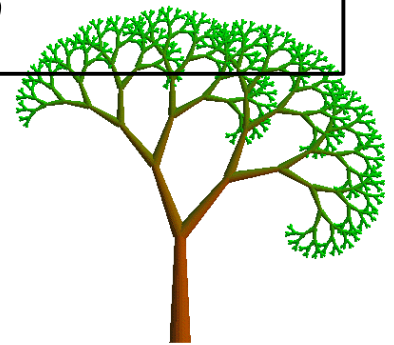




Query Tuning Foundations

Execution Plan

B - Find the Plan



Muriel Fry - Version V1.4 - Sept 2009

- ***Execution Plan***
 - A - Row Source Operations
 - ***B - Find the Plan***
 - C - Interpret the Plan



Execution Plan

- The execution plan of a SQL statement is composed of small building blocks called “row sources.”
- The combination of row sources for a statement is called the execution plan.
- By using parent-child relationships, the explain plan can be displayed in a tree (Text or Graphical)

ORACLE

Interpret the Plan

An execution plan is the output of the optimizer and is presented to the execution engine for implementation. It instructs the execution engine as to the operations it needs to perform to retrieve the data required by a query most efficiently. The implementation of these operations is down to the execution engine to implement.

When a statement is executed, the optimizer performs many steps. Each of these steps either retrieves rows of data physically from the database or prepares them in some way for the user issuing the statement. The combination of steps that are used to run a statement is called an *execution plan*.

An execution plan includes an access method for each table that the statement accesses and an ordering of the tables (the join order). The optimizer also uses different methods to combine the rows from multiple tables (join method). The steps of the execution plan are not performed in the order in which they are numbered. there is parent – child relationship between steps.

<Note.235530.1> Recommended Methods for Obtaining a Formatted Explain Plan

Where to find Execution Plan

1 - PLAN_TABLE – Theoretical plan

Plan_table Output ('TYPICAL')

Plan_table Output ('ALL')

Plan_table Output (1)& (2) ('ADVANCED')

Plan_table AUTOTRACE (1) & (2)

2 - V\$SQL_PLAN – Actual plan

9i v\$sql_plan – Method 1

9i v\$sql_plan – Method 2 (1)&(2)

9i v\$sql_plan Output with rowsource execution statistics

10g v\$sql_plan - DBMS_XPLAN Output

V\$sql_plan Output ('ADVANCED')

V\$sql_plan Output with rowsource execution statistics

3 - DBA_HIST_SQL_PLAN - DBMS_XPLAN output

DBA_HIST_SQL_PLAN - AWR report awrsqrpt.sql

4 – STAT\$SQL_PLAN Statspack sprepsql report

5 - Event 10046 trace file - & tkprof

6 - Event 10132 trace file

7 - 10053 trace file

8 - Process state Trace file

Other views with info about SQL execution



Objectives

- After completing this lesson, you should be able to do the following:
 - Gather/Display Execution plan
 - Understand differences between Plans available
 - Understand purpose of information available with Plan

ORACLE

Objectives

Objective from this part is to be more familiar with plan itself.



Where to find Execution Plan

- 1 - PLAN_TABLE (explain plan for ...)
- 2 - V\$SQL_PLAN (Library Cache)
- 3 - DBA_HIST_SQL_PLAN (AWR)
- 4 - STATS\$SQL_PLAN (Statspack)
- 5 - Event 10046 trace file & tkprof
- 6 - Event 10132 trace file
- 7 - Event 10053 trace file
- 8 - Process state dump tracefile in 10g

ORACLE

Where to Find Execution Plan

<Note.235530.1> Recommended Methods for Obtaining a Formatted Explain Plan To gather plans.

There are many other available explain plan scripts, the most useful of which is probably:

<Note.215187.1> SQLTXPLAIN.SQL - Enhanced Explain Plan and related diagnostic info for one SQL statement.

Following slides are describing more in depth where/how to find execution plan. They will include examples, you'll be able to compare methods.

The EXPLAIN PLAN command allows you to view the execution plan that the optimizer might to execute a SQL statement. This command is very useful because it outlines the plan that the optimizer may use and inserts it in a table called **PLAN_TABLE** without executing the SQL statement.

V\$SQL_PLAN provides a way of examining the execution plan for cursors that were recently executed. Information in V\$SQL_PLAN is very similar to the output of an EXPLAIN PLAN statement:

- **EXPLAIN PLAN** shows a theoretical plan that can be used if this statement were to be executed.

- **V\$SQL_PLAN** contains the actual plan used.
- **AWR** infrastructure and statspack.snap, if level is > 5, store plans from top sqls. Plans are recorded into dba_hist_sql_plan and stats\$sql_plan.
- Plan and row source operations are dumped in **10046 trace file** at cursor closure time in "STAT#" lines.
- **Event 10132** is undocumented event which dump plan in a trace file at the end of sql compilation.
- **Event 10053** which is used to dump CBO computations may include plan.
- Starting 10gR2, when we **dump process state** (or **errorstack from a process**), execution plan is included in trace file generated.



1 - PLAN_TABLE – Theoretical plan

- Explain plan for ...
- Generates optimizer execution plan
- Stores plan in a PLAN table

```
EXPLAIN PLAN FOR
  SELECT e.ename, d.dname
  FROM emp e, dept d
  WHERE e.deptno = d.deptno;
Explained.
```

- Advantage : SQL is not Execute
- Disadvantage : Maybe NOT actual Plan, Bind not Peeked

ORACLE®

PLAN_TABLE Theoretical plan

The EXPLAIN PLAN command is used to generate the execution plan that the optimizer uses to execute a SQL statement. It does not execute the statement but simply produces the plan that may be used and inserts this plan into a table.

Complete list / version is available in documentation :

- http://st-doc.us.oracle.com/9.0/9202/server.920/a96533/ex_plan.htm#17522
- http://st-doc.us.oracle.com/10/101/server.101/b10752/ex_plan.htm#17522
- http://st-doc.us.oracle.com/10/102/server.102/b14211/ex_plan.htm#sthref1934

DBMS_XPLAN package methods can be used to display the execution plan generated by the EXPLAIN PLAN command

DBMS_XPLAN Package: Overview

The DBMS_XPLAN package provides an easy way to display the output of the EXPLAIN PLAN command. It supplies one table function that can be used to retrieve and display the execution plan. **DISPLAY** formats and displays the contents of a plan table

There 3 are Parameters :

- TABLE_NAME default PLAN_TABLE
- STATEMENT_ID default NULL
- FORMAT default TYPICAL, see below :

'**TYPICAL**': This is the default. Display most information of the explain plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. This excludes only PROJECTION, ALIAS and REMOTE SQL information (see below).

'**ALL**': Maximum user level, like typical with additional informations (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed as well as SQL statements generated for parallel execution (9i))

'**ADVANCED**' Outline data

Plan_table Output ('TYPICAL')

```
Explain plan for select * from EMP where ename = 'KING';
```

```
select plan_table_output from table(DBMS_XPLAN.DISPLAY());
```

Plan hash value: 351108634

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	27	2
* 1	TABLE ACCESS FULL	EMP	1	27	2

Predicate Information (identified by operation id):

1 - filter("EMP"."ENAME"='KING')

Note

- outline "01" used for this statement

ORACLE

PLAN_TABLE Output ('TYPICAL')

To use EXPLAIN PLAN, you must:

- First use the EXPLAIN PLAN command to explain a SQL statement
- Retrieve the plan steps by using the DISPLAY method in the DBMS_XPLAN package

DBMS_XPLAN.DISPLAY is displaying below information

Plan hash value:

Numerical representation of the SQL plan for this cursor. Comparing one PLAN_HASH_VALUE to another easily identifies whether or not two plans are the same (rather than comparing the two plans line by line).

Plan operations ordered by Id

Each row in the output table corresponds to a single step in the execution plan. Note that the step Ids with asterisks are listed in the Predicate Information section.

Predicate Information (identified by operation id):

The filter to be applied will only return rows where `""EMP"."ENAME"='KING'""`. Each row is checked to determine that it matches the condition so that the engine will only output rows for which `("EMP"."ENAME"='KING')`. Filter Conditions filter rows after fetch operation and output filtered rows.

Access Conditions can be seen in **“Predicate Information”** Section (In that case, Oracle accesses only matching rows)

Note

This section outputs useful information such

- Dynamic sampling used
- Rule based Optimizer used
- Outline used
- SQL profile used
- Star transformation used

Plan_table Output ('ALL')

```
explain plan for
select * from emp@DD where ename= 'KING'
union all
select * from emp where ename= 'KING';
```

```
select plan_table_output from table(DBMS_XPLAN.DISPLAY(null,null,'ALL'))
```

Id	Operation	Name	Rows	Bytes	Cost	Inst	IN-OUT
0	SELECT STATEMENT		4	232	4		
1	UNION-ALL						
2	REMOTE		1	37	2	DD	R->S
* 3	TABLE ACCESS FULL	EMP	3	195	2		

Predicate Information (identified by operation id):

3 - filter("EMP"."ENAME"='KING')

Remote SQL Information (identified by operation id):

2 - SELECT "EMPNO","ENAME","JOB","SAL","COMM","DEPTNO"
FROM "EMP" "EMP"
WHERE "ENAME"='KING' (accessing 'DD.FR.ORACLE.COM')

ORACLE

PLAN_TABLE Output ('ALL')

'ALL': Maximum user level, like typical with additional informations (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed as well as SQL statements generated for parallel execution)

In that example, 'ALL' parameter is used instead of 'TYPICAL', this displays section :

Remote SQL Information (identified by operation id) : which is useful with distributed SQLs as Remote SQL is displayed.

Note section does not exist because there is nothing specific associated with that SQL

Query Block Name / Object Alias (identified by operation id): A query block is a self-contained DML against a table. A query block can be a top-level DML or a subquery.

Column Projection Information (identified by operation id): We can see that Oracle has provided the columns to be projected at each stage of the execution plan. This information can be extremely useful for debugging complex statements that are joining incorrectly or inefficiently.

