Ex. No: 6 **QUERY TUNING and NORMALIZATION**

Aim:

To execute a minimum of 15 different Query tuning commands along with reasoning .Also ,check and convert your application that it satisfies the maximum Normal form.

```
SQL> select * from vehicles;

        ID MODEL      MANUFACTURE COLOR          MILEAGE     REG_NO
---------- ---------- ----------- ---------- ---------- ----------
         1 suzuki            2000 white              45         99
         3 maruthi           2012 yellow             70         97
         2 toyoto            2008 black              30         94
         5 benz              2004 black              50         98
         4 hondo             2020 red                60         90
         9 acer                   brown                         11
         8 mahi              2004 sandal             80         23
         6 mercedes          2014 pink                6         44

8 rows selected.
```

```
SQL> SELECT * FROM CUSTOMERS;

CUSTO CUST_NAME  MODEL
----- ---------- ----------
1     HARINI     suzuki
2     RITHIKA    toyoto
3     RAJI       acer
4     POOJA      mahi
```

Procedure:

1) SELECT fields instead of using SELECT *

If a table has many fields and many rows, this taxes database resources by querying a lot of unnecessary data.

```
SQL> select model,color from vehicles;

MODEL      COLOR
---------- ----------
suzuki     white
maruthi    yellow
toyoto     black
benz       black
hondo      red
acer       brown
mahi       sandal
mercedes   pink

8 rows selected.
```

2) Use WHERE instead of HAVING to define filters

HAVING statements are calculated after WHERE statements. If the intent is to filter a query based on conditions, a WHERE statement is more efficient.

```
SQL> select model,color from vehicles where mileage>60;

MODEL      COLOR
---------- ----------
maruthi    yellow
mahi       sandal
```

3) Select only required columns:

Avoid unnecessary details to have faster results

```
SQL> select cust_name from customers;

CUST_NAME
----------
HARINI
RITHIKA
RAJI
POOJA
```

4) Use EXISTS, IN suitably in your query

- Usually IN has the slowest execution. ○ IN is effective when a large portion is in the sub-query.
- EXISTS is useful.

```
SQL> select model,color from vehicles where exists(select customer_id from customers where
customer_id>2 and vehicles.model=customers.model);

MODEL      COLOR
---------- ----------
acer       brown
mahi       sandal
```

5) Use EXISTS Instead of DISTINCT when using joins which includes tables having the one- to-many relationship.

```
SQL> select v.model,v.color,c.cust_name from vehicles v join customers c on v.model=c.model
 where exists(select 1 from customers c where c.model=v.model);

MODEL      COLOR      CUST_NAME
---------- ---------- ----------
suzuki     white      HARINI
toyoto     black      RITHIKA
acer       brown      RAJI
mahi       sandal     POOJA
```

Distinct sorts the retrieved rows before suppressing the duplicate rows

6) **Use UNION ALL instead of UNION.**

Union all returns all the records retrieved by queries

```
SQL> select model from vehicles union all select model from customers;

MODEL
----------
suzuki
maruthi
toyoto
benz
hondo
acer
mahi
mercedes
suzuki
toyoto
acer

MODEL
----------
mahi

12 rows selected.
```

7) **Use proper conditions in WHERE statement**

Using AND in where clause is preferable because it avoids data alias

```
SQL> select * from customers where customer_id>2 and model='mahi';

CUSTO CUST_NAME  MODEL
----- ---------- ----------
4     POOJA      mahi
```

8) **Use DECODE to stay away from the checking of same columns or joining a similar table monotonously. It can likewise be made used instead of GROUP BY or ORDER BY statement.** It avoids joining the same table repeatedly

```
SQL> select id,model,
  2  case
  3  when mileage<20 then 'low'
  4  when mileage between 20 and 55 then 'medium'
  5  when mileage>55 then 'high'
  6  else 'unknown'
  7  end as mileage_range
  8  from vehicles;

        ID MODEL      MILEAGE
---------- ---------- -------
         1 suzuki     medium
         3 maruthi    high
         2 toyoto     medium
         5 benz       medium
         4 hondo      high
         9 acer       unknown
         8 mahi       high
         6 mercedes   low

8 rows selected.
```

9) **Use non-column expression on one side of the query** Because it will be processed earlier.

```
SQL> select model from vehicles where mileage+10 > 55;

MODEL
----------
maruthi
benz
hondo
mahi
```

10) **SQL optimization technique concerns the use of Exists()**

```
SQL> select * from  customers c  where exists(select 1 from vehicles v where v.model=c.model);

CUSTO CUST_NAME  MODEL
----- ---------- ----------
1     HARINI     suzuki
2     RITHIKA    toyoto
3     RAJI       acer
4     POOJA      mahi
```

```
SQL> select * from  vehicles v  where exists(select 1 from customers c where c.model=v.model);

        ID MODEL      MANUFACTURE COLOR          MILEAGE     REG_NO
---------- ---------- ----------- ---------- ---------- ----------
         1 suzuki            2000 white              45         99
         2 toyoto            2008 black              30         94
         9 acer                   brown                      11
         8 mahi              2004 sandal             80         23
```

