TEXAS COLLEGE OF MANAGEMENT & IT

DEPARTMENT OF INFORMATION TECHNOLOGY

Database Management System

Project Name : texas_DBMS_3rd_sem

Submitted By : Gauranga Gautam

Submitted To : Sushil Bhattarai

Table of Contents

1. Introduction

The purpose of this report is to outline the significance of the database project for Texas College of Management and IT. This project aims to improve data management, streamline processes, and enhance overall efficiency at the college.

2. Database Design

MySQL

MySQL is an open-source relational database management system. It is widely used for storing, organizing, and retrieving data. MySQL uses Structured Query Language (SQL) to interact with the database. It offers features like data security, scalability, and high performance. MySQL is commonly used for web applications, and it can be accessed and managed through command-line tools or graphical interfaces.

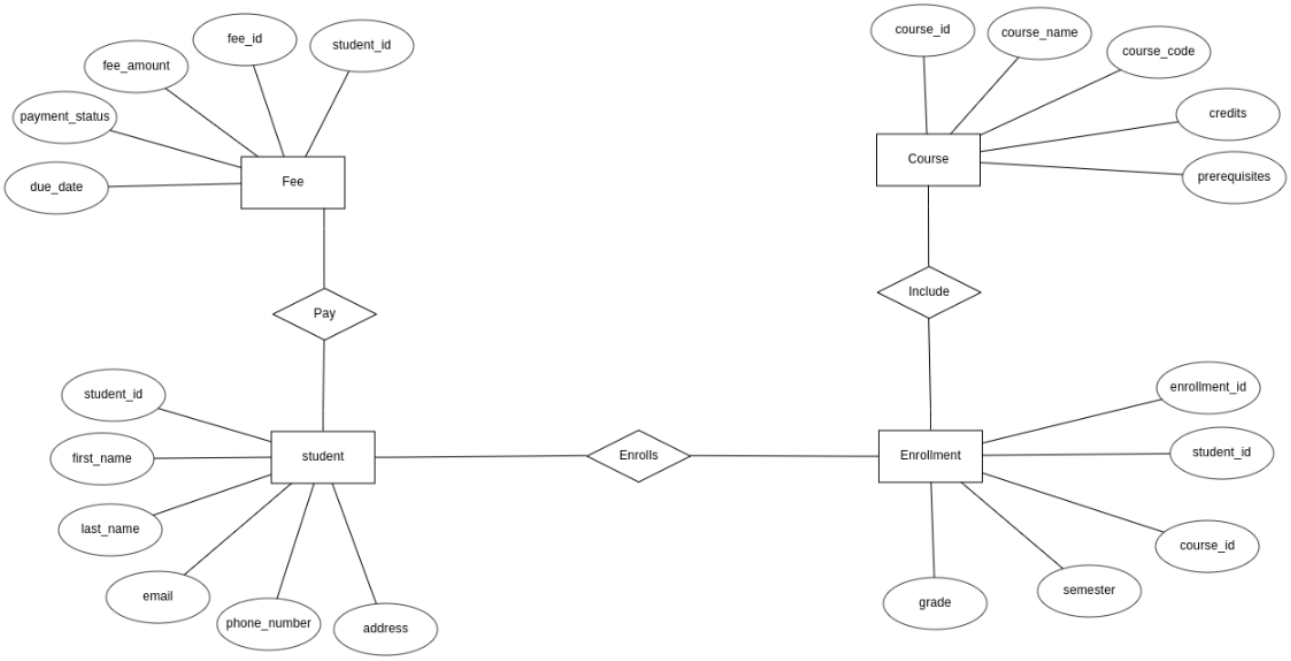
Tables

- **Student Table:** - Columns: student_id (Primary Key), name, address, phone_number, email
- **Course Table:** - Columns: course_id (Primary Key), course_name, course_code, credits, prerequisites
- **Enrollment Table:** - Columns: enrollment_id (Primary Key), student_id (Foreign Key), course_id (Foreign Key), semester, grade
- **Fee Table:** - Columns: fee_id (Primary Key), student_id (Foreign Key), fee_amount, due_date, payment_status

Relationships

- Student to Enrollment: One-to-Many relationship (One student can enroll in many courses).
- Course to Enrollment: One-to-Many relationship (One course can have many enrollments).
- Student to Fee: One-to-Many relationship (One student can have multiple fee records).

