Unit 5. Query Processing

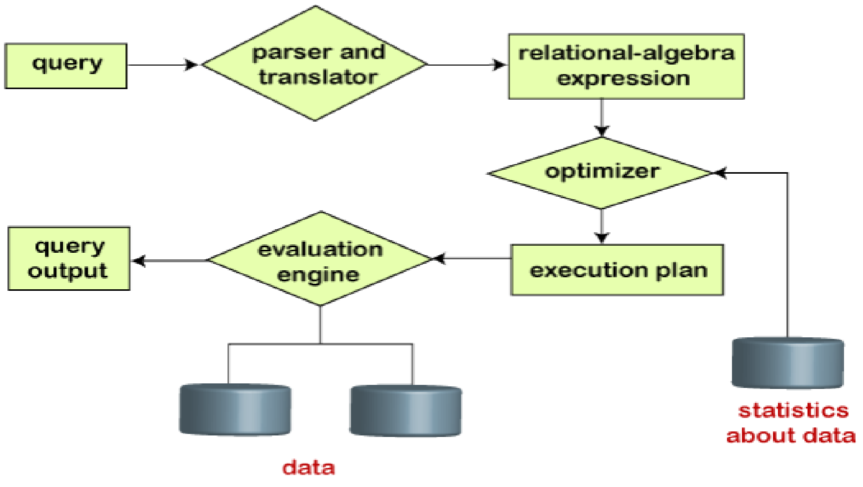
https://github.com/sanjeevlcc/notes_2081/blob/main/DBMS_BIM_BSCIT_BCA/LABS_ON_AIR/14_LAB%2013%20%20Query%20Processing.txt

5.1. Introduction to Query Processing

- Query processing is the series of operations performed by a database management system (DBMS) to interpret and execute user queries efficiently.
- The objective is to transform a high-level declarative query (written in SQL) into a low-level procedural form that the database can execute while optimizing for performance and resource usage.

Key Components of Query Processing

1. **Query Parsing and Translation**
   - The SQL query is parsed to check for syntax and semantic errors.
   - A parse tree or query structure is generated to represent the query logically.
2. **Query Optimization**
   - Multiple strategies (execution plans) are evaluated to execute the query.
   - The most efficient plan, based on cost estimation, is selected.



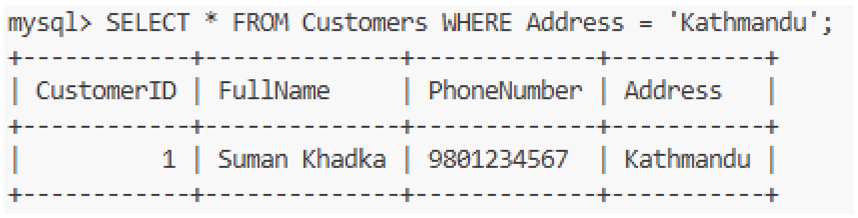
Steps in query processing

3. **Query Evaluation**
   - The selected execution plan is executed by the DBMS.
   - Intermediate results may be stored temporarily to compute the final output.

Stages of Query Processing

1. **Parsing**
   The DBMS parses the SQL query into a parse tree:
   - **Syntax Checking:** Verifies that the query follows SQL rules.
   - **Semantic Checking:** Ensures that referenced tables, columns, and data types exist.

SELECT * FROM Customers WHERE Address = 'Kathmandu';



```
mysql> SELECT * FROM Customers WHERE Address = 'Kathmandu';
+------------+--------------+-------------+-----------+
| CustomerID | FullName     | PhoneNumber | Address   |
+------------+--------------+-------------+-----------+
|          1 | Suman Khadka | 9801234567  | Kathmandu |
+------------+--------------+-------------+-----------+
```

The system verifies that the Customers table and Address column exist.

2. **Query Decomposition**
   - The query is broken into smaller subqueries or steps.
   - Logical equivalences are used to simplify the query.

3. **Optimization**
   - The query optimizer evaluates multiple execution plans.
   - Cost-based optimization considers factors like:
     - Disk I/O operations.
     - CPU time.
     - Memory usage.

4. **Execution**
   - The optimized plan is executed, and the query result is returned to the user.
   - Indexes, joins, and aggregations are handled efficiently.

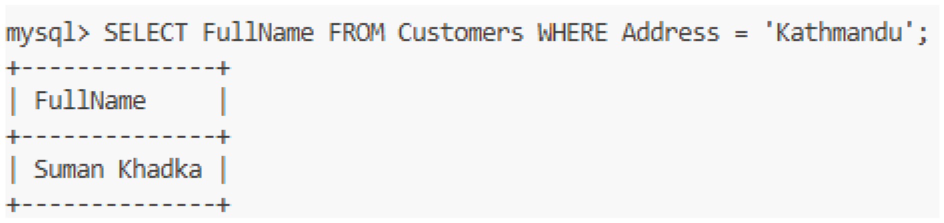
Goals of Query Processing

- Minimize resource usage (CPU, memory, and disk I/O).
- Provide accurate results as quickly as possible.
- Ensure scalability and support for large datasets.

Example of Query Processing

Query: Fetch all customers from Kathmandu.

SELECT FullName FROM Customers WHERE Address = 'Kathmandu';



```
mysql> SELECT FullName FROM Customers WHERE Address = 'Kathmandu';
+--------------+
| FullName     |
+--------------+
| Suman Khadka |
+--------------+
```

Steps:

1. **Parsing:** The query is checked for correctness.
2. **Optimization:** Evaluate if using an index on the Address column will improve performance.
3. **Execution:**
    - Retrieve rows where Address = 'Kathmandu'.
    - Extract the FullName field for each matching record.
4. **Result:**
   A list of customer names from Kathmandu is displayed.

5.2. Query Cost estimation

- Query cost estimation is a critical step in query processing that evaluates the computational resources required to execute a query.
- The goal is to predict and minimize the time and resources consumed, ensuring optimal performance.
- Cost estimation is a vital part of query optimization, as it helps the database management system (DBMS) choose the most efficient execution plan.

Factors Influencing Query Cost

1. **Disk I/O Cost**
    - Measures the cost of reading and writing data blocks from storage.
    - Often the most significant contributor to query cost.
2. **CPU Cost**
    - Includes the cost of processing rows, performing calculations, and executing operations such as joins or aggregations.
3. **Memory Usage**
    - Relates to the amount of temporary storage (RAM) needed for operations like sorting and hashing.

4. **Network Cost**
   - If the database is distributed, network communication between nodes adds to the cost.

Components of Query Cost

1. **Access Costs**
   - Cost of retrieving data from storage (tables or indexes).
   - Example: Sequential scan vs. index scan.
2. **Join Costs**
   - Cost of combining rows from two or more tables.
   - Depends on join algorithms (nested loops, hash join, merge join).
3. **Sorting Costs**
   - Sorting rows for operations like ORDER BY or GROUP BY.
4. **Intermediate Result Costs**
   - Cost of storing and retrieving temporary results during multi-step operations.

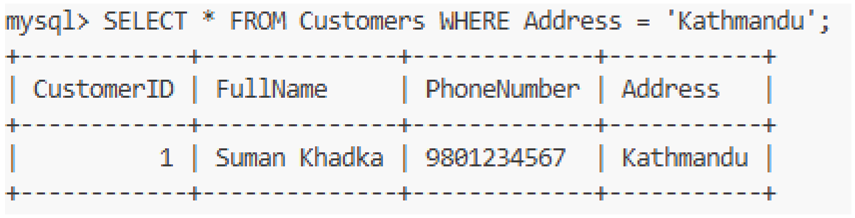
Cost Estimation Techniques

1. **Cardinality Estimation**
   - Predicts the number of rows a query will return.
   - Based on statistics like table size, data distribution, and column selectivity.
2. **Cost Models**
   - DBMS uses cost models to estimate resource usage.
   - Weights are assigned to operations like disk I/O and CPU usage.
3. **Use of Indexes**
   - Evaluates if using an index will reduce the cost of data retrieval.

Example: Query Cost Estimation

Query: Fetch all customers from Kathmandu.

