CHAPTER 10

TOPIC OBJECTIVES

- **Using Explain Plan**
- **Optimizer Rules**
- **Tuning SQL Statements**

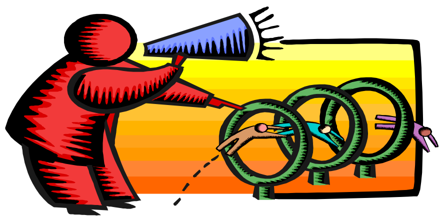
EXPLAIN PLAN

What is going on with my SQL Statements?

The EXPLAIN PLAN diagnostic tool allows you to observe how ORACLE executes SQL statements. The EXPLAIN PLAN statement displays the execution plan selected by the OPTIMIZER for SELECT, INSERT, UPDATE, and DELETE statements. Before you can use this performance diagnostic tool you must:

- Create a plan table
- Understand what information the optimizer will load into this table
- Be able to maintain and synchronize the rows in the plan table

THE PLAN_TABLE COLUMNS(2)



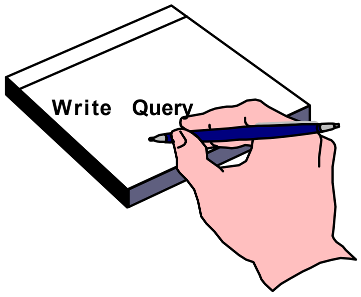
Three new columns to use

There are new columns in the PLAN-TABLE that help estimate resource cost of statements.

- CPU_COST - This column holds numbers that represent the CPU cost of the statement. This column will be NULL if the rule-based optimizer is used.
- IO_COST - This column holds numbers that represent the disk I/O of the statement. This column will be NULL if the rule-based optimizer is used.
- TEMP_SPACE - This column holds numbers that represent the size, in bytes, of any temporary space the statements might require. If the rule-based optimizer is used or if the statement does not require any temporary space, this column will be NULL.

To use these columns make sure you have generated statistics on the schema objects you are evaluating.

EXPLAIN PLAN SYNTAX



When using EXPLAIN PLAN, the actual SQL statement is not executed. Instead, a plan tree is inserted into the PLAN_TABLE. A query must then be written against the PLAN_TABLE to interpret the results.

The identification and informational columns of the PLAN_TABLE are:

COLUMN	DESCRIPTION
STMT DESCRIPTION	A description of the statement
TABLE_NAME	The table in which the output from EXPLAIN PLAN is stored. The default is PLAN_TABLE.
STATEMENT_ID	The statement description
TIMESTAMP	The date and time EXPLAIN PLAN was executed.
REMARKS	Comments about the EXPLAIN results.
OPERATION	The operation performed in this step.
OPTIONS	Options about the operation.
OBJECT_NODE	The database link used to reference an object.
OBJECT_OWNER	The user that owns the table or view
OBJECT_NAME	Database, table, or index name
OBJECT_INSTANCE	The position of the object as it occurred in the statement. Numbering is from left to right, outer to inner. View expansion results in unpredictable numbers.
OBJECT_TYPE	Description of the object
SEARCH_COLUMNS	Not currently used.
ID	The user assigned id number for this statement and run
PARENT_ID	The id of the next execution step that operates on the output of the ID step
POSITION	Order of processing for steps have the same PARENT_ID
COST	Places a COST NUMBER in this column if the cost-based optimizer was used.
OTHER	Other information for this run, such as the query for remote node execution.

OPERATIONS IDENTIFIED BY THE EXPLAIN PLAN

Operations column of the PLAN_TABLE	Description
SELECT STATEMENT	
UPDATE STATEMENT	
DELETE STATEMENT	
INSERT STATEMENT	
AND-EQUAL	Used for WHERE clause equalities. Each comparison includes a non-unique indexed column from which ROWIDs are obtained and intersected. Indexes are “MERGED”.
CONNECT BY	Retrieval based on a tree-walk
CONCATENATION	UNION ALL for a group of tables (ORs rule)
COUNT	A count operation
FILTER	A restriction of the rows returned
FIRST ROW	A retrieval of only the first row of a query
FOR UPDATE	A retrieval that row locks on selected rows
INDEX	A retrieval from an index Options: UNIQUE SCAN index search for unique values RANGE SCAN index search for range between
INTERSECTION	A retrieval of rows common to two tables. The tables are sorted first.
MERGE JOIN	A join formed by merging two sorted sets of data Options: OUTER -- an outer join MINUS -- a retrieval of rows in table 1 but not in table 2
NESTED LOOPS	A join on two child operations (each row returned by the first child operation is then operated on by the second child operation). One of three methods used to join tables. The other two are SORT MERGE and INDEX CLUSTER
PROJECTION	Selecting a subset of the columns
REMOTE	Retrieval across SQL*NET
SEQUENCE	An operation involving the sequence generator
SORT	A retrieval of rows ordered by one or more columns
SORT UNIQUE	UNIQUE -- sort to produce unique values
SORT GROUP BY	GROUP BY -- sort for grouping
SORT JOIN	JOIN -- sort for merge join
SORT ORDER BY	ORDER BY -- sort for order by

EXPLAIN PLAN TABLE COLUMNS

Operations Column of the PLAN_TABLE	Description
TABLE ACCESS	A retrieval from a base table
TABLE ACCESS BY ROWID	BY ROWID table access by ROWID
TABLE ACCESS FULL	FULL able access by Full Table Scan
	CLUSTER table access by Cluster key
UNION	A retrieval of unique rows from two tables UNIONed together dropping the duplicates
VIEW	A retrieval from a view of a table
OPTIMIZER	Choose, RULE, FIRST ROWS, ALL_ROWS

