



Symbiosis Institute of Technology

A DBMS Project Report on Integrated Student Management System



Under the Guidance of

Dr. Vijayshree Khedkar

Group Members:

| Sr. No. | Name | PRN | Division |
|---------|-----------------|-------------|----------|
| 1 | Abhishek Rajput | 22070122007 | CSE-A1 |
| 2 | Aditya Raj | 22070122009 | CSE-A1 |
| 3 | Archit Patil | 22070122025 | CSE-A1 |
| 4 | Arnav Jain | 22070122030 | CSE-A1 |

Department of Computer Science

Academic Session 2023-24

Program Name: B. Tech SYCS

Semester: IV

Table of Contents

| | |
|---|-----------|
| Title of Project | 3 |
| About Project | 3 |
| ER Diagram | 4 |
| Use Case Diagram | 5 |
| Anomalies..... | 6 |
| 1. Insertion Anomalies | 6 |
| 2. Updation Anomalies: | 6 |
| 3. Deletion Anomalies: | 6 |
| Anomalies occurring in our database | 7 |
| Insertion Anomaly: | 7 |
| Deletion Anomaly: | 7 |
| Updation Anomaly: | 7 |
| Functional dependencies | 8 |
| Functional dependencies of our Relational Schema | 10 |
| Normalization | 12 |
| Types of normalization | 12 |
| Normalization occurring in our database | 13 |
| What is MySQL? | 16 |
| Implementation of the tables in MySQL | 17 |
| Queries implemented | 31 |
| Function | 35 |
| Procedure | 37 |
| Trigger | 39 |
| AWS and its Implementation | 41 |
| Front end sample design | 42 |
| Github URL..... | 43 |
| References | 44 |

Title of Project

Integrated Student Management System

About Project

In today's expanding landscape it is crucial to have a streamlined and comprehensive approach to managing administrative and academic functions. Our project introduces a Database Management System (DBMS) specifically designed to meet these requirements. Our system offers a solution for student management incorporating key functionalities such as fee management, hostel accommodation, student records, course administration, library management, timetable scheduling, faculty coordination, attendance tracking and grade management.

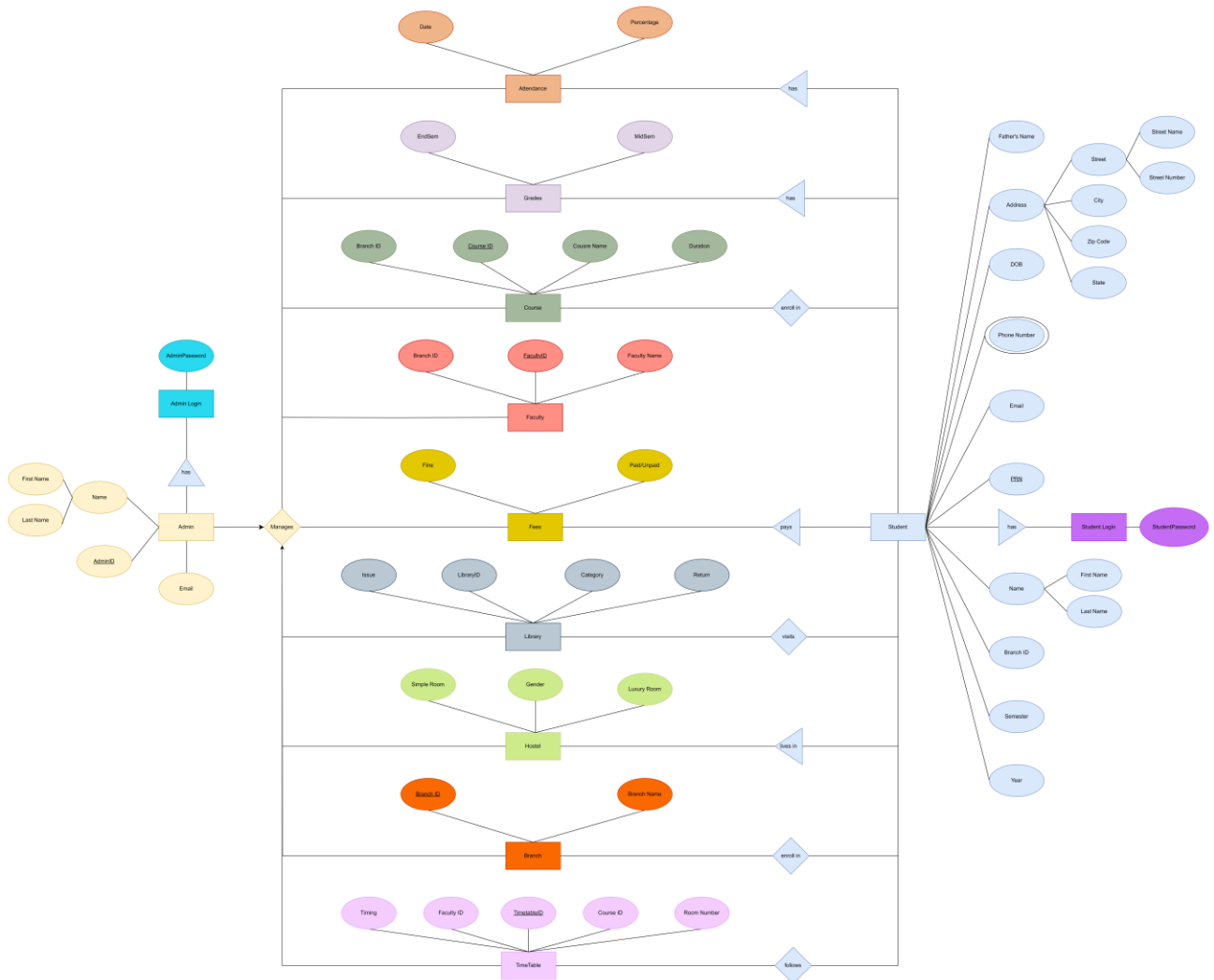
The fee management module of our system simplifies transactions while ensuring transparency and efficiency. Managing hostel accommodations becomes effortless with our process for allocating and maintaining student housing. The student module acts as a repository for all information related to students – an essential component for smooth administration. Course administration is enhanced by features that Facilitate planning and management.

A pivotal feature of our system is the timetable scheduling function that optimizes resource utilization and minimizes scheduling conflicts – resulting in benefits for both students and faculty members. The faculty coordination module provides a platform, for managing faculty information and schedules – an aspect of smooth academic operations.

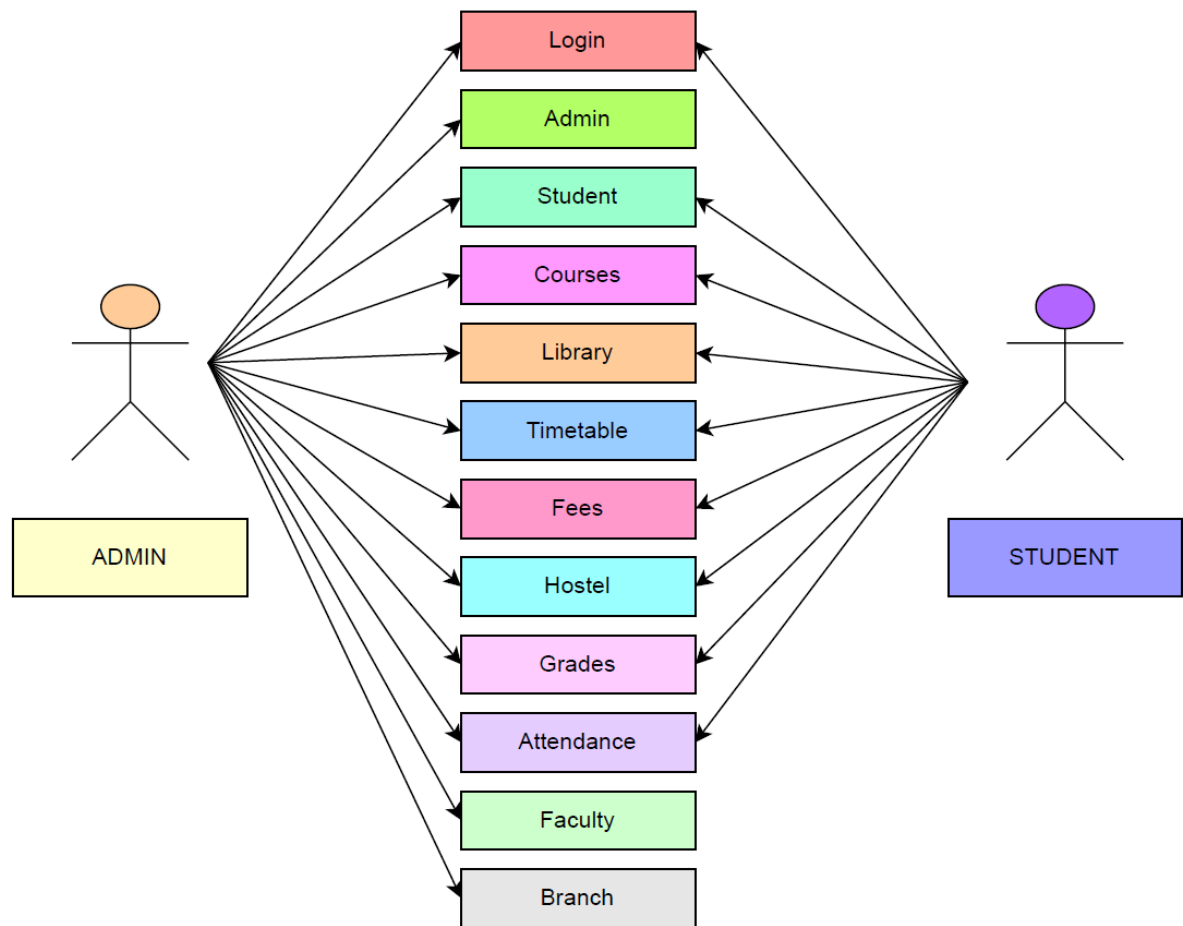
Attendance tracking is automated to ensure accuracy and convenience while the grading system guarantees an efficient process, for evaluating student performance.

