Report

Subject: Plan of Execution of Queries.

Students: CHARFAOUI Younes, BOURBAI Ismail.

Github Repository: https://github.com/IsmailBourbie/master-one-practical-work/tree/master/db_dm/PW03

Step One:

We've installed the oracle 11g desktop class in our computers, and we've used Datagrip IDE for manipulating and queries the DBMS

Step Two:

In this step we create the table provided in the practical work sheet .Let's Execute the Following Line of SQL Script, the other queries are in the github repos.

```
CREATE TABLE LINEORDER (
    LO_ORDERKEY       INTEGER,
    LO_LINENUMBER     INTEGER,
    LO_CUSTKEY        INTEGER     NOT NULL,
    LO_PARTKEY        INTEGER     NOT NULL,
    LO_SUPPKEY        INTEGER     NOT NULL,
    LO_ORDERDATE      INTEGER     NOT NULL,
    LO_ORDERPRIOTITY  VARCHAR(15) NOT NULL,
    LO_SHIPPRIOTITY   INTEGER,
    LO_QUANTITY       INTEGER,
    LO_EXTENDEDPRICE  INTEGER,
    LO_ORDTOTALPRICE  INTEGER,
    LO_DISCOUNT       INTEGER,
    LO_REVENUE        INTEGER,
    LO_SUPPLYCOST     INTEGER,
    LO_TAX            INTEGER,
    LO_COMMITDATE     INTEGER     NOT NULL,
    LO_SHIPMODE       VARCHAR(10) NOT NULL
);
```

Figure 1 Example of Create Query

Step Three:

In This Step we have used the SQL*Loader to load our prepared data into the tables, the following screen shot demonstrate one of the tables that:

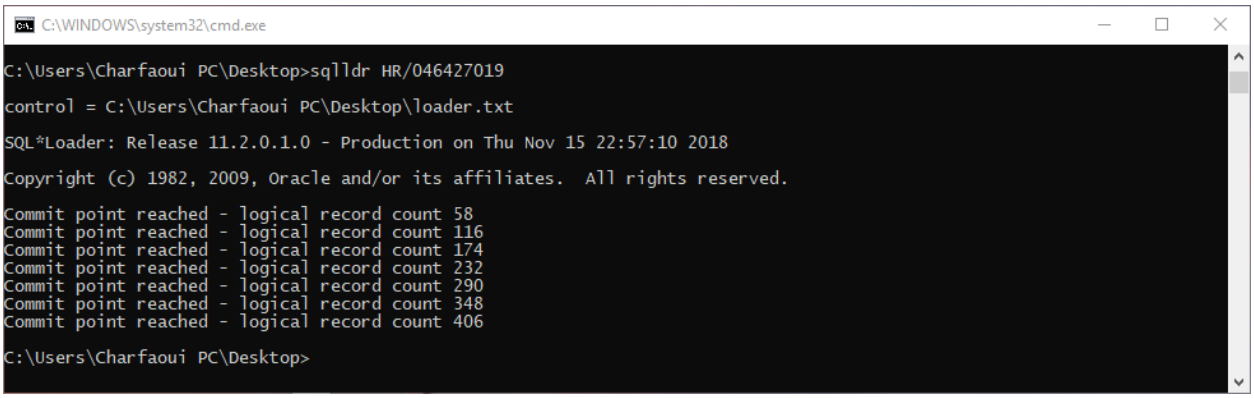

Figure 2 SQL Loader Demonstration

In The Loader file we specifty the tale and from which file we are takiing the data, here is an model:

LOAD DATA INFILE “path_of_datafile" INTO TABLE “name_of_table” FIELDS TERMINATED BY separator (columns)

The other file are in the repo: https://github.com/IsmailBourbie/master-one-practical-work/tree/master/db_dm/PW03/loader_files

Step Four:

For a DBA there is very handy tool called Explain Plan, and this one help the DBA for tracking how the query is executed to help it debug and find an optimal way to reduce the time and make accurate result, in this Practical Work we explored different queries and its plan, Here is the Query and the corresponding plan:

EXPLAIN PLAN FOR SELECT * FROM DATES;

```
PLAN_TABLE_OUTPUT
Plan hash value: 2372408698

-------------------------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |   306 | 58752 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| DATES |   306 | 58752 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Note
-----
   - dynamic sampling used for this statement (level=2)
```