In order to load the PLAN_TABLE with rows concerning a particular SQL statement you would execute the following SQL: Review and execute the scripts big2.sql and big3.sql in order to run the following explain plan selects. ORACLE> @big2 @big3

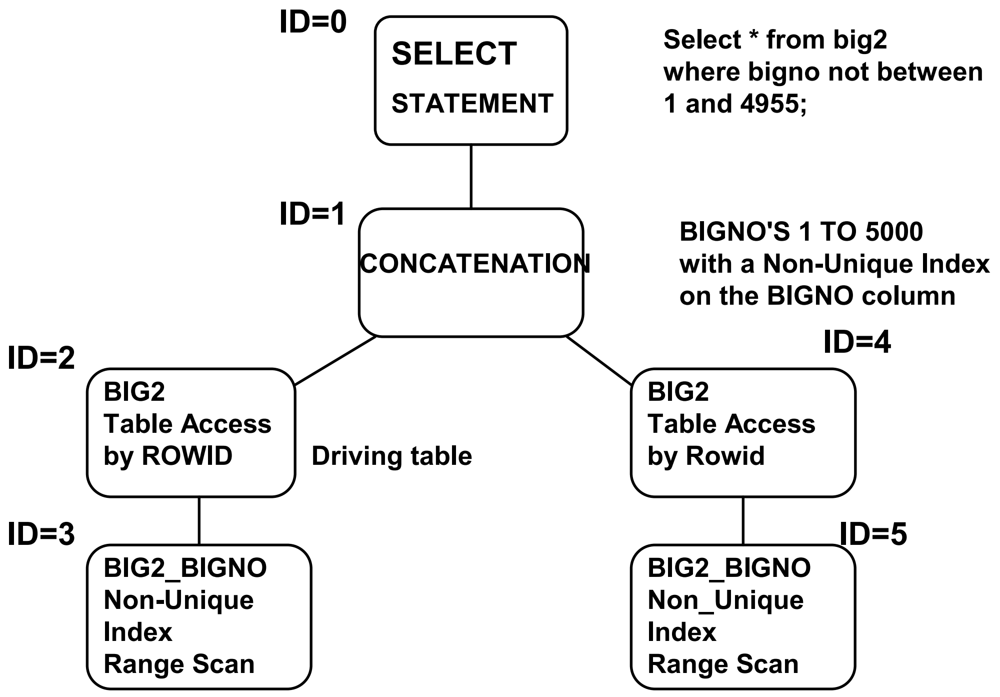
```
EXPLAIN PLAN
        [SET STATEMENT_ID = ‘STMT DESCRIPTION’]
        [INTO table_name]
        FOR   SQL_STATEMENT;
```

EXAMPLES:

```
SQL>  EXPLAIN PLAN FOR
      SELECT *
      FROM BIG2
      WHERE BIGNO NOT BETWEEN 1 AND 4955;
```

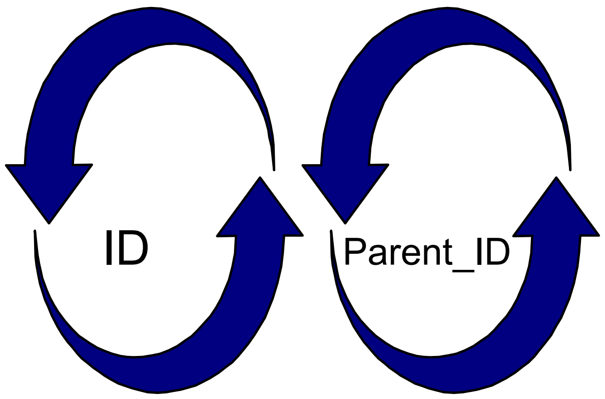
To see the plan results you would execute dbms_xplan(display)

PLAN_TREE ANALYSIS


ID=0
SELECT STATEMENT
Select * from big2 where bigno not between 1 and 4955;
ID=1
CONCATENATION
BIGNO'S 1 TO 5000 with a Non-Unique Index on the BIGNO column
ID=2
BIG2 Table Access by ROWID
Driving table
ID=4
BIG2 Table Access by Rowid
ID=3
BIG2_BIGNO Non-Unique Index Range Scan
ID=5
BIG2_BIGNO Non_Unique Index Range Scan

```
select id, parent_id,object_name,object_type,operation,options
from plan_table  order by id;   OR
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY)

COL   OBJECT_NAME     FORMAT A12
COL   OBJECT_TYPE     FORMAT A12
COL   OPERATION       FORMAT A12
COL   OPTIONS         FORMAT A12

ID    PARENT_ID OBJECT_NAME  OBJECT_TYPE  OPERATION    OPTIONS
---   --------- -----------  -----------  ------------ --------
0                                         Select statement
1         0                               Concatenation
2         1     BIG2                      Table Access  By ROWID
3         2     BIG2_BIGNO   Non-Unique   Index         RangeScan
4         1     BIG2                      Table Access  By ROWID
5         4     BIG2_BIGNO   Non-Unique   Index         RangeScan
```

NESTED OUTPUT USING “START WITH” AND “CONNECT BY”



```
ORACLE>  COL “Query Plan”  format a80

ORACLE>  select   LPAD (‘  ‘, 2*(LEVEL-1))  ||
                  OPERATION  ||      ‘  ‘      ||
                  OPTIONS     ||  ‘  ‘   ||
                  OBJECT_NAME  ||   ‘  ‘      ||
                  DECODE  (ID, 0, ‘COST =  ‘  ||  POSITION)
                  “ Query Plan”
             from PLAN_TABLE
             START WITH ID = 0
             CONNECT BY PRIOR ID = PARENT_ID;
```

See explain.sql or the dbms_xplan command seeing the above code.

```
Query Plan
-----------------------------------------------------------------------------------------------------
Select  Statement                    COST=1
        Concatenation
                Table access by ROWID BIG2
                        Index Range Scan Big2_BIGNO
                Table access by ROWID BIG2
                        Index Range Scan BIG2_BIGNO
```

Read the output from the Explain Plan query by reading from the inside out and top bottom in a clockwise fashion.