This comprehensive student management system, driven by a database management system (DBMS) goes beyond being a technological solution. It is a tool, which simplifies processes while significantly improving the quality of education delivery.

ER Diagram



Use Case Diagram



Anomalies

In the terms of DBMS, anomalies refers to undesirable or inconsistency in data that occurs due to one or more database operations. These anomalies can occur in various situations, typically when performing data manipulation operations such as insert, update, or delete. There are three main types of anomalies in DBMS:

1. Insertion Anomalies:

Insert anomalies occur when certain attributes cannot be inserted into the database due to missing additional data.

- **Redundancy Anomalies:** When new records are added, redundant data may be introduced, resulting in duplicated information.
- **Constraint Violation:** An insertion will be refused and the database may become inconsistent if the data is not consistent with the integrity constraints specified in the schema (such as unique, primary and foreign key constraints).
- **Incompleteness:** When necessary data is absent, insertion anomalies may also occur, resulting in problems with data accuracy and integrity.

2. Deletion Anomalies:

Delete anomalies occur when deleting a certain row inadvertently leads to the deletion of other important data.

- **Partial Update:** Inconsistencies may arise if a record's data is only updated in part.
- **Inconsistent Update:** Inconsistencies may arise when relevant data is updated in one table without also being updated in other tables.

3. Updation Anomalies:

Update anomalies occur when the same data is repeated in multiple rows, and changes are made in some but not all instances.

- **Loss of Data:** Errors in record deletion may result in the loss of related data.
- **Unintended Deletion:** A user may inadvertently remove crucial information from the database.

Anomalies occurring in our database

Insertion Anomaly:

In our schema, insertion anomalies can occur when trying to insert a new student into the Student table before they are assigned a branch or enrolled in any courses. Since BranchID is a foreign key in the Student table, a student must belong to a branch. If a student is inserted before the branch is assigned, it would violate referential integrity constraints.

Deletion Anomaly:

Deletion anomalies may occur when deleting data from the Faculty table. If a faculty member is deleted, it might cause unintended deletion of their related records in TimeTable table if referential integrity constraints are not properly established.

Updation Anomaly:

If faculty member's details change, updating these details could lead to inconsistencies if the updated information is not propagated correctly. For example, if a faculty member's branch ID changes but it's not updated in TimeTable table, the timetable might display outdated information.

Functional Dependencies

In a relational database management system (DBMS), various types of dependencies define the relationships between attributes in tables. Here are the key types of dependencies with examples:

1. Functional Dependency:

If one attribute uniquely determines another within the same table, it's called a functional dependency, denoted as $P \rightarrow Q$.

Example:

In the "Branch" table:

$\text{BranchID} \rightarrow \text{BranchName}$ (BranchName is functionally dependent on BranchID).

2. Fully-Functional Dependency:

An attribute is fully functionally dependent on another if it depends on it and not on any proper subset of it.

Example:

In the "TimeTable" table:

$\{\text{TimetableID}, \text{CourseID}\} \rightarrow \text{FacultyID}, \text{RoomNumber}, \text{Timing}$ (FacultyID, RoomNumber, Timing are fully functionally dependent on both TimetableID, CourseID).

3. Transitive Dependency:

When an indirect relationship causes functional dependency, it's called a transitive dependency.

Example:

In the "Faculty" table:

$\{\text{FacultyID} \rightarrow \text{BranchID}\}$ and $\{\text{BranchID} \rightarrow \text{BranchName}\}$, then $\{\text{FacultyID} \rightarrow \text{BranchName}\}$ is a transitive dependency. (The faculty's ID indirectly determines the branch name through the branch ID.)

If $P \rightarrow Q$ and $Q \rightarrow R$, then $P \rightarrow R$ is a transitive dependency.

4. Multivalued Dependency:

It occurs when the existence of one or more rows in a table implies one or more other rows in the same table.

Example:

If a table has attributes P, Q, and R, and Q and R depend on P, it's represented as $P \twoheadrightarrow Q \twoheadrightarrow R$.

5. **Partial Dependency:**

Partial Dependency happens when a non-prime attribute is functionally dependent on part of a candidate key.

Example:

In the "Hostel" table:

Gender is partially dependent on PRN (The gender attribute depends only on part of the candidate key, PRN.)

SimpleRoom and LuxuryRoom are partially dependent on PRN (These attributes depend only on part of the candidate key, PRN.)

These dependencies are essential in database design to ensure data integrity and eliminate anomalies.

Functional Dependencies of our Relational Schema

Admin Table:

AdminID -> FirstName, LastName, Email

Email -> AdminID

Branch Table:

BranchID -> BranchName

Student Table:

PRN -> FirstName, LastName, Year, BranchID, Semester, DOB, FathersName, Email,
AddressStreetName, AddressStreetNumber, AddressZipCode, AddressState, AddressCity

Email -> PRN

BranchID -> BranchName

PhoneNumber Table:

PRN -> PhoneNumber1, PhoneNumber2

AdminLogin Table:

AdminID -> AdminPassword

StudentLogin Table:

PRN -> StudentPassword

Hostel Table:

PRN -> Gender, SimpleRoom, LuxuryRoom

Library Table:

LibraryID -> Category, Issue, Return1

Faculty Table:

FacultyID -> FacultyName, BranchID

Course Table:

CourseID -> CourseName, Duration, BranchID

TimeTable Table:

TimetableID -> CourseID, FacultyID, RoomNumber, Timing

Grades Table:

PRN -> MidSem, EndSem

Attendance Table:

PRN -> Percentage, Date1

Fees Table:

PRN -> PaidUnpaid, Fine

Normalization

Normalization in Database Management Systems (DBMS) is a process of organizing and structuring the data in a relational database to reduce data redundancy and eliminate certain types of data anomalies. The major objectives of normalisation are to guarantee effective data storage, data integrity and consistency, and ease of handling and maintaining over time. The process of normalization follows a set of rules, typically referred to as normal forms.

Types of normalization

1. First Normal Form (1NF): In 1NF, each column in a table must contain atomic (indivisible) values, and each entry in the table must be unique. It eliminates the possibility of storing multiple values in a single column.

2. Second Normal Form (2NF): A table is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key. This means that each non-key attribute is dependent on the entire primary key, not just a part of it.

3. Third Normal Form (3NF): A table is in 3NF if it is in 2NF, and it has no transitive dependencies. Transitive dependencies occur when a non-key attribute is dependent on another non-key attribute, which is itself dependent on the primary key.

4. Boyce-Codd Normal Form (BCNF): BCNF is a stricter form of 3NF in which for every non-trivial functional dependency, the left-hand side must be a superkey (a set of attributes that can uniquely identify a row in the table). BCNF eliminates partial dependencies, where a non-key attribute is dependent on only part of the primary key.

5. Fourth Normal Form (4NF): 4NF addresses multi-valued dependencies, which occur when an attribute depends on multiple values of another attribute in the same row. 4NF eliminates such dependencies.

6. Fifth Normal Form (5NF): 5NF addresses cases where a table has multiple candidate keys, and a non-key attribute depends on a proper subset of the candidate keys. 5NF eliminates these dependencies.