Example from simple SQL with 'ALL' parameters :

```
SQL> explain plan for select * from emp where ename= 'KING';
```

```
select plan_table_output from table(DBMS_XPLAN.DISPLAY(null,null,'ALL'));
```

```
Plan hash value: 3956160932
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time		

0	SELECT STATEMENT		1	32	3 (0)	00:00:01		
* 1	TABLE ACCESS FULL	EMP	1	32	3 (0)	00:00:01		

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

```
-----
1 - SEL$1 / EMP@SEL$1
```

```
Predicate Information (identified by operation id):
```

```
-----
1 - filter("ENAME"='KING')
```

```
Column Projection Information (identified by operation id):
```

```
-----
1 - "EMP"."EMPNO"[NUMBER,22], "ENAME"[VARCHAR2,10],
    "EMP"."JOB"[VARCHAR2,9], "EMP"."MGR"[NUMBER,22],
    "EMP"."HIREDATE"[DATE,7], "EMP"."SAL"[NUMBER,22],
    "EMP"."COMM"[NUMBER,22], "EMP"."DEPTNO"[NUMBER,22]
```

Plan_table Output('ADVANCED')

```
var A number
exec :A := 0;
explain plan for select avg(T.i) from T where T.j= :A;
```

```
select plan_table_output from table(dbms_xplan.display('PLAN_TABLE',null, 'ADVANCED'));
```

```
-----
| Id | Operation                | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT         |      |    1  |    7  |    56  (2)| 00:00:01 |
|  1 |   SORT AGGREGATE         |      |    1  |    7  |          |          |
|*  2 |    TABLE ACCESS FULL    | T     | 5000  | 35000 |    56  (2)| 00:00:01 |
-----
```

```
...
Outline Data
-----
/*+
  BEGIN_OUTLINE_DATA
  FULL(@"SEL$1" "T"@"SEL$1")
  OUTLINE_LEAF(@"SEL$1")
  ALL_ROWS
  OPTIMIZER_FEATURES_ENABLE('10.2.0.3')
  IGNORE_OPTIM_EMBEDDED_HINTS
  END_OUTLINE_DATA
*/
```

ORACLE

PLAN_TABLE Output ('ADVANCED')

ADVANCED format is only available from 10gR2 and onwards.

It is "non documented"/unsupported output format which includes all sections from ALL format plus this one

- Outline Data : This is as set of hints to reproduce plan
- Outline Data : This section may be useful for support to reproduce PLAN from customer

This is same section which is displayed in 10053:

<Note.403626.1> How to get an outline hint from 10053 trace

There is no Bind Peeking when EXPLAIN PLAN FOR is used.

When ADVANCED format is used with v\$sql_plan, there is one more section :

➤ "Peeked Binds (identified by position):" → (see v\$sql_plan Output ('ADVANCED') slide)

That is exactly the same example which is used with plan_table and v\$sql_plan so we can compare output:

- Note that Plans are **NOT** the same,
- Plan reported by EXPLAIN PLAN is **NOT** actual plan

```
CREATE TABLE T(i number, j number, c2 char(100));
declare
  i number;
begin
  for i in 1..10000 loop
    insert into T values(i, '1','HELLO'||i);
  end loop;
update T set j= 0 where i >= 9999;
commit;
end;
/
create index J on T(j);
exec dbms_stats.gather_table_stats(NULL,'T', cascade=> TRUE,-
    method_opt => 'for all columns size 5');

var A number
exec :A := 0;

explain plan for select avg(T.i) from T where T.j= :A;
```

```
select plan_table_output from table(dbms_xplan.display(null,null,'ADVANCED'));
```

Plan hash value: 2966233522

Id		Operation		Name		Rows		Bytes		Cost (%CPU)		Time	

0		SELECT STATEMENT				1		7		56 (2)		00:00:01	
1		SORT AGGREGATE				1		7					
* 2		TABLE ACCESS FULL		T		5000		35000		56 (2)		00:00:01	

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

- 1 - SEL\$1
- 2 - SEL\$1 / T@SEL\$1

Outline Data

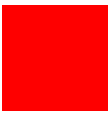
```
/*+
  BEGIN_OUTLINE_DATA
  FULL(@"SEL$1" "T"@"SEL$1")
  OUTLINE_LEAF(@"SEL$1")
  ALL_ROWS
  OPTIMIZER_FEATURES_ENABLE('10.2.0.3')
  IGNORE_OPTIM_EMBEDDED_HINTS
  END_OUTLINE_DATA
*/
```

Predicate Information (identified by operation id):

- 2 - filter("T"."J"=TO_NUMBER(:A))

Column Projection Information (identified by operation id):

- 1 - (#keys=0) AVG("T"."I")[22]
- 2 - "T"."I"[NUMBER,22]



Plan_table – AUTOTRACE

- Sql*Plus facility
- Introduced with Oracle7.3
- Needs a **PLAN_TABLE**
- Needs the **PLUSTRACE** role to retrieve statistics from some v\$ views
- Produces execution plan and/or statistics after running the query
- Maybe NOT the ACTUAL plan when Bind peeking (recursive EXPLAIN PLAN FOR)

ORACLE

PLAN_TABLE AUTOTRACE

The **PLAN_TABLE** table must be available in the current schema to use AUTOTRACE.

The **PLUSTRACE** role is created and granted to the DBA role by running the supplied script
\$ORACLE_HOME/sqlplus/admin/plustrce.sql

The PLUSTRACE role contains select privilege on three v\$ views; these privileges are necessary to generate AUTOTRACE statistics. The PLUSTRACE role can be granted manually to non-DBA users. This is the contents of the plustrce.sql script:

```
drop role plustrace;
create role plustrace;
grant select on v_$sesstat to plustrace;
grant select on v_$statname to plustrace;
grant select on v_$session to plustrace;
grant plustrace to dba with admin option;
```

- <http://st-doc.us.oracle.com/9.0/9202/server.920/a90842/ch9.htm#1006527>
- http://st-doc.us.oracle.com/10/102/server.102/b14211/ex_plan.htm

19.1.4 EXPLAIN PLAN Restrictions

Oracle does not support EXPLAIN PLAN for statements performing implicit type conversion of date bind variables. With bind variables in general, the EXPLAIN PLAN output might not represent the real execution plan.

Plan_table Output - (AUTOTRACE)

```
var A number
exec :A := 0;
set autotrace on
select avg(T.i) from T where T.j= :A;
      AVG(T.I)
-----
      9999.5
```

```
Execution Plan
-----
0  SELECT STATEMENT Optimizer=ALL_ROWS (Cost=56 Card=1 Bytes=7)
1  0  SORT (AGGREGATE)
2  1  TABLE ACCESS (FULL) OF 'T' (TABLE) (Cost=56 Card=5000 Bytes=35000)

Statistics
-----
          0  recursive calls
          0  db block gets
          3  consistent gets
          0  physical reads
          0  redo size
       414  bytes sent via SQL*Net to client
       504  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

ORACLE

Example with AUTOTRACE ON - Query with Bind Variable

Same example is used in v\$sql_plan output and AUTOTRACE output to compare them :

- Note that Plans are **NOT** the same,
- Plan reported by AUTOTRACE is **NOT** actual plan

```
CREATE TABLE T(i number, j number, c2 char(100));
```

```
declare
  i number;
begin
  for i in 1..10000 loop
    insert into T values(i, '1','HELLO'||i);
  end loop;
  update T set j=0 where i >= 9999;
commit;
```

```

end;
/

create index J on T(j);
exec dbms_stats.gather_table_stats(NULL,'T', cascade=> TRUE,-
                                method_opt => 'for all columns size 5');

```

```

var A number
exec :A := 0;
set autotrace on
select avg(T.i) from T where T.j= :A;

```

```

      AVG(T.I)
-----
      9999.5

```

Execution Plan

```

-----
0 SELECT STATEMENT Optimizer=ALL_ROWS (Cost=56 Card=1 Bytes=7)
1 0 SORT (AGGREGATE)
2 1  TABLE ACCESS (FULL) OF 'T' (TABLE) (Cost=56 Card=5000 Bytes=35000)

```

Statistics

```

-----
      0 recursive calls
      0 db block gets
      3 consistent gets
      0 physical reads
      0 redo size
    414 bytes sent via SQL*Net to client
    504 bytes received via SQL*Net from client
      2 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)

```

1 rows processed



2 - V\$SQL_PLAN - Actual plan

- New in 9i
- Shows the "actual" execution plan
 - Explain Plan gives a theoretical plan
- Contains the execution plan of every cursor in the library cache (including childs)
- Links to V\$SQL
 - address, hash_value, CHILD_NUMBER (9i)
 - SQL_ID, CHILD_NUMBER (10g)
- Enabling execution trace (10046) is another way to display actual execution plan.
 - Plan are the same, information available different

ORACLE

V\$SQL_PLAN Actual plan

This view provides a way of examining the execution plan for cursors that were recently executed. The information in this view is very similar to the output from the PLAN_TABLE.

V\$SQL_PLAN shows the plan for a cursor rather than for all cursors associated with a SQL statement. The difference is that a SQL statement can have more than one cursor associated with it, with each cursor further identified by a **CHILD_NUMBER**. For example, the same statement executed by different users has different cursors associated with it if the object being referenced is in a different schema. The V\$SQL_PLAN table can be used to see the different plans for different child cursors of the same statement.

On 10g DBMS_XPLAN can be used to query v\$sql_plan(dbms_xplan.display_cursor)

On 9i There is no package to read v\$sql_plan. See 2 methods in

- "9i v\$sql_plan – Output – 1" Slide
- "9i v\$sql_plan – Output – 2" Slide

Other useful links to dump Actual plan in 9i

- <http://www.scaleabilities.co.uk/content/view/26/9/>
- <http://jonathanlewis.wordpress.com/2006/12/12/plans-in-memory/>
- http://asktom.oracle.com/pls/asktom/f?p=100:11:0:::P11_QUESTION_ID:10353453905351

Explain Plan with Bind Variable Peeking without Executing the Statement :

- http://www.jlcomp.demon.co.uk/faq/how_to_explain.html

9i v\$sql_plan – Method 1

```
SQL> SeLeCt * from dept d WHERE d.deptno= 10;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK

```
select plan_table_output from TABLE( dbms_xplan.display (
  'DYN_PLAN'
  , (select rawtohex(address)||'_'||child_number x
      from v$sql where sql_text like 'SeLeCt%')
  , 'ALL')) ;
```

OR

```
select plan_table_output from TABLE( dbms_xplan.display (
  'DYN_PLAN'
  , (select rawtohex(address)||'_'||child_number x
      from v$sql where hash_value=&HV and child_number=&CH)
  , 'ALL')) ;
```

ORACLE

9i V\$SQL_PLAN – Method 1

Prerequisite to output plan from v\$sql_plan

```
create view DYN_PLAN to be able to use DBMS_XPLAN on it
connect / as sysdba
create or replace view DYN_PLAN as
select rawtohex(address) || '_' || child_number statement_id, sysdate timestamp,
operation, options, object_node, object_owner, object_name, 0 object_instance,
optimizer, search_columns, id, parent_id, position, cost, cardinality, bytes, other_tag,
partition_start, partition_stop, partition_id, other, distribution, cpu_cost, io_cost,
temp_space, access_predicates, filter_predicates from v$sql_plan;
create public synonym DYN_PLAN for DYN_PLAN;
```

9i v\$sql_plan Output - Method 1