Entity-Relationship Diagram (ERD)



3. Database Implementation

Create Tables

```
MariaDB [(none)]> create database texas_DBMS_3rd_sem;
Query OK, 1 row affected (0.010 sec)
```

```
MariaDB [(none)]> use  texas_DBMS_3rd_sem;
Database changed
```

Student Table:

- Columns: student_id (Primary Key), first_name, last_name, address, phone_number, email

```sql
CREATE TABLE Student (
  student_id INT AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(255),
  last_name VARCHAR(255),
  address VARCHAR(255),
  phone_number VARCHAR(20),
  email VARCHAR(255)
);
```

Breakdown of Student Table

- **student_id** (INT): Unique identifier, auto-increment, and primary key.

- **first_name** (VARCHAR(255)): Student's first name.
- **last_name** (VARCHAR(255)): Student's last name.
- **address** (VARCHAR(255)): Student's address.
- **phone_number** (VARCHAR(20)): Student's phone number.
- **email** (VARCHAR(255)): Student's email address.

Course Table

- Columns: course_id (Primary Key), course_name, course_code, credits, prerequisites

```
CREATE TABLE Course
( course_id INT AUTO_INCREMENT PRIMARY KEY,
course_name VARCHAR(255),
course_code VARCHAR(10),
credits INT,
prerequisites VARCHAR(255)
);
```

Breakdown of Course Table

- "course_id" (INT): Unique identifier, auto-increment, and primary key.
- "course_name" (VARCHAR(255)): Name of the course.
- "course_code" (VARCHAR(10)): Code or identifier for the course.
- "credits" (INT): Number of credits assigned to the course.
- "prerequisites" (VARCHAR(255)): Any prerequisites or requirements for the course.

Enrollment Table:

- Columns: enrollment_id (Primary Key), student_id (Foreign Key), course_id (Foreign Key), semester, grade

```
CREATE TABLE Enrollment
( enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
student_id INT,
course_id INT,
semester VARCHAR(20),
grade VARCHAR(2),
FOREIGN KEY (student_id) REFERENCES Student(student_id),
FOREIGN KEY (course_id) REFERENCES Course(course_id) );
```

Breakdown of Enrollment Table

- **Columns:**
  - "enrollment_id" (INT): Unique identifier, auto-increment, and primary key.
  - "student_id" (INT): Foreign key referencing the "student_id" in the "Student" table.
  - "course_id" (INT): Foreign key referencing the "course_id" in the "Course" table.
  - "semester" (VARCHAR(20)): The semester in which the enrollment occurred.
  - "grade" (VARCHAR(2)): The grade achieved in the course.
- **Foreign Key Constraints:**
  - "student_id" references the "student_id" column in the "Student" table.
  - "course_id" references the "course_id" column in the "Course" table.

Fee Table:

- Columns: fee_id (Primary Key), student_id (Foreign Key), fee_amount, due_date, payment_status

```
CREATE TABLE Fee ( fee_id INT AUTO_INCREMENT PRIMARY KEY,
student_id INT,
fee_amount DECIMAL(10, 2),
due_date DATE,
payment_status VARCHAR(20),
FOREIGN KEY (student_id) REFERENCES Student(student_id) );
```

Breakdown of Fee Table

- **Columns:**
  - "fee_id" (INT): Unique identifier, auto-increment, and primary key.
  - "student_id" (INT): Foreign key referencing the "student_id" in the "Student" table.
  - "fee_amount" (DECIMAL(10, 2)): The amount of the fee with two decimal places.
  - "due_date" (DATE): The date by which the fee is due.
  - "payment_status" (VARCHAR(20)): The status of the fee payment.
- **Foreign Key Constraint:**
  - "student_id" references the "student_id" column in the "Student" table.

```
MariaDB [texas_DBMS_3rd_sem]> show tables;
+-----------------------------+
| Tables_in_texas_DBMS_3rd_sem |
+-----------------------------+
| Course                      |
| Enrollment                  |
| Fee                         |
| Student                     |
+-----------------------------+
4 rows in set (0.000 sec)
```

