How to Read and Interpret an Explain Plan

NZOUG Webinary
June 25, 2010

Daniel A. Morgan

- Oracle ACE Director 
- University of Washington Oracle Instructor for 10 years
- Morgan of Morgan's Library on the web
  - www.morganslibrary.org
- Member UKOUG and Oracle Applications User Group
- Conference Speaker
  - OpenWorld, Collaborate, Kaleidoscope, Canada, Chile, Denmark, Estonia, Finland, Germany, Japan, New Zealand, Norway, Sweden, UK & US
- Beta tester for Oracle and TimesTen Databases

Syllabus

- Explain Plan Explained?
- Creating Explain Plans
- Reading and Interpreting Explain Plans
- Some Final Thoughts (if time permits)

Discussion

Explain Plan Explained

What is an Explain Plan

- When a SQL statement being explained the statement is not run
- The optimizer chooses an execution plan
- The plan information is made visible in a global temporary table named **`PLAN$`** usually access via the synonym **`PLAN_TABLE`**

Optimizer

- Rule Based Optimizer (RBO)
  - 14 rules that only apply to version 7 object types
- Cost Based Optimizer (CBO)
  - Introduced in Oracle 7
  - Initially terrible, that began changing significantly in 8*i*
  - The CBO has been the best choice since 9.2.0.4
  - The better the information provided the optimizer the better its choices (most of the time)
- What is true in one version of the optimizer likely is not be true in another. Good 9i code may be bad 10g code. Good 10g code may not perform as expected in 11g.

The RBO

Access paths and their ranking	
Path 1	Single Row by Rowid
Path 2	Single Row by Cluster Join
Path 3	Single Row by Hash Cluster key with Unique or Primary Key
Path 4	Single Row by Unique or Primary Key
Path 5	Clustered Join
Path 6	Hash Cluster Key
Path 7	Indexed Cluster Key
Path 8	Composite Index
Path 9	Single-Column Index
Path 10	Bounded Range Search on Indexed Columns
Path 11	Unbounded Range Search on Indexed Columns
Path 12	Sort-Merge Join
Path 13	MAX or MIN of Indexed Column
Path 14	ORDER BY on Indexed Column
Path 15	Full Table Scan

The CBO

- Artificial intelligence uses information about your data to calculate the best access path
- Statistics must be collected using DBMS_STATS
- How do you determine if statistics are current?

```
CREATE TABLE demo AS SELECT * FROM dba_tables;

SELECT COUNT(*) FROM demo;

SELECT table_name, num_rows, last_analyzed
FROM user_tables
ORDER BY 1;

exec dbms_stats.gather_table_stats(USER, 'DEMO');

SELECT table_name, num_rows, last_analyzed
FROM user_tables
ORDER BY 1;
```

Demo

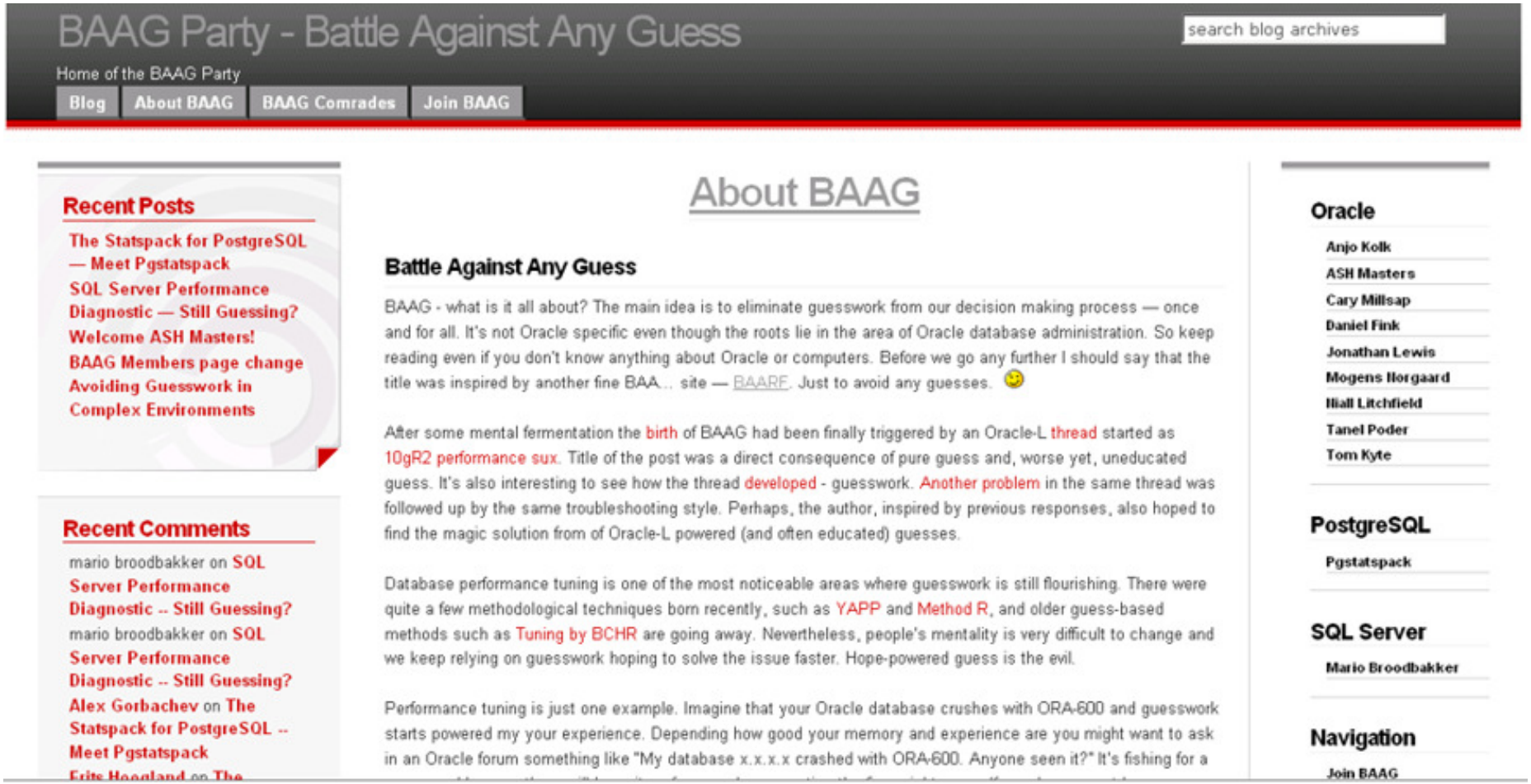
Optimizer Compatibility

```
SQL> set serveroutput on

SQL> DECLARE
  2   ver    VARCHAR2(30);
  3   compat VARCHAR2(30);
  4  BEGIN
  5    dbms_utility.db_version(ver, compat);
  6    dbms_output.put_line('Version: ' || ver ||' compatible: ' || compat);
  7  END;
  8  /
Version: 11.1.0.7.0 Compatible: 10.2.0.4.0

PL/SQL procedure successfully completed.
```

This server is not taking advantage of 11.1.0.7 optimizer enhancements

I have seen this with many customers. It is usually the result of an in-place upgrade or patch. The result is that the production server may behave differently from the development and test servers.

Tuning Methodology



Discussion

Creating Explain Plans

Explain Plan Creation

```
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;


EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

Demo

Legacy Explain Plan Report Creation

- Uses a recursive query that selects fixed columns from the plan table

```
EXPLAIN PLAN
SET STATEMENT_ID = 'abc' FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

SELECT LPAD(' ',2*(level-1)) || operation || ' ' || options ||' ' || object_name
|| ' ' || DECODE(id,0,'Cost = ' || position) QUERY_OUTPUT
FROM plan_table
START WITH id = 0
AND statement_id = 'abc'
CONNECT BY PRIOR id = parent_id
AND statement_id = 'abc';
```

Demo

Contemporary Explain Plan Report Creation

- Uses a pipelined table function built into the DBMS_XPLAN built-in package to dynamically display plan information

```
EXPLAIN PLAN
SET STATEMENT_ID = 'abc' FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

SELECT * FROM TABLE(dbms_xplan.display);
```

Demo

Output an Explain Plan

The old method (pre-9i):

```
SELECT LPAD(' ', 2*(level-1)) || operation || ' ' ||
  options ||' ' || object_name || ' ' ||
  DECODE(id,0,'Cost = ' || position) QUERY_OUTPUT
FROM plan_table
START WITH id = 0
AND statement_id = 'abc'
CONNECT BY PRIOR id = parent_id
AND statement_id = 'abc';
```

The current method (9i or above):

```
SELECT * FROM TABLE(dbms_xplan.display);

CREATE OR REPLACE VIEW xplan AS
SELECT * FROM TABLE(dbms_xplan.display);
```

Discussion

Reading and Interpreting Explain Plans

The SQL Challenge

The SQL challenge ... find the best way to return values present in two different tables

```
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

```
SELECT srvr_id
FROM servers
WHERE srvr_id IN (
  SELECT srvr_id
  FROM serv_inst);
```

```
SELECT srvr_id
FROM servers s
WHERE EXISTS (
  SELECT srvr_id
  FROM serv_inst i
  WHERE s.srvr_id = i.srvr_id);
```

```
SELECT DISTINCT s.srvr_id
FROM servers s, serv_inst i
WHERE s.srvr_id = i.srvr_id;
```

The SQL Challenge

The SQL challenge ... find the best way to return values present in two different tables

```
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

```
SELECT srvr_id
FROM servers
WHERE srvr_id IN (
  SELECT srvr_id
  FROM serv_inst);
```

```
SELECT srvr_id
FROM servers s
WHERE EXISTS (
  SELECT srvr_id
  FROM serv_inst i
  WHERE s.srvr_id = i.srvr_id);
```

```
SELECT DISTINCT s.srvr_id
FROM servers s, serv_inst i
WHERE s.srvr_id = i.srvr_id;
```

The SQL Challenge

The SQL challenge ... find the best way to return values present in two different tables

```
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

```
SELECT srvr_id
FROM servers
WHERE srvr_id IN (
  SELECT srvr_id
  FROM serv_inst);
```

```
SELECT srvr_id
FROM servers s
WHERE EXISTS (
  SELECT srvr_id
  FROM serv_inst i
  WHERE s.srvr_id = i.srvr_id);
```

```
SELECT DISTINCT s.srvr_id
FROM servers s, serv_inst i
WHERE s.srvr_id = i.srvr_id;
```

Demo

And there are more ways that range from the sublime

```
SELECT srvr_id
FROM servers
WHERE srvr_id IN (
  SELECT i.srvr_id
  FROM serv_inst i, servers s
  WHERE i.srvr_id = s.srvr_id);
```

```
SELECT DISTINCT s.srvr_id
FROM servers s, serv_inst i
WHERE s.srvr_id(+) = i.srvr_id;
```

```
WITH q AS (
  SELECT DISTINCT s.srvr_id
  FROM servers s, serv_inst i
  WHERE s.srvr_id = i.srvr_id)
SELECT * FROM q;
```

to the ridiculous

```
SELECT DISTINCT srvr_id
FROM servers
WHERE srvr_id NOT IN (
  SELECT srvr_id
  FROM servers
  MINUS
  SELECT srvr_id
  FROM serv_inst);
```

