K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: A1 Roll No.:
16010120015 YASH
16010120006 DIKSHITA
16010120018 JINAY

Experiment / assignment / tutorial No. 7

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Implementing indexing and query processing

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

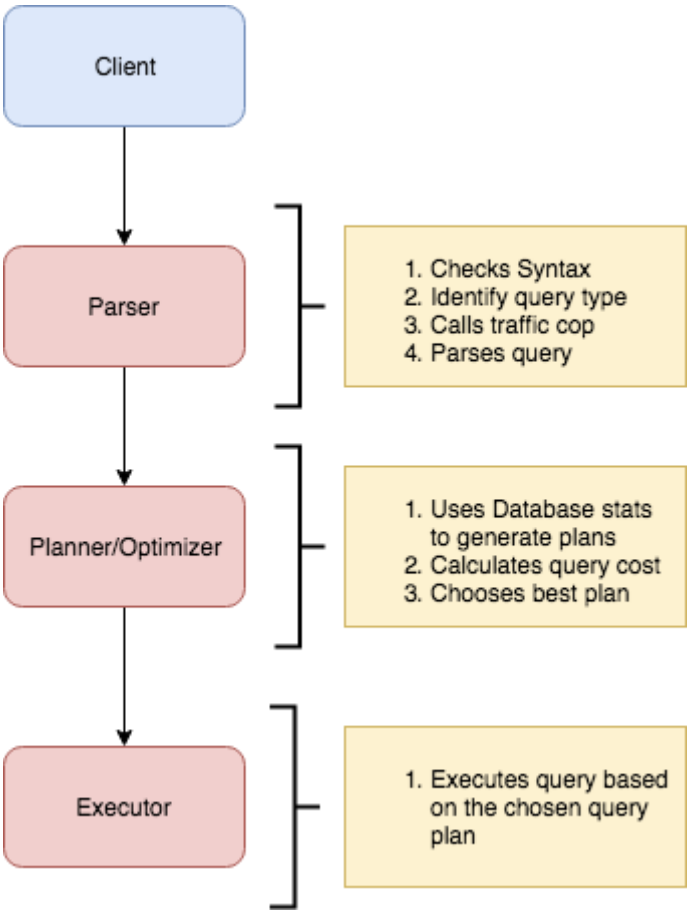
Expected Outcome of Experiment:
CO 3: Use SQL for relational database creation , maintenance and query processing

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

Theory



Postgres query life cyce

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 PostgreSQL is based on pure cost-based optimization;

EXPLAIN command:

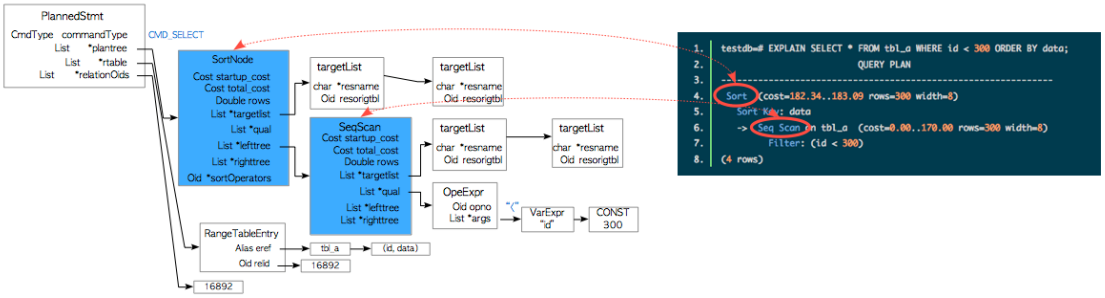
EXPLAIN command:

This command displays the execution plan that the PostgreSQL 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 PostgreSQL displays the plan tree itself. A specific example is shown below.

1. testdb=# 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.