```
select plan_table_output from TABLE( dbms_xplan.display
('DYN_PLAN'
, (select rawtohex(address)||'_'||child_number x
    from v$sql
    where hash_value=793380943
    and child_number=0)
, 'ALL')) ;
```

```
-----
|Id|Operation                               | Name   | Rows | Cost |
-----
| 0|SELECT STATEMENT                        |        |      |      |
| 1| TABLE ACCESS BY INDEX ROWID          | DEPT   |    1 |    1 |
|*2|  INDEX UNIQUE SCAN                    | PK_DEPT|    1 |      |
-----
2 - access("DEPTNO"=10)
```

ORACLE

Example to output plan from v\$sql_plan using DYN_PLAN

```
SQL> SeLeCt * from dept d WHERE d.deptno= 10;
DEPTNO DNAME          LOC
-----
```

```
10 ACCOUNTING      NEW YORK
```

```
select hash_value,child_number from v$sql where sql_text like 'SeLeCt%';
```

```
HASH_VALUE CHILD_NUMBER
-----
793380943      0
```

```
select plan_table_output from TABLE( dbms_xplan.display (
'DYN_PLAN'
, (select rawtohex(address)||'_'||child_number x
    from v$sql where hash_value=793380943 and child_number=0)
, 'ALL')) ;
```

9i v\$sql_plan – Method 2

```
select address,hash_value,sql_text from v$sqltext
  where sql_text like '%--%';

create table P1 as select * from v$sql_plan
  where hash_value=&hash_value and address = '&address';

alter table P1 add (STATEMENT_ID VARCHAR2(30));
alter table P1 add (timestamp date);

select distinct child_number from P1;
-- For each child, dump plan like below
update P1 set timestamp=sysdate where child_number = &CHLD;
commit;
select plan_table_output from
  table(dbms_xplan.display('P1',null,'ALL'));
```

ORACLE

9i V\$SQL_PLAN – Method 2

```
select /* SUP01 */ e.*
  from emp e, dept d
 where e.deptno = d.deptno;
```

```
select address,hash_value,sql_text
  from v$sqltext
 where sql_text like '%SUP01%';
```

```
ADDRESS  HASH_VALUE  SQL_TEXT
-----  -
6B619714  232169476  like '%SUP01%'
6B7FA1A0  1805350536  select /* SUP01 */ e.* from emp e,
```

dept d where e.deptno = d.dep

```
create table P1 as
select * from v$sql_plan where hash_value=1805350536 and address='6B7FA1A0';
```

```
alter table P1 add (STATEMENT_ID VARCHAR2(30));
alter table P1 add (timestamp date);
select distinct child_number from P1;
```

CHILD_NUMBER

```
-----
          0
          1
```

```
update P1 set timestamp=sysdate where child_number = &CHLD;
Enter value for chld: 0
```

```
select plan_table_output from table(dbms_xplan.display('P1',null,'ALL'));
```

```
-----
| Id | Operation                | Name      | Rows  | Bytes | Cost |
-----
|  0 | SELECT STATEMENT          |           |       |       |      |
|  1 |   NESTED LOOPS            |           |       |       |      |
|  2 |     TABLE ACCESS FULL    | EMP       |       |       |      |
|*  3 |       INDEX UNIQUE SCAN   | PK_DEPT   |       |       |      |
-----
```

3 - access("DEPTNO"="DEPTNO")

Note: rule based optimization

```
update P1 set timestamp=sysdate where child_number = &CHLD;
Enter value for chld: 1
```

```
select plan_table_output from table(dbms_xplan.display('P1',null,'ALL'));
```

```
-----
| Id | Operation                | Name      | Rows  | Bytes | Cost |
-----
|  0 | SELECT STATEMENT          |           |       |       | 2    |
|  1 |   NESTED LOOPS            |           | 14    | 560   | 2    |
|  2 |     TABLE ACCESS FULL    | EMP       | 14    | 518   | 2    |
|*  3 |       INDEX UNIQUE SCAN   | PK_DEPT   | 1     | 3     |      |
-----
```

3 - access("DEPTNO"="DEPTNO")



9i v\$sql_plan with Row Source Execution Statistics

```
SQL> SELeCt * from dept d WHERE d.deptno= 10;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK

```
HASH_VALUE CHI SQL_TEXT
```

```
2007160847 - 0 SELeCt * from dept d WHERE d.deptno= 10
```

Plan Table

Operation	Name	Starts	E-Rows	A-Rows	Buffers
TABLE ACCESS BY INDEX ROWID	DEPT	1	1	2	1
INDEX UNIQUE SCAN	PK_DEPT	1	1	1	1

ORACLE

9i v\$sql_plan with execution Row Source Statistics

information comes from v\$sql_plan_statistics_all :

<http://st-doc.us.oracle.com/9.0/9202/server.920/a96533/sqlviews.htm#36258>

Below is query which has been used to output plan in slide. Estimated and Actual Row information are both in same chart, and so that's easy to compare them.

Note 214106.1 Using TKProf to compare actual and predicted row counts

Other columns can be used to output execution information.

```
set pagesize 600
set tab off
set linesize 140
set echo off
set long 4000
```

```

col TQID format A4
col "SLAVE SQL" format A95 WORD_WRAP
col address format A12
col sql_hash format A15
col exec format 9999
col sql_text format A75 WORD_WRAP
repfooter off;
set timing off veri off space 1 flush on pause off termout on numwidth 10;

col "HASH_VALUE CHILD Nb" format A14
select hash_value||' - '|| child_number "HASH_VALUE CHILD Nb", sql_text
from v$sql where (hash_value,CHILD_NUMBER)
= (select hash_value, child_number from v$sql where sql_text like 'SELeCt%');

select '| Operation | Name | Starts | E-Rows |
A-Rows | Buffers |' as "Plan Table" from dual
union all /* QWEKLOIPYRTJHH7 */
select ' -----
-----'
from dual
union all
select rpad('| '||substr(lpad(' ',1*(depth))||operation|| decode(options, null,'',' '
' ||options), 1, 33), 34, ' ')||'|' ||
rpad(substr(object_name||' ',1, 21), 22, ' ')||'|' ||
lpad(starts, 9, ' ') || '|' || lpad(cardinality, 9, ' ') || '|' ||
lpad(outrows, 9, ' ') || '|' || lpad(buffers, 9, ' ') || '|'
as "Explain plan"
from (select /*+ no_merge */
pa.HASH_VALUE, pa.ID, pa.DEPTH, pa.POSITION, pa.OPERATION,
pa.OPTIONS, pa.COST COST, pa.CARDINALITY CARDINALITY,
pa.BYTES BYTES, pa.OBJECT_NODE, pa.OBJECT_OWNER,
pa.OBJECT_NAME, pa.OTHER_TAG, pa.PARTITION_START,
pa.PARTITION_STOP, pa.DISTRIBUTION, pa.last_starts starts,
pa.last_OUTPUT_ROWS outrows,
(pa.last_cr_buffer_gets + pa. last_cu_buffer_gets) buffers
from v$sql_plan_statistics_all pa
where (pa.hash_value,pa.CHILD_NUMBER) = (
select hash_value, child_number
from v$sql
where sql_text like 'SELeCt%')
)
union all select ' -----
-----' from dual;

```



10g v\$sql_plan DBMS_XPLAN Output

DBMS_XPLAN.DISPLAY_CURSOR

Last executed statement from v\$sql_plan

```
select * from dept where deptno=10;
```

```
select plan_table_output from  
table(dbms_xplan.display_cursor()));
```

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	1
1	TABLE ACCESS BY INDEX ROWID	DEPT	1	1
* 2	INDEX UNIQUE SCAN	PK_DEPT	1	

Predicate Information (identified by operation id):

2 - access("DEPT"."DEPTNO"=10)

ORACLE

10g V\$SQL_PLAN – DBMS_XPLAN Output

This package runs with the privileges of the calling user, not the package owner (SYS).

The table function **DISPLAY_CURSOR** requires select privileges on the following fixed views: V\$SQL_PLAN, V\$SESSION, and V\$SQL_PLAN_STATISTICS_ALL. These privileges are automatically granted as part of the SELECT_CATALOG_ROLE.

dbms_xplan.display_cursor(SQL_ID, CURSOR_CHILD_NO, FORMAT)

if SQL_ID is NULL, then this outputs plan from last executed SQL (See Bug 6501604)

```
SELECT e.ename, d.dname FROM emp e, dept d WHERE e.deptno =d.deptno and d.deptno= 10;  
select plan_table_output from table (dbms_xplan.display_cursor(null,null, 'TYPICAL'));
```

More information available with these parameters (see Detail in "Plan_Table Output ('ALL')\" slide)

```
select plan_table_output from table (dbms_xplan.display_cursor('c41yb2spsmdp', 0, 'ALL'));
```

If session &SID is running a very long SQL. PLAN can be dumped during its execution with following SQL (SID is session_ID from session running SQL) :

```
select t.*
from v$session s
table(dbms_xplan.display_cursor(s.sql_id,s.sql_child_number,'TYPICAL')) t
where sid=&SID;
```

V\$sql_plan Output ('ADVANCED')

```
var A number
exec :A := 0;
select avg(T.i) from T where T.j= :A;
```

```
select plan_table_output from table(dbms_xplan.display_cursor(null, null, 'ADVANCED'));
```

```
-----
| Id |Operation                                | Name | Rows | Bytes | Cost (%CPU)| Time     |
-----+-----+-----+-----+-----+-----+-----+
|  0 |SELECT STATEMENT                        |      |      |      |  2  (100)|          |
|  1 |  SORT AGGREGATE                        |      |    1 |    7 |           |          |
|  2 |    TABLE ACCESS BY INDEX ROWID       | T     |    2 |   14 |  2   (0)| 00:00:01 |
|* 3 |      INDEX RANGE SCAN                  | J     |    2 |      |  1   (0)| 00:00:01 |
-----
```

...
Outline Data

```
-----
/*+
  BEGIN_OUTLINE_DATA
  IGNORE_OPTIM_EMBEDDED_HINTS
  OPTIMIZER_FEATURES_ENABLE('10.2.0.1')
  ALL_ROWS
  OUTLINE_LEAF(@"SEL$1")
  INDEX(@"SEL$1" "T"@"SEL$1" ("T"."J"))
  END_OUTLINE_DATA
*/
```

Peeked Binds (identified by position):

```
-----
1 - :A (NUMBER): 0
```

ORACLE

V\$SQL_PLAN Output ('ADVANCED')

query with Bind Variable : Example from v\$sql_plan output

This is SAME example used in plan_table output to compare results : Plans are *NOT* the same.