SELECT * FROM Customers WHERE Address = 'Kathmandu';



```
mysql> SELECT * FROM Customers WHERE Address = 'Kathmandu';
+------------+--------------+-------------+-----------+
| CustomerID | FullName     | PhoneNumber | Address   |
+------------+--------------+-------------+-----------+
|          1 | Suman Khadka | 9801234567  | Kathmandu |
+------------+--------------+-------------+-----------+
```

1. **Access Methods**:
   - **Sequential Scan**: Read all rows in the Customers table and filter where Address = 'Kathmandu'.
   - **Index Scan**: Use an index on the Address column to directly locate matching rows.
2. **Cost Comparison**:
   - Sequential Scan: High disk I/O cost for large tables.
   - Index Scan: Lower cost if an index exists on the Address column.

Comparing Query Plans

Plan 1: Sequential Scan

Cost: Disk I/O = High, CPU = Medium, Memory = Low

Plan 2: Index Scan

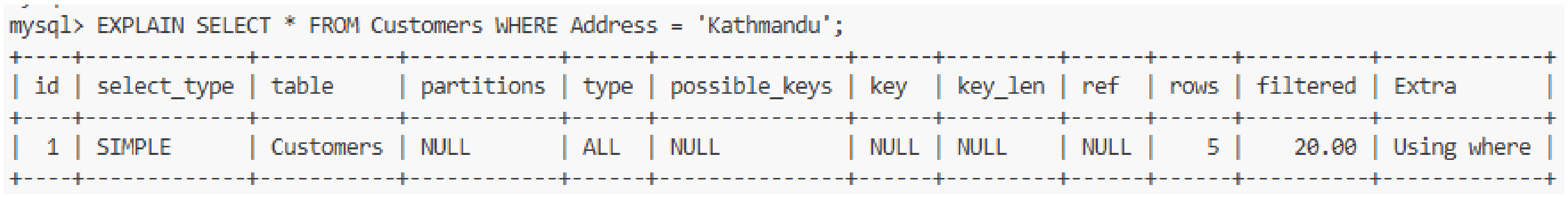
Cost: Disk I/O = Low, CPU = Low, Memory = Low

Optimized Plan Selected: Index Scan (if an index is available).

Tools for Cost Estimation

1. **Explain Plan**
   - Many DBMSs (e.g., MySQL, PostgreSQL) provide EXPLAIN or EXPLAIN ANALYZE commands to visualize query execution plans and their estimated costs.

EXPLAIN SELECT * FROM Customers WHERE Address = 'Kathmandu';



```
mysql> EXPLAIN SELECT * FROM Customers WHERE Address = 'Kathmandu';
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | Customers | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    5 |    20.00 | Using where |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
```

In this particular case, the EXPLAIN output shows that MySQL will use a SIMPLE access type to read data from the Customers table. It will examine all rows in the table (ALL) and then filter them based on the WHERE clause. The Extra column indicates that the query will use a Using where optimization.

2. **Statistics**
    - DBMS relies on table and index statistics to improve cost estimation accuracy.

Key Takeaways

- Query cost estimation ensures efficient query execution by predicting resource usage.
- Factors like disk I/O, CPU, memory, and network significantly impact costs.
- Using indexes and optimizing access methods can dramatically reduce query costs.
- Tools like EXPLAIN help developers and DBAs analyze and optimize query performance.

5.3. Query Operations

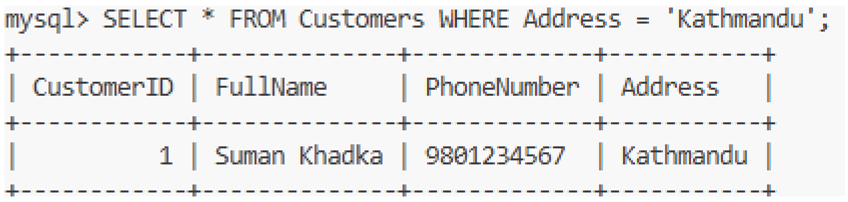
- Query operations are fundamental tasks performed by a database management system (DBMS) to process and execute a query.
- These operations include selection, projection, joins, aggregation, and set operations, among others.
- Query operations are the building blocks of more complex SQL queries and are executed as part of the query plan.

Types of Query Operations

1. **Selection**
    - Filters rows from a table based on specific conditions.
    - In SQL, this corresponds to the WHERE clause.

    Example: Fetch customers from Kathmandu.

    SELECT * FROM Customers WHERE Address = 'Kathmandu';

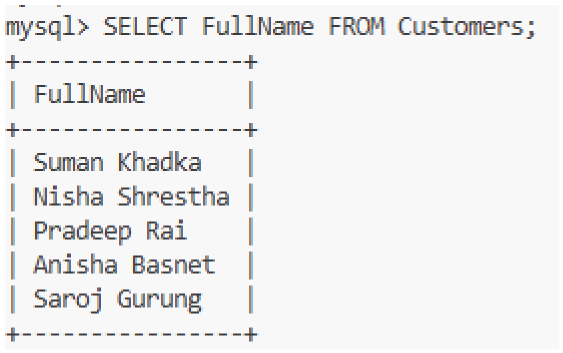


```
mysql> SELECT * FROM Customers WHERE Address = 'Kathmandu';
+------------+--------------+-------------+-----------+
| CustomerID | FullName     | PhoneNumber | Address   |
+------------+--------------+-------------+-----------+
|          1 | Suman Khadka | 9801234567  | Kathmandu |
+------------+--------------+-------------+-----------+
```

2. **Projection**
    - Selects specific columns from a table, reducing the number of attributes in the result.
    - In SQL, this is defined in the SELECT clause.

    Example: Fetch only the names of all customers.

    SELECT FullName FROM Customers;



```
mysql> SELECT FullName FROM Customers;
+----------------+
| FullName       |
+----------------+
| Suman Khadka   |
| Nisha Shrestha |
| Pradeep Rai    |
| Anisha Basnet  |
| Saroj Gurung   |
+----------------+
```

3. **Join**
    - Combines rows from two or more tables based on a related column.
    - **Types: Inner Join, Outer Join (Left, Right, Full), Cross Join.**

Example: List orders with customer names and menu items.

