Advanced Database Management System

Course Code: CS-320
Course Title: Advanced Database Management Systems
Submitted To: Umer Latif
Student Name: Zain
Registration No: 2023-BS-CS-157

---

1. Introduction

TUF Retail Solutions is a retail analytics company that provides business intelligence (BI) and analytical services to large supermarket chains across South Asia. The company uses an Oracle-based data warehouse that stores millions of records related to customers, products, transactions, stores, and regions. These datasets support real-time dashboards and periodic analytical reports used by management for decision-making.

Recently, the BI team observed significant performance degradation in a complex analytical query used in dashboards. This query involves multiple table joins, regional and date-based filtering, and aggregation of sales data. Despite the presence of basic indexing, the query execution time exceeds **45 seconds**, which negatively impacts real-time reporting and operational efficiency.

This lab focuses on: - Designing a relational database schema - Writing a complex multi-table join query - Analyzing the execution plan using Oracle tools - Identifying performance bottlenecks - Proposing effective query optimization strategies

---

2. Entity Relationship (ER) Diagram

The database consists of the following main entities: - **Regions** - **Customers** - **Categories** - **Products** - **Stores** - **Transactions**

Each transaction links a customer, product, and store, while stores and customers are associated with specific regions. Products are grouped into categories.

---

3. Database Schema (DDL Scripts)

3.1 Regions Table

```
CREATE TABLE regions (
    region_id INT PRIMARY KEY,
    region_name VARCHAR(100)
);
```

3.2 Customers Table

```sql
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    region_id INT,
    FOREIGN KEY (region_id) REFERENCES regions(region_id)
);
```

3.3 Categories Table

```sql
CREATE TABLE categories (
    category_id INT PRIMARY KEY,
    category_name VARCHAR(100)
);
```

3.4 Products Table

```sql
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category_id INT,
    FOREIGN KEY (category_id) REFERENCES categories(category_id)
);
```

3.5 Stores Table

```sql
CREATE TABLE stores (
    store_id INT PRIMARY KEY,
    store_name VARCHAR(100),
    region_id INT,
    FOREIGN KEY (region_id) REFERENCES regions(region_id)
);
```

3.6 Transactions Table

```sql
CREATE TABLE transactions (
    transaction_id INT PRIMARY KEY,
    customer_id INT,
    store_id INT,
    product_id INT,
    transaction_date DATE,
    amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (store_id) REFERENCES stores(store_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

4. Data Insertion Scripts

4.1 Regions Data

```
INSERT INTO regions VALUES
(1,'Punjab'), (2,'Sindh'), (3,'KPK'),
(4,'Balochistan'), (5,'Islamabad');
```

4.2 Other Tables

Sample data for **customers**, **products**, **stores**, and **transactions** can be inserted by repeating IDs from **1 to 25** to simulate a large dataset suitable for performance testing.

5. Complex Multi-Join Query

```
SELECT
    c.name AS customer_name,
    cat.category_name,
    s.store_name,
    r.region_name,
    SUM(t.amount) AS total_sales
FROM transactions t
JOIN customers c ON t.customer_id = c.customer_id
JOIN products p ON t.product_id = p.product_id
JOIN categories cat ON p.category_id = cat.category_id
JOIN stores s ON t.store_id = s.store_id
JOIN regions r ON s.region_id = r.region_id
WHERE r.region_name = 'Punjab'
AND t.transaction_date BETWEEN '2024-01-01' AND '2024-03-31'
GROUP BY c.name, cat.category_name, s.store_name, r.region_name
HAVING SUM(t.amount) > 5000;
```

Query Features

- Six-table joins
- Regional filtering
- Date range filtering
- Aggregation using `SUM()`
- `HAVING` clause for conditional aggregation

---

6. Execution Plan

```
EXPLAIN PLAN FOR
SELECT ...;
```

Typical Execution Plan Output

- Full table scan on **transactions**
- Hash joins between large tables
- Nested loop joins for smaller tables
- Index scans on primary keys

---

7. Execution Plan Analysis

7.1 Join Order

- Execution begins with the **transactions** table (largest table)
- Followed by joins with customers, products, stores, and regions

7.2 Access Paths

- Full table scan on transactions
- Index scans on dimension tables

7.3 Performance Bottlenecks

- Full table scan on the transactions table
- No index on `transaction_date`
- Region filter applied after joins
- High number of intermediate rows

---

8. Optimization Recommendations

8.1 Indexing

```
CREATE INDEX idx_trans_date ON transactions(transaction_date);
CREATE INDEX idx_store_region ON stores(region_id);
CREATE INDEX idx_customer_region ON customers(region_id);
```

8.2 Query Optimization

- Apply filters as early as possible
- Reduce intermediate result sets using subqueries

8.3 Materialized Views (Conceptual)

```
CREATE MATERIALIZED VIEW quarterly_sales AS
SELECT ...;
```

8.4 Partitioning

- Partition the **transactions** table by `transaction_date`

---

9. Estimated vs Actual Rows

- Estimated rows were significantly lower than actual rows
- Optimizer preferred hash joins due to large data volume
- Index nested loops were avoided

---

10. Conclusion

The poor performance of the analytical query was mainly caused by full table scans, missing indexes, and late application of filters. By applying proper indexing, query rewriting, partitioning, and materialized views, query execution time can be significantly reduced. These optimizations enhance reporting speed, improve system scalability, and support efficient decision-making in large-scale retail data warehouses.