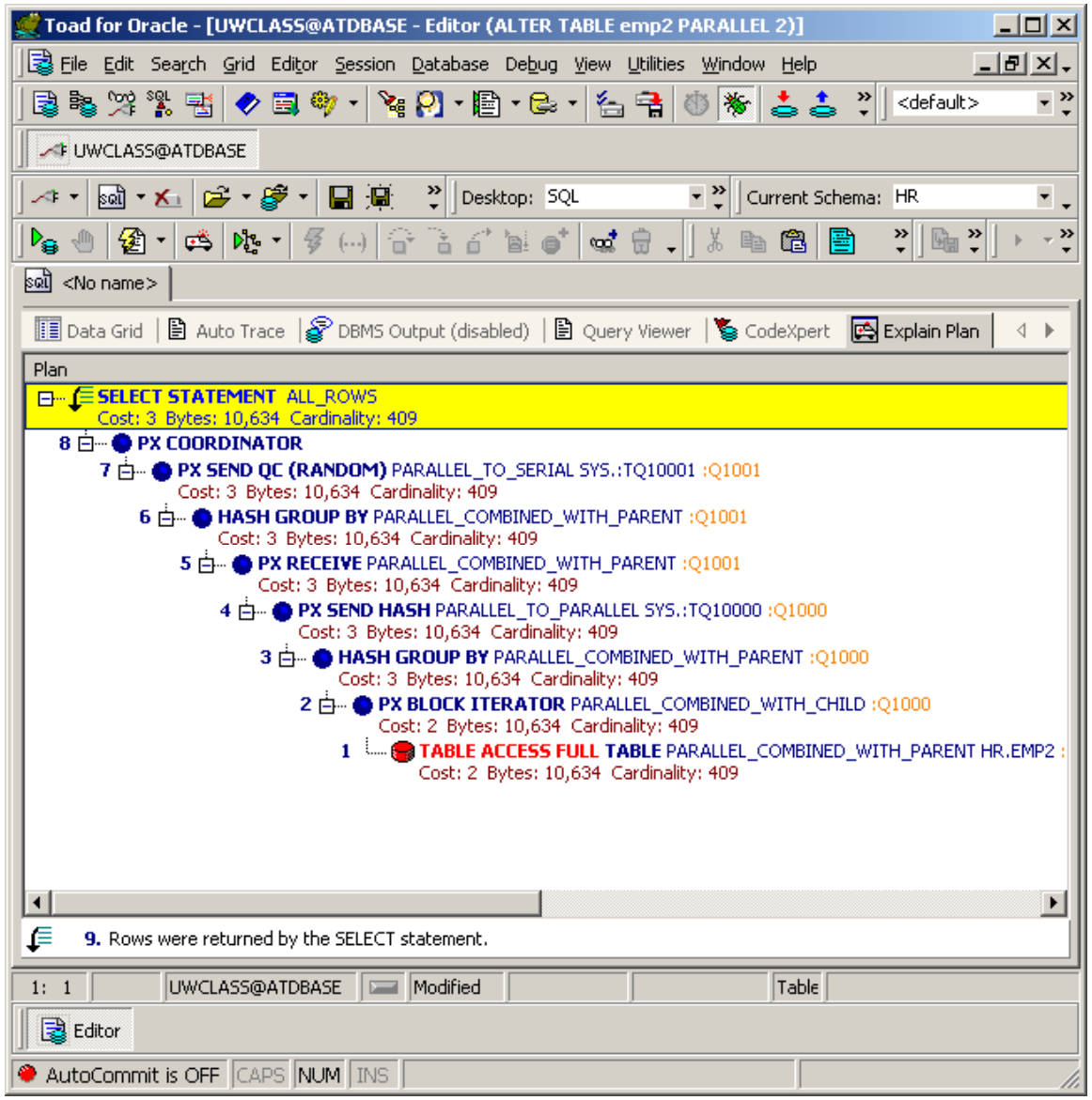
```
SELECT srvr_id
FROM (
  SELECT srvr_id, SUM(cnt) SUMCNT
  FROM (
    SELECT DISTINCT srvr_id, 1 AS CNT
    FROM servers
    UNION ALL
    SELECT DISTINCT srvr_id, 1
    FROM serv_inst)
  GROUP BY srvr_id)
WHERE sumcnt = 2;
```

Create Explain Plan

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers s
WHERE EXISTS (
  SELECT srvr_id
  FROM serv_inst i
  WHERE s.srvr_id = i.srvr_id);

SELECT * FROM TABLE(dbms_xplan.display);
```

Demo

TOAD Plans


Toad for Oracle - [UWCLASS@ATDBASE - Editor (ALTER TABLE emp2 PARALLEL 2)]
File Edit Search Grid Editor Session Database Debug View Utilities Window Help
<default>
UWCLASS@ATDBASE
Desktop: SQL
Current Schema: HR
<No name>
Data Grid
Auto Trace
DBMS Output (disabled)
Query Viewer
CodeXpert
Explain Plan
Plan
SELECT STATEMENT ALL_ROWS
Cost: 3 Bytes: 10,634 Cardinality: 409
8 PX COORDINATOR
7 PX SEND QC (RANDOM) PARALLEL_TO_SERIAL SYS.:TQ10001 :Q1001
Cost: 3 Bytes: 10,634 Cardinality: 409
6 HASH GROUP BY PARALLEL_COMBINED_WITH_PARENT :Q1001
Cost: 3 Bytes: 10,634 Cardinality: 409
5 PX RECEIVE PARALLEL_COMBINED_WITH_PARENT :Q1001
Cost: 3 Bytes: 10,634 Cardinality: 409
4 PX SEND HASH PARALLEL_TO_PARALLEL SYS.:TQ10000 :Q1000
Cost: 3 Bytes: 10,634 Cardinality: 409
3 HASH GROUP BY PARALLEL_COMBINED_WITH_PARENT :Q1000
Cost: 3 Bytes: 10,634 Cardinality: 409
2 PX BLOCK ITERATOR PARALLEL_COMBINED_WITH_CHILD :Q1000
Cost: 2 Bytes: 10,634 Cardinality: 409
1 TABLE ACCESS FULL TABLE PARALLEL_COMBINED_WITH_PARENT HR.EMP2
Cost: 2 Bytes: 10,634 Cardinality: 409
9. Rows were returned by the SELECT statement.
1: 1
UWCLASS@ATDBASE
Modified
Table
Editor
AutoCommit is OFF
CAPS
NUM
INS

TOAD Plans

1 2 3 4

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |   107 |  2782 |     3  (34)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR          |          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)    | :TQ10001 |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,01 | P->S | QC (       |
|   3 |    HASH GROUP BY         |          |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,01 | PCWP |            |
|   4 |     PX RECEIVE           |          |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,01 | PCWP |            |
|   5 |      PX SEND HASH        | :TQ10000 |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,00 | P->P | HASH       |
|   6 |       HASH GROUP BY      |          |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,00 | PCWP |            |
|   7 |        PX BLOCK ITERATOR |          |   107 |  2782 |     2   (0)| 00:00:01 |  Q1,00 | PCWC |            |
|   8 |         TABLE ACCESS FULL| EMP2     |   107 |  2782 |     2   (0)| 00:00:01 |  Q1,00 | PCWP |            |
-------------------------------------------------------------------------------------------------------------

                                                Note
                                                -----
                                          - dynamic sampling used for this statement
```