- First, read the most indented clauses. If there is a tie, read from the top to the bottom

EXPLAIN PLAN OPERATION DEFINITIONS

Operation	Option	Definition
TABLE ACCESS	FULL	FULL TABLE SCAN - one row is read at a time until all rows are read.
	CLUSTER	Table scan on an Index Cluster Table Key
	HASH	A Hash Key is matched against a matching Hash value
	BY ROWID	Accesses a row by its ROWID usually from an Index
	UNIQUE SCAN	A single row is returned from a unique index
	FULL	A full scan of the index is performed. Because the index is in sorted order. It should be better than a full table scan and sort.
	FAST FULL	The rows do not have to be returned in sorted order. Utilizes multiblock I/O and can be run in parallel. All the blocks of the index are scanned
	RANGE SCAN	A range of values is returned from an index. Used with operators like <>, LIKE% or _, or BETWEEN.
	RANGE SCAN desc	A range scan in descending order
	AND EQUAL	Up to fine indexes can be merged. Each index is processed with the results merged with the results of another index.
UNION ALL		A set operation used for UNION ALL. Returns all rows from both queries. Does not eliminate duplicates or sort.
VIEW		A VIEW is a SET operation. A view will be evaluated and processed first, then merged with the rest of the query
HASH JOIN		One of the tables has in memory a bitmap created and a hashing function is used to find the matching rows in the second table.
MERGE JOIN		Tables are sorted and then merged
NESTED LOOPS		One of the tables is read one row at a time and a match is produced by accessing the second table. Usually, the second table has an index.
OUTER JOIN		With an OUTER JOIN, one of the tables is a complete table. All rows from that table will be returned regardless if there is a match. Performs a FULL TABLE SCAN.
SEQUENCE		when the NEXTVAL or CURRVAL is processed against a sequence number.

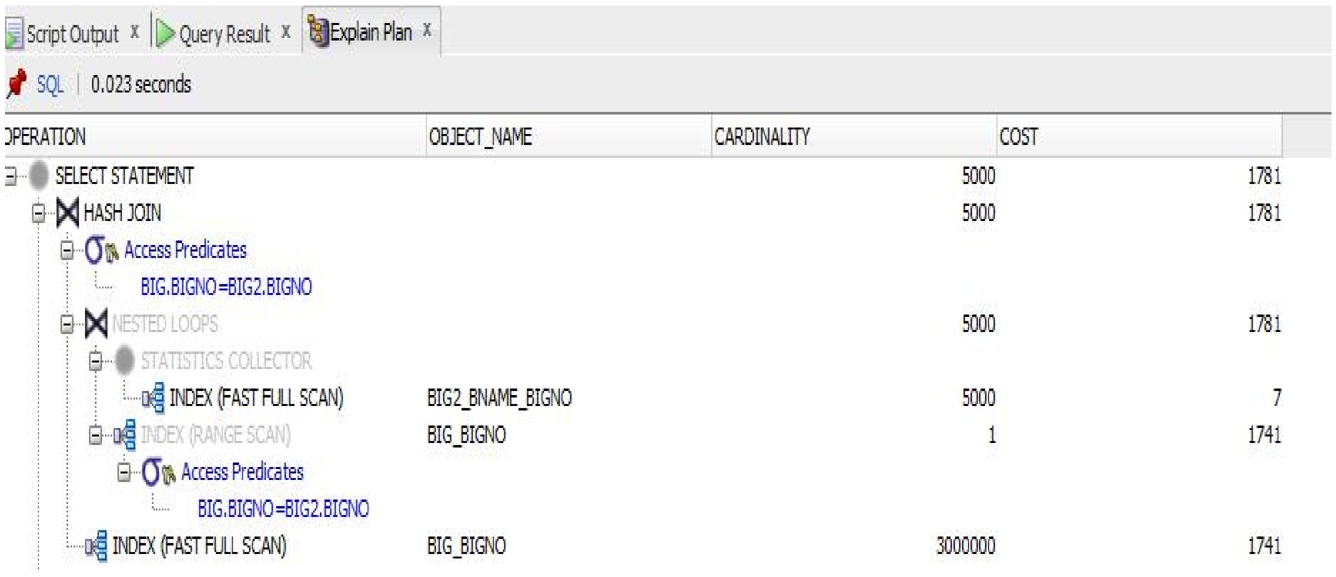
EXPLAIN PLAN OPERATIONS DEFINITIONS - PART 2

Operation	Option	Definition
SORT AGGREGATE		Used with aggregate functions such as SUM, MIN, MAX, COUNT, and AVG
SORT ORDER BY		Used to sort rows with the ORDER BY clause
CONCATENATION		A row operation that will combine the results of an IN clause.
FOR UPDATE		places a lock on the rows covered by the Select statement. This prevents updates against these rows while the query is running.
FILTER		restricts rows by a where clause i.e. WHERE EMPNO = 7900
INTERSECTION		A set operation used for INTERSECT Returns rows common to both queries.
MINUS		A set operation used for MINUS. Returns rows that are unique to the first query.
UNION		A set operation used for UNION. Returns all rows that are unique to both queries. Performs a sort.