This plan, using index, is actual plan.

```
drop table T;
CREATE TABLE T(i number, j number, c2 char(100));
declare
  i number;
begin
  for i in 1..10000 loop
    insert into T values(i, '1','HELLO'||i);
  end loop;
```

```

update T set j='0' where i >= 9999;
commit;
end;
/

```

```

create index J on T(j);

```

```

exec dbms_stats.gather_table_stats(NULL,'T',cascade=>TRUE, -
      method_opt =>'for all columns size 5');
alter system flush shared_pool;

```

```

var A number
exec :A := 0;

```

```

SeLeCt avg(T.i)
  from T where T.j= :A;

```

```

      9999.5

```

```

select plan_table_output
  from table(dbms_xplan.display_cursor(null,null,'ADVANCED'));

```

```

SQL_ID 5zumxwut52h7s, child number 0
-----

```

```

Plan hash value: 798443784

```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	

0	SELECT STATEMENT				2 (100)		
1	SORT AGGREGATE		1	7			
2	TABLE ACCESS BY INDEX ROWID	T	2	14	2 (0)	00:00:01	
* 3	INDEX RANGE SCAN	J	2		1 (0)	00:00:01	

```

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

```

- 1 - SEL\$1
- 2 - SEL\$1 / T@SEL\$1
- 3 - SEL\$1 / T@SEL\$1

Outline Data

```
/*+
  BEGIN_OUTLINE_DATA
  IGNORE_OPTIM_EMBEDDED_HINTS
  OPTIMIZER_FEATURES_ENABLE('10.2.0.1')
  ALL_ROWS
  OUTLINE_LEAF(@"SEL$1")
  INDEX(@"SEL$1" "T"@"SEL$1" ("T"."J"))
  END_OUTLINE_DATA
*/
```

Peeked Binds (identified by position):

1 - :A (NUMBER): 0

Predicate Information (identified by operation id):

3 - access("T"."J"=:A)

Column Projection Information (identified by operation id):

1 - (#keys=0) AVG("T"."I")[22]
2 - "T"."I"[NUMBER,22]
3 - "T".ROWID[ROWID,10]



V\$sql_plan Output with *rowsource execution statistics*

- query ran with statistics_level = ALL
- Format Parameters in DBMS_XPLAN
 - ALLSTATS with/without LAST
 - IOSTATS & MEMSTATS with/without LAST
 - RUNSTAT & RUNSTAT_LAST (for V10.1 compatibility)
- Columns Output
 - CBO Estimation Columns
 - Cost / E-Rows / E-Time / E-Bytes
 - OMem / 1Mem
 - Row source execution statistics Columns
 - Starts / A-Rows / A-Time / Buffers
 - Reads / Writes
 - Used-Mem & 0/1/M / Used-Tmp & Max-Tmp

ORACLE

V\$SQL_PLAN Output with rowsource execution statistics

When **statistics_level=ALL** during SQL execution, then you can dump Rowsource execution statistics by using **ALLSTAT**, **IOSTAT**, **MEMSTAT**.

By adding **LAST**, rowsource execution statistics from last execution are displayed.

STATISTICS_LEVEL can be set to **ALL** at **session** or **system** level. It's possible to get these rowsource execution statistics with hint **gather_plan_statistics**.

Interesting comment : <http://jonathanlewis.wordpress.com/2007/04/26/heisenberg/>

Columns Name Description :

- **E-Rows** are CBO Estimated Rows (result from its computations)
- **Cost** is cumulative Cost at that operation ID
- **E-Time** is translation from Cost into time (That's possible with CPU Cost)
- **OMem** and **1Mem** is estimated memory to perform operation in Optimal mode vs 1 pass Mode.

- **Starts** Column can be useful in complex plan. It displays how many times operation ran. This information is not available in 10046 trace.
- **A-Rows** Column is the most important additional information. It shows the actual number of rows returned by this step and is very useful for finding problems with the Optimizer estimated cardinality shown in the **E-Rows** column.
- **A-Time** is cumulative execution time at operation ID. This is same time (tim=) information found in 10046 trace in Rows source Line (STAT# lines).
- **Buffers** is the cumulative number of buffer gets at Operation ID (logical IO)
- **Reads** Cumulative number of buffer which were read on disk
- **Writes** Cumulative number of buffer which were write on disk
- **Used-Mem** available with LAST parameter, it reports memory used by operation. If (0) then operation has been done in memory (OPTIMAL), if (1) the operation has been done in 1-pass, and if (M) then Multi-Pass was necessary.
- **0/1/M** is available without LAST parameter and report Nb of times operation was done in OPTIMAL/1-PASS/Multi-PASS
- **Used-Tmp** available with LAST parameter, it reports size used in Temporary segment because operation was not done in OPTIMAL mode.
- **Max-Tmp** available without LAST parameter, is same as previous column but only the maximum value from all execution is reported.

```
SELECT e.ename, d.dname FROM emp e, dept d WHERE e.deptno =d.deptno and d.deptno= 10;
```

```

ENAME      DNAME
-----
CLARK      ACCOUNTING
KING       ACCOUNTING
MILLER     ACCOUNTING

```

```
select * from table(dbms_xplan.display_cursor (null, null, 'TYPICAL IOSTATS LAST'));
```

```
SQL_ID c41yb2spsmdpn, child number 0
```

```
-----
SELECT e.ename, d.dname FROM emp e, dept d WHERE e.deptno =d.deptno and d.deptno= 10
```

```
Plan hash value: 568005898
```

```

-----
|Id|Operation                                |Name   |Starts|E-Rows|Cost (%CPU)|
-----
| 1|NESTED LOOPS                            |       |     1|     5|      4  (0)|
| 2| TABLE ACCESS BY INDEX ROWID DEPT      |DEPT   |     1|     1|      1  (0)|
|* 3| INDEX UNIQUE SCAN PK_DEPT              |PK_DEPT|     1|     1|      0  (0)|
|* 4| TABLE ACCESS FULL EMP                  |EMP    |     1|     5|      3  (0)|
-----

```

```

-----
|Id|Operation                                |Name   |E-Time|A-Rows|A-Time  |Buffers|
-----
| 1|NESTED LOOPS                            |       |00:01|     3|00:00.01|     10|
| 2| TABLE ACCESS BY INDEX ROWID DEPT      |DEPT   |00:01|     1|00:00.01|      2|
|* 3| INDEX UNIQUE SCAN PK_DEPT              |PK_DEPT|      |     1|00:00.01|      1|
|* 4| TABLE ACCESS FULL EMP                  |EMP    |00:01|     3|00:00.01|      8|
-----

```

Predicate Information (identified by operation id):

```
-----  
3 - access("D"."DEPTNO"=10)  
4 - filter("E"."DEPTNO"=10)
```

Example to see, plan with 'TYPICAL' format output with row source execution statistics if they exist, from each child for sql_id = '&sql_id' which are still in shared_pool :

```
select t.*  
from v$sql s  
  , table(dbms_xplan.display_cursor(s.sql_id, s.child_number, 'ALLSTATS LAST')) t  
where s.sql_id = '&sql_id' ;
```

Another example

with ALLSTATS format parameter to get information on Logical IO (Buffers), Physical IO (reads/Writes) and Memory (estimation and usage)

EMP1 has been built from EMP and many "insert into emp1 select * from emp1"

```
var B1 varchar2(10)  
exec :B1 := 'A%';  
alter session set statistics_level = all;  
  
select /*+ leading (emp1 dept) use_hash(dept) */ empno, ename, dname  
  from emp1, dept  
 where emp1.deptno=dept.deptno  
       and ename like :B1;
```

EMPNO	ENAME	DNAME
7876	ADAMS	RESEARCH
7499	ALLEN	SALES

...
16384 rows selected.

```
select *  
from table(dbms_xplan.display_cursor (null,null, -  
  'TYPICAL +PEEKED_BINDS ALLSTATS LAST'));
```

```
SQL_ID  cfz510bsax5yv, child number 0  
-----  
Plan hash value: 343414133
```

Plan output is so large it is cut in 3 parts to display it

Id	Operation	Name	Starts	E-Rows	Cost (%CPU)	E-Time
* 1	HASH JOIN		1	1	7 (15)	00:00:01
* 2	TABLE ACCESS FULL	EMP1	1	1	3 (0)	00:00:01
3	TABLE ACCESS FULL	DEPT	1	4	3 (0)	00:00:01

Id	Operation	A-Rows	A-Time	Buffers	Reads	Writes
* 1	HASH JOIN	16384	00:00:00.68	715	48	48
* 2	TABLE ACCESS FULL	16384	00:00:00.08	708	0	0
3	TABLE ACCESS FULL	4	00:00:00.01	7	0	0

Id	Operation	Name	OMem	lMem	Used-Mem	Used-Tmp
* 1	HASH JOIN		1159K	1159K	200K (1)	1024
* 2	TABLE ACCESS FULL	EMP1				
3	TABLE ACCESS FULL	DEPT				

Peeked Binds (identified by position):

1 - (VARCHAR2(30), CSID=46): 'A%'

Predicate Information (identified by operation id):

1 - access("EMP1"."DEPTNO"="DEPT"."DEPTNO")
 2 - filter("ENAME" LIKE :B1)

To perform query , :

- EMP1 table is accessed, This operation cost is 3.
- We read 708 blocks (all were in memory reads = writes = 0)
- We see That CBO estimated that only 1 row will be returned by predicate ("ENAME" LIKE :B1) (gathered statistics before to do subsequent fill EMP1 with insert into EMP1 select * from EMP1)
- Actual Rows number of rows returned is 16386.
- When there is a query tuning exercise, this kind of discrepancy helps to identify where statistics are not accurate
- Build HASH table. "Hash Table" from the "build" table (EMP1) is larger than the available memory, Oracle adopts will perform HASH JOIN in 1Pass using temporary space.
- Oracle starts to read the DEPT table, using Full Table SCAN. Cost operation is 3
- Actual Rows = Estimated Rows = 4 (we have accurate statistics)
- We read 7 blocks and all were in memore (reads= writes=0)
- Oracle uses the same hash function on the join column(s) to probe the hash table, Cost is 7

RUNSTATS_LAST

still exist for 10.1 compatibility. There is less information available

```
select empno, ename, dname
  from emp, dept
 where emp.deptno=dept.deptno
    and ename like :B1;
```

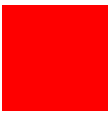
```
select *
  from table(dbms_xplan.display_cursor (null,null,'ALL RUNSTATS_LAST'));
```