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.

```
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.

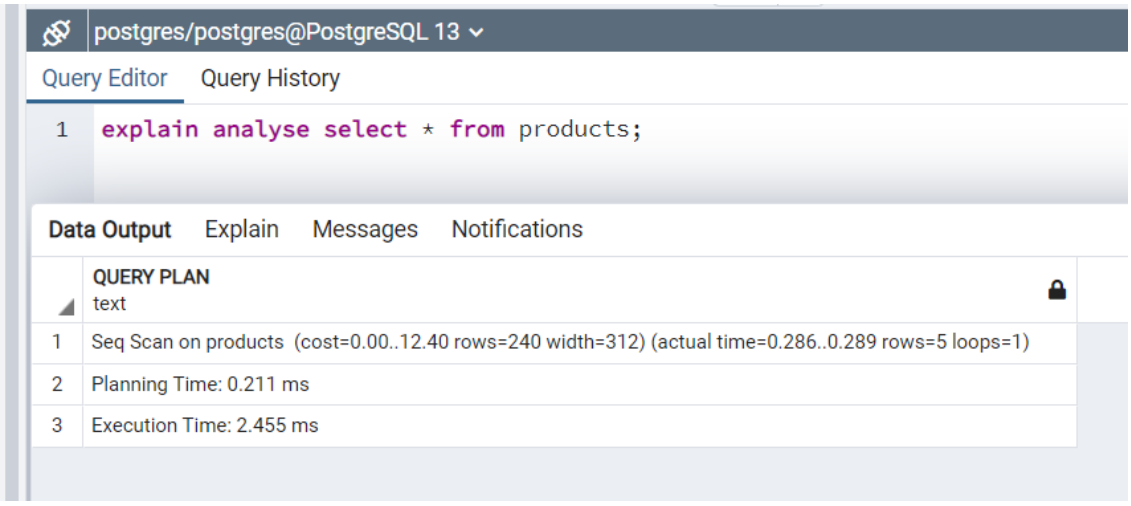
Query plans for select with where clause can be sequential scan, Index Scan, Index only Scan, Bitmap Index Scan etc.

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

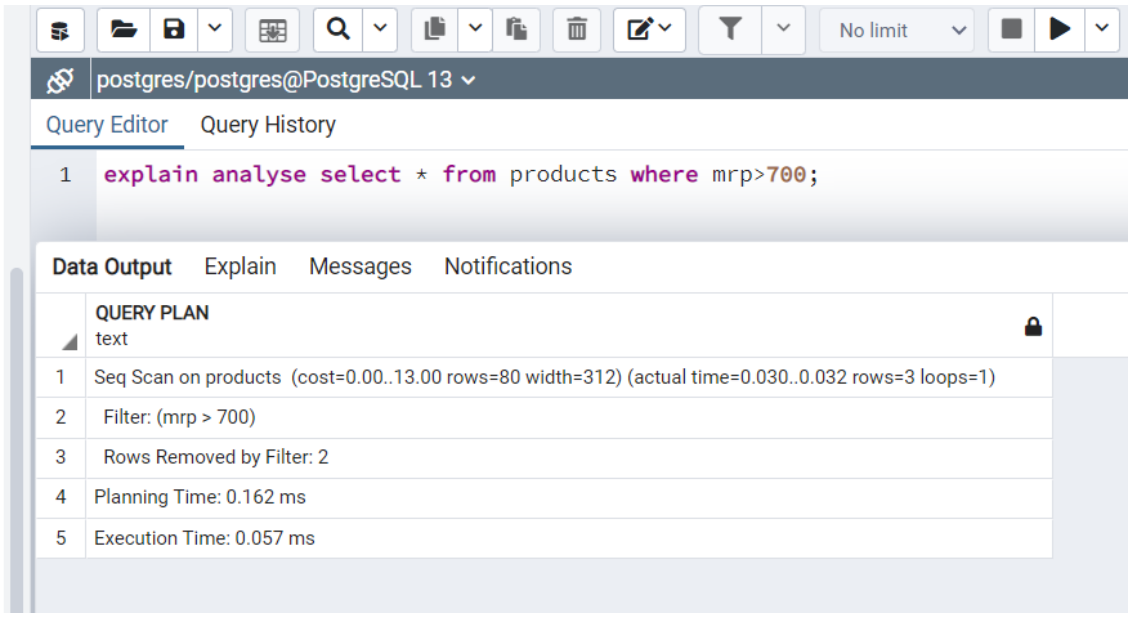
Implementation Screenshots:

Demonstrate query processing for the following types of query on your database

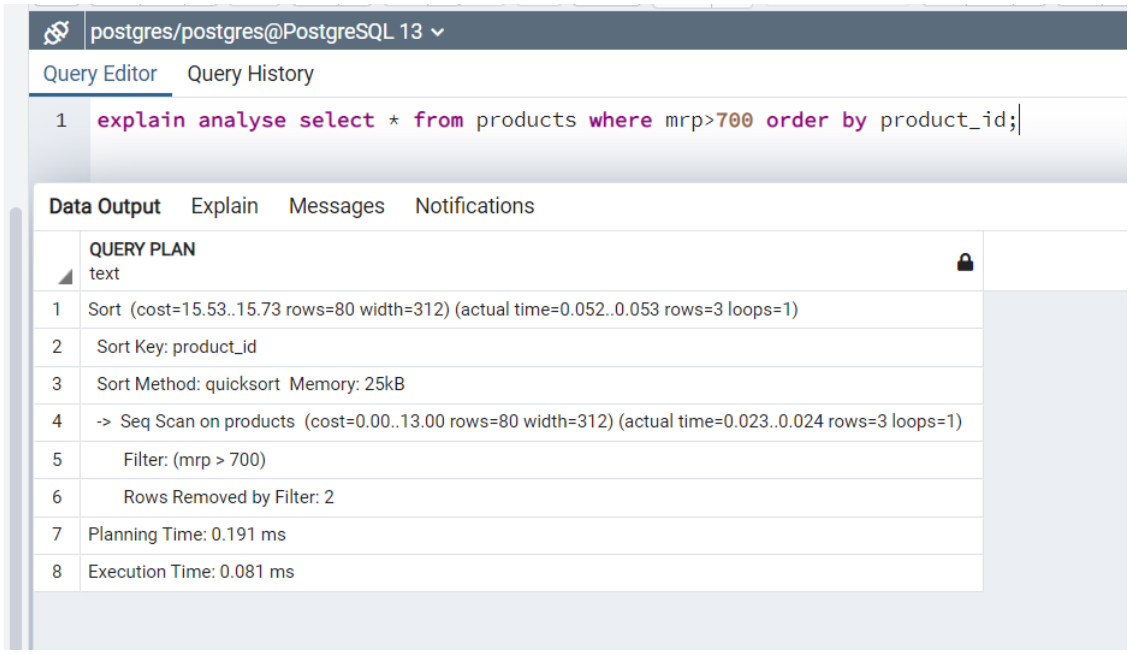
a. Simple select query



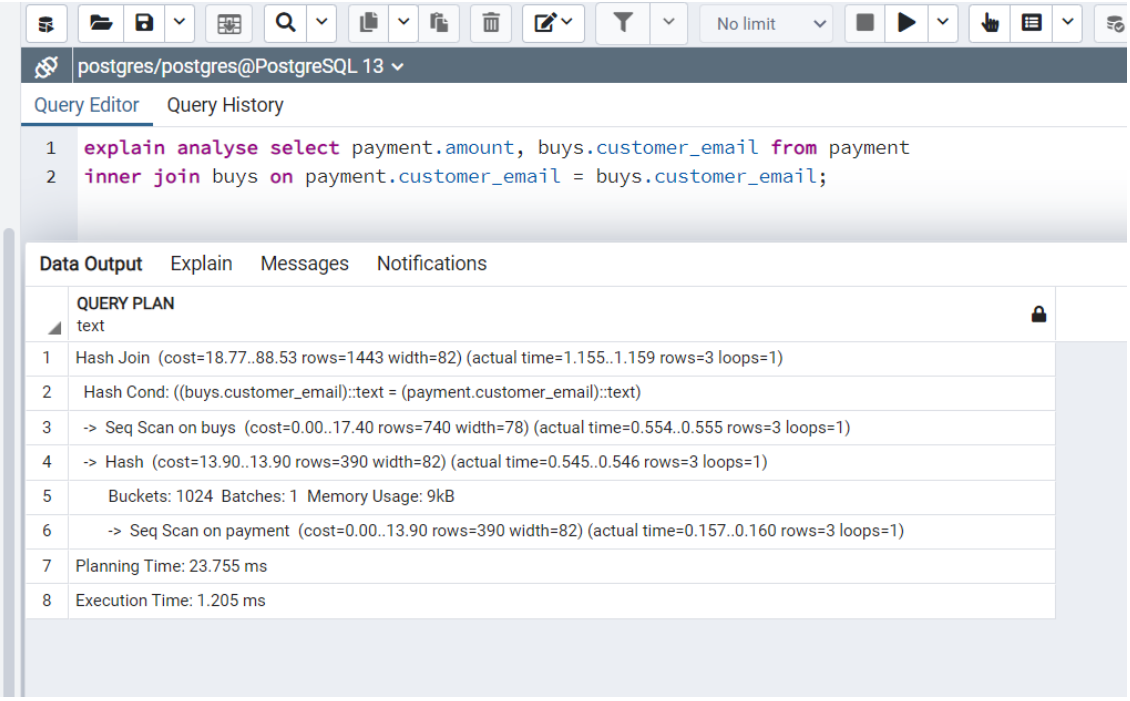
b. Select query with where clause



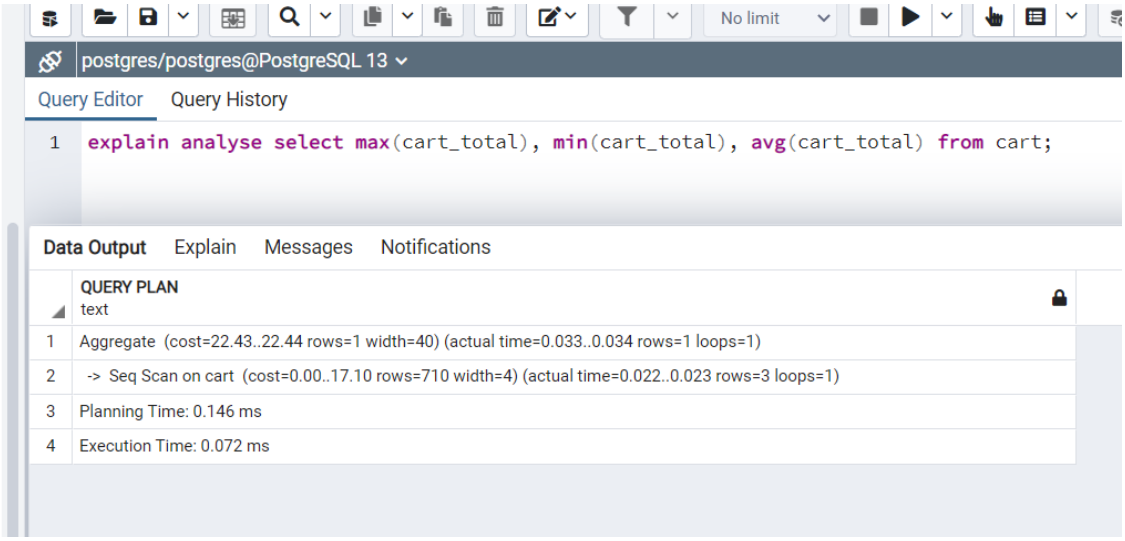
c. Select query with order by query



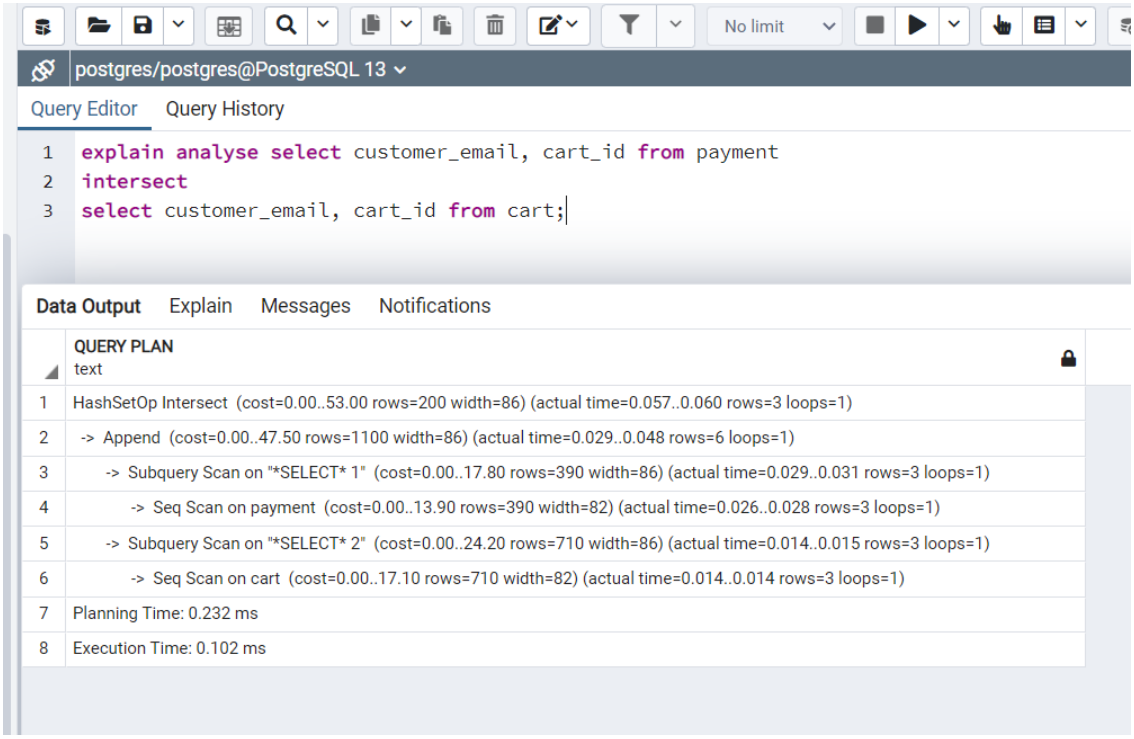
d. Select query with JOIN

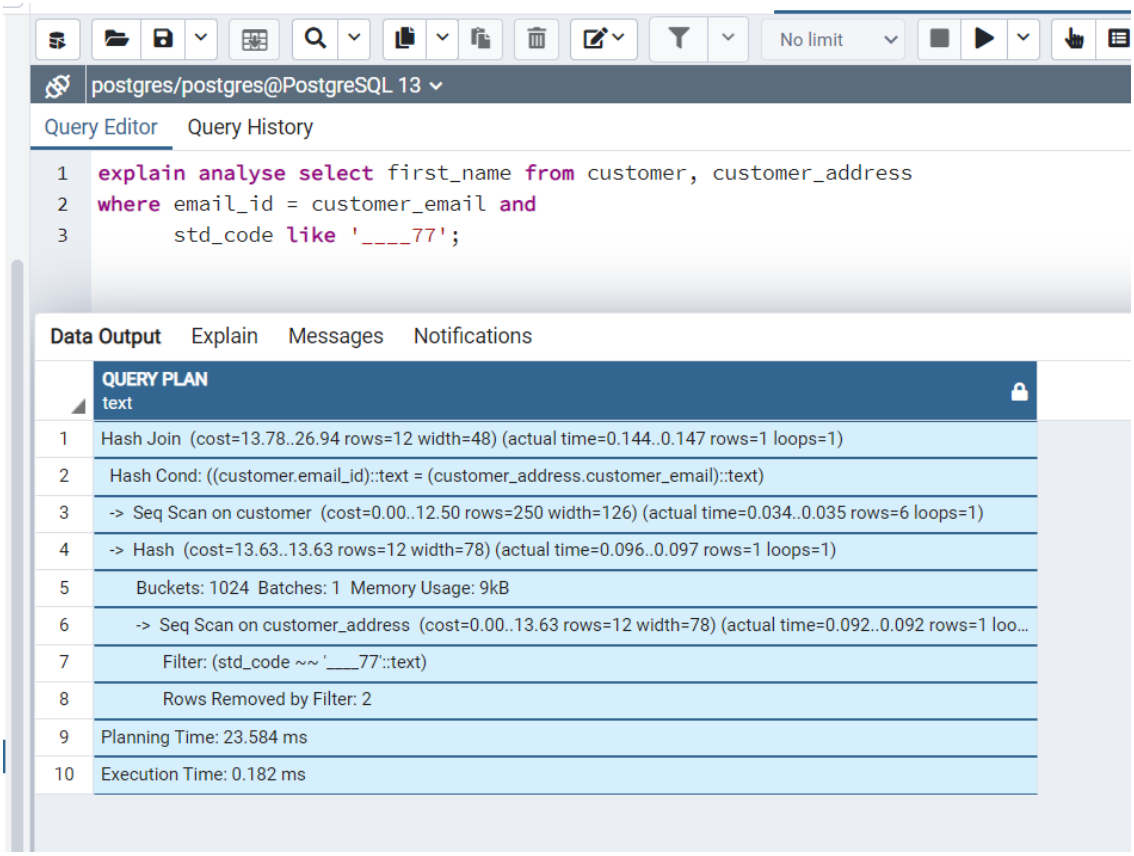


e. Select query with aggregation

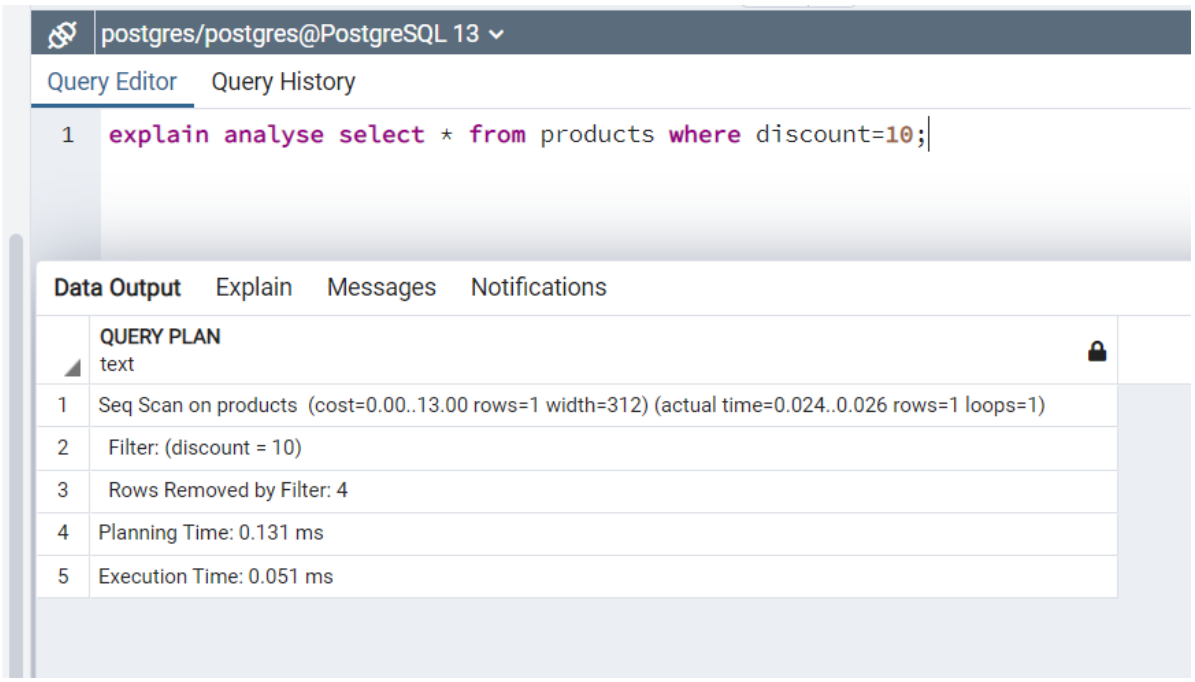

No limit
