: AccessPath: IndexRange
  Index: EMP_DEPT_IDX
         Cost: 2.000851  Degree: 1  Resp: 2.000851  Card: 13.000000  Bytes: 0.000000
```

GENERAL PLANS

This puts it all together doing the joins

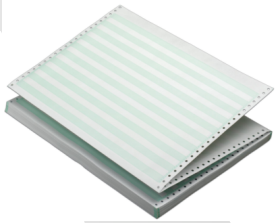
```
GENERAL PLANS
*************************************
Considering cardinality-based initial join order.
Permutations for Starting Table :0
Join order[1]:  DEPT[DEPT]#0  EMP[EMP]#1
***************
Now joining: EMP[EMP]#1
***************
NL Join
  Outer table: Card: 4.000000  Cost: 3.001465  Resp: 3.001465  Degree: 1
. . . (The other joins (Sort-Merge and Hash), also other possible
permutations of the joins)
```


THESE OTHER USEFUL SECTIONS

- Peeked values of the binds in SQL statement
  - Can be useful to understand how these values can change a plan
- SYSTEM STATISTICS INFORMATION
  - Mostly nice to know info
  - System stats generally should be left to the defaults
- Plan Table
  - Shows the plan chosen in traditional explain plan format
- Outline Data
  - This shows the hints that would be used to get this plan
  - This would be the baseline plan if captured

A LOOK AT A 10053 TRACE

orcl_ora_9556_MY_TRACE.trc



QUESTIONS

ORACLE TRACE DATA EVENTS 10046 AND 10053

Ric Van Dyke
Sr. DBA
Oracle Ace

ABOUT ZIONE SOLUTIONS

- Leading System Integrator & IT Service Provider
- Strong Big Data & Business Intelligence Expertise
- Highly Skilled Pool of Resources
- Oracle Service Partner
- Cloud & Managed Services
- Experience Delivering "Industry Best Practices"
- Agile Scrum-Based Application Life Cycle Management

Zione Solutions, LLC.
37000 Grand River Avenue, STE 355
Farmington Hills, MI 48335

Phone: (248)-442-7404
Email : contact@zionesolutions.com
Site : www.zionesolutions.com

Thanks and have a great day!