4. Insert Data into Tables

Insert into Student Table:

```sql
INSERT INTO Student (student_id, first_name, last_name, address,
phone_number, email)
VALUES
    (1, 'Sarita', 'Shrestha', 'Kathmandu, Nepal', '9841-234567',
'sarita@email.com'),
    (2, 'Rajesh', 'Sharma', 'Pokhara, Nepal', '9845-678912',
'rajesh@email.com'),
    (3, 'Anita', 'Gurung', 'Lalitpur, Nepal', '9813-456789',
'anita@email.com'),
    (4, 'Dipak', 'Khadka', 'Bhaktapur, Nepal', '9801-234567',
'dipak@email.com'),
    (5, 'Sunita', 'Thapa', 'Biratnagar, Nepal', '9815-678912',
'sunita@email.com'),
    (6, 'Ramesh', 'KC', 'Butwal, Nepal', '9856-789123', 'ramesh@email.com');
```

```
MariaDB [texas_DBMS_3rd_sem]> select * from Student;
+------------+------------+-----------+------------------+--------------+------------------+
| student_id | first_name | last_name | address          | phone_number | email            |
+------------+------------+-----------+------------------+--------------+------------------+
|          1 | Sarita     | Shrestha  | Kathmandu, Nepal | 9841-234567  | sarita@email.com |
|          2 | Rajesh     | Sharma    | Pokhara, Nepal   | 9845-678912  | rajesh@email.com |
|          3 | Anita      | Gurung    | Lalitpur, Nepal  | 9813-456789  | anita@email.com  |
|          4 | Dipak      | Khadka    | Bhaktapur, Nepal | 9801-234567  | dipak@email.com  |
|          5 | Sunita     | Thapa     | Biratnagar, Nepal| 9815-678912  | sunita@email.com |
|          6 | Ramesh     | KC        | Butwal, Nepal    | 9856-789123  | ramesh@email.com |
+------------+------------+-----------+------------------+--------------+------------------+
6 rows in set (0.000 sec)
```

Breakdown of Insert into Student Table

- In this code, each row in the "Student" table is defined by its own 'INSERT INTO' statement. The 'VALUES' keyword is used to specify the columns (student_id, first_name,

last_name, address, phone_number, and email) and provide the corresponding data for each row.

Insert into Course Table:

```sql
INSERT INTO Course (course_id, course_name, course_code, credits,
prerequisites)
VALUES
    (101, 'Introduction to Cyber Security', 'CS101', 3, NULL),
    (102, 'Cyber Security Fundamentals', 'CS201', 3, 'CS101'),
    (103, 'Network Security', 'NS301', 4, 'CS101'),
    (104, 'Ethical Hacking', 'EH401', 4, 'CS101'),
    (105, 'Privacy and Data Protection', 'PDP501', 2, NULL),
    (106, 'Advanced Cyber Threats', 'ACT601', 3, 'CS201'),
    (107, 'Incident Response and Recovery', 'IRR701', 3, 'CS301');
```

```
MariaDB [texas_DBMS_3rd_sem]> select * from Course;
+-----------+---------------------------------+-------------+---------+---------------+
| course_id | course_name                     | course_code | credits | prerequisites |
+-----------+---------------------------------+-------------+---------+---------------+
|       101 | Introduction to Cyber Security  | CS101       |       3 | NULL          |
|       102 | Cyber Security Fundamentals     | CS201       |       3 | CS101         |
|       103 | Network Security                | NS301       |       4 | CS101         |
|       104 | Ethical Hacking                 | EH401       |       4 | CS101         |
|       105 | Privacy and Data Protection     | PDP501      |       2 | NULL          |
|       106 | Advanced Cyber Threats          | ACT601      |       3 | CS201         |
|       107 | Incident Response and Recovery  | IRR701      |       3 | CS301         |
+-----------+---------------------------------+-------------+---------+---------------+
7 rows in set (0.000 sec)
```

Breakdown of Insert into Course Table

- The columns (course_id, course_name, course_code, credits, and prerequisites) and the pertinent information for each course are specified using the 'VALUES' keyword. We can effectively add the data required for each course to the 'Course' table using this process.