Id		Operation	Name	Starts	E-Rows	A-Rows	A-Time Buffers

*	1	HASH JOIN		1	1	2	00:00.01 15
*	2	TABLE ACCESS FULL	EMP1	1	1	2	00:00.01 7
	3	TABLE ACCESS FULL	DEPT	1	4	4	00:00.01 8

Predicate Information (identified by operation id):

```
-----
 1 - access("EMP1"."DEPTNO"="DEPT"."DEPTNO")
 2 - filter("ENAME" LIKE :B1)
```



3 - DBA_HIST_SQL_PLAN DBMS_XPLAN output

DBMS_XPLAN.DISPLAY_AWR

- execution plan of all children associated to the SQL ID 'atfwcg8anrykp':

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY_AWR('atfwcg8anrykp'));
```

- **DBA_HIST_SQLSTAT** provides information about execution statistics (SQL_ID, PLAN_HASH_VALUE)
- execution plan of all stored SQL containing the string 'SUPPORT':

```
SELECT tf.*  
FROM DBA_HIST_SQLTEXT ht, table  
      (DBMS_XPLAN.DISPLAY_AWR(ht.sql_id  
                              ,null,null, 'ALL' )) tf  
WHERE ht.sql_text like '%SUPPORT%';
```



ORACLE

DBA_HIST_SQL_PLAN DBMS_XPLAN Output

On 10g, Plans from v\$sql_plan may be saved by AWR infrastructure.

The AWR is part of the new intelligent infrastructure of Oracle Database 10g that is used by many of the new solutions (such as ADDM for analysis). The AWR automatically collects, processes, and maintains system-performance statistics for problem-detection and self-tuning purposes and stores the statistics persistently in the database.

The statistics collected and processed by the AWR include:

SQL statements that produce the highest load on the system, based on criteria such as elapsed time, CPU time, buffer gets, and so on.

The DBA_HIST views contain historical data stored in the database.

The **DBMS_XPLAN.DISPLAY_AWR** function can be used to look up an historical SQL statement captured in Oracle 10g's Automatic Workload Repository (AWR), and display its execution plan. This gives a seven-day

rolling window of history that you can access.

Syntax

```
DBMS_XPLAN.DISPLAY_AWR(  
    sql_id IN VARCHAR2,  
    plan_hash_value IN NUMBER DEFAULT NULL,  
    db_id IN NUMBER DEFAULT NULL,  
    format IN VARCHAR2 DEFAULT TYPICAL);
```

To use the **DISPLAY_AWR** functionality, the calling user must have SELECT privilege on DBA_HIST_SQL_PLAN, DBA_HIST_SQLTEXT, and V\$DATABASE, otherwise it will show an appropriate error message.

Example to dump ALL plans from sql_id = '&sql_id' with ADVANCED output format (Bind variable values, if any, could be dumped sometimes)

```
SELECT * FROM  
table(DBMS_XPLAN.DISPLAY_AWR('&sql_id',null,null,'ADVANCED'));
```

Primary Key from DBA_HIST_SQL_PLAN is based on following columns :

- DBID
- SQL_ID
- PLAN_HASH_VALUE
- ID

To know, when this plan has been recorded, it's possible to use **TIMESTAMP** column from **DBA_HIST_SQL_PLAN** using SQL_ID, PLAN_HASH_VALUE information available with plan output and ID=0 (assuming we only have 1 value on DBID).



DBA_HIST_SQL_PLAN – AWR awrsqrpt report

- **Report is available using awrsqrpt.sql script**
 - -> choose begin snapshot
 - -> choose end snapshot
 - -> choose hash value
 - -> get report
- On 10gR2, **DBMS_XPLAN.DISPLAY_AWR** can be used

ORACLE®

AWR report



4- Statspack sprepsql report

- **exec statspack.snap(i_snap_level => 6)**

-> then Plan from Top SQLs are recorded in 'stats\$sql_plan'

- **Report is available using sprepsql.sql script**

- -> choose begin snapshot
- -> choose end snapshot
- -> choose hash bvalue
- -> get report

- On 10gR2, **DBMS_XPLAN.DISPLAY** can be used

```
select * from table(dbms_xplan.display('stats$sql_plan', null, 'all', 'hash_value=1054118158 and snap_id=4'));
```

ORACLE

Statspack sprepsql report

On 9i , Statspack with level > =6 records plan from v\$sql_plan into stats\$sql_plan.

<Note 209197.1> Using Statspack to Record Explain Plan Details

Example how to get PLAN from Top SQLs when snapshot level = 7

DB Name	DB Id	Instance	Inst Num	Release	Cluster	Host
Z9207	165158323	z9207	1	9.2.0.7.0	NO	mufry-fr2
	Snap Id	Snap Time	Sessions	Curs/Sess	Comment	
Begin Snap:	3	12-Jul-06 19:18:44	8	5.3		
End Snap:	4	12-Jul-06 19:19:46	8	2.6		
Elapsed:		1.03 (mins)				

...

SQL ordered by Gets for DB: Z9207 Instance: z9207 Snaps: 3 -4

CPU Elapsed

Buffer Gets	Executions	Gets per Exec	%Total	Time (s)	Time (s)	Hash Value
-------------	------------	---------------	--------	----------	----------	------------

9,738	1	9,738.0	55.6	2.81	5.57	1054118158
-------	---	---------	------	------	------	------------

Module: SQL*Plus

SeLeCt /*+ ORDERED */ max(T1.NAME_HSTID) from T1, T2, T3 where
T1.NAME_HSTID=T2.NAME_HSTID and T1.NAME_HSTID = T3.NAME_HSTID

....

SQL> @?/rdbms/admin/sprepsql

DB Id	DB Name	Inst Num	Instance
-------	---------	----------	----------

165158323	Z9207	1	z9207
-----------	-------	---	-------

Completed Snapshots

Instance	DB Name	Snap Id	Snap Started	Snap Level	Comment
z9207	Z9207	1	12 Jul 2006 19:16	5	
		2	12 Jul 2006 19:17	7	
		3	12 Jul 2006 19:18	7	
		4	12 Jul 2006 19:19	7	

Specify the Begin and End Snapshot Ids

~~~~~

Enter value for begin\_snap: 3

Begin Snapshot Id specified: 3

Enter value for end\_snap: 4

End Snapshot Id specified: 4

Specify the Hash Value

~~~~~

Enter value for hash_value: 1054118158

Hash Value specified is: 1054118158

Specify the Report Name

~~~~~

The default report file name is sp\_3\_4\_1054118158. To use this name, press <return> to continue, otherwise enter an alternative.

Enter value for report\_name:

Using the report name sp\_3\_4\_1054118158

STATSPACK SQL report for Hash Value: 1054118158 Module: SQL\*Plus

| DB Name | DB Id     | Instance | Inst Num | Release   | Cluster | Host      |
|---------|-----------|----------|----------|-----------|---------|-----------|
| Z9207   | 165158323 | z9207    | 1        | 9.2.0.8.0 | NO      | mufry-fr2 |

| Start Id | Start Time         | End Id | End Time           | Duration(mins) |
|----------|--------------------|--------|--------------------|----------------|
| 3        | 12-Jul-06 19:18:44 | 4      | 12-Jul-06 19:19:46 | 1.03           |

SQL Statistics

~~~~~

-> CPU and Elapsed Time are in seconds (s) for Statement Total and in milliseconds (ms)

for Per Execute

	Statement Total	Per Execute	% Snap Total
	-----	-----	-----
Buffer Gets:	9,738	9,738.0	55.58
Disk Reads:	9,672	9,672.0	100.00
Rows processed:	1	1.0	
CPU Time(s/ms):	3	2,814.0	
Elapsed Time(s/ms):	6	5,565.9	
Sorts:	0	.0	
Parse Calls:	1	1.0	
Invalidations:	0		
Version count:	1		
Sharable Mem(K):	12		
Executions:	1		

SQL Text

~~~~~

SeLeCt /\*+ ORDERED \*/ max(T1.NAME\_HSTID) from T1, T2, T3 where  
T1.NAME\_HSTID=T2.NAME\_HSTID and T1.NAME\_HSTID=T3.NAME\_HSTID

Known Optimizer Plan(s) for this Hash Value

~~~~~

Shows all known Optimizer Plans for this database instance, and the Snap Id's they were first found in the shared pool. A Plan Hash Value will appear multiple times if the cost has changed

-> ordered by Snap Id

First Snap Id	First Snap Time	Plan Hash Value	Cost
-----	-----	-----	-----
4 12 Jul 06 19:19		1522189954	1590

Plans in shared pool between Begin and End Snap Ids

~~~~~

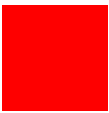
Shows the Execution Plans found in the shared pool between the begin and end snapshots specified. The values for Rows, Bytes and Cost shown below are those which existed at the time the first-ever snapshot captured this plan - these values often change over time, and so may not be indicative of current values

-> Rows indicates Cardinality, PHV is Plan Hash Value

-> ordered by Plan Hash Value

| Operation         | PHV/Object Name             | Rows  | Bytes | Cost  |
|-------------------|-----------------------------|-------|-------|-------|
| -----             | -----                       | ----- | ----- | ----- |
| SELECT STATEMENT  | ---- <b>1522189954</b> ---- |       |       | 1590  |
| SORT AGGREGATE    |                             | 1     | 39    |       |
| HASH JOIN         |                             | 49K   | 1M    | 1590  |
| HASH JOIN         |                             | 49K   | 1M    | 1035  |
| TABLE ACCESS FULL | T1                          | 60K   | 767K  | 496   |
| TABLE ACCESS FULL | T2                          | 49K   | 634K  | 496   |
| TABLE ACCESS FULL | T3                          | 78K   | 1001K | 496   |

End of Report



## 5 - Event 10046 trace file & tkprof

- Row source operations (ACTUAL) in trc file
- Execution plan (Theoretical) can be generated with Tkprof using  
EXPLAIN = U/P
- Segment statistics are available since 9.2
  - These are cumulative statistics
  - They help to read complex execution plan
- There is no predicate information in trace file

ORACLE

### Note References

**<Note.376442.1>** Recommended Method for Obtaining 10046 trace for Tuning

**<Note.39817.1>** Interpreting Raw SQL\_TRACE and DBMS\_SUPPORT.START\_TRACE output

**<Note.295656.1 >** Row Source Operation Statistics in SQL\_TRACE Files

**<Note.312368.1>** Why plan is missing in TKPROF output

- [http://download-uk.oracle.com/docs/cd/B19306\\_01/server.102/b14211/sqltrace.htm#sthref2058](http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14211/sqltrace.htm#sthref2058)

Sometimes, same SQL which ran several times can have a change of plan along these executions. This can occur when :

"Misses in library cache during parse: #NB#"

+ "Misses in library cache during execute: #NB# " is > 1

In that case, the safest way to have plan for a cursor is to avoid aggregation from tkprof.