Normalization occurring in our database

For 1NF:

Atomic Values: Each attribute in all the tables contain atomic values.

For example, In Student Table (FirstName, LastName, Year, BranchID, Semester, DOB, FathersName, Email, AddressStreetName, AddressStreetNumber, AddressZipCode, AddressState, and AddressCity) all represent singular pieces of information without containing multiple values.

| | PRN | FirstName | LastName | Year | BranchID | Semester | DOB | FathersName | Email | AddressStreetName | AddressStreetNumber | AddressZipCode | AddressState | AddressCity |
|---|------|-----------|----------|------|----------|----------|------------|-------------|--------------------|-------------------|---------------------|----------------|---------------|-------------|
| ▶ | 1001 | Abhishek | Rajput | 2 | 5001 | 4 | 2000-05-15 | Mr. Rajput | abhishek@gmail.com | Main Street | 123 | 400001 | Maharashtra | Mumbai |
| | 1002 | Aditya | Raj | 3 | 5001 | 6 | 1999-08-20 | Mr. Raj | aditya@gmail.com | Park Avenue | 456 | 110001 | Delhi | New Delhi |
| | 1003 | Archit | Patil | 1 | 5002 | 2 | 2001-02-10 | Mr. Patil | archit@gmail.com | Broadway | 789 | 560001 | Karnataka | Bangalore |
| | 1004 | Arnav | Jain | 4 | 5002 | 8 | 1998-11-25 | Mr. Jain | arnav@gmail.com | Green Street | 1011 | 600001 | Tamil Nadu | Chennai |
| | 1005 | Aryan | Sharma | 2 | 5003 | 4 | 2000-07-12 | Mr. Sharma | aryan@gmail.com | High Street | 1314 | 700001 | West Bengal | Kolkata |
| | 1006 | Avinash | Gupta | 3 | 5003 | 6 | 1999-09-18 | Mr. Gupta | avinash@gmail.com | River Road | 1516 | 500001 | Telangana | Hyderabad |
| | 1007 | Amita | Singh | 1 | 5004 | 2 | 2001-03-22 | Mr. Singh | amita@gmail.com | Lake Street | 1718 | 380001 | Gujarat | Ahmedabad |
| | 1008 | Akash | Kumar | 4 | 5004 | 8 | 1998-12-30 | Mr. Kumar | akash@gmail.com | Gandhi Road | 1920 | 250001 | Uttar Pradesh | Agra |
| | 1009 | Ananya | Yadav | 2 | 5005 | 4 | 2000-06-08 | Mr. Yadav | ananya@gmail.com | Market Street | 2122 | 560002 | Karnataka | Mysore |
| | 1010 | Ankit | Verma | 3 | 5005 | 6 | 1999-10-14 | Mr. Verma | ankit@gmail.com | Station Road | 2324 | 110002 | Delhi | New Delhi |
| | 1011 | Aditi | Shah | 1 | 5006 | 2 | 2001-04-26 | Mr. Shah | aditi@gmail.com | Victoria Street | 2526 | 600002 | Tamil Nadu | Coimbatore |
| | 1012 | Ayesha | Malik | 4 | 5006 | 8 | 1998-11-03 | Mr. Malik | ayasha@gmail.com | Church Street | 2728 | 700002 | West Bengal | Howrah |
| * | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

No Repeating Groups: There are no repeating groups or arrays of data within any attribute. Each attribute represents a single piece of information about a student without needing to repeat values within the same field.

For example, In the Student table, before the PhoneNumber table was introduced, phone numbers might have been stored directly as attributes. This could lead to repeating groups if a student has multiple phone numbers. By placing phone numbers in a separate table, we avoid this issue and ensure that each attribute in the Student table represents a single piece of information without repeating groups.

| | PRN | PhoneNumber 1 | PhoneNumber 2 |
|---|------|---------------|---------------|
| ▶ | 1001 | 9650104541 | 9650104541 |
| | 1002 | 7720860456 | 9650104542 |
| | 1003 | 8639914092 | 9650104543 |
| | 1004 | 9377852020 | 9650104544 |
| | 1005 | 9234567890 | 9876543210 |
| | 1006 | 9345678901 | 9012345678 |
| | 1007 | 9456789012 | 8901234567 |
| | 1008 | 9567890123 | 9890123456 |
| | 1009 | 9678901234 | 8789012345 |
| | 1010 | 6789012345 | 9678901234 |
| | 1011 | 8890123456 | 9567890123 |
| | 1012 | 8901234567 | 8456789012 |
| * | NULL | NULL | NULL |

For 2NF:

No Partial Dependencies: Each attribute in all the tables have no Partial Dependencies.

For example, Each attribute in the Student table directly relates to student-specific information, There are no attributes that depend only on a subset of the primary key. (FirstName, LastName, Year, BranchID, Semester, DOB, FathersName, Email, AddressStreetName, AddressStreetNumber, AddressZipCode, AddressState, and AddressCity) are all attributes that describe individual aspects of a student's profile. Each of these attributes is directly associated with the PRN, indicating that there are no partial dependencies.

For 3NF:

No Transitive Dependencies: Each attribute in all the tables no Transitive Dependencies.

For example, In the Student table, we ensured no transitive dependencies exist which means that all non-prime attributes are directly dependent on the primary key and not on other non-prime attributes. BranchID is dependent on the PRN (primary key), not on any other non-prime attribute like Year or Semester. Similarly, AddressState depends on PRN, not on AddressCity or any other attribute.

For BCNF:

Functional Dependencies: To ensure that all the tables satisfies the requirements of Boyce-Codd Normal Form (BCNF), we need to ensure that for every non-trivial functional dependency ($X \rightarrow Y$) in the table, X is a superkey.

For example, In the student table,

PRN \rightarrow FirstName, LastName, Year, BranchID, Semester, DOB, FathersName, Email, AddressStreetName, AddressStreetNumber, AddressZipCode, AddressState, AddressCity

All non-prime attributes (FirstName, LastName, Year, BranchID, Semester, DOB, FathersName, Email, AddressStreetName, AddressStreetNumber, AddressZipCode, AddressState, AddressCity) are fully functionally dependent on the primary key (PRN) which means Student table satisfies the condition where all non-trivial functional dependencies are determined by the superkey (PRN), so it is in Boyce-Codd Normal Form (BCNF).

For 4NF:

No Multi-Valued Dependencies (MVDs): Each attribute in all the tables have no multi-valued dependencies.

For example, In the Student table, Each attribute is uniquely determined by the primary key (PRN), and there are no sets of attributes that exhibit non-trivial dependencies on each other.

For 5NF:

No Further Decomposition Possible: In all the tables, No Further Decomposition is Possible.

For example, The Student table is already in a state where no further decomposition is possible without losing information or introducing redundancy. Each attribute is functionally dependent on the primary key (PRN), and there are no dependencies between non-key attributes.

What is MySQL?

Popular for its dependability, speed, and user-friendliness, MySQL is an open-source relational database management system (RDBMS). It was created by MySQL AB at first, but Oracle Corporation currently owns it. MySQL is a relational database that arranges data into tables with rows and columns and uses SQL for data management and querying. Because MySQL is open-source and offers a wide range of customization options at a reasonable cost, it is especially appealing to developers and organizations.

Because MySQL is cross-platform compatible, it can be used with a variety of operating systems and platforms, including Windows, Linux, macOS, and more. Furthermore, MySQL is known for its fast performance, which makes it ideal for applications that need to respond quickly by guaranteeing effective data storage and retrieval.

MySQL prioritizes security by providing tools like access control, user authentication, and data encryption. Its sizable and vibrant user base offers excellent documentation, resources, and assistance. Moreover, it complies with ACID, guaranteeing consistency and dependability of transactions.

MySQL provides support for replication, clustering, and other high-availability techniques to guarantee data redundancy and fault tolerance. It is a well-liked option for web apps since it also integrates readily with a wide range of development frameworks and programming languages. For many years, MySQL has been a mainstay of the web development ecosystem, powering dynamic websites and applications. It is frequently used in conjunction with technologies like PHP, Python, and Ruby on Rails.