5

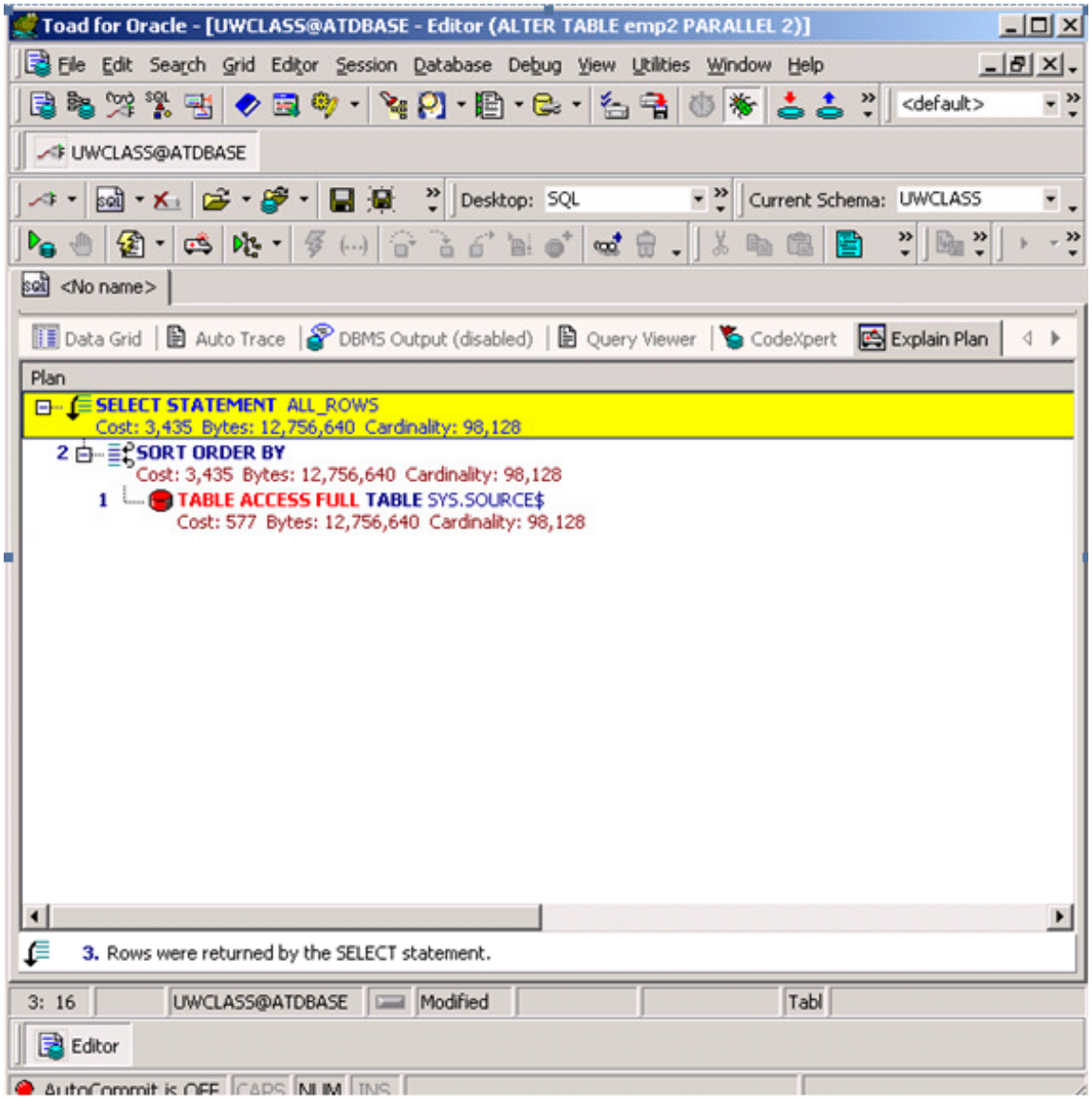
Correct

1. ID
2. Operation
3. Name
4. Cost
5. IN - OUT

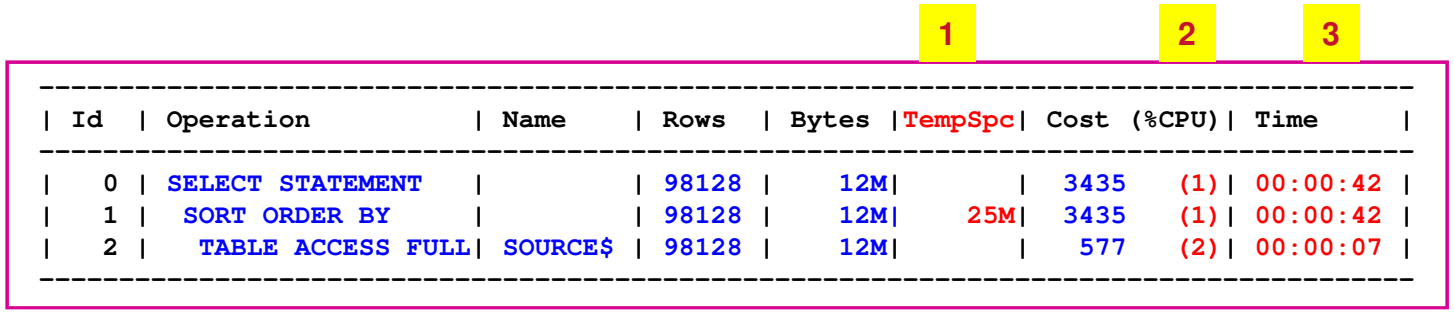
Error / Missing

1. Rows (Cardinality) are wrong
2. Bytes values are wrong
3. CPU% of cost not reported
4. Time not reported
5. Dynamic sampling not reported

TOAD Plans


Toad for Oracle - [UWCLASS@ATDBASE - Editor (ALTER TABLE emp2 PARALLEL 2)]
File Edit Search Grid Editor Session Database Debug View Utilities Window Help
<default>
UWCLASS@ATDBASE
Desktop: SQL
Current Schema: UWCLASS
<No name>
Data Grid
Auto Trace
DBMS Output (disabled)
Query Viewer
CodeXpert
Explain Plan
Plan
SELECT STATEMENT ALL_ROWS
Cost: 3,435 Bytes: 12,756,640 Cardinality: 98,128
2 SORT ORDER BY
Cost: 3,435 Bytes: 12,756,640 Cardinality: 98,128
1 TABLE ACCESS FULL TABLE SYS.SOURCE$
Cost: 577 Bytes: 12,756,640 Cardinality: 98,128
3. Rows were returned by the SELECT statement.
3: 16
UWCLASS@ATDBASE
Modified
Tabl
Editor

TOAD Plans



Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		98128	12M		3435 (1)	00:00:42
1	SORT ORDER BY		98128	12M	25M	3435 (1)	00:00:42
2	TABLE ACCESS FULL	SOURCE$	98128	12M		577 (2)	00:00:07

Correct

1. Operation
2. Name
3. Rows
4. Bytes
5. Cost

Error / Missing

1. Missing swap to TEMP tablespace
2. CPU% of cost not reported
3. Time not reported

Use DBMS_XPLAN Because

- Always current for the database version
  - Patched when the Oracle database is patched
  - Upgraded when the Oracle database is upgraded
- Always understands all Oracle data types
- Always accurately reflect what the optimizer is thinking
- Nothing to install or maintain on the client
- Free (in all Oracle databases)

An Explain Plan Report

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 2840037858
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

How to read an Explain Plan Report

- Begin reading with the line most indented to the right
- If two lines are indented equally the top line is normally executed first
- Sum costs with similar indents in the indent group
- Use the CPU percentage to determine the portion of the cost that is CPU
- (Cost - CPU% of Cost) = Disk I/O

Reading an Explain Plan Report

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 2840037858
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

1. **Start with the most indented: Read 999 rows, ~13KB from the SERV_INST table's primary key index**
2. **Since there is no CPU percentage the cost indicates it will read 3 blocks**

Reading an Explain Plan Report

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 2840037858
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

3. Sort for the query of the PK_SERV_INST index for unique values

Reading an Explain Plan Report

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 2840037858
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

4. **Read one row, 13 bytes from the SERVER table's primary key index: The cost is negligible**

Reading an Explain Plan Report

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 2840037858
------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

5. **Use a NESTED LOOP to join the results of the two index queries**
6. **The cost after this operation will be 4 of which 25% is CPU (3+1=4)**

Reading an Explain Plan Report

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 2840037858
------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

7. The result returned to the end-user will be 11 rows (286 bytes)

A Slightly More Complex Example

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

1. Read 141 rows, about 0.5K of disk, which is 1 block
2. Sort the query result for unique values

A Slightly More Complex Example

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

3. Read 999 rows, about 4K of disk, which is 3 blocks
4. Sort the query result for unique values

A Slightly More Complex Example

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

5. Subtract the result of the IX_SERV_INST query from the result of the PK_SERVERS query

A Slightly More Complex Example

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

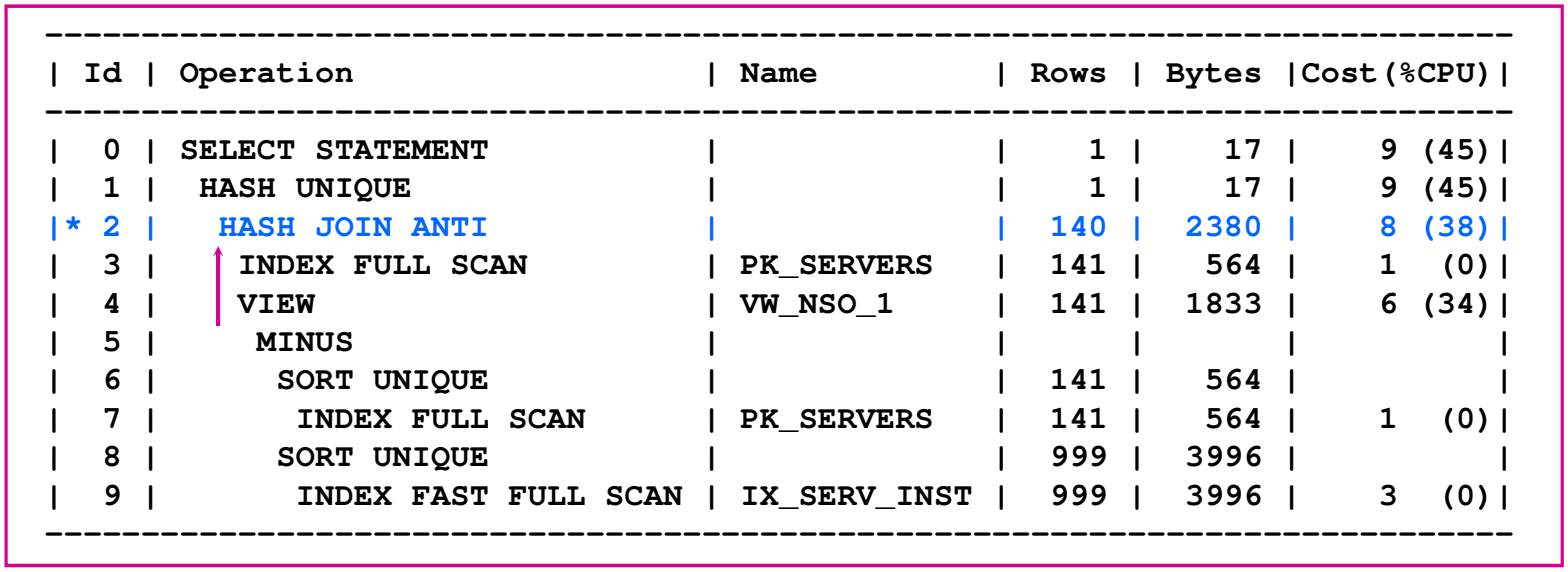
6. And materialize the result of the subtraction as a view
7. The cost up to now has been 4 (3+1). Now the cost is 6 of which 34% (2) is CPU

A Slightly More Complex Example

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

8. Perform a second full scan of the PK_SERVERS index
9. The cost had been 6 we just added one (0% is CPU so it is disk i/o) making the total 7

A Slightly More Complex Example



```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

10. Join the results in the view with the results of the index read
11. The cost has gone from 7 (6+1) to 8 of which 38%, or 3, is CPU

A Slightly More Complex Example

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

12. Use a hashing algorithm to collect a set of unique values for the result set
13. The cost has gone from 8 to 9 of which 45%, or 4, is CPU

A Slightly More Complex Example

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

14. The result returned to the end-user will be 1 row (17 bytes)
15. The total cost is 9 of which 45% (4) is CPU. The balance (5) is disk i/o.

Plans With Errors

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

 ------------------------------------------------------------------------------
 | Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
 ------------------------------------------------------------------------------
 |  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
 |  1 |  INTERSECTION            |               |      |       |            |
 |  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
 |  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
 |  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
 |  5 |    INDEX FAST FULL SCAN| IX_SERV_INST |  999 | 3996  |     3   (0)|
 ------------------------------------------------------------------------------
```

Can you find the error?