SELECT

Orders.OrderID,

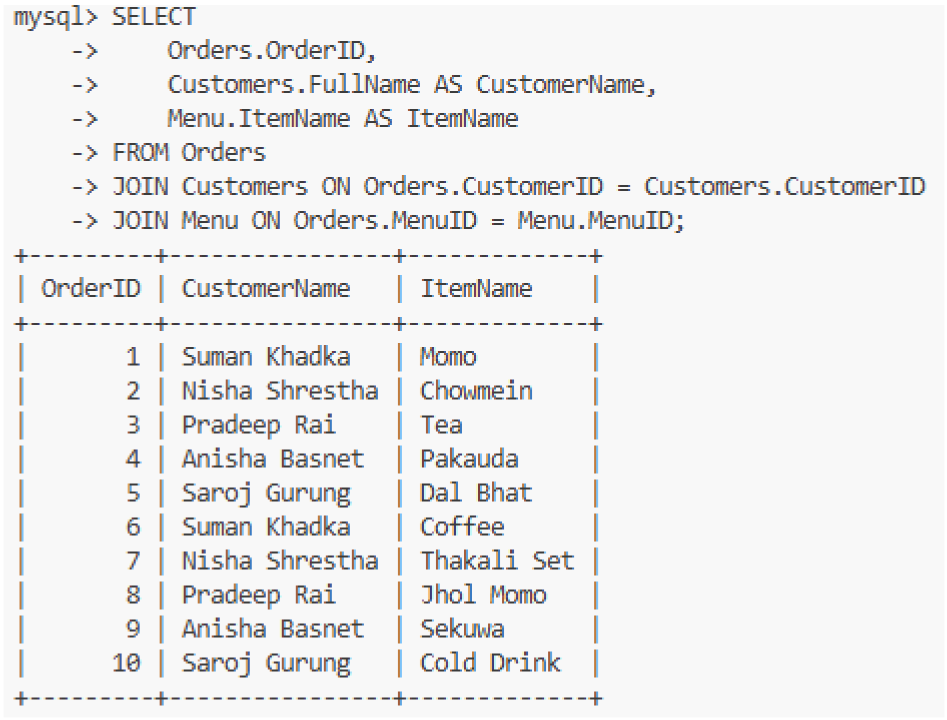
Customers.FullName AS CustomerName,

Menu.ItemName AS ItemName

FROM Orders

JOIN Customers ON Orders.CustomerID = Customers.CustomerID

JOIN Menu ON Orders.MenuID = Menu.MenuID;



```
mysql> SELECT
    ->     Orders.OrderID,
    ->     Customers.FullName AS CustomerName,
    ->     Menu.ItemName AS ItemName
    -> FROM Orders
    -> JOIN Customers ON Orders.CustomerID = Customers.CustomerID
    -> JOIN Menu ON Orders.MenuID = Menu.MenuID;
+---------+----------------+-------------+
| OrderID | CustomerName   | ItemName    |
+---------+----------------+-------------+
|       1 | Suman Khadka   | Momo        |
|       2 | Nisha Shrestha | Chowmein    |
|       3 | Pradeep Rai    | Tea         |
|       4 | Anisha Basnet  | Pakauda     |
|       5 | Saroj Gurung   | Dal Bhat    |
|       6 | Suman Khadka   | Coffee      |
|       7 | Nisha Shrestha | Thakali Set |
|       8 | Pradeep Rai    | Jhol Momo   |
|       9 | Anisha Basnet  | Sekuwa      |
|      10 | Saroj Gurung   | Cold Drink  |
+---------+----------------+-------------+
```

4. Aggregation

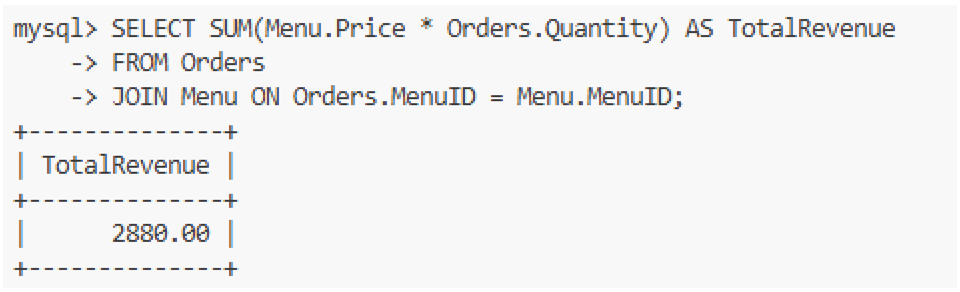
- Performs calculations on groups of rows, such as SUM, COUNT, AVG, MIN, MAX.
- Often used with the GROUP BY clause.

Example: Calculate total revenue.

*SELECT SUM(Menu.Price * Orders.Quantity) AS TotalRevenue*

FROM Orders

JOIN Menu ON Orders.MenuID = Menu.MenuID;



```
mysql> SELECT SUM(Menu.Price * Orders.Quantity) AS TotalRevenue
    -> FROM Orders
    -> JOIN Menu ON Orders.MenuID = Menu.MenuID;
+--------------+
| TotalRevenue |
+--------------+
|      2880.00 |
+--------------+
```

5. **Set Operations**

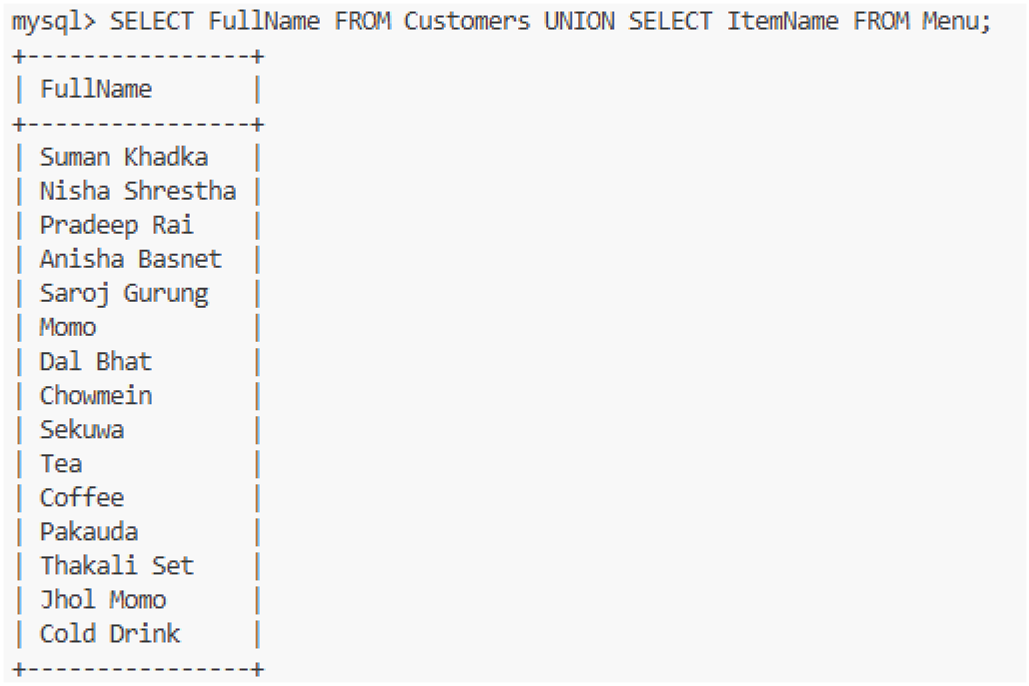
   - Combines results from multiple queries. Operations include UNION, INTERSECT, and EXCEPT/MINUS.

     Example: Combine customer names from two tables.

     SELECT FullName FROM Customers

     UNION

     SELECT ItemName FROM Menu;



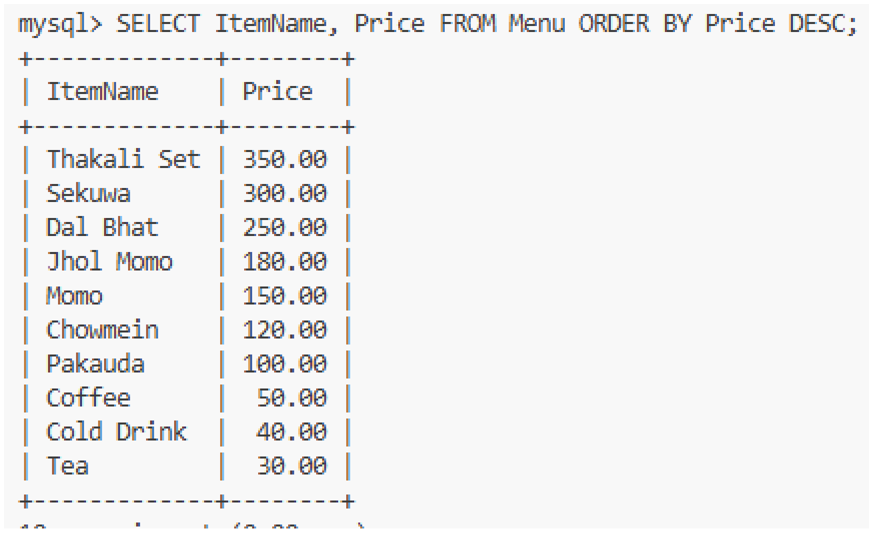
```
mysql> SELECT FullName FROM Customers UNION SELECT ItemName FROM Menu;
+----------------+
| FullName       |
+----------------+
| Suman Khadka   |
| Nisha Shrestha |
| Pradeep Rai    |
| Anisha Basnet  |
| Saroj Gurung   |
| Momo           |
| Dal Bhat       |
| Chowmein       |
| Sekuwa         |
| Tea            |
| Coffee         |
| Pakauda        |
| Thakali Set    |
| Jhol Momo      |
| Cold Drink     |
+----------------+
```

6. **Sorting**

   - Orders rows in ascending (ASC) or descending (DESC) order.
   - Uses the ORDER BY clause.

   Example: List menu items sorted by price.