EXPLAIN PLAN EXAMPLES

The following examples shows the plan for a two-table join where both sides of the WHERE clause are indexed:

```
      DELETE FROM PLAN_TABLE;
ORACLE>    alter session set OPTIMIZER_MODE = all_rows;
           (In this example choose and rule produce the same results)
ORACLE>    EXPLAIN PLAN
           FOR
           SELECT BIG.BIGNO, BIG2.BNAME
           FROM BIG,
           BIG2
           WHERE BIG.BIGNO = BIG2.BIGNO;

ORACLE>  COL "Query Plan"  format a80

ORACLE> SELECT * FROM TABLE(dbms_xplan.display);
Query Plan
------------------------------------------------------------------------------------------------
Select  Statement                    COST=23
         NESTED LOOPS
                  Table  access      FULL   BIG2
                          Index Range Scan BIG_BIGNO
```



Script Output | Query Result | Explain Plan

SQL | 0.023 seconds

```
OPERATION                          OBJECT_NAME        CARDINALITY      COST
SELECT STATEMENT                                             5000      1781
  HASH JOIN                                                  5000      1781
    Access Predicates
      BIG.BIGNO=BIG2.BIGNO
    NESTED LOOPS                                             5000      1781
      STATISTICS COLLECTOR
        INDEX (FAST FULL SCAN)     BIG2_BNAME_BIGNO          5000         7
      INDEX (RANGE SCAN)           BIG_BIGNO                    1      1741
        Access Predicates
          BIG.BIGNO=BIG2.BIGNO
    INDEX (FAST FULL SCAN)         BIG_BIGNO              3000000      1741
```

OPTIMIZING SQL STATEMENTS

NOT TRANSLATIONS

NOT STATEMENT	TRANSLATION
NOT >	<=
NOT >=	<
NOT <	>=
NOT <=	>
NOT BETWEEN	> larger number or < smallest number

The above chart indicates the replacements which can be used instead of the NOT statement. Some reasons for avoiding the NOT statement are:

- The use of !=, NOT=, <>, or NOT LIKE causes ORACLE not to use an index for that portion of the query
- The optimizer assumes that queries containing NOT usually retrieve more rows than they skip. So ORACLE will avoid using an Index.
- When NOT is used with mathematical operators other than =, ORACLE transforms it from a NOT so that indexes may be used.

The set operators UNION, MINUS and INTERSECT can usually be used to avoid NOT =.

NOTE: The query path is a result of weighing all search criteria paths against each other using the cost and determining the table which will be the driving table. Every AND is considered separately.

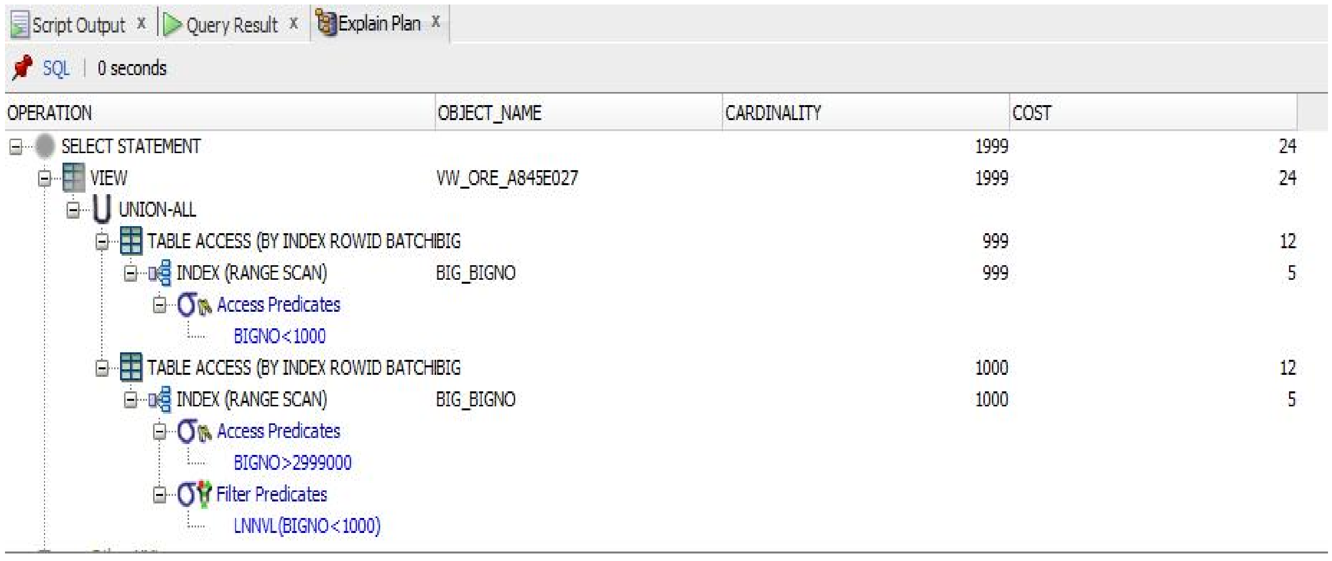
OPTIMIZING NOT BETWEEN

```
ORACLE>    ALTER SESSION SET OPTIMIZER_MODE = ALL_ROWS;
```