Reading an Explain Plan

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

```
-------------------------------------------------------------------------------
| Id | Operation               | Name          | Rows | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT        |               |  141 | 4560  |     6  (84)|
|  1 |  INTERSECTION           |               |      |       |            |
|  2 |   SORT UNIQUE NOSORT    |               |  141 |  564  |     2  (50)|
|  3 |    INDEX FULL SCAN      | PK_SERVERS    |  141 |  564  |     1   (0)|
|  4 |   SORT UNIQUE           |               |  999 | 3996  |     4  (25)|
|  5 |    INDEX FAST FULL SCAN| IX_SERV_INST  |  999 | 3996  |     3   (0)|
-------------------------------------------------------------------------------
```

1. Read 999 rows, ~4K from the SERV_INST table's index IX_SERV_INST: The cost is 3

Reading an Explain Plan

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

```
-------------------------------------------------------------------------------
| Id | Operation               | Name          | Rows | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT        |               |  141 | 4560  |     6  (84)|
|  1 |  INTERSECTION           |               |      |       |            |
|  2 |   SORT UNIQUE NOSORT    |               |  141 |  564  |     2  (50)|
|  3 |    INDEX FULL SCAN      | PK_SERVERS    |  141 |  564  |     1   (0)|
|  4 |   SORT UNIQUE           |               |  999 | 3996  |     4  (25)|
|  5 |    INDEX FAST FULL SCAN| IX_SERV_INST  |  999 | 3996  |     3   (0)|
-------------------------------------------------------------------------------
```

2. Sort the IX_SERV_INST index entries
3. The additional cost is 1 (3+1=4) and 25% of the cost of 4 is CPU (4 x 0.25 = 1): The math works

Reading an Explain Plan

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

 ------------------------------------------------------------------------------
 | Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
 ------------------------------------------------------------------------------
 |  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
 |  1 |  INTERSECTION            |               |      |       |            |
 |  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
 |  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
 |  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
 |  5 |    INDEX FAST FULL SCAN| IX_SERV_INST |  999 | 3996  |     3   (0)|
 ------------------------------------------------------------------------------
```

4. Read 141 rows, 0.5K, from the primary key of the SERVERS table: The cost is 1
5. This line is indented so it is not added, directly, to the cost of operations 4 and 5

Reading an Explain Plan

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

```
-------------------------------------------------------------------------------
| Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
|  1 |  INTERSECTION            |               |      |       |            |
|  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
|  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
|  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
|  5 |    INDEX FAST FULL SCAN  | IX_SERV_INST  |  999 | 3996  |     3   (0)|
-------------------------------------------------------------------------------
```

6. A SORT UNIQUE NOSORT is used to remove potential duplicate rows
7. The additional cost is 1 (1+1=2) and 50% of the cost of 2 is CPU (2 x 0.50 = 1): The math works again

Reading an Explain Plan

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

```
-------------------------------------------------------------------------------
| Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
|  1 |  INTERSECTION            |               |      |       |            |
|  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
|  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
|  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
|  5 |    INDEX FAST FULL SCAN  | IX_SERV_INST  |  999 | 3996  |     3   (0)|
-------------------------------------------------------------------------------
```

8. **Perform an intersection of the two result sets**

Reading an Explain Plan

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

```
-------------------------------------------------------------------------------
| Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT         |               |  141 |  4560 |      6 (84)|
|  1 |  INTERSECTION            |               |      |       |            |
|  2 |   SORT UNIQUE NOSORT     |               |  141 |   564 |      2 (50)|
|  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |   564 |      1  (0)|
|  4 |   SORT UNIQUE            |               |  999 |  3996 |      4 (25)|
|  5 |    INDEX FAST FULL SCAN  | IX_SERV_INST  |  999 |  3996 |      3  (0)|
-------------------------------------------------------------------------------
```

9. Add the two costs (2+4=6): The math works
10. 25% of the 4 is CPU (1) and 50% of the 2 is CPU (1) and (1+1=2). Is 84% of 6 equal to 2?

Explain Plan Demos (if time permits)

- Bitmap Indexes
- Parallel Query
- Partition Pruning
- Temp Space Usage

Demo

Bitmap Indexes

```
EXPLAIN PLAN FOR
SELECT *
FROM serv_inst
WHERE location_code = 30386
OR ws_id BETWEEN 326 AND 333;


-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                            |     2 |   148 |     3   (0)| 00:00:01 |
|   1 |  CONCATENATION                |                            |       |       |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID | SERV_INST                  |     1 |    74 |     1   (0)| 00:00:01 |
|   3 |    BITMAP CONVERSION TO ROWIDS|                            |       |       |            |          |
|*  4 |     BITMAP INDEX RANGE SCAN   | BIX_SERV_INST_WS_ID        |       |       |            |          |
|*  5 |   TABLE ACCESS BY INDEX ROWID | SERV_INST                  |     1 |    74 |     1   (0)| 00:00:01 |
|   6 |    BITMAP CONVERSION TO ROWIDS|                            |       |       |            |          |
|*  7 |     BITMAP INDEX SINGLE VALUE | BIX_SERV_INST_LOCATION_CODE |       |       |            |          |
-------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("WS_ID">=326 AND "WS_ID"<=333)
   5 - filter(LNNVL("WS_ID">=326) OR LNNVL("WS_ID"<=333))
   7 - access("LOCATION_CODE"=30386)
```

Query Rewrite Example

```
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("WS_ID">=326 AND "WS_ID"<=333)
   5 - filter(LNNVL("WS_ID">=326) OR LNNVL("WS_ID"<=333))
   7 - access("LOCATION_CODE"=30386)
```

Parallel Query (PQ)

```
SQL> EXPLAIN PLAN FOR
  2  SELECT SUM(salary)
  3  FROM emp2
  4  GROUP BY department_id;

Explained.

SQL> SELECT plan_table_output FROM table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------------------------
Plan hash value: 3939201228
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------------|
|   0 | SELECT STATEMENT         |          |   107 |  2782 |     3  (34)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR          |          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)    | :TQ10001 |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,01 | P->S | QC (       |
|   3 |    HASH GROUP BY         |          |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,01 | PCWP |            |
|   4 |     PX RECEIVE           |          |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,01 | PCWP |            |
|   5 |      PX SEND HASH        | :TQ10000 |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,00 | P->P | HASH       |
|   6 |       HASH GROUP BY      |          |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,00 | PCWP |            |
|   7 |        PX BLOCK ITERATOR |          |   107 |  2782 |     2   (0)| 00:00:01 |  Q1,00 | PCWC |            |
|   8 |         TABLE ACCESS FULL| EMP2     |   107 |  2782 |     2   (0)| 00:00:01 |  Q1,00 | PCWP |            |
-------------------------------------------------------------------------------------------------------------------
Note
-----
   - dynamic sampling used for this statement
```

Partition Pruning

```
explain plan for
select * from part_zip where state = 'CA';
----------------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |     3 |    72 |     2   (0)| 00:00:01 |       |       |
|   1 |  PARTITION HASH SINGLE|          |     3 |    72 |     2   (0)| 00:00:01 |     1 |     1 |
|*  2 |   TABLE ACCESS FULL   | PART_ZIP |     3 |    72 |     2   (0)| 00:00:01 |     1 |     1 |
----------------------------------------------------------------------------------------------------


explain plan for
select * from part_zip where state = 'HI';
----------------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |     3 |    72 |     2   (0)| 00:00:01 |       |       |
|   1 |  PARTITION HASH SINGLE|          |     3 |    72 |     2   (0)| 00:00:01 |     2 |     2 |
|*  2 |   TABLE ACCESS FULL   | PART_ZIP |     3 |    72 |     2   (0)| 00:00:01 |     2 |     2 |
----------------------------------------------------------------------------------------------------


explain plan for
select * from part_zip where zipcode LIKE '%5%';
----------------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |     2 |    48 |     2   (0)| 00:00:01 |       |       |
|   1 |  PARTITION HASH ALL   |          |     2 |    48 |     2   (0)| 00:00:01 |     1 |     3 |
|*  2 |   TABLE ACCESS FULL   | PART_ZIP |     2 |    48 |     2   (0)| 00:00:01 |     1 |     3 |
----------------------------------------------------------------------------------------------------
```

Temp Space Usage

```
SQL> SELECT * FROM TABLE(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 995087943

-------------------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         | 98128 |    12M|       |  3435   (1)| 00:00:42 |
|   1 |  SORT ORDER BY     |         | 98128 |    12M|    25M|  3435   (1)| 00:00:42 |
|   2 |   TABLE ACCESS FULL| SOURCE$ | 98128 |    12M|       |   577   (2)| 00:00:07 |
-------------------------------------------------------------------------------------------
```

Discussion

Final Thoughts

Common Issues

- Explain Plans are not everything and not what will necessarily happen
- You can Explain Plan any DML statement
- Join Syntax Consistency (traditional vs ANSI)
- Missing Joins (Cartesian Products)
- Myth Busting

This Is One Way To Look At It

```
SQL> EXPLAIN PLAN FOR
  2  SELECT COUNT(*)
  3  FROM parent p, child c
  4  WHERE p.parent_id = c.parent_id;

SQL> select * From table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 3584092213
------------------------------------------------------------------------------------------
| Id  | Operation           | Name   | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |        |     1 |    10 |       |  3163   (5)| 00:00:38 |
|   1 |  SORT AGGREGATE     |        |     1 |    10 |       |            |          |
|*  2 |   HASH JOIN         |        |  1500K|    14M|  8312K|  3163   (5)| 00:00:38 |
|   3 |    TABLE ACCESS FULL| PARENT |   500K|  2442K|       |   380   (4)| 00:00:05 |
|   4 |    TABLE ACCESS FULL| CHILD  |  1500K|  7324K|       |  1106   (5)| 00:00:14 |
------------------------------------------------------------------------------------------


SQL> EXPLAIN PLAN FOR
  2  SELECT COUNT(*)
  3  FROM parent p, child c
  4  WHERE p.parent_id = c.parent_id
  5  AND c.birth_date is NOT NULL;

SQL> select * From table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 3584092213
------------------------------------------------------------------------------------------
| Id  | Operation           | Name   | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |        |     1 |    16 |       |  3037   (5)| 00:00:37 |
|   1 |  SORT AGGREGATE     |        |     1 |    16 |       |            |          |
|*  2 |   HASH JOIN         |        |   999K|    15M|  8312K|  3037   (5)| 00:00:37 |
|   3 |    TABLE ACCESS FULL| PARENT |   500K|  2442K|       |   380   (4)| 00:00:05 |
|*  4 |    TABLE ACCESS FULL| CHILD  |   999K|    10M|       |  1116   (6)| 00:00:14 |
------------------------------------------------------------------------------------------
```

This Is Another

```
SQL> set timing on

SQL> SELECT COUNT(*)
  2  FROM parent p, child c
  3  WHERE p.parent_id = c.parent_id;

  COUNT(*)
----------
   1500000

Elapsed: 00:00:00.59




SQL> SELECT COUNT(*)
  2  FROM parent p, child c
  3  WHERE p.parent_id = c.parent_id
  4  AND birth_date is NOT NULL;

  COUNT(*)
----------
   1000000

Elapsed: 00:00:00.53
```

They are both important

Seeing What's Real

- DBMS_XPLAN.DISPLAY_CURSOR

```
SELECT /* XPLAN_CURSOR */ DISTINCT s.srvr_id
FROM servers s, serv_inst I
WHERE s.srvr_id = i.srvr_id;

SELECT sql_id
FROM gv$sql
WHERE sql_text LIKE '%XPLAN_CURSOR%';

SELECT * FROM
TABLE(dbms_xplan.display_cursor('cpm9ss48qd32f'));
```

DBMS_XPLAN.DISPLAY_CURSOR

