Batch: D2
Roll No.: 16010122323

Experiment / assignment / tutorial No. 7
Grade: AA / AB / BB / BC / CC / CD

Title: Implementing indexing and query processing

Objective: To understand Query Processing and implement indexing to improve query execution plans

Expected Outcome of Experiment:
CO 3: Use SQL for relational database creation , maintenance and query processing

Books/ Journals/ Websites referred:

1. Dr. P.S. Deshpande, SQL and PL/SQL for Oracle 10g.Black book, Dreamtech Press
2. www.db-book.com
3. Korth, Slberchatz, Sudarshan : "Database Systems Concept", 5th Edition , McGraw Hill
4. Elmasri and Navathe,"Fundamentals of database Systems", 4th Edition,PEARSON Education.

Resources used: PostgreSQL/MySQL

Theory:

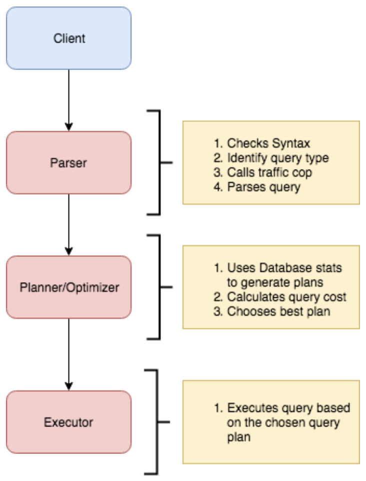
A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.

While creating index, it should be taken into consideration which all columns will be used to make SQL queries and create one or more indexes on those columns.

To add an index for a column or a set of columns, you use the `CREATE INDEX` statement as follows:

```
CREATE INDEX index_name ON table_name (column_list)
```

Query life cycle



Planner and Executor:

The planner receives a query tree from the rewriter and generates a (query) plan tree that can be processed by the executor most effectively.

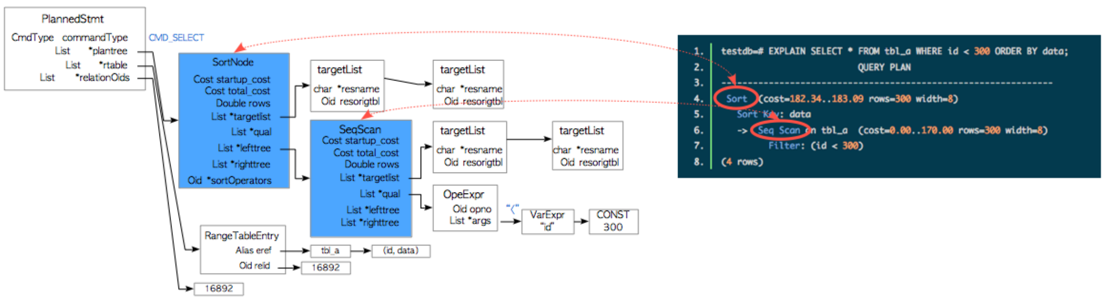
The planner in Database is based on pure cost-based optimization -

EXPLAIN command:

This command displays the execution plan that the PostgreSQL/MySQL planner generates for the supplied statement. The execution plan shows how the table(s) referenced by the statement will be scanned — by plain sequential scan, index scan, etc. — and if multiple tables are referenced, what join algorithms will be used to bring together the required rows from each input table.
As in the other RDBMS, the EXPLAIN command in Database displays the plan tree itself. A specific example is shown below:-

```
Database: testdb=#
1. EXPLAIN SELECT * FROM tbl_a WHERE id < 300 ORDER BY data;
2. QUERY PLAN
3. ---------------------------------------------------------------
4. Sort (cost=182.34..183.09 rows=300 width=8)
5. Sort Key: data
6. -> Seq Scan on tbl_a (cost=0.00..170.00 rows=300 width=8)
7. Filter: (id < 300)
8. (4 rows)
```

A simple plan tree and the relationship between the plan tree and the result of the EXPLAIN command in PostgreSQL.



```
testdb=# EXPLAIN SELECT * FROM tbl_a WHERE id < 300 ORDER BY data;
                            QUERY PLAN
---------------------------------------------------------------
 Sort  (cost=182.34..183.09 rows=300 width=8)
   Sort Key: data
   ->  Seq Scan on tbl_a  (cost=0.00..170.00 rows=300 width=8)
         Filter: (id < 300)
(4 rows)
```

Nodes

The first thing to understand is that each indented block with a preceeding "->" (along with the top line) is called a node. A node is a logical unit of work (a "step" if you will) with an associated cost and execution time. The costs and times presented at each node are cumulative and roll up all child nodes.