Implementation of the tables in MySQL

Database Creation

```
CREATE DATABASE studentdb;  
USE studentdb;
```

Table Creation

-- Admin Table

```
CREATE TABLE Admin(  
    AdminID INT PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50),  
    Email VARCHAR(50) UNIQUE  
);
```

```
CREATE TABLE BRANCH(  
    BranchID INT PRIMARY KEY,  
    BranchName VARCHAR(50)  
);
```

-- Student Table

```
CREATE TABLE Student (  
    PRN INT PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50),  
    Year INT,  
    BranchID INT,  
    Semester INT,  
    DOB DATE,  
    FathersName VARCHAR(50),  
    Email VARCHAR(50) UNIQUE,  
    AddressStreetName VARCHAR(50),  
    AddressStreetNumber VARCHAR(50),  
    AddressZipCode VARCHAR(10),  
    AddressState VARCHAR(50),  
    AddressCity VARCHAR(50),  
    FOREIGN KEY (BranchID) REFERENCES Branch(BranchID)  
);
```

-- PhoneNumber Table

```
CREATE TABLE PhoneNumber (  
    PRN INT PRIMARY KEY,  
    PhoneNumber1 VARCHAR(15) UNIQUE,
```

```
    PhoneNumber2 VARCHAR(15) UNIQUE,  
    FOREIGN KEY (PRN) REFERENCES Student(PRN)  
);
```

-- AdminLogin Table

```
CREATE TABLE AdminLogin (  
    AdminID INT PRIMARY KEY,  
    AdminPassword VARCHAR(50),  
    FOREIGN KEY (AdminID) REFERENCES Admin(AdminID)  
);
```

-- StudentLogin Table

```
CREATE TABLE StudentLogin (  
    PRN INT PRIMARY KEY,  
    StudentPassword VARCHAR(50),  
    FOREIGN KEY (PRN) REFERENCES Student(PRN)  
);
```

-- Hostel Table

```
CREATE TABLE Hostel (  
    PRN INT PRIMARY KEY,  
    Gender VARCHAR(6),  
    SimpleRoom VARCHAR(3),  
    LuxuryRoom VARCHAR(3),  
    FOREIGN KEY (PRN) REFERENCES Student(PRN)  
);
```

-- Library Table

```
CREATE TABLE Library (  
    LibraryID INT,  
    Category VARCHAR(50),  
    Issue DATE,  
    Return1 DATE  
);
```

-- Faculty Table

```
CREATE TABLE Faculty (  
    FacultyID INT PRIMARY KEY,  
    FacultyName VARCHAR(50),  
    BranchID INT,  
    FOREIGN KEY (BranchID) REFERENCES Branch(BranchID)  
);
```

-- Course Table

```
CREATE TABLE Course (  
    CourseID INT PRIMARY KEY,  
    CourseName VARCHAR(50),  
    Duration VARCHAR(50),  
    BranchID INT,  
    FOREIGN KEY (BranchID) REFERENCES Branch(BranchID)  
);
```

-- TimeTable Table

```
CREATE TABLE TimeTable (  
    TimetableID INT PRIMARY KEY,  
    CourseID INT,  
    FacultyID INT,  
    RoomNumber VARCHAR(50),  
    Timing VARCHAR(50),  
    FOREIGN KEY (FacultyID) REFERENCES Faculty(FacultyID),  
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)  
);
```

-- Grades Table

```
CREATE TABLE Grades (  
    PRN INT PRIMARY KEY,  
    MidSem FLOAT,  
    EndSem FLOAT,  
    FOREIGN KEY (PRN) REFERENCES Student(PRN)  
);
```

-- Attendance Table

```
CREATE TABLE Attendance (  
    PRN INT PRIMARY KEY,  
    Percentage FLOAT,  
    Date1 DATE,  
    FOREIGN KEY (PRN) REFERENCES Student(PRN)  
);
```

-- Fees Table

```
CREATE TABLE Fees (  
    PRN INT PRIMARY KEY,  
    PaidUnpaid VARCHAR(10),  
    Fine FLOAT,  
    FOREIGN KEY (PRN) REFERENCES Student(PRN)  
);
```

Table Insertion

-- Insert values into Admin Table

```
INSERT INTO Admin (AdminID, FirstName, LastName, Email)
VALUES
```

```
(1, 'Raj', 'Patel', 'raj@gmail.com'),
(2, 'Priya', 'Sharma', 'priya@gmail.com'),
(3, 'Amit', 'Gupta', 'amit@gmail.com'),
(4, 'Sneha', 'Desai', 'sneha@gmail.com');
```

-- Insert values into Branch Table

```
INSERT INTO BRANCH (BranchID, BranchName)
VALUES
```

```
(5001, 'Computer Science Engineering'),
(5002, 'Electrical Engineering'),
(5003, 'Mechanical Engineering'),
(5004, 'Civil Engineering'),
(5005, 'Artificial Intelligence Engineering'),
(5006, 'Robotics And Automation Engineering');
```

-- Insert values into Student Table

```
INSERT INTO Student (PRN, FirstName, LastName, Year, BranchID, Semester, DOB,
FathersName, Email, AddressStreetName, AddressStreetNumber, AddressZipCode,
AddressState, AddressCity)
VALUES
```

```
(1001, 'Abhishek', 'Rajput', 2, 5001, 4, '2000-05-15', 'Mr. Rajput', 'abhishek@gmail.com',
'Main Street', '123', '400001', 'Maharashtra', 'Mumbai'),
(1002, 'Aditya', 'Raj', 3, 5001, 6, '1999-08-20', 'Mr. Raj', 'aditya@gmail.com', 'Park Avenue',
'456', '110001', 'Delhi', 'New Delhi'),
(1003, 'Archit', 'Patil', 1, 5002, 2, '2001-02-10', 'Mr. Patil', 'archit@gmail.com', 'Broadway',
'789', '560001', 'Karnataka', 'Bangalore'),
(1004, 'Arnav', 'Jain', 4, 5002, 8, '1998-11-25', 'Mr. Jain', 'arnav@gmail.com', 'Green Street',
'1011', '600001', 'Tamil Nadu', 'Chennai'),
(1005, 'Aryan', 'Sharma', 2, 5003, 4, '2000-07-12', 'Mr. Sharma', 'aryan@gmail.com', 'High
Street', '1314', '700001', 'West Bengal', 'Kolkata'),
(1006, 'Avinash', 'Gupta', 3, 5003, 6, '1999-09-18', 'Mr. Gupta', 'avinash@gmail.com', 'River
Road', '1516', '500001', 'Telangana', 'Hyderabad'),
(1007, 'Amita', 'Singh', 1, 5004, 2, '2001-03-22', 'Mr. Singh', 'amita@gmail.com', 'Lake
Street', '1718', '380001', 'Gujarat', 'Ahmedabad'),
(1008, 'Akash', 'Kumar', 4, 5004, 8, '1998-12-30', 'Mr. Kumar', 'akash@gmail.com', 'Gandhi
Road', '1920', '250001', 'Uttar Pradesh', 'Agra'),
(1009, 'Ananya', 'Yadav', 2, 5005, 4, '2000-06-08', 'Mr. Yadav', 'ananya@gmail.com', 'Market
Street', '2122', '560002', 'Karnataka', 'Mysore'),
```

(1010, 'Ankit', 'Verma', 3, 5005, 6, '1999-10-14', 'Mr. Verma', 'ankit@gmail.com', 'Station Road', '2324', '110002', 'Delhi', 'New Delhi'),
(1011, 'Aditi', 'Shah', 1, 5006, 2, '2001-04-26', 'Mr. Shah', 'aditi@gmail.com', 'Victoria Street', '2526', '600002', 'Tamil Nadu', 'Coimbatore'),
(1012, 'Ayesha', 'Malik', 4, 5006, 8, '1998-11-03', 'Mr. Malik', 'ayesha@gmail.com', 'Church Street', '2728', '700002', 'West Bengal', 'Howrah');

-- Insert values into PhoneNumber Table

```
INSERT INTO PhoneNumber (PRN, PhoneNumber1, PhoneNumber2)
VALUES
(1001, '9650104541', '9650104541'),
(1002, '7720860456', '9650104542'),
(1003, '8639914092', '9650104543'),
(1004, '9377852020', '9650104544'),
(1005, '9234567890', '9876543210'),
(1006, '9345678901', '9012345678'),
(1007, '9456789012', '8901234567'),
(1008, '9567890123', '9890123456'),
(1009, '9678901234', '8789012345'),
(1010, '6789012345', '9678901234'),
(1011, '8890123456', '9567890123'),
(1012, '8901234567', '8456789012');
```

-- Insert values into AdminLogin Table

```
INSERT INTO AdminLogin (AdminID, AdminPassword)
VALUES
(1, 'password1'),
(2, 'password2'),
(3, 'password3'),
(4, 'password4');
```

