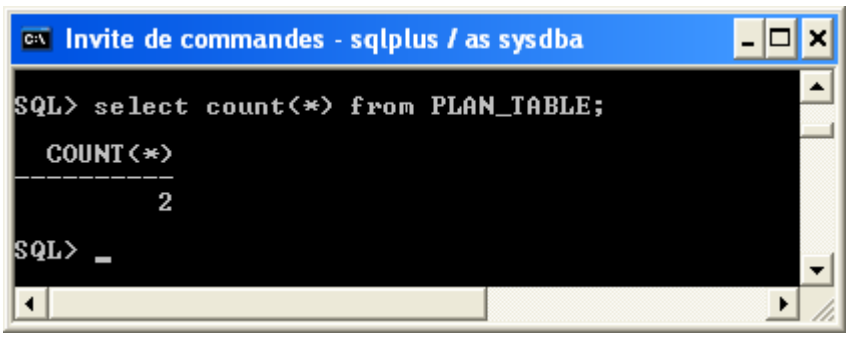
EXPLAIN PLAN & EXECUTION PLAN

EXPLAIN PLAN parses a query and records the "plan" that Oracle devises to execute it. By examining this plan, you can find out if Oracle is picking the right indexes and joining your tables in the most efficient manner. The Explain Plan statement does not go through the same code path that the optimizer uses when determining a plan for execution.

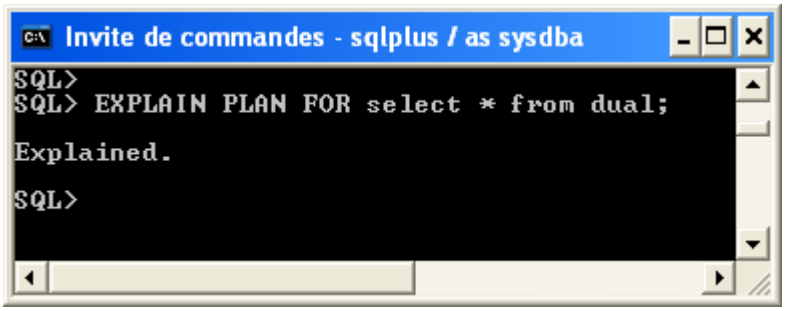
If you do an Explain, you're actually looking at a theoretical plan, not the 'actual' plan.

1. Execution plans can and do change as the underlying optimizer inputs change.
2. EXPLAIN PLAN output shows how the database would run the SQL statement when the statement was explained.
3. This plan can differ from the actual execution plan a SQL statement uses because of differences in the execution environment and explain plan environment.
4. Explain plan is blind to the bind, It presumes *all binds are varchar2's* regardless of how the developer is binding

