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

Batch: A2	**Roll No.: 1911027**
Experiment / assignment / tutorial No. 9	
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/MySQL

Theory:

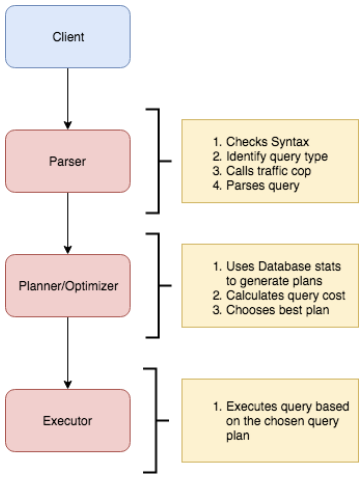
A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.

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



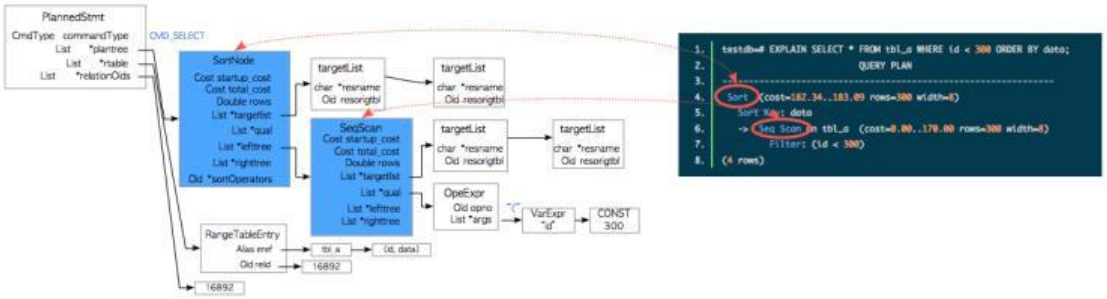
Planner and Executor:
The planner receives a query tree from the rewriter and generates a (query) plan tree that can be processed by the executor most effectively.

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.



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 :

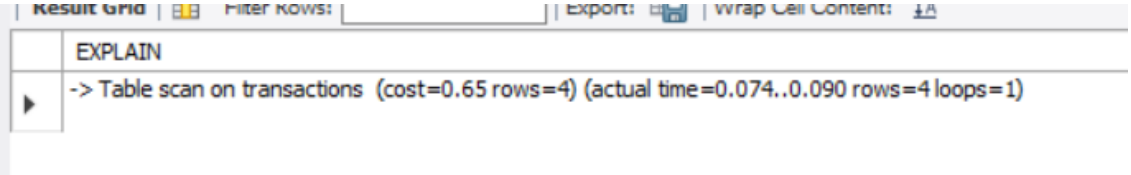
Comprehend how indexes improves the performance of query applied for your database . Demonstrate for the following types of query on your database

a. **Simple select query**
b. **Select query with where clause**
c. **Select query with order by query**
d. **Select query with JOIN**
e. **Select query with aggregation**

A) Simple select query:

Before creating index:

explain analyze select * from transactions;

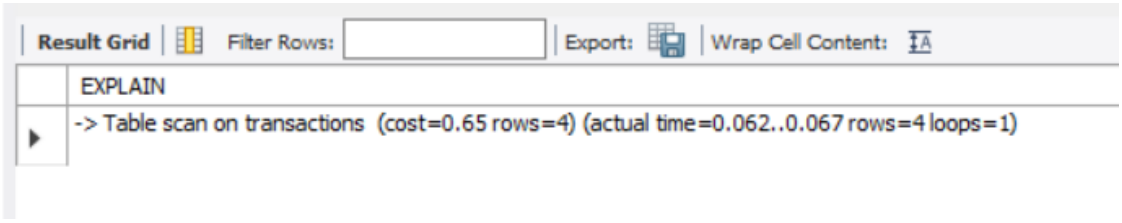


Creating index:

create index index1 on transactions(Trans_ID);

After creating index:

explain analyze select * from transactions;



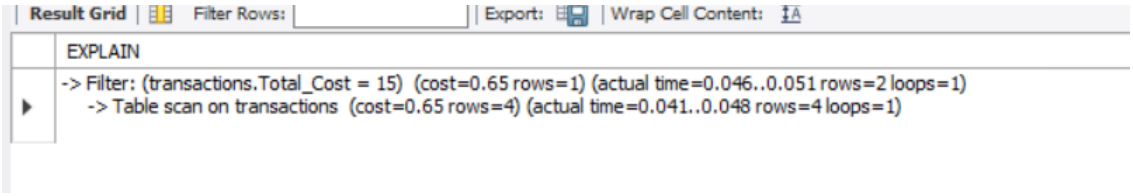
Before creating index actual time= 0.074