- If aggregation = YES only 1 plan will be displayed
- If aggregation = NO, you'll get 1 plan by "row source information" available for that cursor

See below example to demonstrate that "explain=" used with tkprof can be misleading

This is same example as PLAN\_TABLE and v\$sql\_plan

```
drop table T;

CREATE TABLE T(i number, j number, c2 char(100));
declare
  i number;
begin
  for i in 1..10000 loop
    insert into T values(i, '1','HELLO' || i);
  end loop;
update T set j='0' where i >= 9999;
commit;
end;
/
create index J on T(j);
exec dbms_stats.gather_table_stats(NULL,'T',cascade=>TRUE, -
                                method_opt =>'for all columns size 5');

alter system flush shared_pool;

var A number
exec :A := 0;

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

SeLeCt avg(T.i) from T where T.j= :A;
9999.5
```

→ trace file z9207\_oracle\_6000.trc generated in user\_dump\_dest

- generate tkprof output with below syntax
- format execution trace
- Do not output recursive SYS SQL
- Provide explain plan from SQL

```
tkprof z9207_oracle_6000.trc tk_oracle_6000.trc sys=NO explain=tc/tc
```

- Review plans which are different
- "Row Source Operation" Actual plan
- "Execution Plan" generated by "explain=tc/tc"

- Look at tk\_z9207\_ora\_6000.trc
- SQL is CURSOR#1
- At the end of trace, row source statistics are dumped in STAT#1 lines

## tk\_ora\_6000.trc (extract)

\*\*\*\*\*

SeLeCt avg(T.i) from T where T.j= :A

| call    | count | cpu  | elapsed | disk | query | current | rows |
|---------|-------|------|---------|------|-------|---------|------|
| Parse   | 1     | 0.07 | 0.13    | 0    | 235   | 0       | 0    |
| Execute | 1     | 0.00 | 0.00    | 0    | 0     | 0       | 0    |
| Fetch   | 2     | 0.00 | 0.00    | 0    | 3     | 0       | 1    |
| total   | 4     | 0.07 | 0.13    | 0    | 238   | 0       | 1    |

Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: 153 (TC)

### Rows Row Source Operation

```

-----
1  SORT AGGREGATE
2  TABLE ACCESS BY INDEX ROWID T
2  INDEX RANGE SCAN J (object id 62829)

```

### Rows Execution Plan

```

-----
0  SELECT STATEMENT GOAL: CHOOSE
1  SORT (AGGREGATE)
2  TABLE ACCESS GOAL: ANALYZED (FULL) OF 'T'

```

Elapsed times include waiting on following events:

| Event waited on             | Times<br>Waited | Max. Wait | Total Waited |
|-----------------------------|-----------------|-----------|--------------|
| SQL*Net message to client   | 2               | 0.00      | 0.00         |
| SQL*Net message from client | 2               | 14.85     | 14.85        |

\*\*\*\*\*

z9207\_ora\_6000.trc

### Dump file d:\oracle\admin\z9207\udump\z9207\_ora\_6000.trc

Wed Sep 05 13:46:50 2007

ORACLE V9.2.0.8.0 - Production vsnsta=0

vsnsql=12 vsnxtr=3

Windows 2000 Version 5.1 Service Pack 2, CPU type 586

Oracle9i Enterprise Edition Release 9.2.0.8.0 - Production

With the Partitioning, OLAP and Oracle Data Mining options

JServer Release 9.2.0.8.0 - Production

Windows 2000 Version 5.1 Service Pack 2, CPU type 586

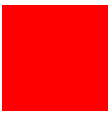
Instance name: z9207

Redo thread mounted by this instance: 1

```

Oracle process number: 8
Windows thread id: 6000, image: ORACLE.EXE
*** 2007-09-05 13:46:50.031
*** SESSION ID:(7.107) 2007-09-05 13:46:50.021
APPNAME mod='SQL*Plus' mh=3669949024 act='' ah=4029777240
=====
PARSING IN CURSOR #1 len=69 dep=0 uid=153 oct=42 lid=153 tim=14560716443 hv=2004533713
ad='69f24eb0'
alter session set events '10046 trace name context forever, level 12'
END OF STMT
...
=====
PARSING IN CURSOR #1 len=36 dep=0 uid=153 oct=3 lid=153 tim=14575565906 hv=2851360509
ad='69b45604'
SeLeCt avg(T.i) from T where T.j= :A
END OF STMT
PARSE #1:c=170245,e=227241,p=0,cr=235,cu=0,mis=1,r=0,dep=0,og=0,tim=14575565902
... recursive SYS sqls
BINDS #1:
  bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=03 oacfl2=0 size=24 offset=0
    bfp=033a45a4 bln=22 avl=01 flg=05
    value=0
EXEC #1:c=0,e=52,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=14575569756
WAIT #1: nam='SQL*Net message to client' ela= 4 p1=1111838976 p2=1 p3=0
FETCH #1:c=0,e=32,p=0,cr=3,cu=0,mis=0,r=1,dep=0,og=4,tim=14575569828
WAIT #1: nam='SQL*Net message from client' ela= 190 p1=1111838976 p2=1 p3=0
FETCH #1:c=0,e=2,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=14575570081
WAIT #1: nam='SQL*Net message to client' ela= 2 p1=1111838976 p2=1 p3=0
*** 2007-09-05 13:47:19.743
WAIT #1: nam='SQL*Net message from client' ela= 14854241 p1=1111838976 p2=1 p3=0
STAT #1 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT AGGREGATE '
STAT #1 id=2 cnt=2 pid=1 pos=1 obj=62828 op='TABLE ACCESS BY INDEX ROWID T '
STAT #1 id=3 cnt=2 pid=2 pos=1 obj=62829 op='INDEX RANGE SCAN J '

```



## 6 - Event 10132 trace file

- Dump plan at the end of compilation
- Plan dumped when HARD parse occurs
- Not available on 8i
- Useful to combine it to 10053 (on 9i)
  - Help to recognize chosen PLAN
- Useful to combine to 10046
  - Provide Cost information
  - predicate information section on 10gR2

ORACLE

### ***Event 10132 trace file***

Examples from 10132 trace over versions (9.2.0.7 – 10.1.0.4 – 10.2.0.1)

Script used to generate trace file

```
alter system flush shared_pool;
drop table T;
CREATE TABLE T(i number, j number, c2 char(100));
declare
  i number;
begin
  for i in 1..10000 loop
    insert into T values(i, '1','HELLO'||i);
  end loop;
update T set j=0 where i >= 9999;
commit;
end;
/

create index J on T(j);
exec dbms_stats.gather_table_stats(NULL,'T',cascade=>TRUE, -
```

```

method_opt =>'for all columns size 5');

var A number
exec :A := 0

alter session set events '10132 trace name context forever, level 1';

select avg(i) from T where j= :A;

alter session set events '10132 trace name context off';

```

## Example 1 : 9.2.0.7

```

Dump file d:\oracle\admin\z9207\udump\z9207_ora_5732.trc
Fri Jul 21 11:42:22 2006
ORACLE V9.2.0.7.0 - Production vsnsta=0
vsnsql=12 vsnxtr=3
Windows 2000 Version 5.1 Service Pack 2, CPU type 586
Oracle9i Enterprise Edition Release 9.2.0.7.0 - Production

```

```

*** 2006-07-21 11:42:22.139
*** SESSION ID:(7.75) 2006-07-21 11:42:22.119
Current SQL statement for this session:
select avg(i) from T where j=:A

```

Plan Table

| -----                       |      |      |       |      |  |
|-----------------------------|------|------|-------|------|--|
| Operation                   | Name | Rows | Bytes | Cost |  |
| -----                       |      |      |       |      |  |
| SELECT STATEMENT            |      | 0    | 0     | 2    |  |
| SORT AGGREGATE              |      | 1    | 7     | 0    |  |
| TABLE ACCESS BY INDEX ROWID | T    | 2    | 14    | 2    |  |
| INDEX RANGE SCAN            | J    | 2    | 0     | 1    |  |
| -----                       |      |      |       |      |  |

## Example 2 : 10.1.0.4

```

Dump file d:\oracle\product\10.1.0\admin\z101\udump\z101_ora_3048.trc
Fri Jul 21 12:15:54 2006
ORACLE V10.1.0.4.0 - Production vsnsta=0
vsnsql=13 vsnxtr=3
Oracle Database 10g Enterprise Edition Release 10.1.0.4.0 - Production

```

```

*** 2006-07-21 12:15:54.576
*** ACTION NAME:() 2006-07-21 12:15:54.476
*** MODULE NAME:(SQL*Plus) 2006-07-21 12:15:54.476
*** SERVICE NAME:(Z101.fr.oracle.com) 2006-07-21 12:15:54.476
*** SESSION ID:(151.5298) 2006-07-21 12:15:54.476

```

```

Current SQL statement for this session:
select avg(i) from T where j= :A

```

Plan Table

```
-----
```

| Operation                   | Name | Rows | Bytes | Cost | Time     |
|-----------------------------|------|------|-------|------|----------|
| SELECT STATEMENT            |      |      |       | 2    |          |
| SORT AGGREGATE              |      | 1    | 7     |      |          |
| TABLE ACCESS BY INDEX ROWID | T    | 2    | 14    | 2    | 00:00:01 |
| INDEX RANGE SCAN            | J    | 2    |       | 1    | 00:00:01 |

```
-----
```

## Example 2 : 10.2.0.1

```
Dump file d:\oracle\product\10.2.0\admin\z102\udump\z102_ora_2360.trc
Fri Jul 21 12:32:34 2006
ORACLE V10.2.0.1.0 - Production vsnsta=0
vsnsql=14 vsnxtr=3
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
```

```
*** 2006-07-21 12:32:34.864
*** ACTION NAME:() 2006-07-21 12:32:34.794
*** MODULE NAME:(SQL*Plus) 2006-07-21 12:32:34.794
*** SERVICE NAME:(Z102) 2006-07-21 12:32:34.794
*** SESSION ID:(111.59780) 2006-07-21 12:32:34.794
```

```
sql_id=cd8pcvmtr16st.
Current SQL statement for this session:
select avg(i) from T where j= :A
```

```
=====
Plan Table
=====
```

| Id | Operation                   | Name | Rows | Bytes | Cost | Time     |
|----|-----------------------------|------|------|-------|------|----------|
| 0  | SELECT STATEMENT            |      |      |       | 2    |          |
| 1  | SORT AGGREGATE              |      | 1    | 7     |      |          |
| 2  | TABLE ACCESS BY INDEX ROWID | T    | 2    | 14    | 2    | 00:00:01 |
| 3  | INDEX RANGE SCAN            | J    | 2    |       | 1    | 00:00:01 |

```
Predicate Information:
```

```
-----
3 - access("J"=:A)
```

```
Content of other_xml column
=====
db_version      : 10.2.0.1
parse_schema    : TC
plan_hash       : 798443784
```

```
Peeked Binds
```

```
=====
Bind variable information
position=1
datatype(code)=2
datatype(string)=NUMBER
precision=0
scale=0
max length=22
```

```

value=0
Outline Data:
/*+
  BEGIN_OUTLINE_DATA
    IGNORE_OPTIM_EMBEDDED_HINTS
    OPTIMIZER_FEATURES_ENABLE('10.2.0.1')
    ALL_ROWS
    OUTLINE_LEAF(@"SEL$1")
    INDEX(@"SEL$1" "T"@"SEL$1" ("T"."J"))
  END_OUTLINE_DATA
*/