   SELECT ItemName, Price FROM Menu ORDER BY Price DESC;



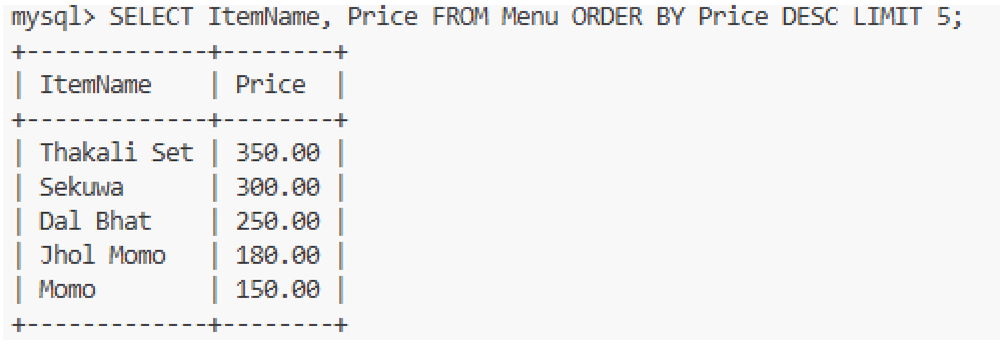
```
mysql> SELECT ItemName, Price FROM Menu ORDER BY Price DESC;
+-------------+--------+
| ItemName    | Price  |
+-------------+--------+
| Thakali Set | 350.00 |
| Sekuwa      | 300.00 |
| Dal Bhat    | 250.00 |
| Jhol Momo   | 180.00 |
| Momo        | 150.00 |
| Chowmein    | 120.00 |
| Pakauda     | 100.00 |
| Coffee      |  50.00 |
| Cold Drink  |  40.00 |
| Tea         |  30.00 |
+-------------+--------+
```

7. **Limiting Results**

   - Restricts the number of rows returned.
   - Uses the LIMIT clause (MySQL) or FETCH FIRST (SQL Server/Oracle).

   Example: Fetch the top 5 most expensive menu items.

   SELECT ItemName, Price FROM Menu ORDER BY Price DESC LIMIT 5;



```
mysql> SELECT ItemName, Price FROM Menu ORDER BY Price DESC LIMIT 5;
+-------------+--------+
| ItemName    | Price  |
+-------------+--------+
| Thakali Set | 350.00 |
| Sekuwa      | 300.00 |
| Dal Bhat    | 250.00 |
| Jhol Momo   | 180.00 |
| Momo        | 150.00 |
+-------------+--------+
```

Operations for Query Optimization

- **Index Utilization:** Efficiently retrieve rows using indexed columns.
- **Partitioning:** Divide large datasets into smaller, manageable parts for faster operations.
- **Caching Intermediate Results:** Store temporary results during complex queries. [REDIS app]

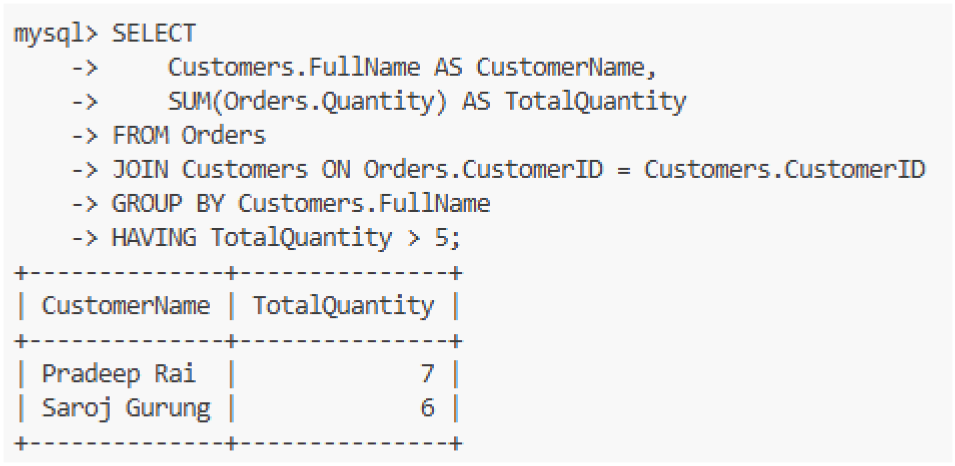
Example: Combining Query Operations

Fetch the names of customers who ordered more than 5 items, along with the total quantity they ordered.

SELECT

  Customers.FullName AS CustomerName,

```
    SUM(Orders.Quantity) AS TotalQuantity
FROM Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
GROUP BY Customers.FullName
HAVING TotalQuantity > 5;
```



```
mysql> SELECT
    ->     Customers.FullName AS CustomerName,
    ->     SUM(Orders.Quantity) AS TotalQuantity
    -> FROM Orders
    -> JOIN Customers ON Orders.CustomerID = Customers.CustomerID
    -> GROUP BY Customers.FullName
    -> HAVING TotalQuantity > 5;
+--------------+---------------+
| CustomerName | TotalQuantity |
+--------------+---------------+
| Pradeep Rai  |             7 |
| Saroj Gurung |             6 |
+--------------+---------------+
```

Explanation:

1. **Join:** Combines Orders and Customers tables.
2. **Aggregation:** Sums up the quantities ordered by each customer.
3. **Group By:** Groups rows by customer name.
4. **Filter (HAVING):** Displays only customers with total quantities greater than 5.

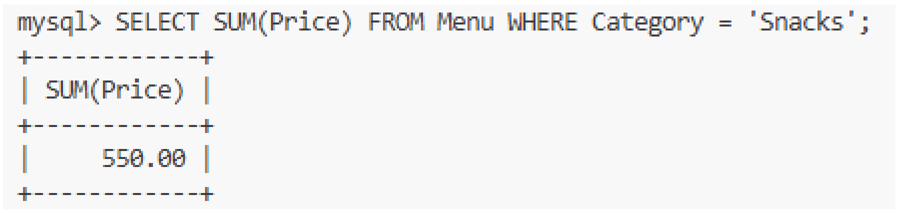
5.4. Evaluation of Expressions

- The evaluation of expressions in query processing involves executing the logical query plan generated during the optimization phase. This step translates the query into a physical execution plan and performs the operations required to retrieve the desired results.
- The main goal of expression evaluation is to ensure that queries are executed efficiently, using minimal resources while providing accurate results.

Steps in Expression Evaluation

1. **Logical Query Plan**
   - The query is represented as a logical structure, such as a tree or directed acyclic graph (DAG).
   - Example:

     SELECT SUM(Price) FROM Menu WHERE Category = 'Snacks';



```
mysql> SELECT SUM(Price) FROM Menu WHERE Category = 'Snacks';
+------------+
| SUM(Price) |
+------------+
|     550.00 |
+------------+
```

Logical Plan:

- **Selection**: Filter rows where Category = 'Snacks'.
- **Projection**: Extract the Price column.
- **Aggregation**: Compute the sum of the Price values.

2. **Physical Query Plan**
   - The logical plan is translated into a physical plan specifying the actual algorithms and access methods (e.g., table scans, index lookups).
   - Example:
     - Use an **Index Scan** on the Category column if indexed.
     - Perform **Hash Aggregation** to compute the sum efficiently.

3. **Execution**
   - The physical plan is executed by the query engine.
   - Intermediate results are produced and processed until the final result is computed.

Evaluation Methods

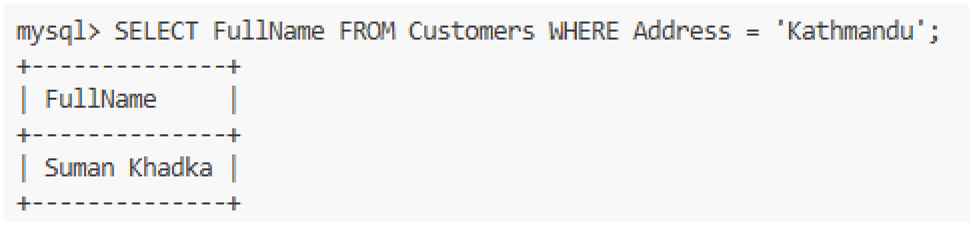
1. **Iterative Execution**
   - Executes one operation at a time, passing intermediate results to the next step.
   - Suitable for simple queries.
2. **Pipelined Execution**
   - Operations are executed in parallel, with intermediate results streamed between operators.
   - Reduces memory overhead and improves performance for large datasets.
3. **Materialized Execution**
   - Intermediate results are stored temporarily before being processed further.
   - Useful for complex queries with multiple subqueries or joins.