Insert into Enrollment Table:

```sql
INSERT INTO Enrollment (enrollment_id, student_id, course_id, semester, grade)
VALUES
    (1, 1, 101, 'Fall 2023', 'A'),
    (2, 2, 102, 'Fall 2023', 'B+'),
    (3, 3, 103, 'Fall 2023', 'A-'),
    (4, 4, 104, 'Fall 2023', 'B'),
    (5, 5, 105, 'Fall 2023', 'A'),
    (6, 6, 101, 'Fall 2023', 'B+'),
    (7, 1, 102, 'Fall 2023', 'A'),
    (8, 2, 103, 'Fall 2023', 'B'),
    (9, 3, 104, 'Fall 2023', 'A-');
```

```
MariaDB [texas_DBMS_3rd_sem]> select * from Enrollment;
+---------------+------------+-----------+-----------+-------+
| enrollment_id | student_id | course_id | semester  | grade |
+---------------+------------+-----------+-----------+-------+
|             1 |          1 |       101 | Fall 2023 | A     |
|             2 |          2 |       102 | Fall 2023 | B+    |
|             3 |          3 |       103 | Fall 2023 | A-    |
|             4 |          4 |       104 | Fall 2023 | B     |
|             5 |          5 |       105 | Fall 2023 | A     |
|             6 |          6 |       101 | Fall 2023 | B+    |
|             7 |          1 |       102 | Fall 2023 | A     |
|             8 |          2 |       103 | Fall 2023 | B     |
|             9 |          3 |       104 | Fall 2023 | A-    |
+---------------+------------+-----------+-----------+-------+
9 rows in set (0.000 sec)
```

Breakdown of Insert into Enrollment Table:

- Each enrollment record in the table is represented by an individual 'INSERT INTO' statement. The 'VALUES' keyword is utilized to specify the columns (enrollment_id, student_id, course_id, semester, and grade) and to provide the corresponding data for each enrollment. This process allows us to efficiently populate the 'Enrollment' table with enrollment details for different students in various courses during the Fall 2023 semester, including their respective grades.

Insert into Fee Table:

```sql
INSERT INTO Fee (fee_id, student_id, fee_amount, due_date,
payment_status)
VALUES
    (1, 1, 1500, '2023-09-30', 'Paid'),
    (2, 2, 1600, '2023-09-30', 'Paid'),
    (3, 3, 1700, '2023-09-30', 'Pending'),
    (4, 4, 1800, '2023-09-30', 'Paid'),
    (5, 5, 1900, '2023-09-30', 'Pending');
```

```
MariaDB [texas_DBMS_3rd_sem]> select * from Fee;
+--------+------------+------------+------------+----------------+
| fee_id | student_id | fee_amount | due_date   | payment_status |
+--------+------------+------------+------------+----------------+
|      1 |          1 |    1500.00 | 2023-09-30 | Paid           |
|      2 |          2 |    1600.00 | 2023-09-30 | Paid           |
|      3 |          3 |    1700.00 | 2023-09-30 | Pending        |
|      4 |          4 |    1800.00 | 2023-09-30 | Paid           |
|      5 |          5 |    1900.00 | 2023-09-30 | Pending        |
+--------+------------+------------+------------+----------------+
5 rows in set (0.000 sec)
```

Breakdown of Insert into Fee Table

- The 'VALUES' keyword is used to specify the columns (fee_id, student_id, fee_amount, due_date, and payment_status) and associated information for each fee record. By following this process, the 'Fee' table can be efficiently filled with financial information for different students, such as the fee amount, due date, and payment status.