-- Insert values into StudentLogin Table

```
INSERT INTO StudentLogin (PRN, StudentPassword)
VALUES
(1001, 'studentpass1'),
(1002, 'studentpass2'),
(1003, 'studentpass3'),
(1004, 'studentpass4'),
(1005, 'studentpass5'),
(1006, 'studentpass6'),
(1007, 'studentpass7'),
(1008, 'studentpass8'),
(1009, 'studentpass9');
```

```
(1010, 'studentpass10'),  
(1011, 'studentpass11'),  
(1012, 'studentpass12');
```

-- Insert values into Hostel Table

```
INSERT INTO Hostel (PRN, Gender, SimpleRoom, LuxuryRoom)  
VALUES  
(1001, 'Male', 'Yes', 'No'),  
(1002, 'Male', 'No', 'Yes'),  
(1003, 'Male', 'Yes', 'No'),  
(1004, 'Male', 'No', 'Yes'),  
(1005, 'Male', 'Yes', 'No'),  
(1006, 'Male', 'Yes', 'No'),  
(1007, 'Female', 'No', 'Yes'),  
(1008, 'Male', 'Yes', 'No'),  
(1009, 'Female', 'No', 'Yes'),  
(1010, 'Male', 'Yes', 'No'),  
(1011, 'Female', 'Yes', 'No'),  
(1012, 'Female', 'No', 'Yes');
```

-- Insert values into Library Table

```
INSERT INTO Library (LibraryID, Category, Issue, Return1)  
VALUES  
(1001, 'Science', '2023-01-10', '2023-02-10'),  
(1002, 'Literature', '2023-02-15', '2023-03-15'),  
(1003, 'Mathematics', '2023-03-20', '2023-04-20'),  
(1004, 'History', '2023-04-25', '2023-05-25'),  
(1001, 'Civics', '2023-05-26', '2023-06-28'),  
(1001, 'Geo', '2023-05-26', Null),  
(1005, 'Science', '2023-03-10', '2023-06-10'),  
(1007, 'Physics', '2023-01-17', Null),  
(1012, 'Mathematics-3', '2023-02-20', '2023-04-20'),  
(1009, 'C++', '2023-07-24', '2023-10-25'),  
(1010, 'Python', '2023-05-26', Null),  
(1004, 'Geo', '2023-05-26', '2023-08-11');
```

-- Insert values into Faculty Table

```
INSERT INTO Faculty (FacultyID, FacultyName, BranchID)  
VALUES  
(101, 'Dr. Sahil Gupta', 5001),  
(102, 'Prof. Aryan Sharma', 5001),  
(103, 'Dr. Arnav Singh', 5001),  
(104, 'Prof. Archit Patel', 5002),
```

(105, 'Dr. Faraj Khan', 5002),
 (106, 'Prof. Dhurmil Verma', 5002),
 (107, 'Dr. Dhruv Joshi', 5003),
 (108, 'Prof. Pranav Reddy', 5003),
 (109, 'Dr. Ankit Malhotra', 5003),
 (110, 'Prof. Aman Gupta', 5004),
 (111, 'Dr. Ansh Sharma', 5004),
 (112, 'Prof. Vijay Singh', 5004),
 (113, 'Dr. Harsh Kumar', 5005),
 (114, 'Prof. Abhay Desai', 5005),
 (115, 'Dr. Ram Patel', 5005),
 (116, 'Prof. Amit Shah', 5006),
 (117, 'Dr. Tarak Mehta', 5006),
 (118, 'Prof. Divyansh Choudhury', 5006);

-- Insert values into Course Table

INSERT INTO Course (CourseID, CourseName, Duration, BranchID)
 VALUES

(601, 'Database Management', '3 months', 5001),
 (602, 'Power Systems', '4 months', 5001),
 (603, 'Thermodynamics', '5 months', 5001),
 (604, 'Structural Analysis', '6 months', 5002),
 (605, 'Java', '5 months', 5002),
 (606, 'Data Structures', '3 months', 5002),
 (607, 'Renewable Energy', '4 months', 5003),
 (608, 'Fluid Mechanics', '5 months', 5003),
 (609, 'Transportation Engineering', '6 months', 5003),
 (610, 'Python Programming', '5 months', 5004),
 (611, 'Machine Learning', '4 months', 5004),
 (612, 'Control Systems', '5 months', 5004),
 (613, 'Construction Management', '6 months', 5005),
 (614, 'Web Development', '5 months', 5005),
 (615, 'Artificial Intelligence', '4 months', 5005),
 (616, 'Heat Transfer', '5 months', 5006),
 (617, 'Automation', '6 months', 5006),
 (618, 'Mobile App Development', '5 months', 5006);

-- Insert values into TimeTable Table

INSERT INTO TimeTable (TimetableID, CourseID, FacultyID, RoomNumber, Timing)
 VALUES

(501, 601, 101, 'Room 101', '9:00 AM - 10:00 AM'),
 (502, 602, 102, 'Room 101', '10:00 AM - 11:00 AM'),
 (503, 603, 103, 'Room CL1', '11:00 AM - 1:00 PM'),

(504, 604, 104, 'Room 102', '9:00 AM - 10:00 AM'),
(505, 605, 105, 'Room 102', '10:00 AM - 11:00 AM'),
(506, 606, 106, 'Room CL2', '11:00 AM - 1:00 PM'),

(507, 607, 107, 'Room 104', '9:00 AM - 10:00 AM'),
(508, 608, 108, 'Room 105', '10:00 AM - 11:00 AM'),
(509, 609, 109, 'Room CL3', '11:00 AM - 1:00 PM'),

(510, 610, 110, 'Room 106', '9:00 AM - 10:00 AM'),
(511, 611, 111, 'Room 106', '10:00 AM - 11:00 AM'),
(512, 612, 112, 'Room CL4', '11:00 AM - 1:00 PM'),

(513, 613, 113, 'Room 108', '9:00 AM - 10:00 AM'),
(514, 614, 114, 'Room 108', '10:00 AM - 11:00 AM'),
(515, 615, 115, 'Room CL5', '11:00 AM - 1:00 PM'),

(516, 616, 116, 'Room 110', '9:00 AM - 10:00 AM'),
(517, 617, 117, 'Room 110', '10:00 AM - 11:00 AM'),
(518, 618, 118, 'Room CL6', '11:00 AM - 1:00 PM');

-- Insert values into Grades Table

INSERT INTO Grades (PRN, MidSem, EndSem)

VALUES

(1001, 85, 90),
(1002, 78, 85),
(1003, 80, 88),
(1004, 75, 82),
(1005, 88, 92),
(1006, 82, 89),
(1007, 85, 90),
(1008, 79, 86),
(1009, 83, 88),
(1010, 77, 84),
(1011, 86, 91),
(1012, 81, 88);

-- Insert values into Attendance Table

INSERT INTO Attendance (PRN, Percentage, Date1)

VALUES

(1001, 90, '2023-01-15'),
(1002, 85, '2023-02-20'),
(1003, 88, '2023-03-25'),
(1004, 82, '2023-04-30'),
(1005, 86, '2023-05-05'),
(1006, 83, '2023-06-10'),
(1007, 88, '2023-07-15'),


```
(1008, 80, '2023-08-20'),  
(1009, 85, '2023-09-25'),  
(1010, 89, '2023-10-30'),  
(1011, 87, '2023-11-05'),  
(1012, 84, '2023-12-10');
```

-- Insert values into Fees Table

```
INSERT INTO Fees (PRN, PaidUnpaid, Fine)  
VALUES
```

```
(1001, 'Paid', 0),  
(1002, 'Unpaid', 50),  
(1003, 'Paid', 0),  
(1004, 'Paid', 0),  
(1005, 'Unpaid', 50),  
(1006, 'Paid', 0),  
(1007, 'Unpaid', 500),  
(1008, 'Paid', 0),  
(1009, 'Paid', 0),  
(1010, 'Unpaid', 250),  
(1011, 'Paid', 0),  
(1012, 'Unpaid', 90);
```

Table Outputs

SELECT * FROM Admin;

| | AdminID | FirstName | LastName | Email |
|---|---------|-----------|----------|-----------------|
| ▶ | 1 | Raj | Patel | raj@gmail.com |
| | 2 | Priya | Sharma | priya@gmail.com |
| | 3 | Amit | Gupta | amit@gmail.com |
| | 4 | Sneha | Desai | sneha@gmail.com |
| • | NULL | NULL | NULL | NULL |

SELECT * FROM Student;

| | PRN | FirstName | LastName | Year | BranchID | Semester | DOB | FathersName | Email | AddressStreetName | AddressStreetNumber | AddressZipCode | AddressState | AddressCity |
|---|------|-----------|----------|------|----------|----------|------------|-------------|--------------------|-------------------|---------------------|----------------|---------------|-------------|
| ▶ | 1001 | Abhishek | Rajput | 2 | 5001 | 4 | 2000-05-15 | Mr. Rajput | abhishek@gmail.com | Main Street | 123 | 400001 | Maharashtra | Mumbai |
| | 1002 | Aditya | Raj | 3 | 5001 | 6 | 1999-08-20 | Mr. Raj | aditya@gmail.com | Park Avenue | 456 | 110001 | Delhi | New Delhi |
| | 1003 | Archit | Patil | 1 | 5002 | 2 | 2001-02-10 | Mr. Patil | archit@gmail.com | Broadway | 789 | 560001 | Karnataka | Bangalore |
| | 1004 | Arnav | Jain | 4 | 5002 | 8 | 1998-11-25 | Mr. Jain | arnav@gmail.com | Green Street | 1011 | 600001 | Tamil Nadu | Chennai |
| | 1005 | Aryan | Sharma | 2 | 5003 | 4 | 2000-07-12 | Mr. Sharma | aryan@gmail.com | High Street | 1314 | 700001 | West Bengal | Kolkata |
| | 1006 | Avinash | Gupta | 3 | 5003 | 6 | 1999-09-18 | Mr. Gupta | avinash@gmail.com | River Road | 1516 | 500001 | Telangana | Hyderabad |
| | 1007 | Amita | Singh | 1 | 5004 | 2 | 2001-03-22 | Mr. Singh | amita@gmail.com | Lake Street | 1718 | 380001 | Gujarat | Ahmedabad |
| | 1008 | Akash | Kumar | 4 | 5004 | 8 | 1998-12-30 | Mr. Kumar | akash@gmail.com | Gandhi Road | 1920 | 250001 | Uttar Pradesh | Agra |
| | 1009 | Ananya | Yadav | 2 | 5005 | 4 | 2000-06-08 | Mr. Yadav | ananya@gmail.com | Market Street | 2122 | 560002 | Karnataka | Mysore |
| | 1010 | Ankit | Verma | 3 | 5005 | 6 | 1999-10-14 | Mr. Verma | ankit@gmail.com | Station Road | 2324 | 110002 | Delhi | New Delhi |
| | 1011 | Aditi | Shah | 1 | 5006 | 2 | 2001-04-26 | Mr. Shah | aditi@gmail.com | Victoria Street | 2526 | 600002 | Tamil Nadu | Coimbatore |
| | 1012 | Ayesha | Malik | 4 | 5006 | 8 | 1998-11-03 | Mr. Malik | ayesha@gmail.com | Church Street | 2728 | 700002 | West Bengal | Howrah |
| • | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

SELECT * FROM PhoneNumber;

| | PRN | PhoneNumber 1 | PhoneNumber 2 |
|---|------|---------------|---------------|
| ▶ | 1001 | 9650104541 | 9650104541 |
| | 1002 | 7720860456 | 9650104542 |
| | 1003 | 8639914092 | 9650104543 |
| | 1004 | 9377852020 | 9650104544 |
| | 1005 | 9234567890 | 9876543210 |
| | 1006 | 9345678901 | 9012345678 |
| | 1007 | 9456789012 | 8901234567 |
| | 1008 | 9567890123 | 9890123456 |
| | 1009 | 9678901234 | 8789012345 |
| | 1010 | 6789012345 | 9678901234 |
| | 1011 | 8890123456 | 9567890123 |
| | 1012 | 8901234567 | 8456789012 |
| • | NULL | NULL | NULL |

SELECT * FROM AdminLogin;

| | AdminID | AdminPassword |
|---|---------|---------------|
| ▶ | 1 | password1 |
| | 2 | password2 |
| | 3 | password3 |
| | 4 | password4 |
| • | NULL | NULL |

SELECT * FROM StudentLogin;

| | PRN | StudentPassword |
|---|------|-----------------|
| ▶ | 1001 | studentpass1 |
| | 1002 | studentpass2 |
| | 1003 | studentpass3 |
| | 1004 | studentpass4 |
| | 1005 | studentpass5 |
| | 1006 | studentpass6 |
| | 1007 | studentpass7 |
| | 1008 | studentpass8 |
| | 1009 | studentpass9 |
| | 1010 | studentpass10 |
| | 1011 | studentpass11 |
| | 1012 | studentpass12 |
| • | NULL | NULL |

SELECT * FROM Hostel;

| | PRN | Gender | SimpleRoom | LuxuryRoom |
|---|------|--------|------------|------------|
| ▶ | 1001 | Male | Yes | No |
| | 1002 | Male | No | Yes |
| | 1003 | Male | Yes | No |
| | 1004 | Male | No | Yes |
| | 1005 | Male | Yes | No |
| | 1006 | Male | Yes | No |
| | 1007 | Female | No | Yes |
| | 1008 | Male | Yes | No |
| | 1009 | Female | No | Yes |
| | 1010 | Male | Yes | No |
| | 1011 | Female | Yes | No |
| | 1012 | Female | No | Yes |
| • | NULL | NULL | NULL | NULL |

SELECT * FROM Library;

| | LibraryID | Category | Issue | Return1 |
|---|-----------|---------------|------------|------------|
| ▶ | 1001 | Science | 2023-01-10 | 2023-02-10 |
| | 1002 | Literature | 2023-02-15 | 2023-03-15 |
| | 1003 | Mathematics | 2023-03-20 | 2023-04-20 |
| | 1004 | History | 2023-04-25 | 2023-05-25 |
| | 1001 | Civics | 2023-05-26 | 2023-06-28 |
| | 1001 | Geo | 2023-05-26 | NULL |
| | 1005 | Science | 2023-03-10 | 2023-06-10 |
| | 1007 | Physics | 2023-01-17 | NULL |
| | 1012 | Mathematics-3 | 2023-02-20 | 2023-04-20 |
| | 1009 | C++ | 2023-07-24 | 2023-10-25 |
| | 1010 | Python | 2023-05-26 | NULL |
| | 1004 | Geo | 2023-05-26 | 2023-08-11 |

SELECT * FROM Faculty;

| | FacultyID | FacultyName | BranchID |
|---|-----------|----------------------|----------|
| ▶ | 101 | Dr. Sahil Gupta | 5001 |
| | 102 | Prof. Aryan Sharma | 5001 |
| | 103 | Dr. Arnav Singh | 5001 |
| | 104 | Prof. Archit Patel | 5002 |
| | 105 | Dr. Faraj Khan | 5002 |
| | 106 | Prof. Dhurmil Verma | 5002 |
| | 107 | Dr. Dhruv Joshi | 5003 |
| | 108 | Prof. Pranav Reddy | 5003 |
| | 109 | Dr. Ankit Malhotra | 5003 |
| | 110 | Prof. Aman Gupta | 5004 |
| | 111 | Dr. Ansh Sharma | 5004 |
| | 112 | Prof. Vijay Singh | 5004 |
| | 113 | Dr. Harsh Kumar | 5005 |
| | 114 | Prof. Abhay Desai | 5005 |
| | 115 | Dr. Ram Patel | 5005 |
| | 116 | Prof. Amit Shah | 5006 |
| | 117 | Dr. Tarak Mehta | 5006 |
| | 118 | Prof. Divyansh Ch... | 5006 |
| • | NULL | NULL | NULL |

SELECT * FROM TimeTable;

| | TimetableID | CourseID | FacultyID | RoomNumber | Timing |
|---|-------------|----------|-----------|------------|---------------------|
| ▶ | 501 | 601 | 101 | Room 101 | 9:00 AM - 10:00 AM |
| | 502 | 602 | 102 | Room 101 | 10:00 AM - 11:00 AM |
| | 503 | 603 | 103 | Room CL1 | 11:00 AM - 1:00 PM |
| | 504 | 604 | 104 | Room 102 | 9:00 AM - 10:00 AM |
| | 505 | 605 | 105 | Room 102 | 10:00 AM - 11:00 AM |
| | 506 | 606 | 106 | Room CL2 | 11:00 AM - 1:00 PM |
| | 507 | 607 | 107 | Room 104 | 9:00 AM - 10:00 AM |
| | 508 | 608 | 108 | Room 105 | 10:00 AM - 11:00 AM |
| | 509 | 609 | 109 | Room CL3 | 11:00 AM - 1:00 PM |
| | 510 | 610 | 110 | Room 106 | 9:00 AM - 10:00 AM |
| | 511 | 611 | 111 | Room 106 | 10:00 AM - 11:00 AM |
| | 512 | 612 | 112 | Room CL4 | 11:00 AM - 1:00 PM |
| | 513 | 613 | 113 | Room 108 | 9:00 AM - 10:00 AM |
| | 514 | 614 | 114 | Room 108 | 10:00 AM - 11:00 AM |
| | 515 | 615 | 115 | Room CL5 | 11:00 AM - 1:00 PM |
| | 516 | 616 | 116 | Room 110 | 9:00 AM - 10:00 AM |
| | 517 | 617 | 117 | Room 110 | 10:00 AM - 11:00 AM |
| | 518 | 618 | 118 | Room CL6 | 11:00 AM - 1:00 PM |
| • | NULL | NULL | NULL | NULL | NULL |

SELECT * FROM Course;

| | CourseID | CourseName | Duration | BranchID |
|---|----------|----------------------------|----------|----------|
| ▶ | 601 | Database Management | 3 months | 5001 |
| | 602 | Power Systems | 4 months | 5001 |
| | 603 | Thermodynamics | 5 months | 5001 |
| | 604 | Structural Analysis | 6 months | 5002 |
| | 605 | Java | 5 months | 5002 |
| | 606 | Data Structures | 3 months | 5002 |
| | 607 | Renewable Energy | 4 months | 5003 |
| | 608 | Fluid Mechanics | 5 months | 5003 |
| | 609 | Transportation Engineering | 6 months | 5003 |
| | 610 | Python Programming | 5 months | 5004 |
| | 611 | Machine Learning | 4 months | 5004 |
| | 612 | Control Systems | 5 months | 5004 |
| | 613 | Construction Management | 6 months | 5005 |
| | 614 | Web Development | 5 months | 5005 |
| | 615 | Artificial Intelligence | 4 months | 5005 |
| | 616 | Heat Transfer | 5 months | 5006 |
| | 617 | Automation | 6 months | 5006 |
| | 618 | Mobile App Development | 5 months | 5006 |
| • | NULL | NULL | NULL | NULL |

SELECT * FROM Grades;

| | PRN | MidSem | EndSem |
|---|------|--------|--------|
| ▶ | 1001 | 85 | 90 |
| | 1002 | 78 | 85 |
| | 1003 | 80 | 88 |
| | 1004 | 75 | 82 |
| | 1005 | 88 | 92 |
| | 1006 | 82 | 89 |
| | 1007 | 85 | 90 |
| | 1008 | 79 | 86 |
| | 1009 | 83 | 88 |
| | 1010 | 77 | 84 |
| | 1011 | 86 | 91 |
| | 1012 | 81 | 88 |
| ● | NULL | NULL | NULL |

SELECT * FROM Attendance;

| | PRN | Percentage | Date1 |
|---|------|------------|------------|
| ▶ | 1001 | 90 | 2023-01-15 |
| | 1002 | 85 | 2023-02-20 |
| | 1003 | 88 | 2023-03-25 |
| | 1004 | 82 | 2023-04-30 |
| | 1005 | 86 | 2023-05-05 |
| | 1006 | 83 | 2023-06-10 |
| | 1007 | 88 | 2023-07-15 |
| | 1008 | 80 | 2023-08-20 |
| | 1009 | 85 | 2023-09-25 |
| | 1010 | 89 | 2023-10-30 |
| | 1011 | 87 | 2023-11-05 |
| | 1012 | 84 | 2023-12-10 |
| ● | NULL | NULL | NULL |

SELECT * FROM Fees;

| | PRN | PaidUnpaid | Fine |
|---|------|------------|------|
| ▶ | 1001 | Paid | 0 |
| | 1002 | Unpaid | 50 |
| | 1003 | Paid | 0 |
| | 1004 | Paid | 0 |
| | 1005 | Unpaid | 50 |
| | 1006 | Paid | 0 |
| | 1007 | Unpaid | 500 |
| | 1008 | Paid | 0 |
| | 1009 | Paid | 0 |
| | 1010 | Unpaid | 250 |
| | 1011 | Paid | 0 |
| | 1012 | Unpaid | 90 |
| ● | NULL | NULL | NULL |

Queries Implemented

1. Retrieve the names of students who have a mid-semester grade above 80.

```
SELECT s.prn,s.FirstName, s.LastName,g.midsem FROM Student s  
JOIN Grades g ON s.PRN = g.PRN WHERE g.MidSem > 80;
```

| | prn | FirstName | LastName | midsem |
|---|------|-----------|----------|--------|
| ▶ | 1001 | Abhishek | Rajput | 85 |
| | 1005 | Aryan | Sharma | 88 |
| | 1006 | Avinash | Gupta | 82 |
| | 1007 | Amita | Singh | 85 |
| | 1009 | Ananya | Yadav | 83 |
| | 1011 | Aditi | Shah | 86 |
| | 1012 | Ayesha | Malik | 81 |

2. Find out the course which has the maximum duration.

```
SELECT * FROM Course WHERE Duration = (SELECT MAX(Duration) FROM Course);
```

| | CourseID | CourseName | Duration | BranchID |
|---|----------|----------------------------|----------|----------|
| ▶ | 604 | Structural Analysis | 6 months | 5002 |
| | 609 | Transportation Engineering | 6 months | 5003 |
| | 613 | Construction Management | 6 months | 5005 |
| | 617 | Automation | 6 months | 5006 |
| ● | NULL | NULL | NULL | NULL |

3. Retrieve the total number of students enrolled in each branch.

```
SELECT b.BranchName, COUNT(s.PRN) AS TotalStudents FROM Branch b  
JOIN Student s ON b.BranchID = s.BranchID  
GROUP BY b.BranchName;
```

| | BranchName | TotalStudents |
|---|-------------------------------------|---------------|
| ▶ | Computer Science Engineering | 2 |
| | Electrical Engineering | 2 |
| | Mechanical Engineering | 2 |
| | Civil Engineering | 2 |
| | Artificial Intelligence Engineering | 2 |
| | Robotics And Automation Engineering | 2 |

4. Retrieve all students who have unpaid fees along with their details.

```
SELECT * FROM Student  
WHERE PRN IN (SELECT PRN FROM Fees WHERE PaidUnpaid = 'Unpaid');
```

| | PRN | FirstName | LastName | Year | BranchID | Semester | DOB | FathersName | Email | AddressStreetName | AddressStreetNumber | AddressZipCode | AddressState | AddressCity |
|---|------|-----------|----------|------|----------|----------|------------|-------------|------------------|-------------------|---------------------|----------------|--------------|-------------|
| ▶ | 1002 | Aditya | Raj | 3 | 5001 | 6 | 1999-08-20 | Mr. Raj | aditya@gmail.com | Park Avenue | 456 | 110001 | Delhi | New Delhi |
| | 1005 | Aryan | Sharma | 2 | 5003 | 4 | 2000-07-12 | Mr. Sharma | aryan@gmail.com | High Street | 1314 | 700001 | West Bengal | Kolkata |
| | 1007 | Amita | Singh | 1 | 5004 | 2 | 2001-03-22 | Mr. Singh | amita@gmail.com | Lake Street | 1718 | 380001 | Gujarat | Ahmedabad |
| | 1010 | Ankit | Verma | 3 | 5005 | 6 | 1999-10-14 | Mr. Verma | ankit@gmail.com | Station Road | 2324 | 110002 | Delhi | New Delhi |
| | 1012 | Ayesha | Malik | 4 | 5006 | 8 | 1998-11-03 | Mr. Malik | ayesha@gmail.com | Church Street | 2728 | 700002 | West Bengal | Howrah |
| • | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

5. Retrieve the students who have a mid-semester grade between 80 and 90, and are enrolled in either the 'Computer Science Engineering' or 'Mechanical Engineering' branch.

```

SELECT * FROM Student
WHERE PRN IN (
    SELECT PRN
    FROM Grades
    WHERE MidSem BETWEEN 80 AND 90
)
AND BranchID IN (
    SELECT BranchID
    FROM Branch
    WHERE BranchName IN ('Computer Science Engineering', 'Mechanical Engineering')
);

```

| | PRN | FirstName | LastName | Year | BranchID | Semester | DOB | FathersName | Email | AddressStreetName | AddressStreetNumber | AddressZipCode | AddressState | AddressCity |
|---|------|-----------|----------|------|----------|----------|------------|-------------|--------------------|-------------------|---------------------|----------------|--------------|-------------|
| ▶ | 1001 | Abhishek | Rajput | 2 | 5001 | 4 | 2000-05-15 | Mr. Rajput | abhishek@gmail.com | Main Street | 123 | 400001 | Maharashtra | Mumbai |
| | 1005 | Aryan | Sharma | 2 | 5003 | 4 | 2000-07-12 | Mr. Sharma | aryan@gmail.com | High Street | 1314 | 700001 | West Bengal | Kolkata |
| | 1006 | Avinash | Gupta | 3 | 5003 | 6 | 1999-09-18 | Mr. Gupta | avinash@gmail.com | River Road | 1516 | 500001 | Telangana | Hyderabad |
| • | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