Figure 3 Result Plan

We made another test with a complex query, here is both of them:

```
EXPLAIN PLAN FOR
select
    sum(lo_extendedprice*lo_discount) as revenue
from
    lineorder, dates
where
    lo_orderdate = d_datekey
    and d_yearmonthnum = 199401
    and lo_discount between 4 and 6
    and lo_quantity between 26 and 35;
```

Figure 4 Complex query for The EXPLAIN PLAN

```
PLAN_TABLE_OUTPUT
Plan hash value: 2553672856

-------------------------------------------------------------------------------
| Id  | Operation           | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |     1 |    78 | 19822   (2)| 00:03:58 |
|   1 |  SORT AGGREGATE     |           |     1 |    78 |            |          |
|*  2 |   HASH JOIN         |           |     1 |    78 | 19822   (2)| 00:03:58 |
|*  3 |    TABLE ACCESS FULL| DATES     |     1 |    26 |     3   (0)| 00:00:01 |
|*  4 |    TABLE ACCESS FULL| LINEORDER |   338K|    16M| 19816   (2)| 00:03:58 |
-------------------------------------------------------------------------------

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

   2 - access("LO_ORDERDATE"="D_DATEKEY")
   3 - filter("D_YEARMONTHNUM"=199401)
   4 - filter("LO_DISCOUNT">=4 AND "LO_DISCOUNT"<=6 AND
              "LO_QUANTITY">=26 AND "LO_QUANTITY"<=35)

Note
-----
   - dynamic sampling used for this statement (level=2)
```

Figure 5 Plan of Execution of Last Query

The Other Examples are in the repository. https://github.com/IsmailBourbie/master-one-practical-work/tree/master/db_dm/PW03/screen_shots

Step Five:

At this point we know how to see the way the query is executing, but how tune some parameter? The way to do that is throughout using hints, hints are a way for telling the DBMS how to execute queries, we 'have used The Nested Loop Hint instead of the default Hash Join to Show the difference, here is the query and the associated plan.

```
EXPLAIN PLAN FOR
select /*+ use_nl(lineorder,dates,part,supplier) */
    sum(lo_revenue), d_year, p_brand
from
    lineorder, dates, part, supplier
where
    lo_orderdate = d_datekey
    and lo_partkey = p_partkey
    and lo_suppkey = s_suppkey
    and p_category = 'MFGR#12'
    and s_region = 'AMERICA'
group by
    d_year, p_brand
order by
    d_year, p_brand;
```

Figure 6 Query Use A Hint to optimize

```
PLAN_TABLE_OUTPUT
Plan hash value: 1339098462

-------------------------------------------------------------------------------------------
| Id  | Operation                | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |             | 13783 |  1763K|   943K  (1)| 03:08:48 |
|   1 |  SORT GROUP BY           |             | 13783 |  1763K|   943K  (1)| 03:08:48 |
|   2 |   NESTED LOOPS           |             | 13783 |  1763K|   943K  (1)| 03:08:48 |
|   3 |    NESTED LOOPS          |             | 72923 |  7833K|   386K  (1)| 01:17:23 |
|*  4 |     HASH JOIN            |             |   269K|    21M| 20536   (2)| 00:04:07 |
|   5 |      VIEW                | VW_GBF_13   |  8094 |   252K|   688   (1)| 00:00:09 |
|   6 |       HASH GROUP BY      |             |  8094 |   189K|   688   (1)| 00:00:09 |
|*  7 |        TABLE ACCESS FULL | PART        |  8094 |   189K|   687   (1)| 00:00:09 |
|   8 |      TABLE ACCESS FULL   | LINEORDER   |  6494K|   322M| 19802   (2)| 00:03:58 |
|*  9 |     TABLE ACCESS FULL    | DATES       |     1 |    26 |     1   (0)| 00:00:01 |
|* 10 |    TABLE ACCESS FULL     | SUPPLIER    |     1 |    21 |     8   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   4 - access("LO_PARTKEY"="ITEM_1")
   7 - filter("P_CATEGORY"='MFGR#12')
   9 - filter("LO_ORDERDATE"="D_DATEKEY")
  10 - filter("S_REGION"='AMERICA' AND "LO_SUPPKEY"="S_SUPPKEY")

Note
-----
   - dynamic sampling used for this statement (level=2)
```

Figure 7 the Plan of Nested Loop Example

In The Operation with id 3 and 2, we see that the DBMS has used Nested Loop in The Plan of executing the query and this demonstrate how we use hints.