```

#### Optimizer environment:

```

optimizer_mode_hinted           = false
optimizer_features_hinted       = 0.0.0
parallel_execution_enabled      = true
parallel_query_forced_dop       = 0
parallel_dml_forced_dop         = 0
parallel_ddl_forced_degree      = 0
parallel_ddl_forced_instances   = 0
_query_rewrite_fudge            = 90
optimizer_features_enable       = 10.2.0.1
_optimizer_search_limit         = 5
cpu_count                       = 1
active_instance_count           = 1
parallel_threads_per_cpu        = 2
hash_area_size                  = 131072
bitmap_merge_area_size          = 1048576
sort_area_size                  = 65536
sort_area_retained_size         = 0
_sort_elimination_cost_ratio    = 0
_optimizer_block_size           = 8192
_sort_multiblock_read_count     = 2
_hash_multiblock_io_count       = 0
_db_file_optimizer_read_count   = 8
_optimizer_max_permutations     = 2000
pga_aggregate_target            = 1048576 KB
_pga_max_size                   = 209700 KB
_query_rewrite_maxdisjunct      = 257
_smm_auto_min_io_size           = 56 KB
_smm_auto_max_io_size           = 248 KB
_smm_min_size                   = 1024 KB
_smm_max_size                   = 104850 KB
_smm_px_max_size                = 524288 KB
_cpu_to_io                      = 0
_optimizer_undo_cost_change     = 10.2.0.1
parallel_query_mode             = enabled

```

```

parallel_dml_mode               = disabled
parallel_ddl_mode               = enabled
optimizer_mode                  = all_rows
sqlstat_enabled                 = false
_optimizer_percent_parallel     = 101
_always_anti_join               = choose
_always_semi_join               = choose
_optimizer_mode_force           = true
_partition_view_enabled         = true
_always_star_transformation     = false
_query_rewrite_or_error         = false
_hash_join_enabled              = true
cursor_sharing                  = exact
_b_tree_bitmap_plans            = true
star_transformation_enabled     = false
_optimizer_cost_model           = choose
_new_sort_cost_estimate         = true
_complex_view_merging           = true
_unnest_subquery                = true
_eliminate_common_subexpr       = true
_pred_move_around               = true
_convert_set_to_join            = false
_push_join_predicate            = true
_push_join_union_view           = true
_fast_full_scan_enabled         = true
_optim_enhance_nnull_detection  = true
_parallel_broadcast_enabled     = true
_px_broadcast_fudge_factor      = 100
_ordered_nested_loop            = true
_no_or_expansion                = false
optimizer_index_cost_adj        = 100
optimizer_index_caching         = 0
_system_index_caching           = 0
_disable_datalayer_sampling     = false
query_rewrite_enabled           = true

```

|                                      |                |                                         |            |
|--------------------------------------|----------------|-----------------------------------------|------------|
| query_rewrite_integrity              | = enforced     | _optimizer_push_down_distinct           | = 0        |
| _query_cost_rewrite                  | = true         | _dml_monitoring_enabled                 | = true     |
| _query_rewrite_2                     | = true         | _optimizer_undo_changes                 | = false    |
| _query_rewrite_1                     | = true         | _predicate_elimination_enabled          | = true     |
| _query_rewrite_expression            | = true         | _nested_loop_fudge                      | = 100      |
| _query_rewrite_jgmigrate             | = true         | _project_view_columns                   | = true     |
| _query_rewrite_fpc                   | = true         | _local_communication_costing_enabled    | = true     |
| _query_rewrite_drj                   | = true         | _local_communication_ratio              | = 50       |
| _full_pwise_join_enabled             | = true         | _query_rewrite_vop_cleanup              | = true     |
| _partial_pwise_join_enabled          | = true         | _slave_mapping_enabled                  | = true     |
| _left_nested_loops_random            | = true         | _optimizer_cost_based_transformation    | = linear   |
| _improved_row_length_enabled         | = true         | _optimizer_mjc_enabled                  | = true     |
| _index_join_enabled                  | = true         | _right_outer_hash_enable                | = true     |
| _enable_type_dep_selectivity         | = true         | _spr_push_pred_refspr                   | = true     |
| _improved_outerjoin_card             | = true         | _optimizer_cache_stats                  | = false    |
| _optimizer_adjust_for_nulls          | = true         | _optimizer_cbqt_factor                  | = 50       |
| _optimizer_degree                    | = 0            | _optimizer_squ_bottomup                 | = true     |
| _use_column_stats_for_function       | = true         | _fic_area_size                          | = 131072   |
| _subquery_pruning_enabled            | = true         | _optimizer_skip_scan_enabled            | = true     |
| _subquery_pruning_mv_enabled         | = false        | _optimizer_cost_filter_pred             | = false    |
| _or_expand_nvl_predicate             | = true         | _optimizer_sortmerge_join_enabled       | = true     |
| _like_with_bind_as_equality          | = false        | _optimizer_join_sel_sanity_check        | = true     |
| _table_scan_cost_plus_one            | = true         | _mmv_query_rewrite_enabled              | = true     |
| _cost_equality_semi_join             | = true         | _bt_mmv_query_rewrite_enabled           | = true     |
| _default_non_equality_sel_check      | = true         | _add_stale_mv_to_dependency_list        | = true     |
| _new_initial_join_orders             | = true         | _distinct_view_unnesting                | = false    |
| _oneside_colstat_for_equijoins       | = true         | _optimizer_dim_subq_join_sel            | = true     |
| _optim_peek_user_binds               | = true         | _optimizer_disable_strans_sanity_checks | = 0        |
| _minimal_stats_aggregation           | = true         | _optimizer_compute_index_stats          | = true     |
| _force temptables_for_gsets          | = false        | _push_join_union_view2                  | = true     |
| workarea_size_policy                 | = auto         | _optimizer_ignore_hints                 | = false    |
| _smm_auto_cost_enabled               | = true         | _optimizer_random_plan                  | = 0        |
| _gs_anti_semi_join_allowed           | = true         | _query_rewrite_setopgrw_enable          | = true     |
| _optim_new_default_join_sel          | = true         | _optimizer_correct_sq_selectivity       | = true     |
| optimizer_dynamic_sampling           | = 2            | _disable_function_based_index           | = false    |
| _pre_rewrite_push_pred               | = true         | _optimizer_join_order_control           | = 3        |
| _optimizer_new_join_card_computation | = true         | _optimizer_cartesian_enabled            | = true     |
| _union_rewrite_for_gs                | = yes_gset_mvs | _optimizer_starplan_enabled             | = true     |
| _generalized_pruning_enabled         | = true         | _extended_pruning_enabled               | = true     |
| _optim_adjust_for_part_skews         | = true         | _optimizer_push_pred_cost_based         | = true     |
| _force_datefold_trunc                | = false        | _sql_model_unfold_forloops              | = run_time |
| statistics_level                     | = typical      | _enable_dml_lock_escalation             | = false    |
| _optimizer_system_stats_usage        | = true         | _bloom_filter_enabled                   | = true     |
| skip_unusable_indexes                | = true         | _update_bji_ipdml_enabled               | = 0        |
| _remove_aggr_subquery                | = true         | _optimizer_extended_cursor_sharing      | = udo      |

```

_dm_max_shared_pool_pct      = 1
_optimizer_cost_hjsmj_multimatch = true
_optimizer_transitivity_retain = true
_px_pwg_enabled              = true
optimizer_secure_view_merging = true
_optimizer_join_elimination_enabled = true
flashback_table_rpi          = non_fbt
_optimizer_cbqt_no_size_restriction = true
_optimizer_enhanced_filter_push = true
_optimizer_filter_pred_pullup  = true
_rowsrc_trace_level           = 0
_simple_view_merging           = true
_optimizer_rownum_pred_based_fkr = true

```

```

_optimizer_better_inlist_costing = all
_optimizer_self_induced_cache_cost = false
_optimizer_min_cache_blocks      = 10
_optimizer_or_expansion           = depth
_optimizer_order_by_elimination_enabled = true
_optimizer_outer_to_anti_enabled = true
_selfjoin_mv_duplicates          = true
_dimension_skip_null             = true
_force_rewrite_enable            = false
_optimizer_star_tran_in_with_clause = true
_optimizer_complex_pred_selectivity = true
_gby_hash_aggregation_enabled    = true

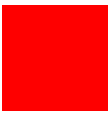
```

### Query Block Registry:

\*\*\*\*\*

SEL\$1 0x792fe60 (PARSER) [FINAL]

Optimizer State Dump: call(in-use=18884, alloc=0), compile(in-use=52584, alloc=0)



## 7 - 10053 trace file

- 10053 is used to report CBO computations to generate plan
- To have the plan helps to read that trace file
- Since 10gR2, plan is dumped at the end
- Before 10gR2, event 10132 can be used to get it (we have plan dump at exactly same time as CBO computations)

ORACLE

### ***Event 10053 trace file***

Example to get plan from 10053 trace. Script used to generate trace file

```
alter system flush shared_pool;

drop table T;
CREATE TABLE T(i number, j number, c2 char(100));

declare
  i number;
begin
  for i in 1..10000 loop
    insert into T values(i, '1','HELLO'||i);
  end loop;
  update T set j=0 where i >= 9999;
  commit;
end;
/
```

```

create index J on T(j);
exec dbms_stats.gather_table_stats(NULL,'T',cascade=>TRUE,-
                                method_opt =>'for all columns size 5');

var A number
exec :A := 0

alter session set events '10053 trace name context forever, level 1';

/* Begin of Only on 9.2.0.8 */
alter session set events '10132 trace name context forever, level 1';
/* End of Only on 9.2.0.8 */

select avg(i) from T where j= :A;
exit