Checking the existence of PLAN_TABLE



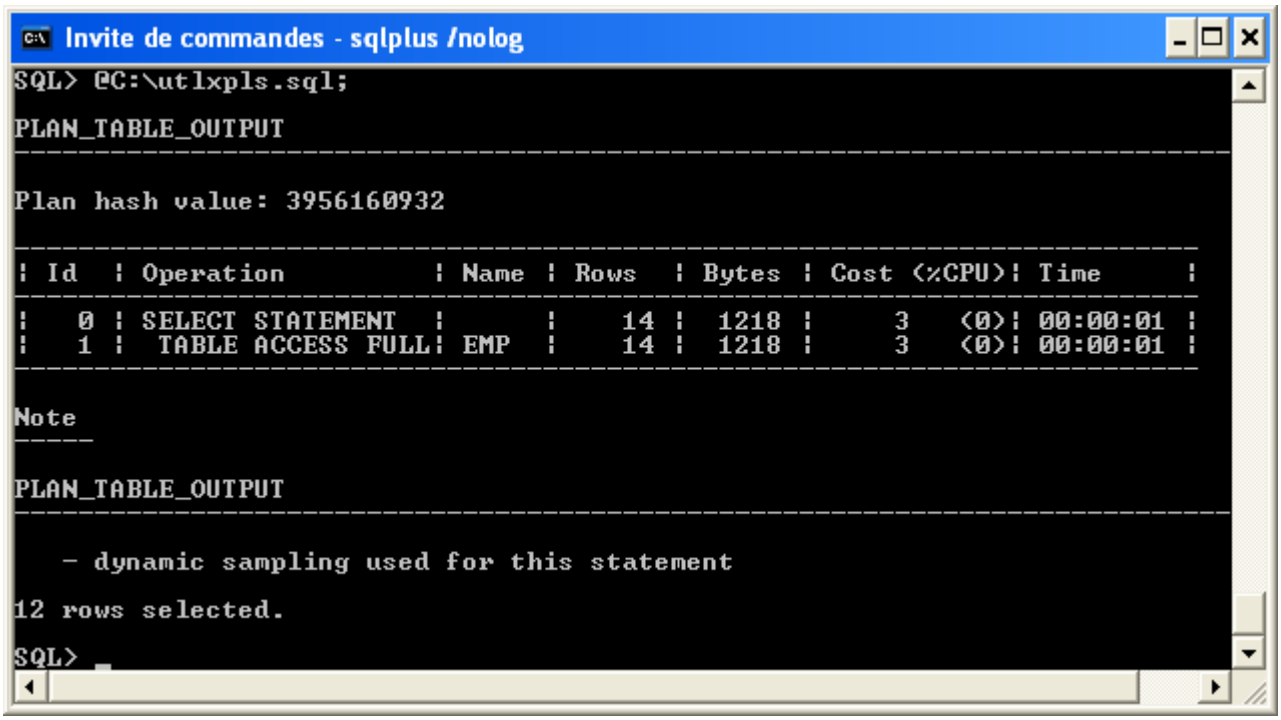
If the plan table is not accessible this table must be created using ORACLE_HOME/rdbms/admin/UTLXPLAN.SQL script



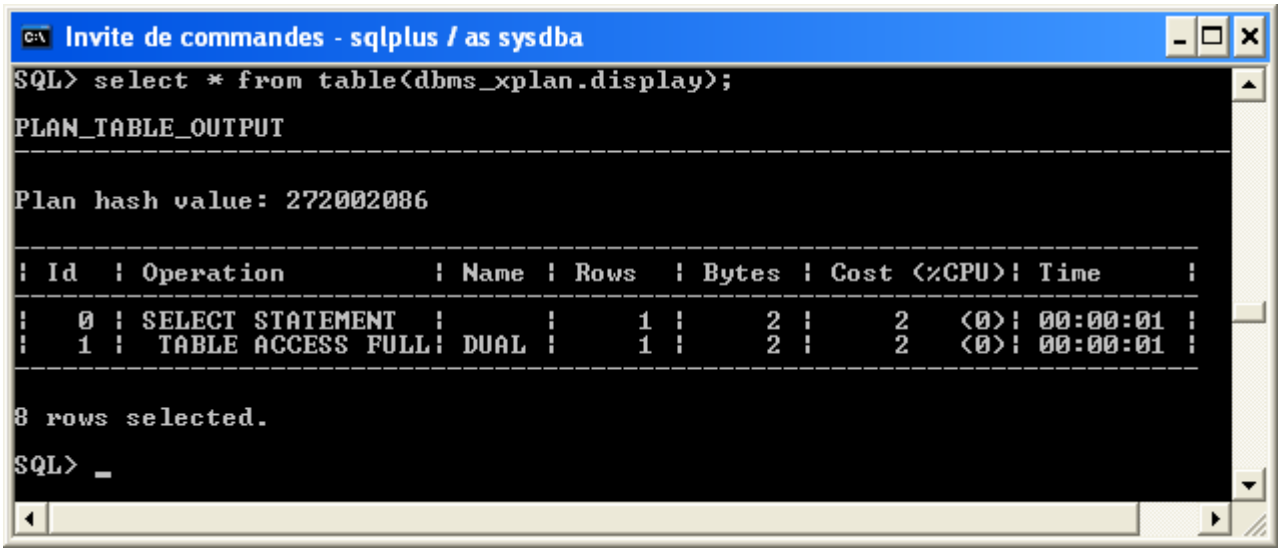
Using the SELECT statement:

```
SQL> SELECT substr (lpad (' ', level-1) || operation || ' (' || options ||
     ')', 1, 30) "Operation", object_name "Object Accessed", COST
     FROM plan_table START WITH id = 0 CONNECT BY PRIOR id=parent_id;
```