(In this example FIRST_ROWS or ALL_ROWS PRODUCE THE SAME RESULTS)

```
ORACLE>    EXPLAIN PLAN
           set statement_id =  'abc'
           FOR
           SELECT BNAME      /* INDEX ON BIGNO  */
           FROM BIG          /* 2% OF THE ROWS RETURNED   */
           WHERE BIGNO NOT BETWEEN 1000 AND 2999000;

ORACLE>  COL “Query Plan”  format a80

ORACLE>  SELECT * FROM
TABLE(dbms_xplan.display('PLAN_TABLE','abc')
ORACLE>  SAVE explain

Query Plan
----------------------------------------------------------------------------------------------------------
Select  Statement                    COST=161
        CONCATENATION
                Table  access BY ROWID   BIG
                        Index Range Scan BIG_BIGNO
                Table  access BY ROWID   BIG
                        Index Range Scan BIG_BIGNO
```

Or



OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1999	24
VIEW	VW_ORE_A845E027	1999	24
UNION-ALL			
TABLE ACCESS (BY INDEX ROWID BATCH	BIG	999	12
INDEX (RANGE SCAN)	BIG_BIGNO	999	5
Access Predicates			
BIGNO<1000			
TABLE ACCESS (BY INDEX ROWID BATCH	BIG	1000	12
INDEX (RANGE SCAN)	BIG_BIGNO	1000	5
Access Predicates			
BIGNO>2999000			
Filter Predicates			
LNNVL(BIGNO<1000)			

OPTIMIZING NOT EQUAL TO

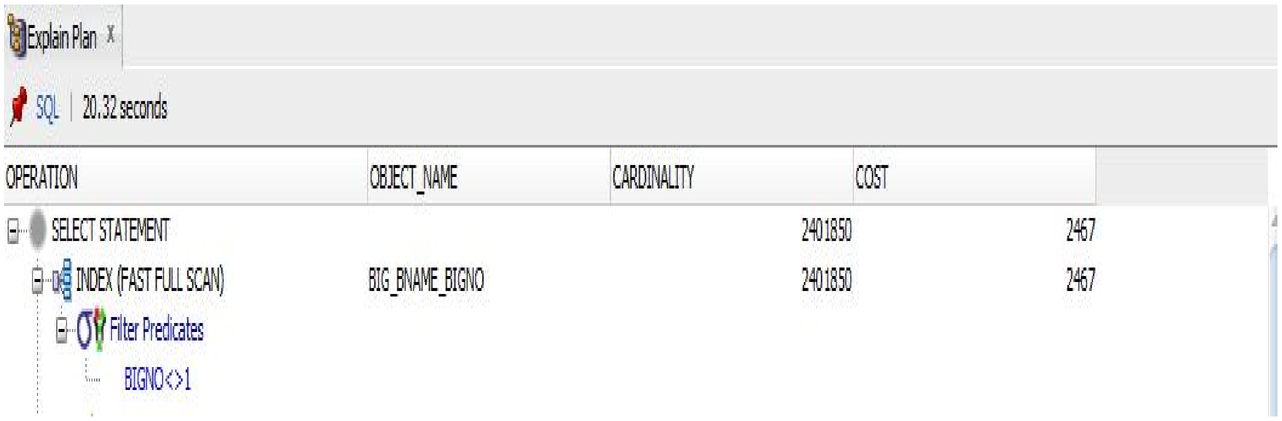
ORACLE> ALTER SESSION SET OPTIMIZER_MODE = ALL_ROWS;
(The use of FIRST_ROWS OR FIRST_ROWS_N produce the same results)

```
ORACLE>    EXPLAIN PLAN
           set statement_id='abc2'
           FOR
           SELECT BNAME    /*  INDEX ON BIGNO  */
           FROM BIG
           WHERE BIGNO <> 1;

     ORACLE>    @explain or

SELECT * FROM

TABLE(dbms_xplan.display('PLAN_TABLE','abc2'))
```

Query Plan

```
------------------------------------------------------------------------------------------------------------
Select  Statement                    COST=317
        TABLE        ACCESS          FULL  BIG
```

or



Explain Plan

SQL | 20.32 seconds

```
OPERATION                        OBJECT_NAME      CARDINALITY   COST
SELECT STATEMENT                                  2401850       2467
  INDEX (FAST FULL SCAN)         BIG_BNAME_BIGNO  2401850       2467
    Filter Predicates
      BIGNO<>1
```

ELIMINATING SQL STATEMENTS

Why is my SQL code running so long?

Use a single SQL statement rather than multiple statements whenever possible.

BAD:

```
UPDATE emp
set ename = ‘MOE’
where empno = 3;

UPDATE emp
set sal = 4000
where empno = 3;
```

GOOD:

```
UPDATE emp
set ename = ‘MOE’, SAL = 4000
where empno = 3;
```

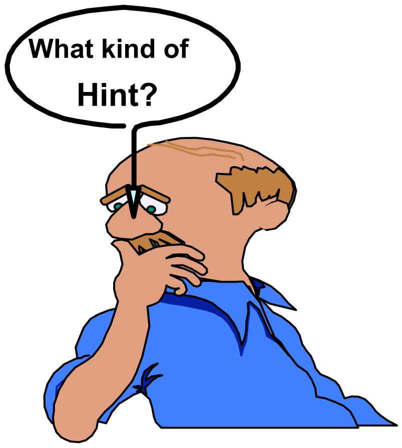
BAD:

```
SELECT * FROM EMP WHERE DEPTNO = 10;
SELECT * FROM EMP WHERE DEPTNO = 20;
SELECT * FROM EMP WHERE DEPTNO = 30;
```