Cost:

It is not the time but a concept designed to estimate the cost of an operation. The first number is start-up cost (cost to retrieve first record) and the second number is the cost incurred to process entire node (total cost from start to finish).

Cost is a combination of 5 work components used to estimate the work required: sequential fetch, non-sequential (random) fetch, processing of row, processing operator (function), and processing index entry.

Rows are the approximate number of rows returned when a specified operation is performed.

(In the case of select with where clause rows returned is

Rows = cardinality of relation * selectivity)

Width is an average size of one row in bytes**.**

Explain Analyze command:

The EXPLAIN ANALYZE option causes the statement to be actually executed, not only planned. Then actual run time statistics are added to the display, including the total elapsed time expended within each plan node (in milliseconds) and the total number of rows it actually returned. This is useful for seeing whether the planner's estimates are close to reality.

Ex: EXPLAIN (ANALYZE) SELECT * FROM foo;

```
QUERY PLAN
— Seq Scan on foo (cost=0.00..18334.10 rows=1000010 width=37) (actual time=0.012..61.524
rows=1000010 loops=1)
Total runtime: 90.944 ms
(2 rows)
```

The command displays the following additional parameters:

- **actual time** is the actual time in milliseconds spent to get the first row and all rows, respectively.
- **rows** is the actual number of rows received with Seq Scan.
- **loops** is the number of times the Seq Scan operation had to be performed.
- **Total runtime** is the total time of query execution.

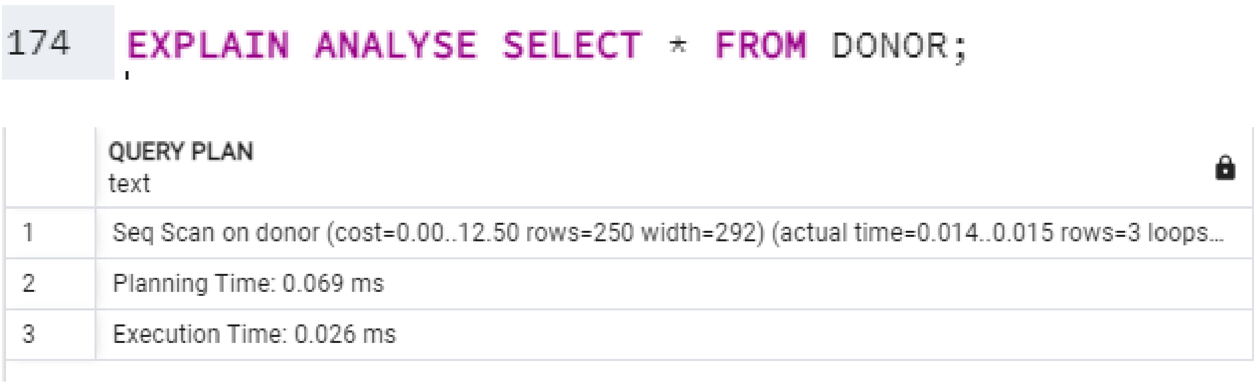
Query plans for select with where clause can be sequential scan, Index Scan, Index only Scan, Bitmap Index Scan etc.

Query plans for joins are Nested loop join, Hash join, Merge join etc.

Implementation Screenshots :

Comprehend how indexes improves the performance of query applied for your database . Demonstrate for the following types of query on your database