postgres/postgres@PostgreSQL 13
Query Editor
Query History
1 explain analyse select max(cart_total), min(cart_total), avg(cart_total) from cart;
Data Output
Explain
Messages
Notifications
QUERY PLAN
text
1 Aggregate (cost=22.43..22.44 rows=1 width=40) (actual time=0.033..0.034 rows=1 loops=1)
2 -> Seq Scan on cart (cost=0.00..17.10 rows=710 width=4) (actual time=0.022..0.023 rows=3 loops=1)
3 Planning Time: 0.146 ms
4 Execution Time: 0.072 ms

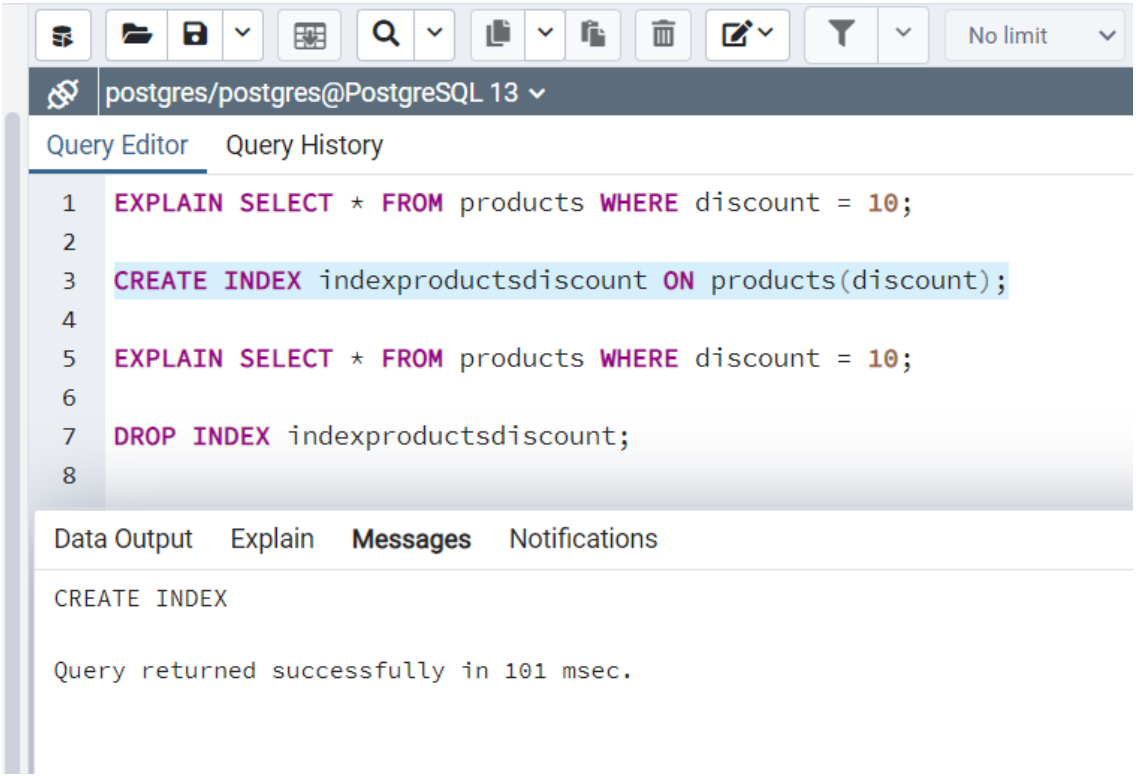
Other:

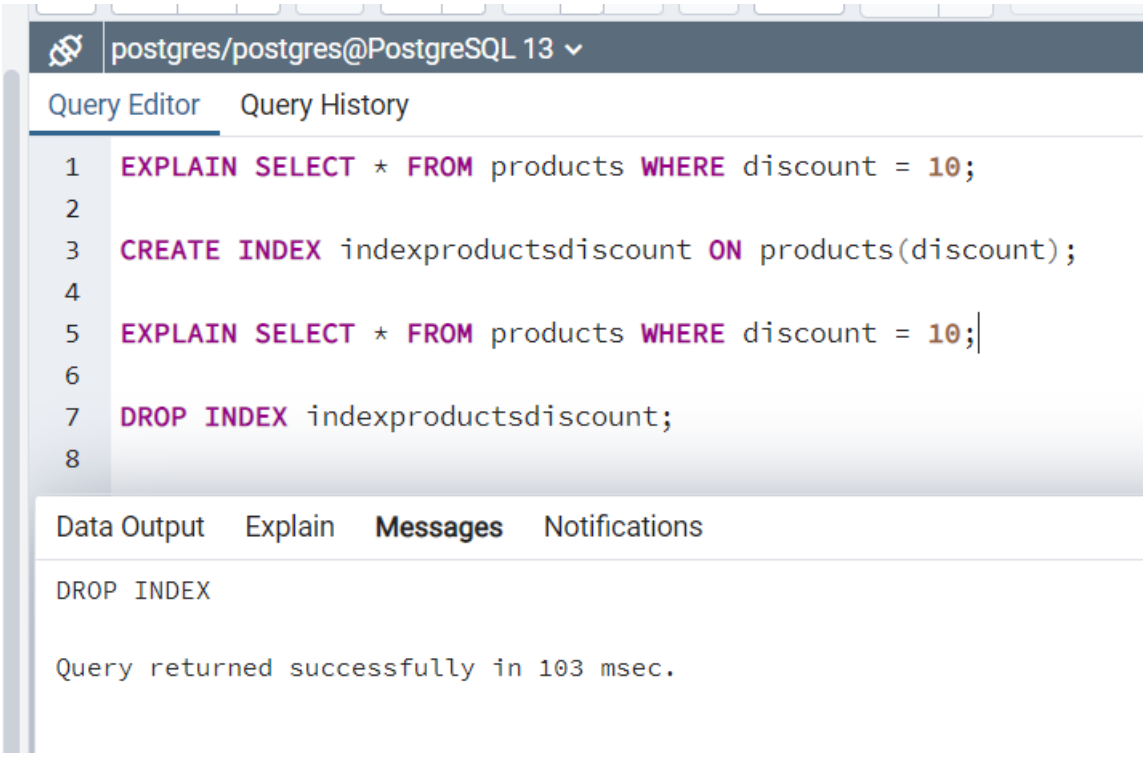

No limit
postgres/postgres@PostgreSQL 13
Query Editor
Query History
1 explain analyse select customer_email, cart_id from payment
2 intersect
3 select customer_email, cart_id from cart;
Data Output
Explain
Messages
Notifications
QUERY PLAN
text
1 HashSetOp Intersect (cost=0.00..53.00 rows=200 width=86) (actual time=0.057..0.060 rows=3 loops=1)
