Experiment 6: Query Optimization Techniques

Prerequisites

Knowledge of SQL, JOIN operations, execution plans, and indexing concepts

---

Lab Objective

To understand various query optimization techniques used to improve SQL query performance and analyze how query restructuring impacts execution efficiency.

Learning Outcomes

By completing this experiment, students will be able to:

- Understand the role of the query optimizer.
- Identify inefficient query patterns.
- Rewrite queries for better performance.
- Apply filtering and join optimization techniques.
- Compare execution plans before and after optimization.
- Analyze performance improvements using cost and execution time.

Introduction

Query optimization is the process of improving the performance of SQL queries without changing their output. Modern relational database systems include a **query optimizer** that determines the most efficient way to execute a query.

However, poorly written queries can result in excessive CPU usage, high I/O operations, and slow response time. By restructuring queries and applying optimization techniques, performance can be significantly improved.

This experiment focuses on practical query rewriting and performance analysis.

Key Concepts Covered

- **Query Optimizer:** Component that selects the most efficient execution plan.
- **Predicate Pushdown:** Applying filters as early as possible.
- **Join Reordering:** Optimizing the order of table joins.
- **Avoiding SELECT ***: Retrieving only required columns.
- **Index-Friendly Conditions:** Writing queries that utilize indexes effectively.
- **Eliminating Redundant Conditions:** Simplifying WHERE clauses.

Experiment Steps

---

1. Analyze an Inefficient Query

Execute a poorly structured query and observe execution details.

Example:

```
EXPLAIN SELECT *
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
WHERE YEAR(e.enrollment_date) = 2025;
```

Observation:

- Function usage on column may prevent index usage.
- Full table scan may occur.

---

2. Rewrite Query for Optimization

Avoid functions on indexed columns and simplify conditions.

Optimized Version:

```
EXPLAIN SELECT s.student_id, s.name, e.enrollment_date
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
WHERE e.enrollment_date BETWEEN '2025-01-01' AND '2025-12-31';
```

Compare:

- Access type
- Rows scanned
- Estimated cost

---

3. Apply Predicate Pushdown

Filter data before performing joins.

Less Efficient:

```
SELECT *
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id;
```

More Efficient:

```
SELECT *
FROM Students s
JOIN Enrollments e
ON s.student_id = e.student_id
WHERE s.age > 20;
```

Observe reduction in processed rows.

---

4. Optimize JOIN Order

Reorder joins based on smaller datasets first.

- Join smaller filtered tables first.
- Ensure indexed columns are used in JOIN conditions.

Use EXPLAIN to compare join order chosen by optimizer.

---

5. Compare Before and After Optimization

Record:

- Execution time
- Rows examined
- Access type
- Query cost

Document performance improvement after optimization.

How It Works

- The query optimizer evaluates multiple execution strategies.
- It selects the plan with the lowest estimated cost.
- Writing efficient queries helps the optimizer choose better plans.
- Early filtering reduces intermediate result size.
- Index-aware conditions significantly improve performance.

Submission Instructions

Students must submit:

- Screenshot of original query execution plan.
- Screenshot of optimized query execution plan.
- Execution time comparison (before and after optimization).
- Brief explanation of optimization techniques applied.

Conclusion

This experiment demonstrates the importance of writing efficient SQL queries. Understanding and applying query optimization techniques is essential for database developers, administrators, and data engineers working with large datasets and performance-critical systems.