Step Six:

In This last step, our goal is to use an optimization technique to make the query work faster as possible for this we used the indexes and horizontal fragmentation, the queries are in the repository and one of the testing query and it's correspond plan are below :

Other Queries: https://github.com/IsmailBourbie/master-one-practical-work/blob/master/db_dm/PW03/queries.sql

```
EXPLAIN PLAN FOR
select
    c_city, s_city, d_year, sum(lo_revenue) as revenue
from
    customer, lineorder, supplier, dates
where
    lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and (c_city='UNITED KI1' or c_city='UNITED KI5')
    and (s_city='UNITED KI1' or s_city='UNITED KI5')
    and d_yearmonth = 'Dec1997'
group by
    c_city, s_city, d_year
order by
    d_year asc, revenue desc;;
```

Figure 8 Query to demonstrate Index Optimization

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                   |     1 |   123 |   132   (2)| 00:00:02 |
|   1 |  SORT ORDER BY                 |                   |     1 |   123 |   132   (2)| 00:00:02 |
|   2 |   HASH GROUP BY                |                   |     1 |   123 |   132   (2)| 00:00:02 |
|   3 |    NESTED LOOPS                |                   |       |       |            |          |
|   4 |     NESTED LOOPS               |                   |     1 |   123 |   130   (0)| 00:00:02 |
|   5 |      NESTED LOOPS              |                   |     1 |   103 |    23   (0)| 00:00:01 |
|   6 |       MERGE JOIN CARTESIAN     |                   |     1 |    51 |     8   (0)| 00:00:01 |
|   7 |        INLIST ITERATOR         |                   |       |       |            |          |
|   8 |         TABLE ACCESS BY INDEX ROWID| SUPPLIER      |    19 |   380 |     8   (0)| 00:00:01 |
|*  9 |          INDEX RANGE SCAN      | S_CITY_IDX        |     8 |       |     2   (0)| 00:00:01 |
|  10 |        BUFFER SORT             |                   |     1 |    31 |     0   (0)| 00:00:01 |
|  11 |         TABLE ACCESS BY INDEX ROWID| DATES         |     1 |    31 |     0   (0)| 00:00:01 |
|* 12 |          INDEX RANGE SCAN      | D_YEARMONTH_IDX   |     1 |       |     0   (0)| 00:00:01 |
|* 13 |       TABLE ACCESS BY INDEX ROWID | LINEORDER      |    11 |   572 |    15   (0)| 00:00:01 |
|* 14 |        INDEX RANGE SCAN        | LO_ORDERDATE_IDX  |    35 |       |     6   (0)| 00:00:01 |
|  15 |      INLIST ITERATOR           |                   |       |       |            |          |
|* 16 |       INDEX RANGE SCAN         | C_CITY_IDX        |   116 |       |     2   (0)| 00:00:01 |
|* 17 |     TABLE ACCESS BY INDEX ROWID| CUSTOMER          |     1 |    20 |   107   (0)| 00:00:02 |
-------------------------------------------------------------------------------------------------------------

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

   9 - access("S_CITY"='UNITED KI1' OR "S_CITY"='UNITED KI5')
  12 - access("D_YEARMONTH"='Dec1997')
  13 - filter("LO_SUPPKEY"="S_SUPPKEY")
  14 - access("LO_ORDERDATE"="D_DATEKEY")
  16 - access("C_CITY"='UNITED KI1' OR "C_CITY"='UNITED KI5')

Database Changes  Database Console
```

Figure 9 Plan after Index Modification

Conclusion:

In This Practical word we saw that oracle DBMS is a great one , its offer loading data from various input , help with monitoring the query plan execution and also give the database administrator the tools and ways to optimize the database structure and the corresponding query via multiple parameters, another future which is a great about it but really made us crazy about oracle is the security provided , really in the first time your hand will be very dirty just to enter into the DBMS and start making some queries, really it was very difficult to just enter into it, lastly I would really recommend this DBMS for A very large enterprise that need security and performance.