```

## Example 1 : 9.2.0.8

```

Dump file d:\oracle\admin\z9207\udump\z9207_ora_1360.trc
Mon Sep 10 16:33:47 2007
ORACLE V9.2.0.8.0 - Production vsnsta=0
vsnsql=12 vsnxtr=3
Windows 2000 Version 5.1 Service Pack 2, CPU type 586
Oracle9i Enterprise Edition Release 9.2.0.8.0 - Production
...
GENERAL PLANS
*****
Join order[1]:  T[T]#0
Best so far: TABLE#: 0  CST:          2  CDN:          2  BYTES:          14
Final - All Rows Plan:
  JOIN ORDER: 1
    CST: 2  CDN: 2  RSC: 2  RSP: 2  BYTES: 14
    IO-RSC: 2  IO-RSP: 2  CPU-RSC: 0  CPU-RSP: 0
-- End of Plan information from 10053
Current SQL statement for this session: (This is from 10132 event)
select avg(i) from T where j= :A
Plan Table
-----

```

| Operation                   | Name | Rows | Bytes | Cost |
|-----------------------------|------|------|-------|------|
| SELECT STATEMENT            |      | 0    | 0     | 2    |
| SORT AGGREGATE              |      | 1    | 7     | 0    |
| TABLE ACCESS BY INDEX ROWID | T    | 2    | 14    | 2    |
| INDEX RANGE SCAN            | J    | 2    | 0     | 1    |

## Example 2 : 10.2.0.1

```

Dump file d:\oracle\product\10.2.0\admin\z102\udump\z102_ora_1516.trc
Fri Jul 21 19:04:07 2006
ORACLE V10.2.0.1.0 - Production vsnsta=0
vsnsql=14 vsnxtr=3
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production

```

```

...
GENERAL PLANS
*****
Considering cardinality-based initial join order.
*****
Join order[1]:  T[T]#0
*****
Best so far: Table#: 0  cost: 2.0027  card: 2.0000  bytes: 14
(newjo-stop-1) k:0, spcnt:0, perm:1, maxperm:80000
*****
Number of join permutations tried: 1
*****
Final - All Rows Plan:  Best join order: 1
  Cost: 2.0027  Degree: 1  Card: 2.0000  Bytes: 14
  Resc: 2.0027  Resc_io: 2.0000  Resc_cpu: 15833
  Resp: 2.0027  Resp_io: 2.0000  Resc_cpu: 15833
kkoipt: Query block SEL$1 (#0)
***** UNPARSED QUERY IS *****
SELECT AVG("T"."I") "AVG(T.I)" FROM "TC"."T" "T" WHERE "T"."J"=:B1
kkoqbc-end

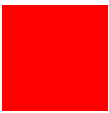
          : call(in-use=16920, alloc=0), compile(in-use=33644, alloc=0)
apadrv-end: call(in-use=16920, alloc=0), compile(in-use=34204, alloc=0)

sql_id=g2ru6d5fj9skr.
Current SQL statement for this session:
select avg(T.i) from T where T.j= :A

=====
Plan Table
=====
-----+-----+-----+-----+-----+-----+
| Id | Operation                                | Name | Rows | Bytes | Cost | Time |
-----+-----+-----+-----+-----+-----+
| 0  | SELECT STATEMENT                        |      |      |      | 2    |      |
| 1  |   SORT AGGREGATE                        |      | 1    | 7    |      |      |
| 2  |    TABLE ACCESS BY INDEX ROWID        | T    | 2    | 14   | 2    | 00:00:01 |
| 3  |     INDEX RANGE SCAN                    | J    | 2    |      | 1    | 00:00:01 |
-----+-----+-----+-----+-----+-----+

Predicate Information:
-----
3 - access("T"."J"=:A)

```



## 8 - Process state Trace file

- This is new in 10gR2
- During its execution, plan is included in process state dump

ORACLE®

### ***Process State Trace File***

#### **Example from process state dump in 10.2.0.1**

Dump file d:\oracle\product\10.2.0\admin\z102\udump\z102\_ora\_2236.trc

```
Fri Jul 21 13:55:08 2006
ORACLE V10.2.0.1.0 - Production vsnsta=0
vsnsql=14 vsnxtr=3
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
...
*** ACTION NAME:() 2006-07-21 13:55:08.597
*** MODULE NAME:(SQL*Plus) 2006-07-21 13:55:08.597
*** SERVICE NAME:(Z102) 2006-07-21 13:55:08.597
*** SESSION ID:(115.45183) 2006-07-21 13:55:08.597
*** 2006-07-21 13:55:08.597
ksedmp: internal or fatal error
Current SQL statement for this session:
select avg(T1.i) from T T1, T T2, T T3 where T3.j= :A
----- Call Stack Trace -----
calling          call      entry          argument values in hex
```

| location   | type    | point      | (? means dubious value) |
|------------|---------|------------|-------------------------|
| -----      | -----   | -----      | -----                   |
| _ksedst+38 | CALLrel | _ksedst1+0 | 0 1                     |

...

=====

Plan Table

=====

| Id | Operation            | Name | Rows | Bytes | Cost | Time     |
|----|----------------------|------|------|-------|------|----------|
| 0  | SELECT STATEMENT     |      |      |       | 962K |          |
| 1  | SORT AGGREGATE       |      | 1    | 7     |      |          |
| 2  | MERGE JOIN CARTESIAN |      | 191M | 1335M | 962K | 03:17:56 |
| 3  | MERGE JOIN CARTESIAN |      | 20K  | 137K  | 101  | 00:00:02 |
| 4  | INDEX RANGE SCAN     | J    | 2    | 6     | 1    | 00:00:01 |
| 5  | BUFFER SORT          |      | 10K  | 39K   | 100  | 00:00:02 |
| 6  | TABLE ACCESS FULL    | T    | 10K  | 39K   | 50   | 00:00:01 |
| 7  | BUFFER SORT          |      | 10K  |       | 961K | 03:17:55 |
| 8  | TABLE ACCESS FULL    | T    | 10K  |       | 49   | 00:00:01 |

Content of other\_xml column

=====

```

db_version      : 10.2.0.1
parse_schema    : TC
plan_hash       : 2261269514

```

Peeked Binds

=====

```

Bind variable information
  position=1
  datatype(code)=2
  datatype(string)=NUMBER
  precision=0
  scale=0
  max length=22
  value=0

```

Outline Data:

/\*+

```

  BEGIN_OUTLINE_DATA

```

...



# Other views with info about SQL execution

- v\$sqlarea
- v\$sqlarea\_plan\_hash
- v\$sql
- v\$sqlstats
- v\$sql\_plan\_statistics
- v\$sql\_plan\_statistics\_all
- DBA\_HIST\_SQLSTAT
- DBA\_HIST\_SQLTEXT
- DBA\_HIST\_SQL\_SUMMARY
- DBA\_HIST\_SQL\_PLAN

ORACLE

## ***Other Views***

Another useful view is V\$SQL\_PLAN\_STATISTICS, which provides the execution statistics of each operation in the execution plan for each cached cursor. Also, the V\$SQL\_PLAN\_STATISTICS\_ALL view concatenates information from V\$SQL\_PLAN with execution statistics from V\$SQL\_PLAN\_STATISTICS and V\$SQL\_WORKAREA. DBMS\_XPLA.DISPLAY\_CURSOR uses this view when ALLSTATS parameter is used.

Oracle 10G R2 provides a new view called V\$SQLSTATS that contains a combination of columns that appear in V\$SQL and V\$SQLAREA. The benefits that V\$SQLSTATS provides are as follows:

Since V\$SQLSTATS contains the entire text of the SQL statement AND its associated performance statistics, we are no longer required to access both the V\$SQLTEXT and V\$SQLAREA to obtain the information we need.

Oracle states that V\$SQLSTATS is faster and more scalable.

The data in V\$SQLAREA has a tendency to get its contents flushed out just when you need to get additional information from it. The V\$SQLSTATS view provides users with a longer access window.

|                            |  |                           |
|----------------------------|--|---------------------------|
| V\$SQL                     |  | V\$SQL_SHARED_CURSOR      |
| V\$SQL_BIND_CAPTURE        |  | V\$SQL_WORKAREA           |
| V\$SQL_BIND_DATA           |  | V\$SQL_WORKAREA_ACTIVE    |
| V\$SQL_BIND_METADATA       |  | V\$SQL_WORKAREA_HISTOGRAM |
| V\$SQL_CURSOR              |  | V\$SQLAREA                |
| V\$SQL_OPTIMIZER_ENV       |  | V\$SQLAREA_PLAN_HASH      |
| V\$SQL_PLAN                |  | V\$SQLSTATS               |
| V\$SQL_PLAN_STATISTICS     |  | V\$SQLTEXT                |
| V\$SQL_PLAN_STATISTICS_ALL |  | V\$SQLTEXT_WITH_NEWLINES  |

## Example :

Set of SQLs useful to look for information when customer gets a plan change and did nothing :

More information when STATISTICS\_LEVEL = ALL (Plan statistics by example)

1st set of SQL is to gather information still in shared\_pool

- query on v\$sql\_plan
- query on v\$sqlstats
- Often DB restarted and info is lost

```

spool SUP01.lst
set echo on
set lines 300 pages 300

select t.*
  from v$sql s
       , table(dbms_xplan.display_cursor(
                s.sql_id, s.child_number, 'ADVANCED ALLSTATS')) t
  where s.sql_id = '&sql_id' ;

SELECT * from v$sqlstats  where sql_id= '&sql_id';

SELECT *
  from table(DBMS_XPLAN.DISPLAY_AWR( '&sql_id', null, null, 'ADVANCED ALLSTATS'));

SELECT sql_id, PLAN_HASH_VALUE , timestamp
  from dba_hist_sql_plan where sql_id='&sql_id' and id=0;

```

2nd set of SQLs is to find same kind of information in AWR history tables

- Query on DBA\_HIST\_SQL\_PLAN (plan + date at which we stored that plan)
- Query on DBA\_HIST\_SQLSTAT to have an idea which plan is good/bad
- Query on DBA\_HIST\_SQLBIND to have an idea how different can be binds

```

col sql_text format A60
set lines 2000

```

```

col value_string format A50
col value_anydata format A50

SELECT * from DBA_HIST_SNAPSHOT;

SELECT * from dba_hist_sqlstat where sql_id = '&sql_id' order by snap_id;

SELECT *
  from dba_hist_sqlbind where sql_id = '&sql_id' order by snap_id, position, dup_position;

exit

```

**From information provided by these SQLs its possible to**

- 1) Find date with good execution
- 2) restore STAT from FAST date (Before, Save current stat – SLOW\_STATS)
- 3) check if problem disappears and Save these new set of STATS – FAST\_STATS
- 4) Root Cause Analysis can be done using these 2 set of STATS

Or

- 1) We can use hint in OUTLINE\_DATA section to force plan to use some access paths

Or

- 1) we can see that nothing changed in execution plan but number of executions is now much higher than before