2 -> Append (cost=0.00..47.50 rows=1100 width=86) (actual time=0.029..0.048 rows=6 loops=1)
3 -> Subquery Scan on "*SELECT* 1" (cost=0.00..17.80 rows=390 width=86) (actual time=0.029..0.031 rows=3 loops=1)
4 -> Seq Scan on payment (cost=0.00..13.90 rows=390 width=82) (actual time=0.026..0.028 rows=3 loops=1)
5 -> Subquery Scan on "*SELECT* 2" (cost=0.00..24.20 rows=710 width=86) (actual time=0.014..0.015 rows=3 loops=1)
6 -> Seq Scan on cart (cost=0.00..17.10 rows=710 width=82) (actual time=0.014..0.014 rows=3 loops=1)
7 Planning Time: 0.232 ms
8 Execution Time: 0.102 ms


No limit
postgres/postgres@PostgreSQL 13
Query Editor
Query History
1 explain analyse select first_name from customer, customer_address
2 where email_id = customer_email and
3 std_code like '____77';
Data Output
Explain
Messages
Notifications
QUERY PLAN
text
1 Hash Join (cost=13.78..26.94 rows=12 width=48) (actual time=0.144..0.147 rows=1 loops=1)
2 Hash Cond: ((customer.email_id)::text = (customer_address.customer_email)::text)
3 -> Seq Scan on customer (cost=0.00..12.50 rows=250 width=126) (actual time=0.034..0.035 rows=6 loops=1)
4 -> Hash (cost=13.63..13.63 rows=12 width=78) (actual time=0.096..0.097 rows=1 loops=1)
5 Buckets: 1024 Batches: 1 Memory Usage: 9kB
6 -> Seq Scan on customer_address (cost=0.00..13.63 rows=12 width=78) (actual time=0.092..0.092 rows=1 loo...
7 Filter: (std_code ~~ '____77'::text)
8 Rows Removed by Filter: 2
9 Planning Time: 23.584 ms
10 Execution Time: 0.182 ms

Indexing:


postgres/postgres@PostgreSQL 13
Query Editor
Query History
1 explain analyse select * from products where discount=10;
Data Output
Explain
Messages
Notifications
QUERY PLAN
text
1 Seq Scan on products (cost=0.00..13.00 rows=1 width=312) (actual time=0.024..0.026 rows=1 loops=1)
2 Filter: (discount = 10)
3 Rows Removed by Filter: 4
4 Planning Time: 0.131 ms
5 Execution Time: 0.051 ms


No limit
postgres/postgres@PostgreSQL 13
Query Editor
Query History
1 EXPLAIN SELECT * FROM products WHERE discount = 10;
2
3 CREATE INDEX indexproductsdiscount ON products(discount);
4
5 EXPLAIN SELECT * FROM products WHERE discount = 10;
6
7 DROP INDEX indexproductsdiscount;
8
Data Output
Explain
Messages
Notifications
CREATE INDEX
Query returned successfully in 101 msec.


postgres/postgres@PostgreSQL 13
Query Editor
Query History
1 EXPLAIN SELECT * FROM products WHERE discount = 10;
2
3 CREATE INDEX indexproductsdiscount ON products(discount);
4
5 EXPLAIN SELECT * FROM products WHERE discount = 10;
6
7 DROP INDEX indexproductsdiscount;
8
Data Output
Explain
Messages
Notifications
DROP INDEX
Query returned successfully in 103 msec.

Implement

- Indexing on foreign key

-Indexing on composite key

-Indexing on secondary key

CREATING INDEX

CREATE INDEX SSNO ON SOLDIERS(SSNO DESC);