```
dbms_xplan.display_cursor(
sql_id          IN VARCHAR2 DEFAULT NULL,
cursor_child_no IN INTEGER DEFAULT 0,
format          IN VARCHAR2 DEFAULT 'TYPICAL')
RETURN dbms_xplan_type_table PIPELINED;
```

Format Constants	
ALIAS	If relevant, shows the "Query Block Name / Object Alias" section
ALLSTATS	A shortcut for 'IOSTATS MEMSTATS'
BYTES	If relevant, shows the number of bytes estimated by the optimizer
COST	If relevant, shows optimizer cost information
IOSTATS	Assuming that basic plan statistics are collected when SQL statements are executed (either by using the gather_plan_statistics hint or by setting the parameter statistics_level to ALL), this format will show IO statistics for ALL (or only for the LAST as shown below) executions of the cursor
LAST	By default, plan statistics are shown for all executions of the cursor. The keyword LAST can be specified to see only the statistics for the last execution
MEMSTATS	Assuming that PGA memory management is enabled (that is, pga_aggregate_target parameter is set to a non 0 value), this format allows to display memory management statistics (for example, execution mode of the operator, how much memory was used, number of bytes spilled to disk, and so on). These statistics only apply to memory intensive operations like hash-joins, sort or some bitmap operators
NOTE	If relevant, shows the note section of the explain plan
PARALLEL	If relevant, shows PX information (distribution method and table queue information)
PARTITION	If relevant, shows partition pruning information
PREDICATE	If relevant, shows the predicate section
PROJECTION	If relevant, shows the projection section
REMOTE	If relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)
ROWS	If relevant, shows the number of rows estimated by the optimizer
RUNSTATS_LAST	Same as IOSTATS LAST: displays the runtime stat for the last execution of the cursor
RUNSTATS_TOT	Same as IOSTATS: displays IO statistics for all executions of the specified cursor

V$SQL

```
SQL> desc v$sql
 Name                                                      Null?    Type
 ----------------------------------------------------------------- --------
 SQL_TEXT                                                           VARCHAR2(1000
 SQL_FULLTEXT                                                       CLOB
 SQL_ID                                                             VARCHAR2(13)
 SHARABLE_MEM                                                       NUMBER
 PERSISTENT_MEM                                                     NUMBER
 RUNTIME_MEM                                                        NUMBER
 SORTS                                                              NUMBER
 FETCHES                                                            NUMBER
 PX_SERVERS_EXECUTIONS                                              NUMBER
 PARSE_CALLS                                                        NUMBER
 DISK_READS                                                         NUMBER
 DIRECT_WRITES                                                      NUMBER
 BUFFER_GETS                                                        NUMBER
 APPLICATION_WAIT_TIME                                              NUMBER
 CONCURRENCY_WAIT_TIME                                              NUMBER
 CLUSTER_WAIT_TIME                                                  NUMBER
 USER_IO_WAIT_TIME                                                  NUMBER
 PLSQL_EXEC_TIME                                                    NUMBER
 JAVA_EXEC_TIME                                                     NUMBER
 ROWS_PROCESSED                                                     NUMBER
 COMMAND_TYPE                                                       NUMBER
 OPTIMIZER_MODE                                                     VARCHAR2(10)
 OPTIMIZER_COST                                                     NUMBER
 OPTIMIZER_ENV                                                      RAW(2000)
 PLAN_HASH_VALUE                                                    NUMBER
 CHILD_NUMBER                                                       NUMBER
 CPU_TIME                                                           NUMBER
 ELAPSED_TIME                                                       NUMBER
 IS_BIND_SENSITIVE                                                  VARCHAR2(1)
 SQL_PROFILE                                                        VARCHAR2(64)
 LAST_ACTIVE_TIME                                                   DATE
 BIND_DATA                                                          RAW(2000)
 IO_INTERCONNECT_BYTES                                              NUMBER
 IO_DISK_BYTES                                                      NUMBER

Abbreviated column list
```

V$SQL_PLAN

```
SQL> desc v$sql_plan
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- --------------
 ADDRESS                                                           RAW(4)
 HASH_VALUE                                                        NUMBER
 SQL_ID                                                            VARCHAR2(13)
 PLAN_HASH_VALUE                                                   NUMBER
 TIMESTAMP                                                         DATE
 OPERATION                                                         VARCHAR2(30)
 OPTIMIZER                                                         VARCHAR2(20)
 ID                                                                NUMBER
 PARENT_ID                                                         NUMBER
 DEPTH                                                             NUMBER
 POSITION                                                          NUMBER
 SEARCH_COLUMNS                                                    NUMBER
 COST                                                              NUMBER
 CARDINALITY                                                       NUMBER
 BYTES                                                             NUMBER
 PARTITION_START                                                   VARCHAR2(64)
 PARTITION_STOP                                                    VARCHAR2(64)
 CPU_COST                                                          NUMBER
 IO_COST                                                           NUMBER
 TEMP_SPACE                                                        NUMBER
 ACCESS_PREDICATES                                                 VARCHAR2(4000)
 FILTER_PREDICATES                                                 VARCHAR2(4000)
 PROJECTION                                                        VARCHAR2(4000)
 TIME                                                              NUMBER
 QBLOCK_NAME                                                       VARCHAR2(30)

Abbreviated column list
```

Any DML Statement Can Be Explained

```
SQL> explain plan for
  2  MERGE INTO bonuses b
  3  USING (
  4    SELECT employee_id, salary, dept_no
  5    FROM employee
  6    WHERE dept_no =20) e
  7  ON (b.employee_id = e.employee_id)
  8  WHEN MATCHED THEN
  9    UPDATE SET b.bonus = e.salary * 0.1
 10    DELETE WHERE (e.salary < 40000)
 11  WHEN NOT MATCHED THEN
 12    INSERT (b.employee_id, b.bonus)
 13    VALUES (e.employee_id, e.salary * 0.05)
 14    WHERE (e.salary > 40000);

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | MERGE STATEMENT       |          |     6 |   546 |    17   (6)| 00:00:01 |
|   1 |  MERGE                | BONUSES  |       |       |            |          |
|   2 |   VIEW                |          |       |       |            |          |
|*  3 |    HASH JOIN OUTER    |          |     6 |   462 |    17   (6)| 00:00:01 |
|*  4 |     TABLE ACCESS FULL| EMPLOYEE |     6 |   234 |     8   (0)| 00:00:01 |
|   5 |     TABLE ACCESS FULL| BONUSES  |     6 |   228 |     8   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

INSERT, UPDATE, DELETE, and MERGE

Any Join Syntax Can Be Used (Traditional or ANSI joins)

```
explain plan for
select distinct i.srvr_id
from servers s, serv_inst i
where s.srvr_id = i.srvr_id;
-------------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |   999 | 25974 |     9  (12)| 00:00:01 |
|   1 |  HASH UNIQUE        |             |   999 | 25974 |     9  (12)| 00:00:01 |
|   2 |   NESTED LOOPS      |             |   999 | 25974 |     8   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| SERV_INST   |   999 | 12987 |     8   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN| PK_SERVERS  |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------



explain plan for
select distinct i.srvr_id
from servers s inner join serv_inst i
on s.srvr_id = i.srvr_id;
-------------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |   999 | 25974 |     9  (12)| 00:00:01 |
|   1 |  HASH UNIQUE        |             |   999 | 25974 |     9  (12)| 00:00:01 |
|   2 |   NESTED LOOPS      |             |   999 | 25974 |     8   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| SERV_INST   |   999 | 12987 |     8   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN| PK_SERVERS  |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

Produce identical plans

... in all currently supported versions ...

Beware of Missing Joins

```
SQL> explain plan for
  2  select s.srvr_id
  3  from servers s, serv_inst i
  4  where s.srvr_id = i.srvr_id;

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
| Id  | Operation          | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |             |   999 | 25974 |     8   (0)| 00:00:01 |
|   1 |  NESTED LOOPS      |             |   999 | 25974 |     8   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| SERV_INST   |   999 | 12987 |     8   (0)| 00:00:01 |
|*  3 |   INDEX UNIQUE SCAN| PK_SERVERS  |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------


SQL> explain plan for
  2  select s.srvr_id
  3  from servers s, serv_inst i;

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                     |  140K|  1788K|   130   (1)| 00:00:02 |
|   1 |  MERGE JOIN CARTESIAN          |                     |  140K|  1788K|   130   (1)| 00:00:02 |
|   2 |   INDEX FAST FULL SCAN         | PK_SERVERS          |   141 |  1833 |     2   (0)| 00:00:01 |
|   3 |   BUFFER SORT                  |                     |   999 |       |   128   (1)| 00:00:02 |
|   4 |    BITMAP CONVERSION TO ROWIDS |                     |   999 |       |     1   (0)| 00:00:01 |
|   5 |     BITMAP INDEX FAST FULL SCAN| BIX_SERV_INST_WS_ID |       |       |            |          |
-----------------------------------------------------------------------------------------------------
```

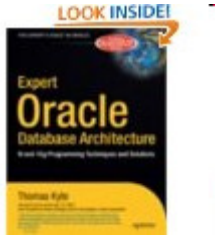
Note the impact of the missing join

Myth Busting


```
SQL> explain plan for
  2  SELECT doc_name
  3  FROM t
  4  WHERE person_id = 221;

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |   216 |  6264 |    64   (4)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T    |   216 |  6264 |    64   (4)| 00:00:01 |
------------------------------------------------------------------------------



SQL> explain plan for
  2  SELECT /*+ INDEX(t ix_t_person_id) */ doc_name
  3  FROM t
  4  WHERE person_id = 221;

