Comprehensive Report

Comprehensive Report: Database Migration, Querying, and Optimization

Prepared by: [Your Name]
Date: [Report Date]

Table of Contents

---

1. Introduction

The objective of this project was to migrate data from a legacy database to a modern DBMS, write efficient SQL queries, and optimize query performance. The ultimate goal was to ensure that the new database structure supported fast and scalable querying, addressing both existing performance issues and future growth.

Key tasks involved:

- Migrating and cleaning the data.
- Designing an optimal schema.
- Writing SQL queries for common use cases.
- Applying optimization techniques to improve query performance.

---

2. Data Migration Process

2.1 Data Export

The first step was to extract data from the legacy system. The legacy database contained student records, course data, enrollments, and department details. Exporting this data involved the following:

- Using CSV and SQL dumps to extract the data.
- Ensuring compatibility with the new DBMS for seamless migration.

2.2 Data Cleansing

Data from the legacy system was prone to inconsistencies, missing values, and duplicates. The data cleansing process involved:

- Removing duplicate records.
- Standardizing formats (e.g., email addresses, course names).
- Handling null values through imputation and deletion where necessary.

2.3 Schema Design

The new database schema was designed with normalization in mind to reduce redundancy and ensure data integrity. Key design goals included:

- Creating normalized tables to store students, courses, departments, and enrollment data.
- Implementing indexing on key columns for faster access.

2.4 Data Import

After designing the schema, the clean data was imported into the new database using appropriate loading mechanisms (bulk insert, SQL import). The new DBMS supported indexing and other performance-enhancing features.

---

3. Database Schema Design

3.1 Tables

- **Students**: Stores basic student information (ID, first name, last name, email).
- **Courses**: Contains course data such as course ID, course name, and department ID.
- **Departments**: Defines the departments offering various courses.
- **Enrollments**: Links students to the courses they are enrolled in.
- **Course_instructors:** Associates instructors with courses they are teaching.
- **Instructors:** Stores instructor details such as instructor ID and name.

3.2 Schema Design Goals

The schema was designed with the following principles in mind:

- **Normalization**: Ensuring that the schema was in third normal form (3NF) to eliminate redundancy and maintain referential integrity.
- **Indexing**: Added indexes to key columns such as `student_id`, `course_id`, and `department_id` for faster query execution.
- **Foreign Key Relationships**: Established foreign keys to link students, courses, and departments.

---

4. Querying Process

4.1 Common Queries

Several SQL queries were designed to retrieve and analyze data from the database. Below are some of the most common queries used:

Fetching Student Information:

Retrieves basic information about students.

sql:

```sql
SELECT first_name, last_name, email
FROM students;
```

Calculating Average Course Enrollment:

Determines the average number of students enrolled in each course.

sql:

```sql
SELECT AVG(student_count) AS average_enrollment
FROM enrollments;
```

Department-wise Student Count:

Calculates the total number of students enrolled in each department.

sql:

```sql
SELECT department_id, COUNT(student_id) AS total_students
FROM enrollments
GROUP BY department_id;
```

Instructor and Course Details:

Retrieves information about instructors and the courses they teach.

sql:

```sql
SELECT i.first_name, i.last_name, c.course_name, c.department_id
FROM instructors i
JOIN course_instructors ci ON i.instructor_id = ci.instructor_id
JOIN courses c ON ci.course_id = c.course_id;
```

Top 10 courses:

Retrieves information about courses in which most of the students are enrolled:

```sql
SELECT c.course_id, c.course_name, COUNT(e.student_id) AS enrollment_count
FROM courses c
JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name
ORDER BY enrollment_count DESC
LIMIT 10;
```

4.2 Query Example:

Department-wise Student Count Query:

This query counts the number of students in each department, which is critical for enrollment analysis and resource allocation.

sql:

```sql
SELECT department_id, COUNT(student_id) AS total_students
FROM enrollments
GROUP BY department_id;
```

---

5. Query Optimization Techniques

Optimizing query performance is crucial for handling large datasets efficiently. The following techniques were applied:

5.1 Indexing

Indexing was applied to frequently queried columns such as `student_id`, `course_id`, and `department_id`. This led to significant improvements in query performance, especially for queries involving joins and WHERE clauses.

5.2 Query Rewriting

Complex queries were rewritten to minimize the use of nested subqueries and joins. This simplification reduced the execution time for most queries.

5.3 Caching

Frequently executed queries were cached to reduce redundant database hits. This was particularly useful for queries involving large datasets, where caching provided faster access to previously retrieved results.

5.4 Partitioning

Large tables were partitioned based on specific criteria (e.g., department or year of enrollment) to speed up access to data subsets and improve query performance.

---

6. Performance Analysis

6.1 Query Execution Time

The performance of queries was analyzed by comparing the execution time before and after optimization. Below is a summary of the results:

with the percentage improvement:

Query Name	Description	Original Time (s)	Optimized Time (s)	Improvement (%)
Query 1	Students in Course	0.2134	0.0144	93.25%

Query 2	Average Enrollment per Instructor	0.0193	0.0059	69.32%
Query 3	Courses by Department	0.0133	0.0060	54.78%
Query 4	Total Students per Department	0.0165	0.0049	70.51%
Query 5	Instructors for BTech CSE Core	0.2184	0.1849	15.33%
Query 6	Top 10 Courses by Enrollment	0.0086	0.0072	15.95%

6.2 Key Findings

- Indexing provided the most significant performance boost, particularly for join operations and filtering.
- Query rewriting improved the efficiency of aggregate functions and reduced the execution time for complex queries.
- Caching was highly effective for queries that were executed repeatedly.

---

7. Key Takeaways and Recommendations

7.1 Key Takeaways

- **Indexing**: The single most impactful technique, improving query performance by up to 93%.
- **Query Simplification**: Reduced complexity in queries resulted in faster execution times for most queries.
- **Caching**: Helped avoid repetitive queries and improved performance for frequently accessed data.
- **Partitioning**: Provided better data access for large tables, especially those queried by departments.

7.2 Recommendations

- **Continuous Monitoring**: Regularly monitor the performance of frequently executed queries to identify areas for improvement.
- **Further Optimization**: Explore further partitioning strategies as data volumes grow.
- **Backup and Recovery**: Implement a robust backup and recovery strategy to ensure data integrity and availability.

---

8. Conclusion

The database migration and optimization project was successful in achieving the desired performance improvements. Query execution times were significantly reduced, and the database is now capable of handling larger datasets more efficiently. Indexing, query rewriting, and caching proved to be highly effective techniques.