11. **Use a single case for each word**

Follow the SQL standard rules for efficient query performance

```
SQL> select model from vehicles where color='black';

MODEL
----------
toyoto
benz
```

12. **Create views with only essential columns** View needn't have extra information

```
SQL> create view cust_view as select cust_name,model from customers;

View created.
```

13. **Create joins with inner joins to avoid Cartesian problem:**

Avoid cartesian product as it creates combination of each record of one table with every record of other

```
SQL> select v.model,v.manufacture,v.color,c.cust_name from vehicles v inner join customers c on v.model=c.model;

MODEL      MANUFACTURE COLOR      CUST_NAME
---------- ----------- ---------- ----------
suzuki            2000 white      HARINI
toyoto            2008 black      RITHIKA
acer                   brown      RAJI
mahi              2004 sandal     POOJA
```

14. Use wildcards only at the end of the phrase

```
SQL> select id,color from vehicles where model like 'm%';

        ID COLOR
---------- ----------
         3 yellow
         8 sandal
         6 pink
```

15.Use BETWEEN instead of less than greater than BETWEEN

is efficient than the relational operators < >

```
SQL> select model from vehicles where mileage between 50 and 100;

MODEL
----------
maruthi
benz
hondo
mahi
```

DESCRIPTION:

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

Database Normal Forms :

1. 1NF (First Normal Form):
2. 2NF (Second Normal Form)
3. 3NF (Third Normal Form)
4. BCNF (Boyce-Codd Normal Form)
5. 4NF (Fourth Normal Form)
6. 5NF (Fifth Normal Form)

First Normal Form:

A relation is in first normal form only if the relational table doesn't contain any

multivalued attributes. (The relation contains only single-valued attributes).

```
SQL> select * from employee;

EMP_I FNAME      LNAME           PHN HIRE_DATE  JOB_I SALAR JOB
----- ---------- ---------- ---------- ---------- ----- ----- -----
1001  Rithika    S          9345542103 20-8-2024  111   50000
1002  harini     S          9003402381 25-10-2024 222   70000
1003  Suriya     A          6878900181 13-6-2024  333   40000
1004  Arun       J           733421053 04-10-2024 444   30000
1005  Lithi      A           810511053 12-12-2024 555   25000
1006  Prakash    A          9009992512 11-06-2024 666   40000 hr
```

Second Normal Form:

A relation is in second normal form if: •It is in first normal form or 1NF •It doesn't contain any partial dependencies. •It shouldn't have any non-prime attribute which is functionally dependent on any proper subset of the candidate key of the relation

```
SQL> create table sub1 as select job_id,job from employee;

Table created.

SQL> select * from sub1;

JOB_I JOB
----- -----
111   IT
222   arch
333   IT
444   arch
555   hr
666   hr
```

```
SQL> create table sub2 as select fname, lname, phn, hire_date from employee;


Table created.
```

```
SQL> select * from sub2;

FNAME      LNAME              PHN HIRE_DATE
---------- ---------- ---------- ----------
Rithika    S          9345542103 20-8-2024
harini     S          9003402381 25-10-2024
Suriya     A          6878900181 13-6-2024
Arun       J           733421053 04-10-2024
Lithi      A           810511053 12-12-2024
Prakash    A          9009992512 11-06-2024
```

```
SQL> create table sub3 as select emp_id, salary from employee;

Table created.

SQL> select * from sub3;

EMP_I SALAR
----- -----
1001  50000
1002  70000
1003  40000
1004  30000
1005  25000
1006  40000

6 rows selected.
```

Third Normal Form :

The relation should be in 2NF If A->B and B->C are two FDs then A->C is called transitive dependency.

```
SQL> create table sub3_1 as select emp_id fname, lname, phn, hire_date , sal
ary from employee;

Table created.

SQL> create table sub3_2 as select job_id,job from employee;

Table created.

SQL> create table sub3_3 as select emp_id, job_id from employee;

Table created.

SQL> select * from sub3_1;

FNAME LNAME             PHN HIRE_DATE  SALAR
----- ---------- ---------- ---------- -----
1001  S          9345542103 20-8-2024  50000
1002  S          9003402381 25-10-2024 70000
1003  A          6878900181 13-6-2024  40000
1004  J           733421053 04-10-2024 30000
1005  A           810511053 12-12-2024 25000
1006  A          9009992512 11-06-2024 40000

6 rows selected.

SQL> select * from sub3_2;

JOB_I JOB
----- -----
111   IT
222   arch
333   IT
444   arch
555   hr
666   hr

6 rows selected.

SQL> select * from sub3_3;

EMP_I JOB_I
----- -----
1001  111
1002  222
1003  333
1004  444
1005  555
1006  666

6 rows selected.
```

Result:

Thus, 15 Query tuning commands and normalization are executed with reasoning.