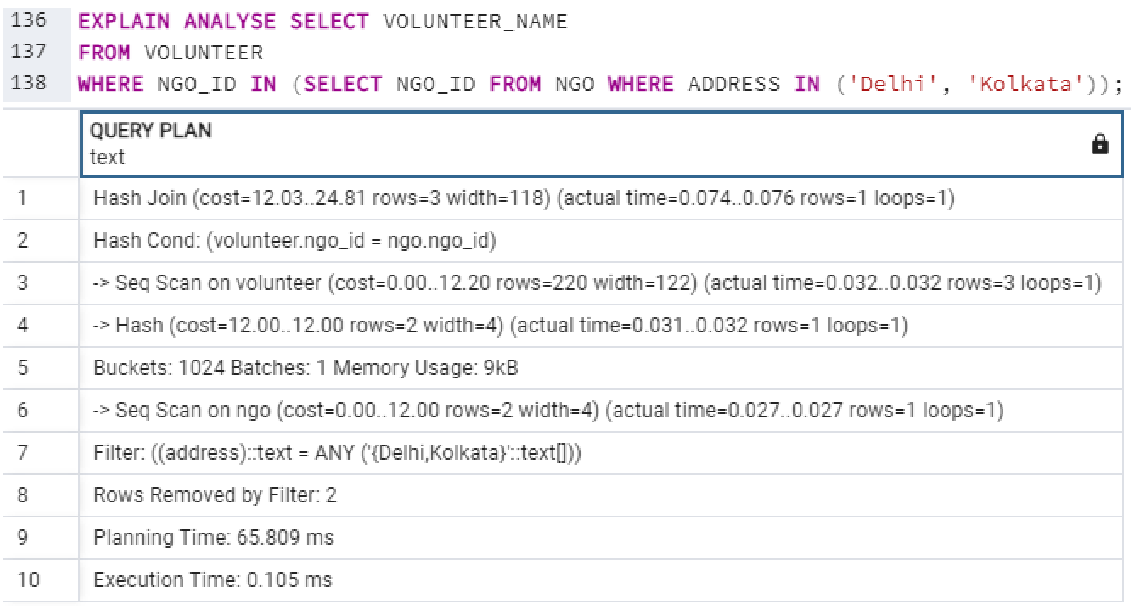
a. **Simple select query**



```
EXPLAIN ANALYSE SELECT * FROM DONOR;
```

	QUERY PLAN text
1	Seq Scan on donor (cost=0.00..12.50 rows=250 width=292) (actual time=0.014..0.015 rows=3 loops...
2	Planning Time: 0.069 ms
3	Execution Time: 0.026 ms

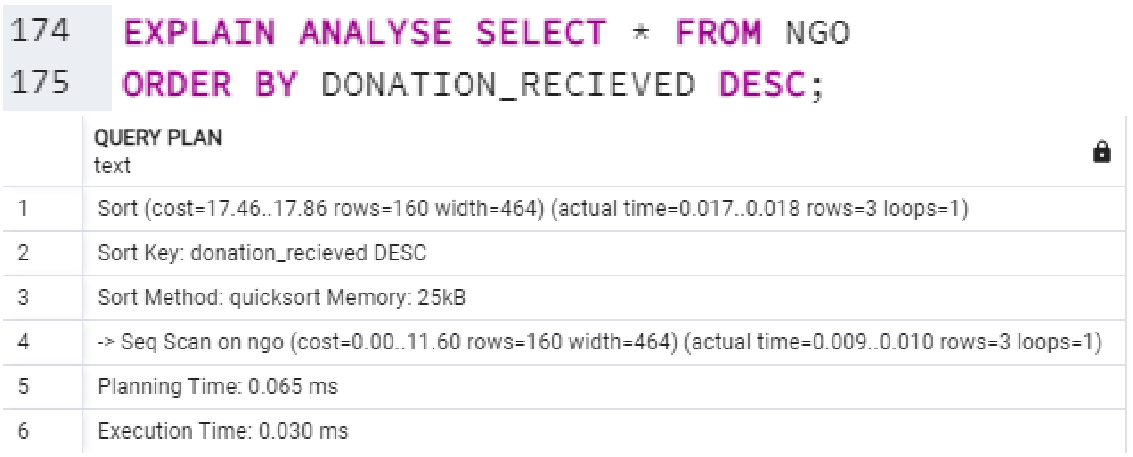
b. Select query with where clause



```sql
EXPLAIN ANALYSE SELECT VOLUNTEER_NAME
FROM VOLUNTEER
WHERE NGO_ID IN (SELECT NGO_ID FROM NGO WHERE ADDRESS IN ('Delhi', 'Kolkata'));
```