Expression Trees

Expression trees represent a query as a hierarchical structure, with nodes as operations and leaves as tables or attributes.

Example Query:

SELECT FullName FROM Customers WHERE Address = 'Kathmandu';



```
mysql> SELECT FullName FROM Customers WHERE Address = 'Kathmandu';
+--------------+
| FullName     |
+--------------+
| Suman Khadka |
+--------------+
```

Expression Tree:

π (FullName)

|

σ (Address = 'Kathmandu')

|

Customers

Explanation:

1. The leaf node is the Customers table.
2. The selection operation (σ) filters rows where Address = 'Kathmandu'.
3. The projection operation (π) extracts the FullName column.

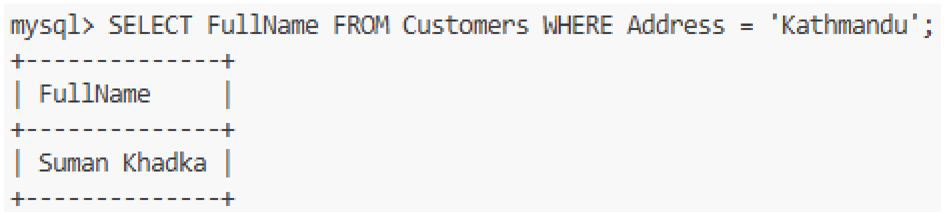
Optimization of Expression Evaluation

1. **Reordering Operations**
   - Apply selective operations (e.g., WHERE) early to reduce the number of rows processed.

     Example: Instead of scanning all rows for projection:

     SELECT FullName FROM Customers WHERE Address = 'Kathmandu';

     First, filter rows (WHERE), then project (SELECT FullName).



```
mysql> SELECT FullName FROM Customers WHERE Address = 'Kathmandu';
+--------------+
| FullName     |
+--------------+
| Suman Khadka |
+--------------+
```

2. **Using Indexes**
   - Speeds up data retrieval, especially for selective conditions.

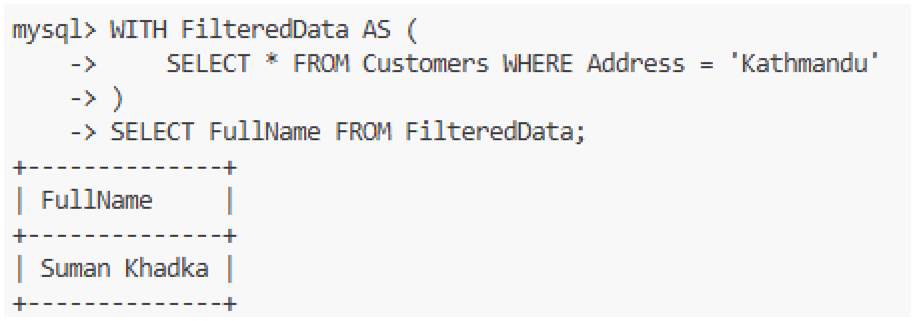
     Example:
     If the Address column is indexed, the condition WHERE Address = 'Kathmandu' will be faster.

3. **Efficient Join Algorithms**
   - Choose join strategies based on data size and distribution:
     - Nested Loop Join (small datasets).
     - Hash Join (large datasets).
     - Merge Join (sorted datasets).

4. **Caching Intermediate Results**
   - Temporarily store results of common subexpressions to avoid redundant computations.

     Example:
     If multiple subqueries reuse the same intermediate result:

```
WITH FilteredData AS (
   SELECT * FROM Customers WHERE Address = 'Kathmandu'
)
SELECT FullName FROM FilteredData;
```



```
mysql> WITH FilteredData AS (
    ->     SELECT * FROM Customers WHERE Address = 'Kathmandu'
    -> )
    -> SELECT FullName FROM FilteredData;
+--------------+
| FullName     |
+--------------+
| Suman Khadka |
+--------------+
```

Example: Complex Expression Evaluation

Query: Find the total quantity of snacks ordered by each customer.

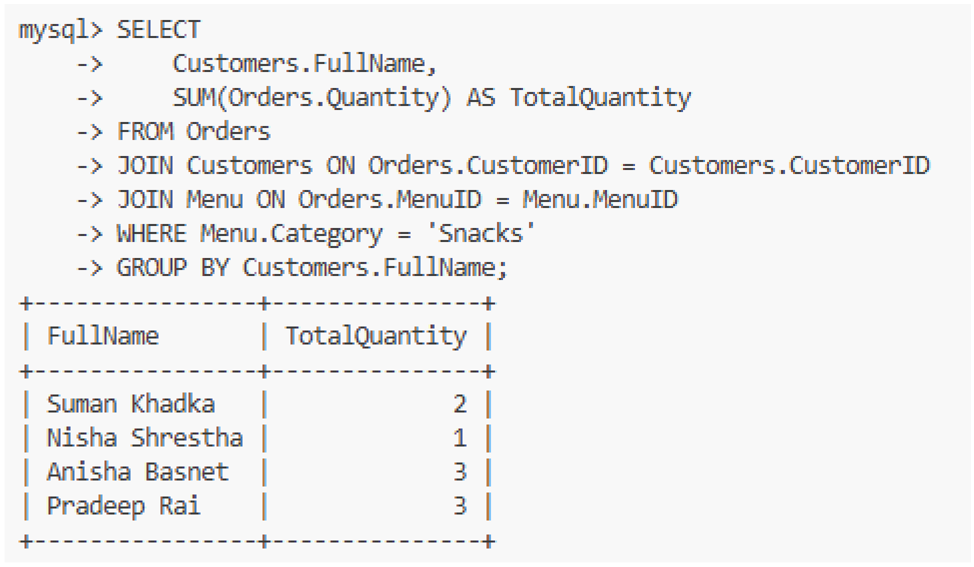
```
SELECT
   Customers.FullName,
   SUM(Orders.Quantity) AS TotalQuantity
FROM Orders
```

JOIN Customers ON Orders.CustomerID = Customers.CustomerID

JOIN Menu ON Orders.MenuID = Menu.MenuID

WHERE Menu.Category = 'Snacks'

GROUP BY Customers.FullName;



```
mysql> SELECT
    ->     Customers.FullName,
    ->     SUM(Orders.Quantity) AS TotalQuantity
    -> FROM Orders
    -> JOIN Customers ON Orders.CustomerID = Customers.CustomerID
    -> JOIN Menu ON Orders.MenuID = Menu.MenuID
    -> WHERE Menu.Category = 'Snacks'
    -> GROUP BY Customers.FullName;
+-----------------+---------------+
| FullName        | TotalQuantity |
+-----------------+---------------+
| Suman Khadka    |             2 |
| Nisha Shrestha  |             1 |
| Anisha Basnet   |             3 |
| Pradeep Rai     |             3 |
+-----------------+---------------+
```

Evaluation Steps:

1. **Join**: Combine Orders, Customers, and Menu using the keys CustomerID and MenuID.
2. **Selection**: Filter rows where Menu.Category = 'Snacks'.
3. **Aggregation**: Compute the sum of Quantity for each customer.
4. **Projection**: Output FullName and TotalQuantity.

Optimized Plan:

- Use indexes on MenuID and Category for efficient joins and filtering.
- Apply filtering (Category = 'Snacks') before aggregation to reduce the data size.

5.5. Query Optimization

➢ Query optimization is a critical phase of query processing in which the database management system (DBMS) selects the most efficient execution plan for a given query.
➢ The objective of query optimization is to reduce query execution time and resource consumption, such as CPU, memory, and disk I/O, while ensuring that the result remains correct.

Types of Query Optimization

1. **Heuristic Optimization**
    - Uses predefined rules or heuristics to transform the query into a more efficient form.
    - Example: Reordering joins or pushing selections closer to the base tables.
2. **Cost-Based Optimization**
    - Evaluates multiple execution plans and selects the one with the lowest estimated cost.
    - Cost is estimated based on factors like disk I/O, CPU usage, and memory requirements.