After creating index actual time= 0.062

B) Select query with where clause:

Before creating index:

explain analyze select * from transactions where Total_cost=15;

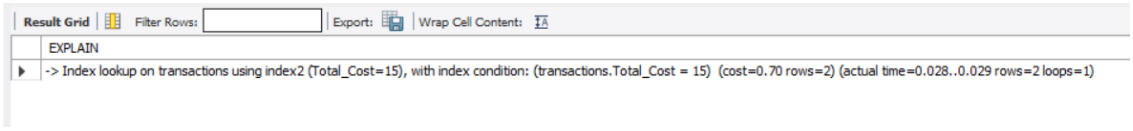


Creating index:

create index index2 on transactions(Total_cost);

After creating index:

explain analyze select * from transactions WHERE Total_cost=15;



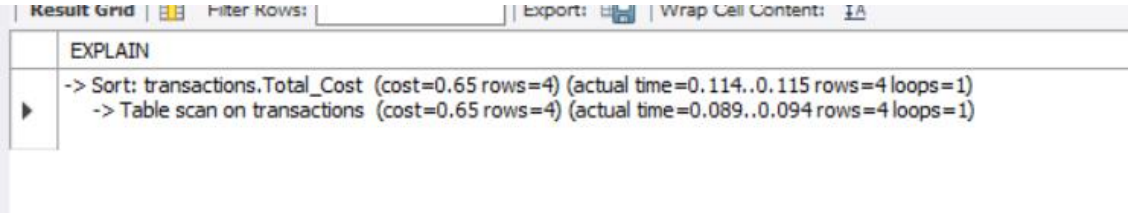
Before creating index actual time= 0.046

After creating index actual time= 0.028

C) Select query with order by clause:

Before creating index:

explain analyze select * from transactions ORDER BY Total_cost;

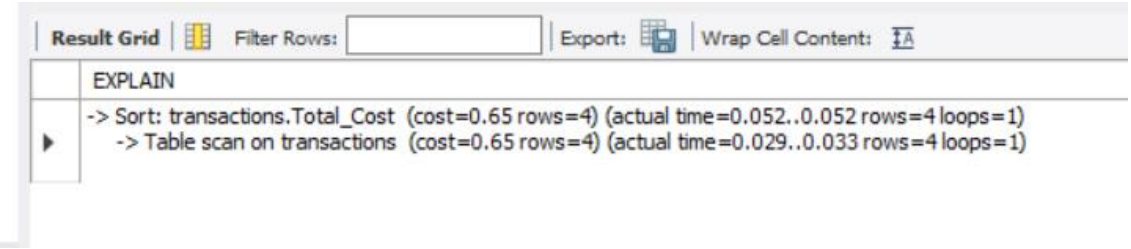


Creating index:

create index index3 on transactions(Total_cost);

After creating index:

explain analyze select * from transactions order by Total_cost;



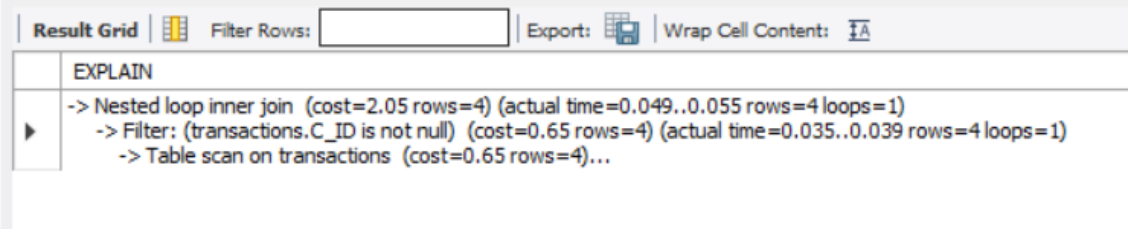
Before creating index actual time= 0.114, 0.089

After creating index actual time= 0.052, 0.029

D) Select query with JOIN

Before creating index:

explain analyze select Cust_ID,C_ID,Trans_ID,C_FName,total_cost from customer join transactions on customer.Cust_id=transactions.C_ID;



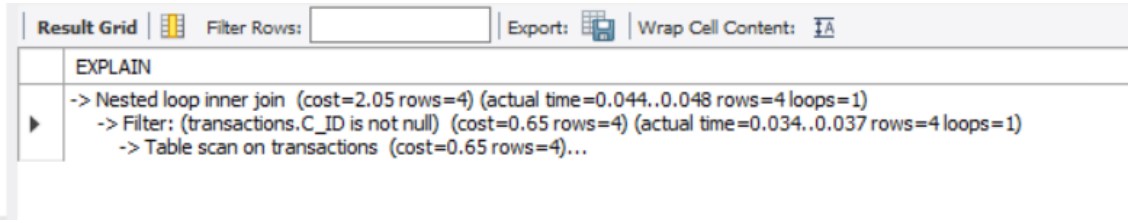
Creating index:

create index index4 on transactions(C_ID);

create index index6 on customer(Cust_ID);

After creating index:

explain analyze select Cust_ID,C_ID,Trans_ID,C_FName,total_cost from customer join transactions on customer.Cust_id=transactions.C_ID;



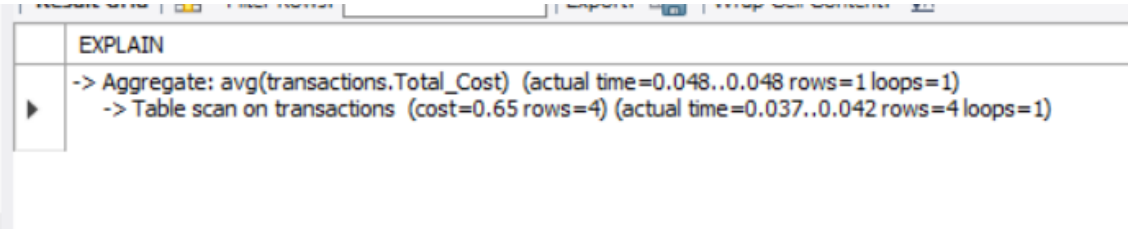
Before creating index actual time= 0.049, 0.035

After creating index actual time= 0.044, 0.034

E) Select quey with aggregation:

Before creating index:

explain analyze select avg(Total_cost) from transactions;

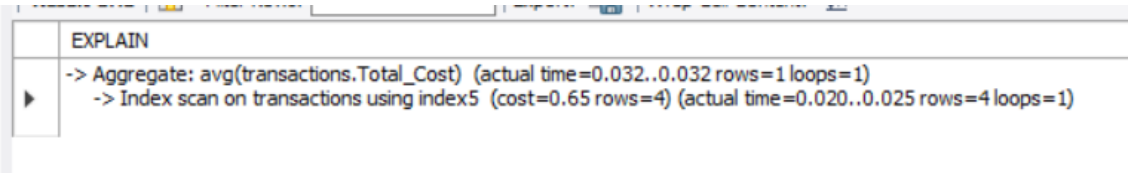


Creating index:

create index index5 on transactions(Total_cost);

After creating index:

explain analyze select avg(Total_cost) from transactions;



Before creating index actual time= 0.048, 0.037

After creating index actual time= 0.032, 0.020

Post Lab Question:

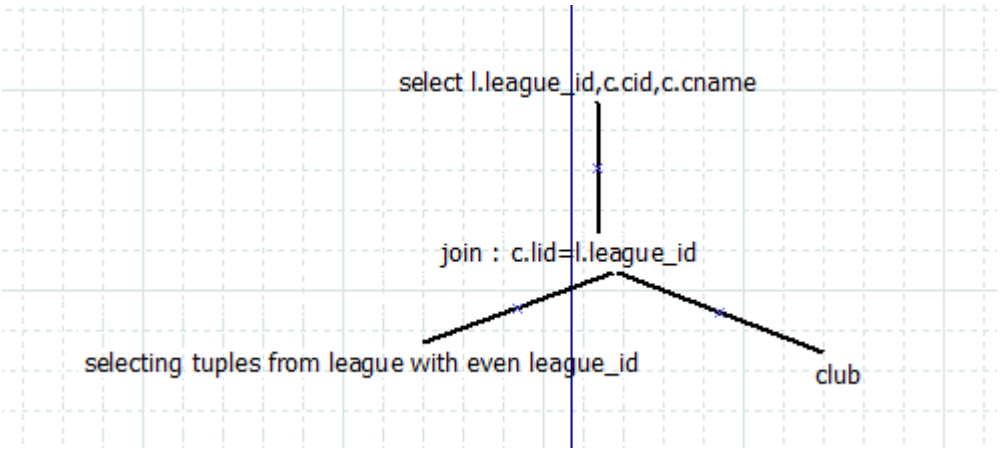
1. **Illustrate with an example Heuristic based query optimization with suitable example**

ANS) Heuristic based optimization uses rule-based optimization approaches for query optimization.
Some of the common heuristic 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.

Eg :
select l.league_id,c.cid,c.cname
from league as l,salary_slip as c
where l.league_id%2=1 and c.lid=l.league_id

In the above query initially select operation is performed so that only the tuples in league having even league_id are selected. After that the join operation is performed. Thus, there are fewer tuples to join.
Query tree :



Conclusion: By performing this experiment completely understood the concept of indexing and query processing in mysql. Also created indexes on our database and seen the reduction in time in execution of queries after creating index.