```
392  //CREATING INDEX
393
394  CREATE INDEX SSNO ON SOLDIERS(SSNO DESC);
395
```

Data Output | Explain | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 450 msec.
```

EXPLAIN SELECT * FROM SOLDIERS WHERE SSNO >604 order by NAME;

```
437
438  EXPLAIN SELECT * FROM SOLDIERS WHERE SSNO >604 order by NAME;
439
440
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=1.19..1.20 rows=4 width=306)
2	[...] Sort Key: name
3	[...] -> Seq Scan on soldiers (cost=0.00..1.15 rows=4 width=306)
4	[...] Filter: (ssno > 604)

Simple select query

EXPLAIN (ANALYZE) SELECT * FROM SOLDIERS;

```
438  EXPLAIN (ANALYZE) SELECT * FROM SOLDIERS;
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on soldiers (cost=0.00..1.12 rows=12 width=306) (actual time=0.575..0.577 rows=12 loops=1)
2	Planning Time: 146.001 ms
3	Execution Time: 0.590 ms

Select query with query

EXPLAIN ANALYZE SELECT * FROM SOLDIERS WHERE NAME = 'YOGENDRA SINGH';

```
402
403  //EXPLAIN ANALYZER
404
405  EXPLAIN ANALYZE SELECT * FROM SOLDIERS WHERE NAME = 'YOGENDRA SINGH';
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on soldiers (cost=0.00..1.15 rows=1 width=306) (actual t...
2	[...] Filter: ((name)::text = 'YOGENDRA SINGH'::text)
3	[...] Rows Removed by Filter: 11
4	Planning Time: 0.089 ms
5	Execution Time: 0.023 ms

Select query with order by query

EXPLAIN ANALYZE SELECT * FROM SOLDIERS WHERE SSNO >604 order by NAME;

```
407  EXPLAIN ANALYZE SELECT * FROM SOLDIERS WHERE SSNO >604 order by NAME;
408
409
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=1.19..1.20 rows=4 width=306) (actual time=0.051..0.052 rows=8 loops=1)
2	[...] Sort Key: name
3	[...] Sort Method: quicksort Memory: 26kB
4	[...] -> Seq Scan on soldiers (cost=0.00..1.15 rows=4 width=306) (actual time=0.009..0.010 rows=8 loops=1)
5	[...] Filter: (ssno > 604)
6	[...] Rows Removed by Filter: 4

Select query with join

EXPLAIN ANALYZE SELECT *FROM RESOURCES
RIGHT JOIN ITEMS
ON resources.rname = items.ideptname;

```
409
410  EXPLAIN ANALYZE SELECT *FROM RESOURCES
411  RIGHT JOIN ITEMS
412  ON resources.rname = items.ideptname;
413
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Hash Left Join (cost=15.85..29.02 rows=250 width=568) (actual time=9.653..9.663 rows=9 loops=1)
2	[...] Hash Cond: (items.ideptname = resources.rname)
3	[...] -> Seq Scan on items (cost=0.00..12.50 rows=250 width=288) (actual time=0.025..0.027 rows=9 loops=1)
4	[...] -> Hash (cost=12.60..12.60 rows=260 width=280) (actual time=9.595..9.596 rows=5 loops=1)
5	[...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	[...] -> Seq Scan on resources (cost=0.00..12.60 rows=260 width=280) (actual time=0.009..0.019 rows=5 loops=1)
7	Planning Time: 30.914 ms
8	Execution Time: 9.751 ms

Select query with aggregation

EXPLAIN ANALYZE SELECT MAX(SALARY) , MIN(SALARY) , AVG(SALARY) FROM SOLDIERS