GOOD:

```
SELECT * FROM EMP WHERE DEPTNO IN (10,20,30);
```

USING HINTS TO CONTROL THE DRIVING TABLE



Listed below are two methods of sending hints to the ORACLE optimizer:

Method 1: (Recommended)

```
        ORACLE>    SELECT   /*+  ORDERED  */ ENAME,LOC
                   FROM EMP, DEPT
                   WHERE EMP.DEPTNO = DEPT.DEPTNO;
                /* EMP IS THE DRIVING TABLE  */
```

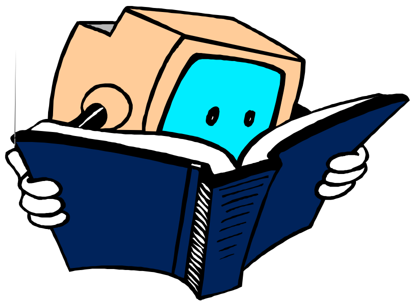
Method 2:

```
        ORACLE>    SELECT /*+  USE_NL(DEPT,EMP)  */ ENAME, LOC
                   FROM DEPT, EMP
                   WHERE EMP.DEPTNO = DEPT.DEPTNO;
                --   DEPT IS THE DRIVING TABLE
```

HINTS:

• FULL	• ROWID	• CLUSTER
• HASH	• INDEX	• INDEX
• INDEX_ASC	• INDEX_DESC	• INDEX_MERGE
• ORDERED	• USE_MERGE	• USE_NL

Lab 9- using the optimizer and explain plan



Create the BIG and BIG2 tables and indexes using the script big1.sql and big2.sql Scripts. Given a non-unique index BIG_BIGNO on the BIGNO column of BIG, and a non-unique index BIG_BNAME on the BNAME column of the BIG table, use EXPLAIN PLAN to determine how ORACLE would process the following utilizing ALL_ROWS and FIRST_ROWS.

```
                    EXPLAIN PLAN FOR
1.    ORACLE>       Select * from big where bigno > 1;
              SELECT * FROM TABLE(dbms_xplan.display
```

HINT: use AUTOTRACE to help provide the information.

```
      ORACLE>       delete from plan_table;
2.    ORACLE>       explain plan for
                    Select * from BIG where bname like 'BET%';
      SQL> select * from table (dbms_xplan.display)
```

3. Using the Optimization type of FIRST_ROWS then ALL_ROWS do:

```
      ORACLE>       delete from plan_table;

      ORACLE>       explain plan for Select MAX(BIGNO) from big;
              SELECT * FROM TABLE(dbms_xplan.display
```

4. Using the Optimization type of ALL_ROWS then FIRST_ROWS do:

```
      ORACLE>       delete from plan_table;
       ORACLE>      explain plan for
                    Select * from big where bigno IN (500, 99500);
              SELECT * FROM TABLE(dbms_xplan.display
```

5. Using the Optimization type of ALL_ROWS and FIRST_ROWS_1000 do:

```
ORACLE>    delete from plan_table;

ORACLE>    SELECT * FROM BIG WHERE BIGNO = 1000 AND
           BNAME = 'ONE THOUSAND';
```

Lab 9- continued

Given non-unique indexes on all columns of the BIG2 table plus a concatenated non-unique index on the BNAME and BIGNO columns of the BIG2 table, use EXPLAIN PLAN to determine how ORACLE10G would execute the following queries:

6. Using the Optimization type of ALL_ROWS and FIRST_ROWS do:

```
ORACLE>    delete from plan_table;

ORACLE>    Select bigno from big2 where bname = 'ONE THOUSAND';
```

7. Using the Optimization type of ALL_ROWS do:

```
ORACLE>    delete from plan_table;

ORACLE>    select bigno, bname from big2
           where lower(BNAME) = 'one thousand';
```

8. Using the Optimization type of ALL_ROWS do:

```
ORACLE>    delete from plan_table;

ORACLE>    select bigno, bname
           from big2
           where bname = ' ONE THOUSAND' and bigno = 1;
```

9. Using the Optimization type of ALL_ROWS and FIRST_ROWS do:

```
ORACLE>    delete from plan_table;

ORACLE>    select big.bigno, big2.bname
           from big, big2
           where big.bigno = ABS(big2.bigno + 0);
```

(THE DRIVING TABLE IS: ______________________)

10. Using the Optimization type of ALL_ROWS do:

```
ORACLE>   delete from plan_table;

ORACLE>   select big.bigno, big2.bnam
          from big, big2
          where ABS(big.bigno) + 0 = big2.bigno;
```

(THE DRIVING TABLE IS: ___________________)

11. The following exercise will have you alter you session setting the optimizer to the type of ALL_ROWS and than perform a series of EXPLAIN PLANs for the same SQL statement except for the value being searched. The main purpose of this is to demonstrate when it is more efficient for ORACLE to perform a full table scan versus using an Index on the BIGNO column. For each of these queries identify the

    Cost   Table Access   %Rows Returned

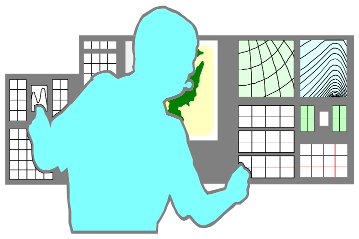
```
1.   ORACLE>   Use the ALL_ROWS Hint;
     ORACLE>   delete from plan_table;

2.   ORACLE>   SELECT * FROM BIG
               WHERE BIGNO > 2999999;

3.   ORACLE>   SELECT * FROM BIG
               WHERE BIGNO > 2995000;
     ORACLE>   delete from plan_table;

4.   ORACLE>   SELECT * FROM BIG
               WHERE BIGNO > 2980000;
     ORACLE>   delete from plan_table;

5.   ORACLE>   SELECT * FROM BIG
               WHERE BIGNO > 2971000;
     ORACLE>   delete from plan_table;

6.   ORACLE>   SELECT * FROM BIG
               WHERE BIGNO > 2960161;

     ORACLE>   delete from plan_table;
```