	QUERY PLAN text
1	Hash Join (cost=12.03..24.81 rows=3 width=118) (actual time=0.074..0.076 rows=1 loops=1)
2	Hash Cond: (volunteer.ngo_id = ngo.ngo_id)
3	-> Seq Scan on volunteer (cost=0.00..12.20 rows=220 width=122) (actual time=0.032..0.032 rows=3 loops=1)
4	-> Hash (cost=12.00..12.00 rows=2 width=4) (actual time=0.031..0.032 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on ngo (cost=0.00..12.00 rows=2 width=4) (actual time=0.027..0.027 rows=1 loops=1)
7	Filter: ((address)::text = ANY ('{Delhi,Kolkata}'::text[]))
8	Rows Removed by Filter: 2
9	Planning Time: 65.809 ms
10	Execution Time: 0.105 ms

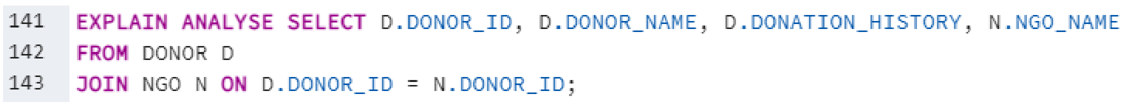
c. Select query with order by query



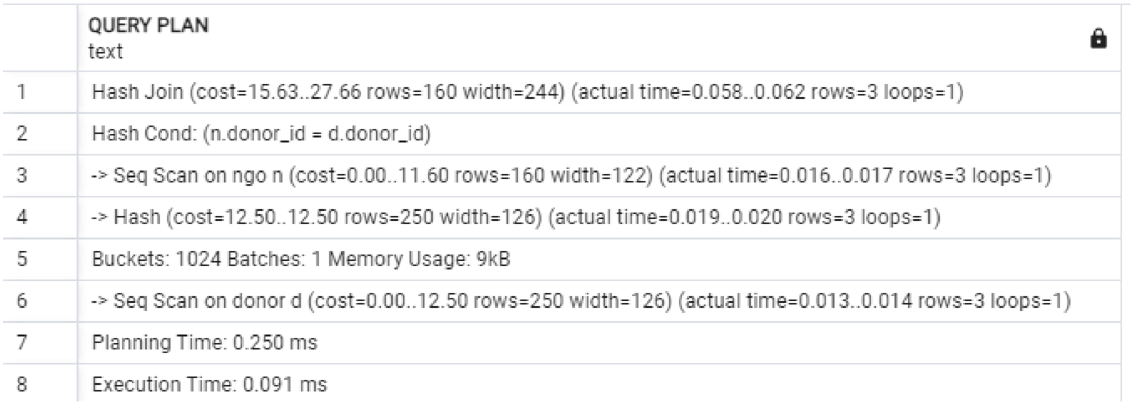
```sql
EXPLAIN ANALYSE SELECT * FROM NGO
ORDER BY DONATION_RECIEVED DESC;
```

	QUERY PLAN text
1	Sort (cost=17.46..17.86 rows=160 width=464) (actual time=0.017..0.018 rows=3 loops=1)
2	Sort Key: donation_recieved DESC
3	Sort Method: quicksort Memory: 25kB
4	-> Seq Scan on ngo (cost=0.00..11.60 rows=160 width=464) (actual time=0.009..0.010 rows=3 loops=1)
5	Planning Time: 0.065 ms
6	Execution Time: 0.030 ms

d. Select query with JOIN

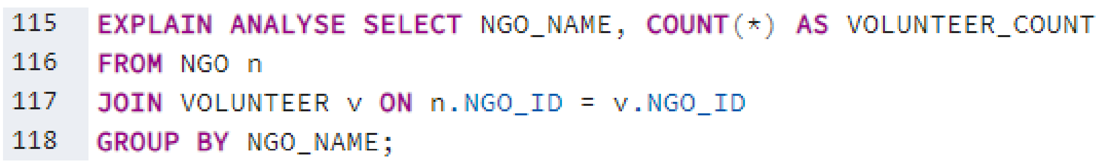


```sql
EXPLAIN ANALYSE SELECT D.DONOR_ID, D.DONOR_NAME, D.DONATION_HISTORY, N.NGO_NAME
FROM DONOR D
JOIN NGO N ON D.DONOR_ID = N.DONOR_ID;
```