Using the utlxpls.sql (utlxpls.sql is a script that Oracle ships):



Using table (dbms_xplan.display) table



Execution PLAN

There are two different methods you can use to look at the execution plan of a SQL statement:

1. EXPLAIN PLAN command - This displays an execution plan for a SQL statement without actually executing the statement.

2. V$SQL_PLAN - A dictionary view introduced in Oracle 9i that shows the execution plan for a SQL statement that has been compiled into a cursor in the cursor cache.

Under certain conditions the plan shown when using EXPLAIN PLAN can be different from the plan shown using V$SQL_PLAN. For example, when the SQL statement contains bind variables the plan shown from using EXPLAIN PLAN ignores the bind variable values while the plan shown in V$SQL_PLAN takes the bind variable values into account in the plan generation process.

Displaying an execution plan has been made easier after the introduction of the dbms_xplan package in Oracle 9i and by the enhancements made to it in subsequent releases.
This packages provides several PL/SQL procedures to display the plan from different sources:

1. EXPLAIN PLAN command
2. V$SQL_PLAN
3. Automatic Workload Repository (AWR)
4. SQL Tuning Set (STS)
5. SQL Plan Baseline (SPM)

TESTCASE WHEN EXPLAIN PLAN IS DIFFERENT FROM EXECUTION PLAN

The Oracle Explain Plan command is widely used to evaluate the plan that the Oracle optimizer will choose for a given SQL statement. Unfortunately, it doesn't always tell the truth. This is due to the fact that the Explain Plan statement does not go through the same code path that the optimizer uses when determining a plan for execution. One of the simplest examples of this behaviour is the case where bind variables are used in a statement. Explain plan ignores them while the optimizer uses them to determine the plan.

```
SQL> create table t (id varchar2 (10), name varchar2 (100));

SQL> insert into t select to_char (object_id), object_name from dba_objects;
51449 rows created.
```

Now we add a little index for lookup performance and gather stats.

```
SQL> create index i on t (id);
Index created.

SQL> exec dbms_stats.gather_table_stats (user,'T', cascade=>true);
PL/SQL procedure successfully completed.
```

Now let's define a bind variable of NUMBER type and set a value for it:

```
SQL> var x number

SQL> exec :x: =99999
PL/SQL procedure successfully completed.
```

Now let's use "explain plan for" to estimate the execution plan:

```
SQL> explain plan for
     Select sum (length (name)) from t where id >:x;

Explained.

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------

Plan hash value:     3694077449

--------------------------------------------------------------------------------
--------
| Id | Operation                    | Name | Rows | Bytes | Cost (%CPU)|
Time     |
--------------------------------------------------------------------------------
--------
|   0 | SELECT STATEMENT             |      |     1 |    29 |    56   (0)|
00:00:01 |
|   1 | SORT AGGREGATE               |      |     1 |    29 |
|          |
|   2 |   TABLE ACCESS BY INDEX ROWID| T    | 2572 | 74588 |    56   (0)|
00:00:01 |
|* 3 |    INDEX RANGE SCAN          | I    |   463 |       |     3   (0)|
00:00:01 |
--------------------------------------------------------------------------------
--------

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

 3 - Access ("ID">: X)

15 rows selected.
```

Explain plan command nicely reports that we'd be using an index range scan, which would be a good thing to do given my test data and search condition.