7. ORACLE>

```
SELECT * FROM BIG
WHERE BIGNO > ?????;
```

(Where ????? identifies the point at which ORACLE uses a FULL TABLE SCAN)

PERFORMANCE DIAGNOSTIC TOOLS



- Explain Plan

  The actual SQL statement IS NOT executed. Instead, a plan tree is inserted into the PLAN_TABLE. You must run a query on the PLAN_TABLE

- SQL Trace Facility

  SQL statement is executed. Use this facility for very long-running queries. This facility records statistics for each SQL statement in a single session, storing statistics in a file with an extension of .trc (Trace Dump File). These statistics are to the nearest second for the PARSE, EXECUTE, and FETCH phase of each SQL statement.

- TKPROF (Trace Kernel Profile)

  Submit a trace dump file (.trc) as input to the TKPROF facility. this facility formats a report, and stores it on the disk with an extension of .prf unless you specify your own extension.

SQL Trace Utility

How is the SQL statement performing?

The SQL Trace Utility provides performance information on each SQL statement.
It tracks the following statistics:

- - Parse, execute, fetch counts
- - CPU Times & elapsed times
- - Physical & logical reads
- - Number of rows processed

SQL Trace utility is enabled by using INIT.ORA parameters:

- TIMED_STATISTICS=TRUE turns on CPU timings to the nearest second
- TRACE_ENABLED=TRUE to use dump files
- MAX_DUMP_FILE_SIZE=no_blocks file output for trace
- USER_DUMP_DEST=default_directory for dumps
- SQL_TRACE=FALSE (default)
  =TRUE (SQL trace facility for every SQL statement)

To activate SQL trace for a single session, enter the following command:

```
ORACLE>    ALTER SESSION SET SQL_TRACE = TRUE;

ORACLE>   ALTER SESSION SET SQL_TRACE=FALSE;
```

TKPROF (Trace Kernel Profile)

Produce a readable report

TKPROF translates a trace file into readable format, and can also generate Explain Plan output. Typing TKPROF by itself will give you a on-line help list. This command is executed from a UNIX or DOS prompt.

```
% or c:orant/bin>     TKPROF
```

TKPROF SYNTAX:

```
%TKPROF trace_file  output_file  [ SORT= sort option1...] print = n
   EXPLAIN=userid/password
```

- trace_file is the input trace file
- output_file is the output file TKPROF has formatted
- SORT sorts the statistics in the output file in sorted order
  Some sort options are

-PRSCNT	Count of times parsed
-PRSCPU	Count time during parsing
-PRSPHR	Physical reads during parsing
-EXECPUT	CPU time during execution
-EXEPHR	Physical reads during execution
-FCHPHR	Physical reads during fetch

- PRINT=n output the first n analyzed statements
- EXPLAIN run the EXPLAIN PLAN statement on all of the SQL statements in the trace. Create a Plan table and delete it when finished.

USING TKPROF

1. To use TKPROF, ensure that your session has the SQL Trace Utility set to true:

```
ORACLE>    ALTER SESSION SET SQL_TRACE TRUE;
```

2. Identify where the .trc file is. You can identify the user dump file and where it was sent to and if SQL TRACE is TRUE by executing the following script:

```
ORACLE>    select name,value   from V$PARAMETER
           where NAME = ‘user_dump_dest’;
```

```
NAME                                VALUE
----------------------------------  -------------------------------------------------
timed_statistics                    TRUE
user_dump_dest                      /u01/app/oracle/admin/orcl/udump
```

3. Execute a sql script which will generate the Trace file with the appropriate statistics.

```
ORACLE>    SELECT * FROM BIG
           WHERE BIGNO = 5000;
```

4. ALTER SESSION SET SQL_TRACE=FALSE;

5. Go to the user_dump_dest and identify the .trc file by looking at the date and time it was created. You can use File Manager for this if necessary.

6. Execute TKPROF on the appropriate .trc file.
   UNIX

```
%tkprof  %ORACLE_HOME%\rdbms\admin\log\ora_1914.trc  mytrace
```

```
C:\oracle\product\11.2.0\db_1\BIN\tkprof80 c:\oracle\rdbms\trace\ora_1914.trc mytrace
```

7. Browse the mytrace.prf file created from tkprof in your “CURRENT” directory.

```
c:\oracle\product\11.2.0\db_1\BIN>  edit mytrace.prf
```

TKPROF COLUMN DEFINITIONS

TKPROF COLUMNS

CALL	COUNT	CPU	ELAPSED	DISK	QUERY	CURRENT	ROWS

CALL	-	Parse, Execute, or Fetch
COUNT	-	Number of times a statement was parsed, executed, or fetched. Multiple data blocks may be read by each fetch call.
CPU	-	Total CPU time in seconds for all parse, execute, or fetch calls for the statement
ELAPSED	-	Total elapsed time in seconds for all parse, execute, or fetch calls for the statement
DISK	-	Total number of data blocks physically read from the data files on disk for all parse, execute, or fetch calls
QUERY	-	Total number of buffers retrieved in consistent mode for all parse, execute, or fetch calls
CURRENT	-	Total number of buffers retrieved in current mode. Buffers are often retrieved in current mode for insert, update, and delete statements
ROWS	-	Total number of rows processed by the SQL statement. The total does not include rows processed by subqueries.

NOTE: For select statements, the number of rows returned appears for the fetch step. For delete, insert, and update statements, the number of rows processed appears for the execute step.

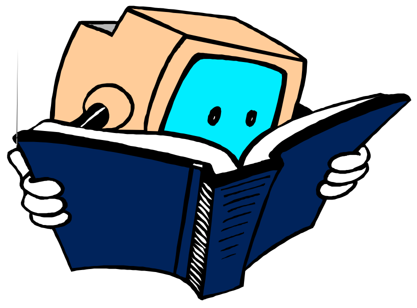
Execute phase is concerned with INSERTS, UPDATES & DELETES

Fetch phase is concerned with SELECTs

SAMPLE TKPROF REPORT

```
C:\$ORACLE_HOME\ADMIN\ORCL\TRACE>  EDIT mytrace.prf

select *
from  staff where id > 50

call      count  cpu     elapsed  disk  query current   rows
--------  -------  --------  --------- -------- -------- -------  ----------
Parse       1      0.09       0.23       6      25       1          0
Execute     1      0.00       0.00       0       0       0          0
Fetch       3      0.01       0.01       1       3       3         40

--------  -------  --------  --------- -------- -------- -------  ----------
total       5      0.10       0.24       7      28       4         40

Misses in library cache during parse: 1
Optimizer hint: CHOOSE
Parsing user id: 5

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call      count  cpu     elapsed  disk  query current   rows
--------  -------  --------  --------- -------- -------- -------  ----------
Parse      3      0.09       0.23       6      25       1          0
Execute    4      0.00       0.14       0       6       0          0
Fetch      5      0.01       0.01       1       3       3         48

Misses in library cache during parse: 1
Misses in library cache during execute: 1

    4  user  SQL statements in session.
    5  internal SQL statements in session.
   10  SQL statements in session.
********************************************************************************
Processed trace file: ora_1941.trc
       1  session in tracefile.
       4  user  SQL statements in tracefile.
       6  internal SQL statements in tracefile.
      10  SQL statements in tracefile.
       9  unique SQL statements in tracefile.
     109  lines in trace file.
```

LAB 10 - USING SQL TRACE AND TKPROF



Using SQL Trace Utility and TKPROF, compare the following queries to see which one appears to be the fastest:

```
1.      ORACLE>     SELECT BIG.BNAME, BIG2.BIGNO
                    FROM BIG, BIG2
                    WHERE BIG.BIGNO = BIG2.BIGNO AND
                    BIG2.BNAME = ‘ONE THOUSAND’;


2.      ORACLE>     SELECT BIG.BNAME
                    FROM BIG
                    WHERE BIGNO IN (SELECT BIGNO FROM
                                    BIG2
                                    WHERE BNAME = ‘ONE THOUSAND’);


3.      ORACLE>     SELECT BIG.BNAME
                    FROM BIG
                    WHERE 0 <  (SELECT COUNT(*)
                               FROM BIG2
                               WHERE BIG2.BNAME = ‘ONE THOUSAND’
                               AND BIG.BIGNO = BIG2.BIGNO);
```

LAB 10 - USING SQL TRACE AND TKPROF (CONTINUED)

4. Fill in the formatted report from TKPROF for the first SQL statement:

   c:\tkprof c:\orant\rdbms80\admin\TRACE\ora______.trc   lab84
   explain=***userid/password***

lab84.prf

****

select big.bname, big2.bigno
from big, big2
where big.bigno = big2.bigno
and big2.bname = 'ONE THOUSAND';

TKPROF COLUMNS

CALL	COUNT	CPU	ELAPSED	DISK	QUERY	CURRENT	ROWS
Parse							
Execute							
Fetch							

Misses in library cache during parse: _______________
Parsing user id: ______userid __________

ROWS	EXECUTION PLAN

LAB 10 - USING SQL TRACE AND TKPROF (CONTINUED)

5. Fill in the formatted report from TKPROF for the first SQL statement:

```
c:\tkprof c:\oracle\rdbms\admin\TRACE\ora______.trc   lab84
explain=userid/password
```

lab84.prf

```
*******************************************************************************************
select big.bname,
from big
where  BIGNO IN  (SELECT BIGNO
                  FROM BIG2
                  WHERE BNAME = 'ONE THOUSAND');
```

TKPROF COLUMNS

CALL	COUNT	CPU	ELAPSED	DISK	QUERY	CURRENT	ROWS
Parse							
Execute							
Fetch							

Misses in library cache during parse: ________________
Parsing user id: _______userid ___________

ROWS	EXECUTION PLAN

LAB 10 - USING SQL TRACE AND TKPROF (CONTINUED)

6. Fill in the formatted report from TKPROF for the first SQL statement:

   c:\tkprof c:\$ORACLE_HOME\rdbms\admin\orcl\udump\ora______.trc lab84 explain=***userid/password***

lab84.prf

```
**********************************************************************************************
select big.bname
from big
where  0 < (SELECT COUNT(*)
                    FROM BIG2
WHERE big2.bname = 'ONE THOUSAND'
            AND BIG.BIGNO = BIG2.BIGNO);
```

TKPROF COLUMNS

CALL	**COUNT**	**CPU**	**ELAPSED**	**DISK**	**QUERY**	**CURRENT**	**ROWS**
Parse							
Execute							
Fetch							

Misses in library cache during parse: _______________
Parsing user id: ______userid __________

ROWS	EXECUTION PLAN

This page left blank intentionally