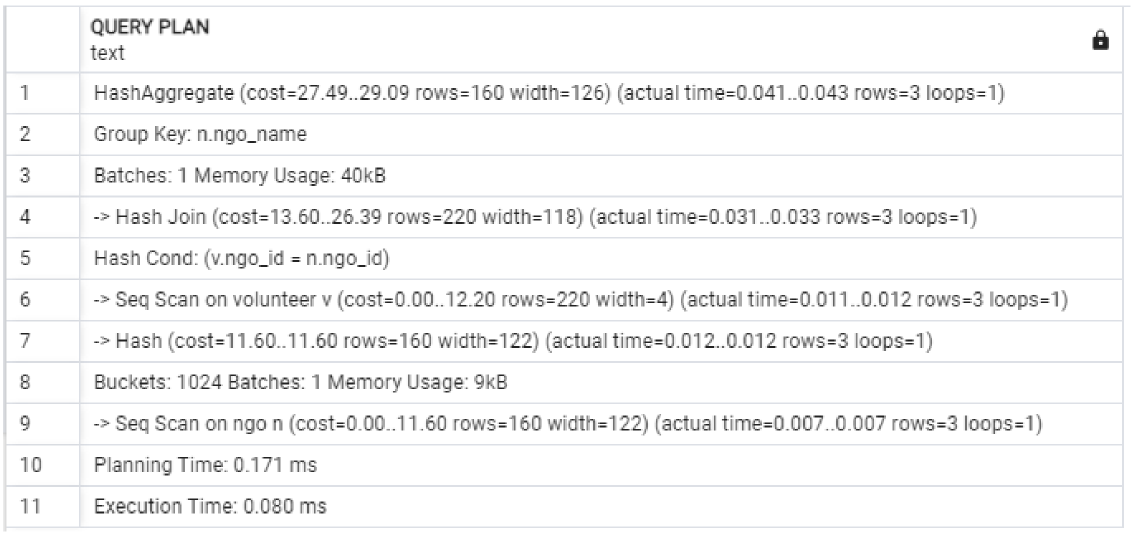


```
   QUERY PLAN
   text
1  Hash Join (cost=15.63..27.66 rows=160 width=244) (actual time=0.058..0.062 rows=3 loops=1)
2  Hash Cond: (n.donor_id = d.donor_id)
3  -> Seq Scan on ngo n (cost=0.00..11.60 rows=160 width=122) (actual time=0.016..0.017 rows=3 loops=1)
4  -> Hash (cost=12.50..12.50 rows=250 width=126) (actual time=0.019..0.020 rows=3 loops=1)
5  Buckets: 1024 Batches: 1 Memory Usage: 9kB
6  -> Seq Scan on donor d (cost=0.00..12.50 rows=250 width=126) (actual time=0.013..0.014 rows=3 loops=1)
7  Planning Time: 0.250 ms
8  Execution Time: 0.091 ms
```

e. Select query with aggregation



```sql
EXPLAIN ANALYSE SELECT NGO_NAME, COUNT(*) AS VOLUNTEER_COUNT
FROM NGO n
JOIN VOLUNTEER v ON n.NGO_ID = v.NGO_ID
GROUP BY NGO_NAME;
```



```
    QUERY PLAN
    text
1   HashAggregate (cost=27.49..29.09 rows=160 width=126) (actual time=0.041..0.043 rows=3 loops=1)
2   Group Key: n.ngo_name
3   Batches: 1 Memory Usage: 40kB
4   -> Hash Join (cost=13.60..26.39 rows=220 width=118) (actual time=0.031..0.033 rows=3 loops=1)
5   Hash Cond: (v.ngo_id = n.ngo_id)
6   -> Seq Scan on volunteer v (cost=0.00..12.20 rows=220 width=4) (actual time=0.011..0.012 rows=3 loops=1)
7   -> Hash (cost=11.60..11.60 rows=160 width=122) (actual time=0.012..0.012 rows=3 loops=1)
8   Buckets: 1024 Batches: 1 Memory Usage: 9kB
9   -> Seq Scan on ngo n (cost=0.00..11.60 rows=160 width=122) (actual time=0.007..0.007 rows=3 loops=1)
10  Planning Time: 0.171 ms
11  Execution Time: 0.080 ms
```

Post Lab Question:

1. **Illustrate with an example Heuristic based query optimization with suitable example**

Conclusion: