Experiment 4: Query Evaluation Plan Analysis

Prerequisites

Knowledge of SQL queries, JOINs, and indexing concepts

Lab Objective

To analyze how SQL queries are executed internally by the database management system using query execution plans and understand the cost and performance implications of different query strategies.

Learning Outcomes

By completing this experiment, students will be able to:

- Generate and interpret query execution plans.
- Understand how the database optimizer chooses execution strategies.
- Identify table scans, index scans, and join methods.
- Compare performance before and after indexing.
- Analyze query cost and execution time.

Introduction

When a SQL query is executed, the database does not directly run it as written. Instead, it generates an **execution plan** that determines how data will be accessed, joined, filtered, and sorted.

Query evaluation plans help developers understand:

- How tables are accessed
- Whether indexes are used
- Which join algorithms are selected
- Estimated cost and execution time

This experiment focuses on analyzing and interpreting execution plans to improve database performance.

Key Concepts Covered

- **Query Optimizer:** Component that determines the most efficient way to execute a query.
- **Execution Plan:** Step-by-step strategy used to retrieve data.
- **Cost Estimation:** Estimated resource usage (CPU, I/O).
- **Table Scan vs Index Scan:** Methods of data retrieval.
- **Join Algorithms:** Nested Loop Join, Hash Join, Merge Join.
- **Execution Time Analysis:** Comparing performance metrics.

Experiment Steps

---

1. Generate Execution Plan

Use the database's EXPLAIN feature to view query execution plans.

Example:

```
EXPLAIN SELECT *

FROM Students s

JOIN Enrollments e ON s.student_id = e.student_id

WHERE s.age > 20;
```

Observe:

- Access type (ALL, index, range, etc.)
- Rows examined
- Possible keys and selected keys
- Join order

---

2. Analyze Table Access Methods

Identify whether:

- Full table scans are being used.
- Indexes are being utilized.

Example observation:

- If access type = ALL → Full table scan
- If access type = index or range → Index usage

Students should record differences in row count and cost.

---

3. Study Join Operations

Observe different join algorithms used by the optimizer.

For large datasets, the system may choose:

- Nested Loop Join
- Hash Join
- Merge Join

Example:

EXPLAIN SELECT s.name, c.course_name

FROM Students s

JOIN Enrollments e ON s.student_id = e.student_id

JOIN Courses c ON e.course_id = c.course_id;

Analyze:

- Join order
- Join type
- Estimated cost

---

4. Compare Before and After Indexing

Step 1 – Without Index

Run EXPLAIN on a query filtering by a non-indexed column.

Step 2 – Create Index

CREATE INDEX idx_age ON Students(age);

Step 3 – Re-evaluate Execution Plan

EXPLAIN SELECT * FROM Students WHERE age > 20;

Compare:

- Access type
- Number of rows scanned
- Query cost
- Execution time

---

5. Performance Comparison

Modify queries and observe changes in execution plan.

Examples:

- Add filtering conditions.
- Reorder JOIN clauses.
- Reduce selected columns instead of using SELECT *.

Analyze how small changes affect:

- Estimated cost
- Rows processed
- Join method selection

How It Works

- The query optimizer evaluates multiple execution strategies.
- It selects the plan with the lowest estimated cost.
- Execution plans show how data is retrieved and processed internally.
- Indexes significantly influence execution efficiency.
- Understanding execution plans helps developers write optimized queries.

Submission Instructions

Students must submit:

- Screenshot of execution plan for a sample query.
- Screenshot showing table scan or index usage.
- Comparison of execution plan before and after indexing.
- Brief analysis explaining performance differences observed.

Conclusion

This experiment enables students to understand the internal workings of query execution and optimization. The ability to interpret execution plans is a critical skill for database developers, database administrators, and performance engineers working with large-scale data systems.