Now let's actually run the statement and see the REAL execution plan *actually used* for the execution. I'll use dbms_xplan.display_CURSOR for this. If you don't pass SQL_ID/child into that function it will just report the last SQL statement executed in your current session. But the key difference between the dbms_xplan.DISPLAY and DISPLAY_CURSOR is that the latter

goes to library cache and fetches the *actual* SQL plan used from there. The explain plan command just reparses the statement and estimates a plan, ignoring any bind variable values and assuming that all bind variables are of type varchar2:

```
SQL> select sum (length (name)) from t where id >: x;

SUM (LENGTH (NAME))
-------------------

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

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
-----
SQL_ID  7zm570j6kj597, child number 0
-------------------------------------
Select sum (length (name)) from t where id >: x


Plan hash value:     2966233522

-------------------------------------------------------------------------------
| Id | Operation          | Name | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      |       |    60 (100)|          |
|   1 | SORT AGGREGATE     |      |    1 |    29 |            |          |
|* 2 |   TABLE ACCESS FULL| T    | 2572 | 74588 |    60   (5)| 00:00:01 |
-------------------------------------------------------------------------------

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

 2 - Filter (TO_NUMBER ("ID")>: X)

19 rows selected.
```

How to read an Oracle SQL Execution Plan?

To execute any SQL statement Oracle has to derive an 'execution plan'. The execution plan of a query is a description of how Oracle will implement the retrieval of data to satisfy a given SQL statement. It is nothing but a tree which contains the order of steps and relationship between them.

The basic rules of execution plan tree is below:

1. An execution plan will contain a root, which has no parents
2. A parent can have one or more children, and its ID will be less than the child(s) ID
3. A child can have only one parent, it is indented to the right; in case of many child's, it will Have the same indentation.

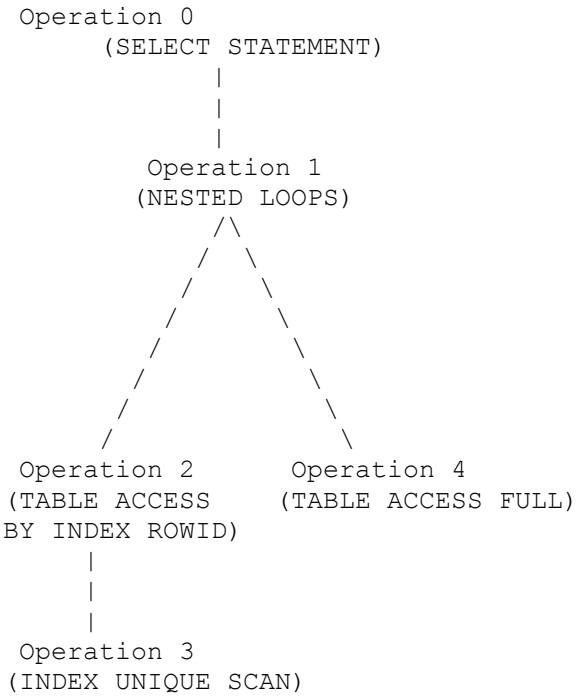
```
SQL> explain plan for
     Select e.deptno, e.ename, d.dname from emp e, dept d
```

```
     Where e.deptno = d.deptno and e.deptno = 10;

Explained.

SQL> SELECT * FROM table (dbms_xplan.display (null, null, 'basic'));

PLAN_TABLE_OUTPUT
-----------------------------------------------
Plan hash value: 568005898

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

Using the rules above, you could say;

1. Operation 0 is the root of the tree; it has one child, Operation 1
2. Operation 1 has two children, which is Operation 2 and 4
3. Operation 2 has one child, which is Operation 3

Below is the graphical representation of the execution plan. If you read the tree; In order to perform Operation 1, you need to perform Operation 2 and 4. Operation 2 comes first; In order to perform 2, you need to perform its Child Operation 3. In order to perform Operation 4, you need to perform Operation 2



1. Operation 3 accesses DEPT table using INDEX UNIQUE SCAN and passes the ROWID to Operation 2.
2. Operation 2 returns all the rows from DEPT table to Operation 1.

3. Operation 1 performs Operation 4 for each row returned by Operation 2.
4. Operation 4 performs a full table scan (TABLE ACCESS FULL) scan and applies the filter E.DEPTNO=10 and returns the rows to Operation 1.
5. Operation 1 returns the final results to Operation 0.