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |   216 |  6264 |   216   (0)| 00:00:03 |
|   1 |  TABLE ACCESS BY INDEX ROWID| T              |   216 |  6264 |   216   (0)| 00:00:03 |
|*  2 |   INDEX RANGE SCAN           | IX_T_PERSON_ID |   216 |       |     1   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------
```

Resources

- Oracle Technology Network
  - http://tahiti.oracle.com
- Morgan of Morgan's Library on the web
  - www.morganslibrary.org
- Tom Kyte
  - http://asktom.oracle.com
  - Tom's Books
- Jonathan Lewis
  - http://www.jlcomp.demon.co.uk/faq
  - Jonathan's Books
- Cary Millsap
  - http://carymillsap.blogspot.com



Questions

Complex Query Plan

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                               |     1 |    66 |   264    (2)| 00:00:04 |
|   1 |  MERGE JOIN OUTER                        |                               |     1 |    66 |   264    (2)| 00:00:04 |
|   2 |   MERGE JOIN OUTER                       |                               |     1 |    44 |   139    (3)| 00:00:02 |
|   3 |    NESTED LOOPS OUTER                    |                               |     1 |    22 |    28    (4)| 00:00:01 |
|   4 |     FAST DUAL                            |                               |     1 |       |     2    (0)| 00:00:01 |
|   5 |     VIEW                                 |                               |     1 |    22 |    26    (4)| 00:00:01 |
|   6 |      VIEW                                |                               |     1 |    18 |    26    (4)| 00:00:01 |
|*  7 |       FILTER                             |                               |       |       |             |          |
|   8 |        SORT AGGREGATE                    |                               |     1 |    80 |             |          |
|*  9 |         HASH JOIN                        |                               |   717 | 57360 |    26    (4)| 00:00:01 |
|* 10 |          HASH JOIN                       |                               |    67 |  4623 |     7   (15)| 00:00:01 |
|  11 |           TABLE ACCESS FULL              | MKTDRIDAILYRESAVAIL           |    17 |   629 |     3    (0)| 00:00:01 |
|* 12 |           TABLE ACCESS FULL              | MKTDRIHRLYRESAVAIL            |    67 |  2144 |     3    (0)| 00:00:01 |
|  13 |          TABLE ACCESS FULL               | MKTHOUR                       |  1752 | 19272 |    19    (0)| 00:00:01 |
|  14 |    VIEW                                  |                               |     1 |    22 |   110    (1)| 00:00:02 |
|  15 |     VIEW                                 |                               |     1 |     9 |   110    (1)| 00:00:02 |
|* 16 |      FILTER                              |                               |       |       |             |          |
|  17 |       SORT AGGREGATE                     |                               |     1 |    19 |             |          |
|  18 |        MERGE JOIN                        |                               |   365 |  6935 |   110    (1)| 00:00:02 |
|* 19 |         TABLE ACCESS BY INDEX ROWID      | MKTSTUDYMODE                  |     2 |    18 |     4    (0)| 00:00:01 |
|  20 |          INDEX FULL SCAN                 | XPKMKTSTUDYMODE               |    16 |       |     1    (0)| 00:00:01 |
|* 21 |         SORT JOIN                        |                               |  1318 | 13180 |   106    (1)| 00:00:02 |
|* 22 |          TABLE ACCESS FULL               | MKTPLAN                       |  1318 | 13180 |   105    (0)| 00:00:02 |
|  23 |   BUFFER SORT                            |                               |     1 |    22 |   153    (1)| 00:00:02 |
|  24 |    VIEW                                  |                               |     1 |    22 |   125    (0)| 00:00:02 |
|  25 |     VIEW                                 |                               |     1 |     9 |   125    (0)| 00:00:02 |
|* 26 |      FILTER                              |                               |       |       |             |          |
|  27 |       SORT AGGREGATE                     |                               |     1 |    54 |             |          |
|* 28 |        HASH JOIN                         |                               |    99 |  5346 |   125    (0)| 00:00:02 |
|  29 |         NESTED LOOPS                     |                               |       |       |             |          |
|  30 |          NESTED LOOPS                    |                               |    81 |  3483 |   109    (0)| 00:00:02 |
|  31 |           NESTED LOOPS                   |                               |    94 |  2632 |    27    (0)| 00:00:01 |
|  32 |            VIEW                          |                               |     1 |     9 |     2    (0)| 00:00:01 |
|* 33 |             COUNT STOPKEY                |                               |       |       |             |          |
|  34 |              FAST DUAL                   |                               |     1 |       |     2    (0)| 00:00:01 |
|  35 |            TABLE ACCESS BY INDEX ROWID   | MKTCASE                       |    94 |  1786 |    25    (0)| 00:00:01 |
|* 36 |             INDEX RANGE SCAN             | XFMKTCASE_MKTDAYAPPV2200_U    |    94 |       |     1    (0)| 00:00:01 |
|* 37 |           INDEX UNIQUE SCAN              | XPKMKTPLAN                    |     1 |       |     0    (0)| 00:00:01 |
|  38 |          TABLE ACCESS BY INDEX ROWID     | MKTPLAN                       |     1 |    15 |     1    (0)| 00:00:01 |
|  39 |         INDEX FAST FULL SCAN             | XPKMKTCASEDRFORECAST          |  4560 | 50160 |    16    (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------
```

Complex Query Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	66	264 (2)	00:00:04
1	MERGE JOIN OUTER		1	66	264 (2)	00:00:04
2	MERGE JOIN OUTER		1	44	139 (3)	00:00:02
3	NESTED LOOPS OUTER		1	22	28 (4)	00:00:01
4	FAST DUAL		1		2 (0)	00:00:01
5	VIEW		1	22	26 (4)	00:00:01
6	VIEW		1	18	26 (4)	00:00:01
* 7	FILTER					
8	SORT AGGREGATE		1	80		
* 9	HASH JOIN		717	57360	26 (4)	00:00:01
* 10	HASH JOIN		67	4623	7 (15)	00:00:01
11	TABLE ACCESS FULL	MKTDRIDAILYRESAVAIL	17	629	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	MKTDRIHRLYRESAVAIL	67	2144	3 (0)	00:00:01
13	TABLE ACCESS FULL	MKTHOUR	1752	19272	19 (0)	00:00:01
14	VIEW		1	22	110 (1)	00:00:02
15	VIEW		1	9	110 (1)	00:00:02
* 16	FILTER					
17	SORT AGGREGATE		1	19		
18	MERGE JOIN		365	6935	110 (1)	00:00:02
* 19	TABLE ACCESS BY INDEX ROWID	MKTSTUDYMODE	2	18	4 (0)	00:00:01
20	INDEX FULL SCAN	XPKMKTSTUDYMODE	16		1 (0)	00:00:01
* 21	SORT JOIN		1318	13180	106 (1)	00:00:02
* 22	TABLE ACCESS FULL	MKTPLAN	1318	13180	105 (0)	00:00:02
23	BUFFER SORT		1	22	153 (1)	00:00:02
24	VIEW		1	22	125 (0)	00:00:02
25	VIEW		1	9	125 (0)	00:00:02
* 26	FILTER					
27	SORT AGGREGATE		1	54		
* 28	HASH JOIN		99	5346	125 (0)	00:00:02
29	NESTED LOOPS					
30	NESTED LOOPS		81	3483	109 (0)	00:00:02
31	NESTED LOOPS		94	2632	27 (0)	00:00:01
32	VIEW		1	9	2 (0)	00:00:01
* 33	COUNT STOPKEY					
34	FAST DUAL		1		2 (0)	00:00:01
35	TABLE ACCESS BY INDEX ROWID	MKTCASE	94	1786	25 (0)	00:00:01
* 36	INDEX RANGE SCAN	XFMKTCASE_MKTDAYAPPV2200_U	94		1 (0)	00:00:01
* 37	INDEX UNIQUE SCAN	XPKMKTPLAN	1		0 (0)	00:00:01
38	TABLE ACCESS BY INDEX ROWID	MKTPLAN	1	15	1 (0)	00:00:01
39	INDEX FAST FULL SCAN	XPKMKTCASEDRFORECAST	4560	50160	16 (0)	00:00:01

Complex Query Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	66	264 (2)	00:00:04
1	MERGE JOIN OUTER		1	66	264 (2)	00:00:04
2	MERGE JOIN OUTER		1	44	139 (3)	00:00:02
3	NESTED LOOPS OUTER		1	22	28 (4)	00:00:01
4	FAST DUAL		1		2 (0)	00:00:01
5	VIEW		1	22	26 (4)	00:00:01
6	VIEW		1	18	26 (4)	00:00:01
* 7	FILTER					
8	SORT AGGREGATE		1	80		
* 9	HASH JOIN		717	57360	26 (4)	00:00:01
* 10	HASH JOIN		67	4623	7 (15)	00:00:01
11	TABLE ACCESS FULL	MKTDRIDAILYRESAVAIL	17	629	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	MKTDRIHRLYRESAVAIL	67	2144	3 (0)	00:00:01
13	TABLE ACCESS FULL	MKTHOUR	1752	19272	19 (0)	00:00:01
14	VIEW		1	22	110 (1)	00:00:02
15	VIEW		1	9	110 (1)	00:00:02
* 16	FILTER					
17	SORT AGGREGATE		1	19		
18	MERGE JOIN		365	6935	110 (1)	00:00:02
* 19	TABLE ACCESS BY INDEX ROWID	MKTSTUDYMODE	2	18	4 (0)	00:00:01
20	INDEX FULL SCAN	XPKMKTSTUDYMODE	16		1 (0)	00:00:01
* 21	SORT JOIN		1318	13180	106 (1)	00:00:02
* 22	TABLE ACCESS FULL	MKTPLAN	1318	13180	105 (0)	00:00:02
23	BUFFER SORT		1	22	153 (1)	00:00:02
24	VIEW		1	22	125 (0)	00:00:02
25	VIEW		1	9	125 (0)	00:00:02
* 26	FILTER					
27	SORT AGGREGATE		1	54		
* 28	HASH JOIN		99	5346	125 (0)	00:00:02
29	NESTED LOOPS					
30	NESTED LOOPS		81	3483	109 (0)	00:00:02
31	NESTED LOOPS		94	2632	27 (0)	00:00:01
32	VIEW		1	9	2 (0)	00:00:01
* 33	COUNT STOPKEY					
34	FAST DUAL		1		2 (0)	00:00:01
35	TABLE ACCESS BY INDEX ROWID	MKTCASE	94	1786	25 (0)	00:00:01
* 36	INDEX RANGE SCAN	XFMKTCASE_MKTDAYAPPV2200_U	94		1 (0)	00:00:01
* 37	INDEX UNIQUE SCAN	XPKMKTPLAN	1		0 (0)	00:00:01
38	TABLE ACCESS BY INDEX ROWID	MKTPLAN	1	15	1 (0)	00:00:01
39	INDEX FAST FULL SCAN	XPKMKTCASEDRFORECAST	4560	50160	16 (0)	00:00:01