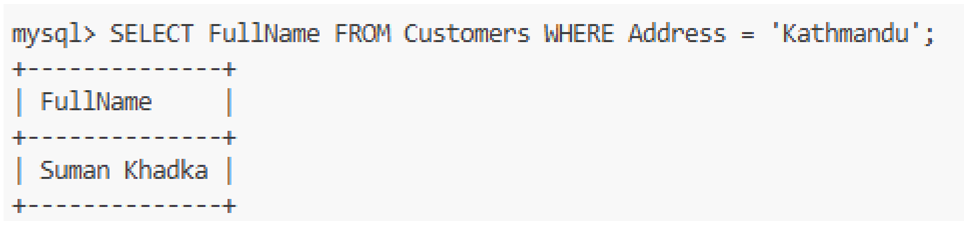
Steps in Query Optimization

1. **Query Parsing and Decomposition**
    - Parse the query into a syntax tree or query graph.
    - Break down the query into smaller subqueries for easier analysis.

2. **Logical Query Optimization**
    - Focuses on the structure of the query.
    - Example transformations:
        - **Selection Pushdown**: Move WHERE clauses closer to base tables to minimize intermediate results.
        - **Projection Pushdown**: Eliminate unused columns early.

    Example:

    SELECT FullName FROM Customers WHERE Address = 'Kathmandu';



```
mysql> SELECT FullName FROM Customers WHERE Address = 'Kathmandu';
+--------------+
| FullName     |
+--------------+
| Suman Khadka |
+--------------+
```

    Optimized Query:

    - Apply the condition Address = 'Kathmandu' (selection) before retrieving FullName (projection).

3. **Physical Query Optimization**
    - Determines the best algorithms and data structures for executing the query.

- Example: Choosing between a nested-loop join, hash join, or merge join.

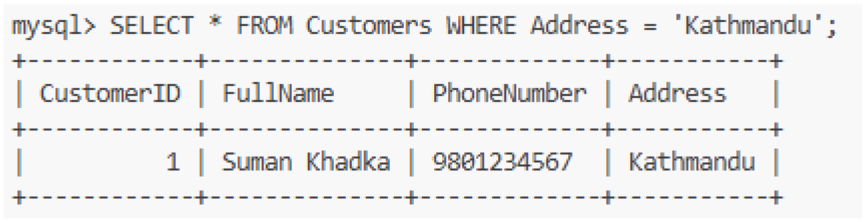
4. **Evaluation and Plan Selection**
   - The optimizer evaluates the cost of various plans and selects the most efficient one.

Optimization Techniques

1. **Index Utilization**
   - Use indexes on columns involved in WHERE, JOIN, or ORDER BY clauses to speed up data retrieval.

Example:
For the query:

SELECT * FROM Customers WHERE Address = 'Kathmandu';



```
mysql> SELECT * FROM Customers WHERE Address = 'Kathmandu';
+------------+--------------+-------------+-----------+
| CustomerID | FullName     | PhoneNumber | Address   |
+------------+--------------+-------------+-----------+
|          1 | Suman Khadka | 9801234567  | Kathmandu |
+------------+--------------+-------------+-----------+
```

Using an index on the Address column will reduce the number of scanned rows.

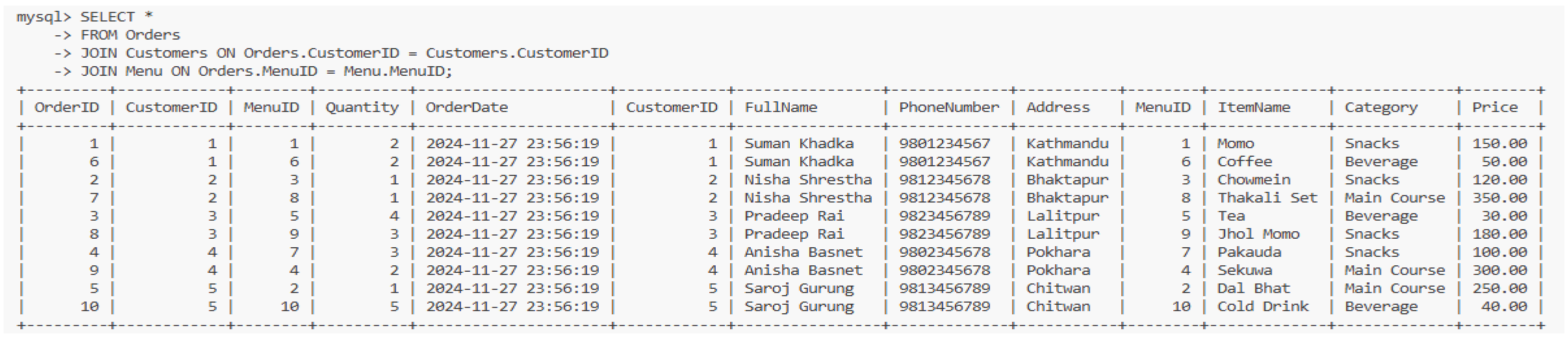
2. **Join Reordering**
   - Rearrange the order of joins to minimize intermediate results.
   - Smaller tables are typically processed first to reduce overall cost.

Example:
SELECT *

FROM Orders

JOIN Customers ON Orders.CustomerID = Customers.CustomerID

JOIN Menu ON Orders.MenuID = Menu.MenuID;



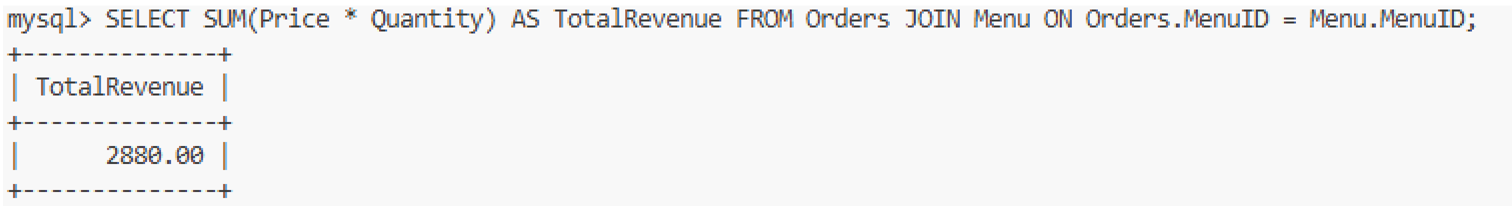
```
mysql> SELECT *
    -> FROM Orders
    -> JOIN Customers ON Orders.CustomerID = Customers.CustomerID
    -> JOIN Menu ON Orders.MenuID = Menu.MenuID;
+---------+------------+--------+----------+---------------------+------------+----------------+-------------+-----------+--------+-------------+-------------+--------+
| OrderID | CustomerID | MenuID | Quantity | OrderDate           | CustomerID | FullName       | PhoneNumber | Address   | MenuID | ItemName    | Category    | Price  |
+---------+------------+--------+----------+---------------------+------------+----------------+-------------+-----------+--------+-------------+-------------+--------+
|       1 |          1 |      1 |        2 | 2024-11-27 23:56:19 |          1 | Suman Khadka   | 9801234567  | Kathmandu |      1 | Momo        | Snacks      | 150.00 |
|       6 |          1 |      6 |        2 | 2024-11-27 23:56:19 |          1 | Suman Khadka   | 9801234567  | Kathmandu |      6 | Coffee      | Beverage    |  50.00 |
|       2 |          2 |      3 |        1 | 2024-11-27 23:56:19 |          2 | Nisha Shrestha | 9812345678  | Bhaktapur |      3 | Chowmein    | Snacks      | 120.00 |
|       7 |          2 |      8 |        1 | 2024-11-27 23:56:19 |          2 | Nisha Shrestha | 9812345678  | Bhaktapur |      8 | Thakali Set | Main Course | 350.00 |
|       3 |          3 |      5 |        4 | 2024-11-27 23:56:19 |          3 | Pradeep Rai    | 9823456789  | Lalitpur  |      5 | Tea         | Beverage    |  30.00 |
|       8 |          3 |      9 |        3 | 2024-11-27 23:56:19 |          3 | Pradeep Rai    | 9823456789  | Lalitpur  |      9 | Jhol Momo   | Snacks      | 180.00 |
|       4 |          4 |      7 |        3 | 2024-11-27 23:56:19 |          4 | Anisha Basnet  | 9802345678  | Pokhara   |      7 | Pakauda     | Snacks      | 100.00 |
|       9 |          4 |      4 |        2 | 2024-11-27 23:56:19 |          4 | Anisha Basnet  | 9802345678  | Pokhara   |      4 | Sekuwa      | Main Course | 300.00 |
|       5 |          5 |      2 |        1 | 2024-11-27 23:56:19 |          5 | Saroj Gurung   | 9813456789  | Chitwan   |      2 | Dal Bhat    | Main Course | 250.00 |
|      10 |          5 |     10 |        5 | 2024-11-27 23:56:19 |          5 | Saroj Gurung   | 9813456789  | Chitwan   |     10 | Cold Drink  | Beverage    |  40.00 |
+---------+------------+--------+----------+---------------------+------------+----------------+-------------+-----------+--------+-------------+-------------+--------+
```

Optimize by starting with the smaller of Customers or Menu tables.

3. **Query Simplification**
   - Remove redundant conditions and operations.
   - Example: Combining multiple WHERE clauses into a single, efficient condition.
   -
4. **Materialized Views**
   - Store precomputed results for frequently executed queries.

   Example: Instead of recalculating total revenue each time:

   SELECT SUM(Price * Quantity) AS TotalRevenue FROM Orders JOIN Menu ON Orders.MenuID = Menu.MenuID;



```
mysql> SELECT SUM(Price * Quantity) AS TotalRevenue FROM Orders JOIN Menu ON Orders.MenuID = Menu.MenuID;
+--------------+
| TotalRevenue |
+--------------+
|      2880.00 |
+--------------+
```

   Use a materialized view with precomputed results.

5. **Avoiding Cartesian Products**
   - Ensure that joins are properly specified to prevent unnecessary combinations of rows.

     Bad Example:

         SELECT * FROM Customers, Orders;

     Optimized Example:

         SELECT *

         FROM Customers

         JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

```
mysql> SELECT * FROM Customers, Orders;
+------------+----------------+-------------+-----------+---------+------------+--------+----------+---------------------+
| CustomerID | FullName       | PhoneNumber | Address   | OrderID | CustomerID | MenuID | Quantity | OrderDate           |
+------------+----------------+-------------+-----------+---------+------------+--------+----------+---------------------+
|          5 | Saroj Gurung   | 9813456789  | Chitwan   |       1 |          1 |      1 |        2 | 2024-11-27 23:56:19 |
|          4 | Anisha Basnet  | 9802345678  | Pokhara   |       1 |          1 |      1 |        2 | 2024-11-27 23:56:19 |
|          3 | Pradeep Rai    | 9823456789  | Lalitpur  |       1 |          1 |      1 |        2 | 2024-11-27 23:56:19 |
|          2 | Nisha Shrestha | 9812345678  | Bhaktapur |       1 |          1 |      1 |        2 | 2024-11-27 23:56:19 |
|          1 | Suman Khadka   | 9801234567  | Kathmandu |       1 |          1 |      1 |        2 | 2024-11-27 23:56:19 |
|          5 | Saroj Gurung   | 9813456789  | Chitwan   |       2 |          2 |      3 |        1 | 2024-11-27 23:56:19 |
|          4 | Anisha Basnet  | 9802345678  | Pokhara   |       2 |          2 |      3 |        1 | 2024-11-27 23:56:19 |
|          3 | Pradeep Rai    | 9823456789  | Lalitpur  |       2 |          2 |      3 |        1 | 2024-11-27 23:56:19 |
|          2 | Nisha Shrestha | 9812345678  | Bhaktapur |       2 |          2 |      3 |        1 | 2024-11-27 23:56:19 |
|          1 | Suman Khadka   | 9801234567  | Kathmandu |       2 |          2 |      3 |        1 | 2024-11-27 23:56:19 |
|          5 | Saroj Gurung   | 9813456789  | Chitwan   |       3 |          3 |      5 |        4 | 2024-11-27 23:56:19 |
|          4 | Anisha Basnet  | 9802345678  | Pokhara   |       3 |          3 |      5 |        4 | 2024-11-27 23:56:19 |
|          3 | Pradeep Rai    | 9823456789  | Lalitpur  |       3 |          3 |      5 |        4 | 2024-11-27 23:56:19 |
|          2 | Nisha Shrestha | 9812345678  | Bhaktapur |       3 |          3 |      5 |        4 | 2024-11-27 23:56:19 |
|          1 | Suman Khadka   | 9801234567  | Kathmandu |       3 |          3 |      5 |        4 | 2024-11-27 23:56:19 |
|          5 | Saroj Gurung   | 9813456789  | Chitwan   |       4 |          4 |      7 |        3 | 2024-11-27 23:56:19 |
|          4 | Anisha Basnet  | 9802345678  | Pokhara   |       4 |          4 |      7 |        3 | 2024-11-27 23:56:19 |
|          3 | Pradeep Rai    | 9823456789  | Lalitpur  |       4 |          4 |      7 |        3 | 2024-11-27 23:56:19 |
|          2 | Nisha Shrestha | 9812345678  | Bhaktapur |       4 |          4 |      7 |        3 | 2024-11-27 23:56:19 |
|          1 | Suman Khadka   | 9801234567  | Kathmandu |       4 |          4 |      7 |        3 | 2024-11-27 23:56:19 |
|          5 | Saroj Gurung   | 9813456789  | Chitwan   |       5 |          5 |      2 |        1 | 2024-11-27 23:56:19 |
|          4 | Anisha Basnet  | 9802345678  | Pokhara   |       5 |          5 |      2 |        1 | 2024-11-27 23:56:19 |
|          3 | Pradeep Rai    | 9823456789  | Lalitpur  |       5 |          5 |      2 |        1 | 2024-11-27 23:56:19 |
```

```
mysql> SELECT *
    -> FROM Customers
    -> JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
+------------+----------------+-------------+-----------+---------+------------+--------+----------+---------------------+
| CustomerID | FullName       | PhoneNumber | Address   | OrderID | CustomerID | MenuID | Quantity | OrderDate           |
+------------+----------------+-------------+-----------+---------+------------+--------+----------+---------------------+
|          1 | Suman Khadka   | 9801234567  | Kathmandu |       1 |          1 |      1 |        2 | 2024-11-27 23:56:19 |
|          1 | Suman Khadka   | 9801234567  | Kathmandu |       6 |          1 |      6 |        2 | 2024-11-27 23:56:19 |
|          2 | Nisha Shrestha | 9812345678  | Bhaktapur |       2 |          2 |      3 |        1 | 2024-11-27 23:56:19 |
|          2 | Nisha Shrestha | 9812345678  | Bhaktapur |       7 |          2 |      8 |        1 | 2024-11-27 23:56:19 |
|          3 | Pradeep Rai    | 9823456789  | Lalitpur  |       3 |          3 |      5 |        4 | 2024-11-27 23:56:19 |
|          3 | Pradeep Rai    | 9823456789  | Lalitpur  |       8 |          3 |      9 |        3 | 2024-11-27 23:56:19 |
|          4 | Anisha Basnet  | 9802345678  | Pokhara   |       4 |          4 |      7 |        3 | 2024-11-27 23:56:19 |
|          4 | Anisha Basnet  | 9802345678  | Pokhara   |       9 |          4 |      4 |        2 | 2024-11-27 23:56:19 |
|          5 | Saroj Gurung   | 9813456789  | Chitwan   |       5 |          5 |      2 |        1 | 2024-11-27 23:56:19 |
|          5 | Saroj Gurung   | 9813456789  | Chitwan   |      10 |          5 |     10 |        5 | 2024-11-27 23:56:19 |
+------------+----------------+-------------+-----------+---------+------------+--------+----------+---------------------+
```

Tools for Query Optimization

1. **EXPLAIN/EXPLAIN ANALYZE**
   - Provides insights into the execution plan for a query.
   - Shows cost estimates and chosen strategies.

   Example (PostgreSQL):

   ```
   EXPLAIN ANALYZE SELECT * FROM Customers WHERE Address = 'Kathmandu';
   ```

```
mysql> EXPLAIN ANALYZE SELECT * FROM Customers WHERE Address = 'Kathmandu';
+--------------------------------------------------------------------------------------------------------------------
-------------------------+
| EXPLAIN
                         |
+--------------------------------------------------------------------------------------------------------------------
-------------------------+
| -> Filter: (Customers.Address = 'Kathmandu')  (cost=0.75 rows=1) (actual time=0.0248..0.0297 rows=1 loops=1)
    -> Table scan on Customers  (cost=0.75 rows=5) (actual time=0.0224..0.0265 rows=5 loops=1)
 |
+--------------------------------------------------------------------------------------------------------------------
-------------------------+
```

2. **Database Statistics**
    - DBMS uses statistics about table sizes, data distribution, and indexes to estimate costs.

Example: Query Optimization in Action

Query: Fetch the names of customers who have placed more than 5 orders.

SELECT Customers.FullName, COUNT(Orders.OrderID) AS TotalOrders

FROM Customers

JOIN Orders ON Customers.CustomerID = Orders.CustomerID

GROUP BY Customers.FullName

HAVING TotalOrders > 5;

```
mysql> SELECT Customers.FullName, COUNT(Orders.OrderID) AS TotalOrders
    -> FROM Customers
    -> JOIN Orders ON Customers.CustomerID = Orders.CustomerID
    -> GROUP BY Customers.FullName
    -> HAVING TotalOrders > 5;
Empty set (0.00 sec)
```

Optimization:

1. Use an index on CustomerID for the JOIN.
2. Apply the HAVING clause after computing the COUNT to avoid unnecessary row processing.
3. Push projection (Customers.FullName) early to reduce data size.

Benefits of Query Optimization

1. **Improved Performance**
    - Faster query execution by reducing resource usage.

2. **Scalability**
    - Efficient queries handle larger datasets without degradation.
3. **Reduced Costs**
    - Minimizes CPU, memory, and storage usage.
4. **Better User Experience**
    - Provides timely results for complex queries.

Q/A

Fill in the Blanks (20 Questions)

1. Query processing involves translating a _______________ plan for execution.
2. _______________ is a critical step in query optimization to predict resource consumption.
3. Disk I/O, CPU usage, and memory requirements are considered in _______________ estimation.
4. The process of retrieving rows based on specific conditions is called _______________.
5. _______________ selects specific columns from a table, reducing the number of attributes.
6. A _______________ combines rows from two or more tables based on a related column.
7. _______________ functions calculate summary statistics like SUM and AVG.
8. Set operations like _______________ combine results from multiple queries.
9. Query evaluation begins with a _______________ query plan.
10. An expression tree represents a query in a _______________ structure.
11. Operations like WHERE are moved closer to base tables in _______________ optimization.
12. A _______________ scan reads all rows in a table sequentially.
13. _______________ scan uses indexes to locate specific rows efficiently.
14. Intermediate results in query execution can be stored using _______________.
15. Joins like _______________ combine rows based on a common key.
16. The HAVING clause is used for filtering groups in _______________ operations.
17. Query optimization aims to reduce _______________ time and resource consumption.
18. Tools like _______________ provide details about the execution plan of a query.
19. _______________ operations involve sorting and grouping rows.
20. Logical and physical query plans are parts of _______________ processing.

Multiple Choice Questions (MCQ) (20 Questions)

1. What is the primary goal of query processing?
   a) Storing data
   b) Executing queries efficiently
   c) Parsing data
   d) Cleaning data

2. Which phase of query processing evaluates the computational cost of a query?
   a) Query Parsing
   b) Query Optimization
   c) Query Cost Estimation
   d) Query Execution

3. What is the logical representation of a query often called?
   a) Execution Plan
   b) Expression Tree
   c) Query Log
   d) Data Map

4. Which clause is used to filter groups in SQL?
   a) WHERE
   b) HAVING
   c) GROUP BY
   d) SELECT

5. Which of the following is an aggregation function?
   a) SUM
   b) JOIN
   c) WHERE
   d) EXPLAIN

6. What is the purpose of query optimization?
   a) To rewrite the query
   b) To minimize execution time and resources
   c) To store the query plan
   d) To create indexes

7. What does a sequential scan do?
   a) Reads only indexed rows
   b) Reads all rows in a table sequentially
   c) Combines rows from multiple tables
   d) Sorts data in ascending order

8. What is a primary feature of heuristic optimization?
   a) Rule-based transformations
   b) Cost estimation
   c) Data visualization
   d) Query execution

9. Which join retrieves matching rows from both tables?
   a) LEFT JOIN
   b) RIGHT JOIN
   c) INNER JOIN
   d) CROSS JOIN

10. How do indexes improve query performance?
   a) By reducing the number of columns
   b) By storing data sequentially
   c) By speeding up data retrieval
   d) By creating a Cartesian product

11. What is the main purpose of query evaluation?
   a) To parse the query
   b) To execute the query efficiently
   c) To create materialized views
   d) To perform aggregation

12. What is the role of the EXPLAIN tool in query processing?
   a) To write queries
   b) To display execution plans
   c) To create indexes
   d) To aggregate data

13. Which optimization technique pushes filtering conditions closer to the base tables?
   a) Join reordering
   b) Selection pushdown
   c) Materialized views
   d) Hash join

14. What is the output of a Cartesian product?
   a) Filtered rows
   b) All possible combinations of rows
   c) Sorted rows
   d) Aggregated values

15. Which of the following operations retrieves specific columns from a table?
   a) Selection

b) Projection
c) Aggregation
d) Join

16. A physical query plan is:
a) Abstract
b) Logical
c) Executable
d) Theoretical

17. What is a materialized view?
a) A temporary table storing intermediate results
b) A logical representation of a query
c) A precomputed result stored in the database
d) A tool for indexing

18. Which clause is used to sort the output of a query?
a) HAVING
b) WHERE
c) ORDER BY
d) GROUP BY

19. Cost estimation in query processing is influenced by:
a) Query syntax
b) Data size and distribution
c) Table names
d) Number of queries

20. Which type of join combines rows even if there are no matching keys?
a) INNER JOIN
b) CROSS JOIN
c) OUTER JOIN
d) NATURAL JOIN

Short Questions (20 Questions)

1. What is query processing?
2. Define query optimization.
3. What is the purpose of cost estimation in query processing?
4. Explain the difference between logical and physical query plans.
5. What is a selection operation?

6. Define projection in the context of SQL.
7. What is a join operation?
8. Explain the role of the HAVING clause.
9. What is an expression tree?
10. How do indexes improve query performance?
11. What are materialized views?
12. What is heuristic optimization?
13. List the factors influencing query cost.
14. Define aggregation and give an example.
15. Explain the concept of query caching.
16. What is a sequential scan?
17. What is the purpose of the ORDER BY clause?
18. Explain selection pushdown with an example.
19. What is the significance of query evaluation?
20. How does the EXPLAIN tool assist in optimization?

Comprehensive Questions (8 Questions)

1. Explain the key phases of query processing and their significance.
2. Discuss query cost estimation and the factors influencing it.
3. Describe the different types of query operations with examples.
4. Explain how expression trees are used in query evaluation with an example.
5. Discuss the techniques of query optimization and their impact on performance.
6. Compare heuristic and cost-based optimization with suitable examples.
7. Illustrate how join operations are optimized in query processing.
8. Write a detailed explanation of the role of tools like EXPLAIN in query optimization.

Answers to Fill in the Blanks

1. *execution*
2. *cost estimation*
3. *query cost*
4. *selection*
5. *projection*
6. *join*
7. *aggregation*
8. *UNION*
9. *logical*
10. *hierarchical*
11. *heuristic*
12. *sequential*
13. *index*
14. *caching*
15. *inner join*
16. *aggregation*
17. *execution*
18. *EXPLAIN*
19. *query*
20. *query*

Answers of MCQ

1. *b) Executing queries efficiently*
2. *c) Query Cost Estimation*
3. *b) Expression Tree*
4. *b) HAVING*

5. *a) SUM*
6. *b) To minimize execution time and resources*
7. *b) Reads all rows in a table sequentially*
8. *a) Rule-based transformations*
9. *c) INNER JOIN*
10. *c) By speeding up data retrieval*
11. *b) To execute the query efficiently*
12. *b) To display execution plans*
13. *b) Selection pushdown*
14. *b) All possible combinations of rows*
15. *b) Projection*
16. *c) Executable*
17. *c) A precomputed result stored in the database*
18. *c) ORDER BY*
19. *b) Data size and distribution*
20. ***20.**c) OUTER JOIN*