```
415
416  //Select query with aggregation
417  EXPLAIN ANALYZE SELECT MAX(SALARY) , MIN(SALARY)  , AVG(SALARY) FROM SOLDIERS
418
419
420
421
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Aggregate (cost=1.21..1.22 rows=1 width=40) (actual time=6.785..6.786 rows=1 loops=1)
2	[...] -> Seq Scan on soldiers (cost=0.00..1.12 rows=12 width=4) (actual time=0.010..0.014 rows=12 loops=1)
3	Planning Time: 0.162 ms
4	Execution Time: 6.815 ms

Indexing on primary key

EXPLAIN ANALYZE SELECT SSNO FROM SOLDIERS

```
418
419  //PRIMARY KEY
420  EXPLAIN ANALYZE SELECT SSNO FROM SOLDIERS
421
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on soldiers (cost=0.00..1.12 rows=12 width=4) (actual time=0.009..0.010 rows=12 loops=1)
2	Planning Time: 0.074 ms
3	Execution Time: 0.018 ms

Indexing on secondary key

EXPLAIN ANALYZE SELECT NAME FROM SOLDIERS

```
422  //SECONDARY KEY
423  EXPLAIN ANALYZE SELECT NAME FROM SOLDIERS
424
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on soldiers (cost=0.00..1.12 rows=12 width=118) (actual time=0.014..0.015 rows=12 loops=1)
2	Planning Time: 0.077 ms
3	Execution Time: 0.028 ms

Indexing on foreign key

CREATE INDEX INDEX2 ON DEPARTMENT(DNAME)

EXPLAIN ANALYZE SELECT DNAME FROM DEPARTMENT

```
429  //FOREIGN KEY
430  CREATE INDEX INDEX2 ON DEPARTMENT(DNAME)
431  EXPLAIN ANALYZE SELECT DNAME FROM DEPARTMENT
432
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on department (cost=0.00..13.30 rows=330 width=104) (actual time=0.028..0.029 rows=10 loops=1)
2	Planning Time: 21.575 ms
3	Execution Time: 0.042 ms

Indexing on composite key

CREATE INDEX INDEX4 ON DEPARTMENT(DNAME,DRNO)

EXPLAIN ANALYZE SELECT DNAME,DRNO FROM DEPARTMENT

```
433  //COMPOSITE KEY
434  CREATE INDEX INDEX4 ON DEPARTMENT(DNAME,DRNO)
435  EXPLAIN ANALYZE SELECT DNAME,DRNO FROM DEPARTMENT
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on department (cost=0.00..1.10 rows=10 width=108) (actual time=0.013..0.015 rows=10 loops=1)
2	Planning Time: 1.524 ms
3	Execution Time: 0.027 ms

Post Lab Question:

Q1. Illustrate with an example Heuristic based query optimization with suitable example

Ans.

Heuristic based optimization uses rule-based optimization approaches for query optimization. These algorithms have polynomial time and space complexity, which is lower than the exponential complexity of exhaustive search-based algorithms.

Some of the common rules are −

- Perform select and project operations before join operations. This is done by moving the select and project operations down the query tree. This reduces the number of tuples available for join.
- Perform the most restrictive select/project operations at first before the other operations.
- Avoid cross-product operation since they result in very large-sized intermediate tables.

Process for heuristics optimization

1. The parser of a high-level query generates an initial internal representation;
2. Apply heuristics rules to optimize the internal representation.
3. A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.
4. The main heuristic is to apply first the operations that reduce the size of intermediate results.

E.g., Apply SELECT and PROJECT operations before applying the JOIN or other binary operations.

Query tree:

- A tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as leaf nodes of the tree, and represents the relational algebra operations as internal nodes.

Query graph:

- A graph data structure that corresponds to a relational calculus expression. It does not indicate an order on which operations to perform first. There is only a single graph corresponding to each query.

- SQL query:

SELECT LNAME, FNAME FROM EMPLOYEE WHERE SALARY < (SELECT MAX (SALARY) FROM EMPLOYEE WHERE DNO=5);

(SELECT MAX (SALARY) FROM EMPLOYEE WHERE DNO=5)

MAXSALARY (σDNO = 5(EMPLOY EE))

The Outer block SELECT LNAME;

FNAME FROM EMPLOY EE WHERE SALARY < C

Q2. List different algorithms of Select, Project , Join with cost

Ans.

1. Select :-
    1) Linear search
    2) Binary search
    3) Using a primary index
    4) Using a primary index to retrieve multiple records: The comparison condition is >, ≤, <, ≥ on a key field with a primary index .
    5) Using a clustering index to retrieve multiple records: The selection condition involves an equality comparison on a non-key attribute with a clustering index.

2. Project:-
    a. CARTESIAN PRODUCT operation
    b. UNION operations
    c. INTERSECTION operations
    d. DIFFERENCE operations
3. Join :-
    a. Nested loop join.
    b. Block nested loop join.
    c. Single-loop join
    d. Sort-merge join
    e. Hash-join