25 + 2 = 27

Complex Query Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	66	264 (2)	00:00:04
1	MERGE JOIN OUTER		1	66	264 (2)	00:00:04
2	MERGE JOIN OUTER		1	44	139 (3)	00:00:02
3	NESTED LOOPS OUTER		1	22	28 (4)	00:00:01
4	FAST DUAL		1		2 (0)	00:00:01
5	VIEW		1	22	26 (4)	00:00:01
6	VIEW		1	18	26 (4)	00:00:01
* 7	FILTER					
8	SORT AGGREGATE		1	80		
* 9	HASH JOIN		717	57360	26 (4)	00:00:01
* 10	HASH JOIN		67	4623	7 (15)	00:00:01
11	TABLE ACCESS FULL	MKTDRIDAILYRESAVAIL	17	629	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	MKTDRIHRLYRESAVAIL	67	2144	3 (0)	00:00:01
13	TABLE ACCESS FULL	MKTHOUR	1752	19272	19 (0)	00:00:01
14	VIEW		1	22	110 (1)	00:00:02
15	VIEW		1	9	110 (1)	00:00:02
* 16	FILTER					
17	SORT AGGREGATE		1	19		
18	MERGE JOIN		365	6935	110 (1)	00:00:02
* 19	TABLE ACCESS BY INDEX ROWID	MKTSTUDYMODE	2	18	4 (0)	00:00:01
20	INDEX FULL SCAN	XPKMKTSTUDYMODE	16		1 (0)	00:00:01
* 21	SORT JOIN		1318	13180	106 (1)	00:00:02
* 22	TABLE ACCESS FULL	MKTPLAN	1318	13180	105 (0)	00:00:02
23	BUFFER SORT		1	22	153 (1)	00:00:02
24	VIEW		1	22	125 (0)	00:00:02
25	VIEW		1	9	125 (0)	00:00:02
* 26	FILTER					
27	SORT AGGREGATE		1	54		
* 28	HASH JOIN		99	5346	125 (0)	00:00:02
29	NESTED LOOPS					
30	NESTED LOOPS		81	3483	109 (0)	00:00:02
31	NESTED LOOPS		94	2632	27 (0)	00:00:01
32	VIEW		1	9	2 (0)	00:00:01
* 33	COUNT STOPKEY					
34	FAST DUAL		1		2 (0)	00:00:01
35	TABLE ACCESS BY INDEX ROWID	MKTCASE	94	1786	25 (0)	00:00:01
* 36	INDEX RANGE SCAN	XFMKTCASE_MKTDAYAPPV2200_U	94		1 (0)	00:00:01
* 37	INDEX UNIQUE SCAN	XPKMKTPLAN	1		0 (0)	00:00:01
38	TABLE ACCESS BY INDEX ROWID	MKTPLAN	1	15	1 (0)	00:00:01
39	INDEX FAST FULL SCAN	XPKMKTCASEDRFORECAST	4560	50160	16 (0)	00:00:01

16 + 109 = 125

Complex Query Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	66	264 (2)	00:00:04
1	MERGE JOIN OUTER		1	66	264 (2)	00:00:04
2	MERGE JOIN OUTER		1	44	139 (3)	00:00:02
3	NESTED LOOPS OUTER		1	22	28 (4)	00:00:01
4	FAST DUAL		1		2 (0)	00:00:01
5	VIEW		1	22	26 (4)	00:00:01
6	VIEW		1	18	26 (4)	00:00:01
* 7	FILTER					
8	SORT AGGREGATE		1	80		
* 9	HASH JOIN		717	57360	26 (4)	00:00:01
* 10	HASH JOIN		67	4623	7 (15)	00:00:01
11	TABLE ACCESS FULL	MKTDRIDAILYRESAVAIL	17	629	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	MKTDRIHRLYRESAVAIL	67	2144	3 (0)	00:00:01
13	TABLE ACCESS FULL	MKTHOUR	1752	19272	19 (0)	00:00:01
14	VIEW		1	22	110 (1)	00:00:02
15	VIEW		1	9	110 (1)	00:00:02
* 16	FILTER					
17	SORT AGGREGATE		1	19		
18	MERGE JOIN		365	6935	110 (1)	00:00:02
* 19	TABLE ACCESS BY INDEX ROWID	MKTSTUDYMODE	2	18	4 (0)	00:00:01
20	INDEX FULL SCAN	XPKMKTSTUDYMODE	16		1 (0)	00:00:01
* 21	SORT JOIN		1318	13180	106 (1)	00:00:02
* 22	TABLE ACCESS FULL	MKTPLAN	1318	13180	105 (0)	00:00:02
23	BUFFER SORT		1	22	153 (1)	00:00:02
24	VIEW		1	22	125 (0)	00:00:02
25	VIEW		1	9	125 (0)	00:00:02
* 26	FILTER					
27	SORT AGGREGATE		1	54		
* 28	HASH JOIN		99	5346	125 (0)	00:00:02
29	NESTED LOOPS					
30	NESTED LOOPS		81	3483	109 (0)	00:00:02
31	NESTED LOOPS		94	2632	27 (0)	00:00:01
32	VIEW		1	9	2 (0)	00:00:01
* 33	COUNT STOPKEY					
34	FAST DUAL		1		2 (0)	00:00:01
35	TABLE ACCESS BY INDEX ROWID	MKTCASE	94	1786	25 (0)	00:00:01
* 36	INDEX RANGE SCAN	XFMKTCASE_MKTDAYAPPV2200_U	94		1 (0)	00:00:01
* 37	INDEX UNIQUE SCAN	XPKMKTPLAN	1		0 (0)	00:00:01
38	TABLE ACCESS BY INDEX ROWID	MKTPLAN	1	15	1 (0)	00:00:01
39	INDEX FAST FULL SCAN	XPKMKTCASEDRFORECAST	4560	50160	16 (0)	00:00:01

Complex Query Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	66	264 (2)	00:00:04
1	MERGE JOIN OUTER		1	66	264 (2)	00:00:04
2	MERGE JOIN OUTER		1	44	139 (3)	00:00:02
3	NESTED LOOPS OUTER		1	22	28 (4)	00:00:01
4	FAST DUAL		1		2 (0)	00:00:01
5	VIEW		1	22	26 (4)	00:00:01
6	VIEW		1	18	26 (4)	00:00:01
* 7	FILTER					
8	SORT AGGREGATE		1	80		
* 9	HASH JOIN		717	57360	26 (4)	00:00:01
* 10	HASH JOIN		67	4623	7 (15)	00:00:01
11	TABLE ACCESS FULL	MKTDRIDAILYRESAVAIL	17	629	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	MKTDRIHRLYRESAVAIL	67	2144	3 (0)	00:00:01
13	TABLE ACCESS FULL	MKTHOUR	1752	19272	19 (0)	00:00:01
14	VIEW		1	22	110 (1)	00:00:02
15	VIEW		1	9	110 (1)	00:00:02
* 16	FILTER					
17	SORT AGGREGATE		1	19		
18	MERGE JOIN		365	6935	110 (1)	00:00:02
* 19	TABLE ACCESS BY INDEX ROWID	MKTSTUDYMODE	2	18	4 (0)	00:00:01
20	INDEX FULL SCAN	XPKMKTSTUDYMODE	16		1 (0)	00:00:01
* 21	SORT JOIN		1318	13180	106 (1)	00:00:02
* 22	TABLE ACCESS FULL	MKTPLAN	1318	13180	105 (0)	00:00:02
23	BUFFER SORT		1	22	153 (1)	00:00:02
24	VIEW		1	22	125 (0)	00:00:02
25	VIEW		1	9	125 (0)	00:00:02
* 26	FILTER					
27	SORT AGGREGATE		1	54		
* 28	HASH JOIN		99	5346	125 (0)	00:00:02
29	NESTED LOOPS					
30	NESTED LOOPS		81	3483	109 (0)	00:00:02
31	NESTED LOOPS		94	2632	27 (0)	00:00:01
32	VIEW		1	9	2 (0)	00:00:01
* 33	COUNT STOPKEY					
34	FAST DUAL		1		2 (0)	00:00:01
35	TABLE ACCESS BY INDEX ROWID	MKTCASE	94	1786	25 (0)	00:00:01
* 36	INDEX RANGE SCAN	XFMKTCASE_MKTDAYAPPV2200_U	94		1 (0)	00:00:01
* 37	INDEX UNIQUE SCAN	XPKMKTPLAN	1		0 (0)	00:00:01
38	TABLE ACCESS BY INDEX ROWID	MKTPLAN	1	15	1 (0)	00:00:01
39	INDEX FAST FULL SCAN	XPKMKTCASEDRFORECAST	4560	50160	16 (0)	00:00:01

And with a few more operations ends up at 153

Complex Query Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	66	264 (2)	00:00:04
1	MERGE JOIN OUTER		1	66	264 (2)	00:00:04
2	MERGE JOIN OUTER		1	44	139 (3)	00:00:02
3	NESTED LOOPS OUTER		1	22	28 (4)	00:00:01
4	FAST DUAL		1		2 (0)	00:00:01
5	VIEW		1	22	26 (4)	00:00:01
6	VIEW		1	18	26 (4)	00:00:01
* 7	FILTER					
8	SORT AGGREGATE		1	80		
* 9	HASH JOIN		717	57360	26 (4)	00:00:01
* 10	HASH JOIN		67	4623	7 (15)	00:00:01
11	TABLE ACCESS FULL	MKTDRIDAILYRESAVAIL	17	629	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	MKTDRIHRLYRESAVAIL	67	2144	3 (0)	00:00:01
13	TABLE ACCESS FULL	MKTHOUR	1752	19272	19 (0)	00:00:01
14	VIEW		1	22	110 (1)	00:00:02
15	VIEW		1	9	110 (1)	00:00:02
* 16	FILTER					
17	SORT AGGREGATE		1	19		
18	MERGE JOIN		365	6935	110 (1)	00:00:02
* 19	TABLE ACCESS BY INDEX ROWID	MKTSTUDYMODE	2	18	4 (0)	00:00:01
20	INDEX FULL SCAN	XPKMKTSTUDYMODE	16		1 (0)	00:00:01
* 21	SORT JOIN		1318	13180	106 (1)	00:00:02
* 22	TABLE ACCESS FULL	MKTPLAN	1318	13180	105 (0)	00:00:02
23	BUFFER SORT		1	22	153 (1)	00:00:02
24	VIEW		1	22	125 (0)	00:00:02
25	VIEW		1	9	125 (0)	00:00:02
* 26	FILTER					
27	SORT AGGREGATE		1	54		
* 28	HASH JOIN		99	5346	125 (0)	00:00:02
29	NESTED LOOPS					
30	NESTED LOOPS		81	3483	109 (0)	00:00:02
31	NESTED LOOPS		94	2632	27 (0)	00:00:01
32	VIEW		1	9	2 (0)	00:00:01
* 33	COUNT STOPKEY					
34	FAST DUAL		1		2 (0)	00:00:01
35	TABLE ACCESS BY INDEX ROWID	MKTCASE	94	1786	25 (0)	00:00:01
* 36	INDEX RANGE SCAN	XFMKTCASE_MKTDAYAPPV2200_U	94		1 (0)	00:00:01
* 37	INDEX UNIQUE SCAN	XPKMKTPLAN	1		0 (0)	00:00:01
38	TABLE ACCESS BY INDEX ROWID	MKTPLAN	1	15	1 (0)	00:00:01
39	INDEX FAST FULL SCAN	XPKMKTCASEDRFORECAST	4560	50160	16 (0)	00:00:01

106+4 = 110

Complex Query Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	66	264 (2)	00:00:04
1	MERGE JOIN OUTER		1	66	264 (2)	00:00:04
2	MERGE JOIN OUTER		1	44	139 (3)	00:00:02
3	NESTED LOOPS OUTER		1	22	28 (4)	00:00:01
4	FAST DUAL		1		2 (0)	00:00:01
5	VIEW		1	22	26 (4)	00:00:01
6	VIEW		1	18	26 (4)	00:00:01
* 7	FILTER					
8	SORT AGGREGATE		1	80		
* 9	HASH JOIN		717	57360	26 (4)	00:00:01
* 10	HASH JOIN		67	4623	7 (15)	00:00:01
11	TABLE ACCESS FULL	MKTDRIDAILYRESAVAIL	17	629	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	MKTDRIHRLYRESAVAIL	67	2144	3 (0)	00:00:01
13	TABLE ACCESS FULL	MKTHOUR	1752	19272	19 (0)	00:00:01
14	VIEW		1	22	110 (1)	00:00:02
15	VIEW		1	9	110 (1)	00:00:02
* 16	FILTER					
17	SORT AGGREGATE		1	19		
18	MERGE JOIN		365	6935	110 (1)	00:00:02
* 19	TABLE ACCESS BY INDEX ROWID	MKTSTUDYMODE	2	18	4 (0)	00:00:01
20	INDEX FULL SCAN	XPKMKTSTUDYMODE	16		1 (0)	00:00:01
* 21	SORT JOIN		1318	13180	106 (1)	00:00:02
* 22	TABLE ACCESS FULL	MKTPLAN	1318	13180	105 (0)	00:00:02
23	BUFFER SORT		1	22	153 (1)	00:00:02
24	VIEW		1	22	125 (0)	00:00:02
25	VIEW		1	9	125 (0)	00:00:02
* 26	FILTER					
27	SORT AGGREGATE		1	54		
* 28	HASH JOIN		99	5346	125 (0)	00:00:02
29	NESTED LOOPS					
30	NESTED LOOPS		81	3483	109 (0)	00:00:02
31	NESTED LOOPS		94	2632	27 (0)	00:00:01
32	VIEW		1	9	2 (0)	00:00:01
* 33	COUNT STOPKEY					
34	FAST DUAL		1		2 (0)	00:00:01
35	TABLE ACCESS BY INDEX ROWID	MKTCASE	94	1786	25 (0)	00:00:01
* 36	INDEX RANGE SCAN	XFMKTCASE_MKTDAYAPPV2200_U	94		1 (0)	00:00:01
* 37	INDEX UNIQUE SCAN	XPKMKTPLAN	1		0 (0)	00:00:01
38	TABLE ACCESS BY INDEX ROWID	MKTPLAN	1	15	1 (0)	00:00:01
39	INDEX FAST FULL SCAN	XPKMKTCASEDRFORECAST	4560	50160	16 (0)	00:00:01

A few more operations but still at 110

Complex Query Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	66	264 (2)	00:00:04
1	MERGE JOIN OUTER		1	66	264 (2)	00:00:04
2	MERGE JOIN OUTER		1	44	139 (3)	00:00:02
3	NESTED LOOPS OUTER		1	22	28 (4)	00:00:01
4	FAST DUAL		1		2 (0)	00:00:01
5	VIEW		1	22	26 (4)	00:00:01
6	VIEW		1	18	26 (4)	00:00:01
* 7	FILTER					
8	SORT AGGREGATE		1	80		
* 9	HASH JOIN		717	57360	26 (4)	00:00:01
* 10	HASH JOIN		67	4623	7 (15)	00:00:01
11	TABLE ACCESS FULL	MKTDRIDAILYRESAVAIL	17	629	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	MKTDRIHRLYRESAVAIL	67	2144	3 (0)	00:00:01
13	TABLE ACCESS FULL	MKTHOUR	1752	19272	19 (0)	00:00:01
14	VIEW		1	22	110 (1)	00:00:02
15	VIEW		1	9	110 (1)	00:00:02
* 16	FILTER					
17	SORT AGGREGATE		1	19		
18	MERGE JOIN		365	6935	110 (1)	00:00:02
* 19	TABLE ACCESS BY INDEX ROWID	MKTSTUDYMODE	2	18	4 (0)	00:00:01
20	INDEX FULL SCAN	XPKMKTSTUDYMODE	16		1 (0)	00:00:01
* 21	SORT JOIN		1318	13180	106 (1)	00:00:02
* 22	TABLE ACCESS FULL	MKTPLAN	1318	13180	105 (0)	00:00:02
23	BUFFER SORT		1	22	153 (1)	00:00:02
24	VIEW		1	22	125 (0)	00:00:02
25	VIEW		1	9	125 (0)	00:00:02
* 26	FILTER					
27	SORT AGGREGATE		1	54		
* 28	HASH JOIN		99	5346	125 (0)	00:00:02
29	NESTED LOOPS					
30	NESTED LOOPS		81	3483	109 (0)	00:00:02
31	NESTED LOOPS		94	2632	27 (0)	00:00:01
32	VIEW		1	9	2 (0)	00:00:01
* 33	COUNT STOPKEY					
34	FAST DUAL		1		2 (0)	00:00:01
35	TABLE ACCESS BY INDEX ROWID	MKTCASE	94	1786	25 (0)	00:00:01
* 36	INDEX RANGE SCAN	XFMKTCASE_MKTDAYAPPV2200_U	94		1 (0)	00:00:01
* 37	INDEX UNIQUE SCAN	XPKMKTPLAN	1		0 (0)	00:00:01
38	TABLE ACCESS BY INDEX ROWID	MKTPLAN	1	15	1 (0)	00:00:01
39	INDEX FAST FULL SCAN	XPKMKTCASEDRFORECAST	4560	50160	16 (0)	00:00:01

19+7 = 26 and 26+2 = 28

Complex Query Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	66	264 (2)	00:00:04
1	MERGE JOIN OUTER		1	66	264 (2)	00:00:04
2	MERGE JOIN OUTER		1	44	139 (3)	00:00:02
3	NESTED LOOPS OUTER		1	22	28 (4)	00:00:01
4	FAST DUAL		1		2 (0)	00:00:01
5	VIEW		1	22	26 (4)	00:00:01
6	VIEW		1	18	26 (4)	00:00:01
* 7	FILTER					
8	SORT AGGREGATE		1	80		
* 9	HASH JOIN		717	57360	26 (4)	00:00:01
* 10	HASH JOIN		67	4623	7 (15)	00:00:01
11	TABLE ACCESS FULL	MKTDRIDAILYRESAVAIL	17	629	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	MKTDRIHRLYRESAVAIL	67	2144	3 (0)	00:00:01
13	TABLE ACCESS FULL	MKTHOUR	1752	19272	19 (0)	00:00:01
14	VIEW		1	22	110 (1)	00:00:02
15	VIEW		1	9	110 (1)	00:00:02
* 16	FILTER					
17	SORT AGGREGATE		1	19		
18	MERGE JOIN		365	6935	110 (1)	00:00:02
* 19	TABLE ACCESS BY INDEX ROWID	MKTSTUDYMODE	2	18	4 (0)	00:00:01
20	INDEX FULL SCAN	XPKMKTSTUDYMODE	16		1 (0)	00:00:01
* 21	SORT JOIN		1318	13180	106 (1)	00:00:02
* 22	TABLE ACCESS FULL	MKTPLAN	1318	13180	105 (0)	00:00:02
23	BUFFER SORT		1	22	153 (1)	00:00:02
24	VIEW		1	22	125 (0)	00:00:02
25	VIEW		1	9	125 (0)	00:00:02
* 26	FILTER					
27	SORT AGGREGATE		1	54		
* 28	HASH JOIN		99	5346	125 (0)	00:00:02
29	NESTED LOOPS					
30	NESTED LOOPS		81	3483	109 (0)	00:00:02
31	NESTED LOOPS		94	2632	27 (0)	00:00:01
32	VIEW		1	9	2 (0)	00:00:01
* 33	COUNT STOPKEY					
34	FAST DUAL		1		2 (0)	00:00:01
35	TABLE ACCESS BY INDEX ROWID	MKTCASE	94	1786	25 (0)	00:00:01
* 36	INDEX RANGE SCAN	XFMKTCASE_MKTDAYAPPV2200_U	94		1 (0)	00:00:01
* 37	INDEX UNIQUE SCAN	XPKMKTPLAN	1		0 (0)	00:00:01
38	TABLE ACCESS BY INDEX ROWID	MKTPLAN	1	15	1 (0)	00:00:01
39	INDEX FAST FULL SCAN	XPKMKTCASEDRFORECAST	4560	50160	16 (0)	00:00:01

110 28+1 more CPU = 139

Complex Query Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	66	264 (2)	00:00:04
1	MERGE JOIN OUTER		1	66	264 (2)	00:00:04
2	MERGE JOIN OUTER		1	44	139 (3)	00:00:02
3	NESTED LOOPS OUTER		1	22	28 (4)	00:00:01
4	FAST DUAL		1		2 (0)	00:00:01
5	VIEW		1	22	26 (4)	00:00:01
6	VIEW		1	18	26 (4)	00:00:01
* 7	FILTER					
8	SORT AGGREGATE		1	80		
* 9	HASH JOIN		717	57360	26 (4)	00:00:01
* 10	HASH JOIN		67	4623	7 (15)	00:00:01
11	TABLE ACCESS FULL	MKTDRIDAILYRESAVAIL	17	629	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	MKTDRIHRLYRESAVAIL	67	2144	3 (0)	00:00:01
13	TABLE ACCESS FULL	MKTHOUR	1752	19272	19 (0)	00:00:01
14	VIEW		1	22	110 (1)	00:00:02
15	VIEW		1	9	110 (1)	00:00:02
* 16	FILTER					
17	SORT AGGREGATE		1	19		
18	MERGE JOIN		365	6935	110 (1)	00:00:02
* 19	TABLE ACCESS BY INDEX ROWID	MKTSTUDYMODE	2	18	4 (0)	00:00:01
20	INDEX FULL SCAN	XPKMKTSTUDYMODE	16		1 (0)	00:00:01
* 21	SORT JOIN		1318	13180	106 (1)	00:00:02
* 22	TABLE ACCESS FULL	MKTPLAN	1318	13180	105 (0)	00:00:02
23	BUFFER SORT		1	22	153 (1)	00:00:02
24	VIEW		1	22	125 (0)	00:00:02
25	VIEW		1	9	125 (0)	00:00:02
* 26	FILTER					
27	SORT AGGREGATE		1	54		
* 28	HASH JOIN		99	5346	125 (0)	00:00:02
29	NESTED LOOPS					
30	NESTED LOOPS		81	3483	109 (0)	00:00:02
31	NESTED LOOPS		94	2632	27 (0)	00:00:01
32	VIEW		1	9	2 (0)	00:00:01
* 33	COUNT STOPKEY					
34	FAST DUAL		1		2 (0)	00:00:01
35	TABLE ACCESS BY INDEX ROWID	MKTCASE	94	1786	25 (0)	00:00:01
* 36	INDEX RANGE SCAN	XFMKTCASE_MKTDAYAPPV2200_U	94		1 (0)	00:00:01
* 37	INDEX UNIQUE SCAN	XPKMKTPLAN	1		0 (0)	00:00:01
38	TABLE ACCESS BY INDEX ROWID	MKTPLAN	1	15	1 (0)	00:00:01
39	INDEX FAST FULL SCAN	XPKMKTCASEDRFORECAST	4560	50160	16 (0)	00:00:01

125v+139 = 264