5. Read Data

Aggregate Functions

In SQL, **aggregate functions** are used to perform calculations on sets of values within a group of rows, typically in a result set produced by a SELECT statement. These functions allow you to summarize and analyze data by calculating values such as sums, averages, counts, maximums, and minimums. Aggregate functions operate on a set of rows and return a single value as the result. They are essential for data analysis and reporting.

Types of Aggregate Functions:

1. **SUM**:
    - The `SUM` function determines the total value by adding up all the values in a numeric column.

- Example: `SELECT COUNT(*) AS Total_Students FROM Student;`

2. **AVG** (Average):
   - The `AVG` function determines the average (mean) value of a numeric column and outputs that value.
   - Example: `SELECT AVG(LENGTH(email)) AS Average_Email_Length FROM Student;`
3. **COUNT**:
   - The `COUNT` function calculates the count by counting the number of columns or non-null values in a row.
   - Example: `SELECT COUNT(phone_number) AS Students_With_Phone FROM Student;`
4. **MAX** (Maximum):
   - The `MAX` function returns the maximum value by locating the highest value within a column.
   - Example: `SELECT MAX(student_id) AS Max_Student_ID FROM Student;`
5. **MIN** (Minimum):
   - The `MIN` function returns the minimum value by locating the lowest value within a column.
   - Example: `SELECT MIN(LENGTH(address)) AS Min_Address_Length FROM Student;`

Joins

- **INNER JOIN**: When there is a match based on the "student_id" column, an INNER JOIN is used to combine the data from the "Student" and "Fee" tables.

```sql
SELECT s.student_id, s.first_name, s.last_name, f.amount_paid
FROM Student s
INNER JOIN Fee f ON s.student_id = f.student_id;
```

- **LEFT JOIN**: All "Student" rows and related "Fee" rows are combined using an LEFT JOIN. When there is no match in "Fee," the "Student" data is retained with NULL values in the "Fee" columns.

```sql
SELECT s.student_id, s.first_name, s.last_name, f.amount_paid
FROM Student s
LEFT JOIN Fee f ON s.student_id = f.student_id;
```

- **RIGHT JOIN**: Similar to an LEFT JOIN, a RIGHT JOIN keeps all "Fee" rows and any matching "Student" rows. If there is no match in "Student," the "Fee" data is retained with NULL values in the "Student"

```sql
SELECT s.student_id, s.first_name, s.last_name, f.amount_paid
FROM Student s
RIGHT JOIN Fee f ON s.student_id = f.student_id;
```

- **FULL OUTER JOIN**: All rows from both the "Student" and the "Fee" tables are included in a FULL OUTER JOIN. Where no match is found, it returns NULL values instead of matching data.

```sql
SELECT s.student_id, s.first_name, s.last_name, f.amount_paid
FROM Student s
FULL OUTER JOIN Fee f ON s.student_id = f.student_id;
```

6. Conclusion

The goal of this project is to create and put into use a well-organized database with four main tables: Student, Course, Enrollment, and Fee. These tables are linked together to manage student records, course information, enrollments, and financial transactions effectively while preserving the accuracy and accessibility of the data.

The database is now fully populated thanks to data insertion, making data administration and retrieval more effective. The ability to analyze data in-depth is made possible by the use of SQL joins and aggregate functions like COUNT, SUM, and AVG. In the end, this project helps the college fulfill its purpose of providing top-notch instruction and administrative effectiveness.

Recommendations

1. **Database Indexing (Opinion):** I'd create indexes on frequently queried columns (e.g., student_id) to enhance query performance.
2. **Error Handling (Opinion):** I'd improve error handling to provide informative error messages for better debugging and troubleshooting.
3. **Security (Opinion):** I'd strengthen database security with proper user authentication and authorization mechanisms, along with data encryption for sensitive information.
4. **Optimization (Opinion):** I'd regularly review and optimize database queries and schema for improved performance using profiling tools to identify bottlenecks.

7. Reference

- Stack Overflow (https://stackoverflow.com)
- Tutorialspoint (https://www.tutorialspoint.com)
- Javatpoint (https://www.javatpoint.com)