6. Retrieve the library books that are currently not returned.

```

SELECT *
FROM Library
WHERE Return1 IS NULL;

```

| | LibraryID | Category | Issue | Return1 |
|---|-----------|----------|------------|---------|
| ▶ | 1001 | Geo | 2023-05-26 | NULL |
| | 1007 | Physics | 2023-01-17 | NULL |
| | 1010 | Python | 2023-05-26 | NULL |

7. Query: Retrieve all students whose last name has exactly six characters:

```

SELECT *
FROM Student

```


WHERE LastName LIKE '_____';

| | PRN | FirstName | LastName | Year | BranchID | Semester | DOB | FathersName | Email | AddressStreetName | AddressStreetNumber | AddressZipCode | AddressState | AddressCity |
|---|------|-----------|----------|------|----------|----------|------------|-------------|--------------------|-------------------|---------------------|----------------|--------------|-------------|
| ▶ | 1001 | Abhishek | Rajput | 2 | 5001 | 4 | 2000-05-15 | Mr. Rajput | abhishek@gmail.com | Main Street | 123 | 400001 | Maharashtra | Mumbai |
| | 1005 | Aryan | Sharma | 2 | 5003 | 4 | 2000-07-12 | Mr. Sharma | aryan@gmail.com | High Street | 1314 | 700001 | West Bengal | Kolkata |
| | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

8. Find the number of books issued in each category from the library.

SELECT Category, COUNT(*) AS BooksIssued

FROM Library

GROUP BY Category;

| | Category | BooksIssued |
|---|---------------|-------------|
| ▶ | Science | 2 |
| | Literature | 1 |
| | Mathematics | 1 |
| | History | 1 |
| | Civics | 1 |
| | Geo | 2 |
| | Physics | 1 |
| | Mathematics-3 | 1 |
| | C++ | 1 |
| | Python | 1 |

9.Retrieve the student details along with their branch names and sorted by branch names in descending order.

SELECT s.prn,s.Firstname,s.lastname,b.branchid,b.BranchName

FROM Student s

INNER JOIN Branch b ON s.BranchID = b.BranchID

ORDER BY b.BranchName DESC;

| | prn | Firstname | lastname | branchid | BranchName |
|---|------|-----------|----------|----------|-------------------------------------|
| ▶ | 1011 | Aditi | Shah | 5006 | Robotics And Automation Engineering |
| | 1012 | Ayesha | Malik | 5006 | Robotics And Automation Engineering |
| | 1005 | Aryan | Sharma | 5003 | Mechanical Engineering |
| | 1006 | Avinash | Gupta | 5003 | Mechanical Engineering |
| | 1003 | Archit | Patil | 5002 | Electrical Engineering |
| | 1004 | Arnav | Jain | 5002 | Electrical Engineering |
| | 1001 | Abhishek | Rajput | 5001 | Computer Science Engineering |
| | 1002 | Aditya | Raj | 5001 | Computer Science Engineering |
| | 1007 | Amita | Singh | 5004 | Civil Engineering |
| | 1008 | Akash | Kumar | 5004 | Civil Engineering |
| | 1009 | Ananya | Yadav | 5005 | Artificial Intelligence Engineering |
| | 1010 | Ankit | Verma | 5005 | Artificial Intelligence Engineering |

10. Create a view to display student details along with their branch names.

```
CREATE VIEW StudentDetailsWithBranch AS  
SELECT s.prn,s.Firstname,s.lastname, b.BranchName  
FROM Student s  
INNER JOIN Branch b ON s.BranchID = b.BranchID;
```

```
SELECT * from studentdetailswithbranch;
```

| | prn | Firstname | lastname | BranchName |
|---|------|-----------|----------|-------------------------------------|
| ► | 1001 | Abhishek | Rajput | Computer Science Engineering |
| | 1002 | Aditya | Raj | Computer Science Engineering |
| | 1003 | Archit | Patil | Electrical Engineering |
| | 1004 | Arnav | Jain | Electrical Engineering |
| | 1005 | Aryan | Sharma | Mechanical Engineering |
| | 1006 | Avinash | Gupta | Mechanical Engineering |
| | 1007 | Amita | Singh | Civil Engineering |
| | 1008 | Akash | Kumar | Civil Engineering |
| | 1009 | Ananya | Yadav | Artificial Intelligence Engineering |
| | 1010 | Ankit | Verma | Artificial Intelligence Engineering |
| | 1011 | Aditi | Shah | Robotics And Automation Engin... |
| | 1012 | Ayesha | Malik | Robotics And Automation Engin... |

Function

A function in SQL is a named collection of SQL statements that have the ability to process data, receive parameters, and return either a series of results or a single value. Functions are usually used for data processing, calculations, and retrieval; they also improve the organisation and reusability of code in SQL queries.

1. Function to calculate the student's grades based on their mid-semester and end-semester grades.

```
DELIMITER $$
CREATE FUNCTION calculate_average_grade(p_prn INT)
RETURNS FLOAT
DETERMINISTIC
BEGIN
    DECLARE v_avg_grade FLOAT;
    SELECT (MidSem + EndSem) / 2 INTO v_avg_grade FROM Grades WHERE PRN =
p_prn;
    RETURN v_avg_grade;
END$$
DELIMITER ;
```

```
SELECT calculate_average_grade(1001) AS AVERAGE;
```

| | |
|---|---------|
| | average |
| ▶ | 87.5 |

2. Function to determine the grade status (Pass/Fail) of a student based on their average grade.

```
DELIMITER $$
CREATE FUNCTION get_student_grade_status(p_prn INT)
RETURNS VARCHAR(50)
DETERMINISTIC
BEGIN
    DECLARE v_grade_status VARCHAR(10);
    DECLARE v_avg_grade FLOAT;
    SELECT calculate_average_grade(p_prn) INTO v_avg_grade FROM DUAL;

    IF v_avg_grade >= 60 THEN
        SET v_grade_status = 'Pass';
    ELSE
        SET v_grade_status = 'Fail';
    END IF;
```

```
    RETURN v_grade_status;  
END $$  
DELIMITER ;
```

```
SELECT get_student_grade_status(1005) AS Grade;
```

| | |
|---|-------|
| | Grade |
| ▶ | Pass |

Procedure

A precompiled collection of SQL statements and procedural logic kept in a database is called a stored procedure. It is used to access or edit data in the database and is shared and reused by multiple programmes. Stored procedures centralise complicated processes and business logic within the database system, improving data security, reducing redundancy, and increasing the efficiency of database operations.

1. Procedure to Add New Faculty.

```
DELIMITER //
CREATE PROCEDURE add_new_faculty(
    IN p_faculty_id INT,
    IN p_faculty_name VARCHAR(50),
    IN p_branch_id INT)
BEGIN
    INSERT INTO Faculty (FacultyID, FacultyName, BranchID)
    VALUES (p_faculty_id, p_faculty_name, p_branch_id);
END //
DELIMITER ;

CALL add_new_faculty(119,'Dr.lakshya',5006);
SELECT * FROM faculty;
```

| | FacultyID | FacultyName | BranchID |
|---|-----------|----------------------|----------|
| ▶ | 101 | Dr. Sahil Gupta | 5001 |
| | 102 | Prof. Aryan Sharma | 5001 |
| | 103 | Dr. Arnav Singh | 5001 |
| | 104 | Prof. Archit Patel | 5002 |
| | 105 | Dr. Faraj Khan | 5002 |
| | 106 | Prof. Dhurmil Verma | 5002 |
| | 107 | Dr. Dhruv Joshi | 5003 |
| | 108 | Prof. Pranav Reddy | 5003 |
| | 109 | Dr. Ankit Malhotra | 5003 |
| | 110 | Prof. Aman Gupta | 5004 |
| | 111 | Dr. Ansh Sharma | 5004 |
| | 112 | Prof. Vijay Singh | 5004 |
| | 113 | Dr. Harsh Kumar | 5005 |
| | 114 | Prof. Abhay Desai | 5005 |
| | 115 | Dr. Ram Patel | 5005 |
| | 116 | Prof. Amit Shah | 5006 |
| | 117 | Dr. Tarak Mehta | 5006 |
| | 118 | Prof. Divyansh Ch... | 5006 |
| | 119 | Dr.lakshya | 5006 |
| * | NULL | NULL | NULL |

2. Procedure which takes student's PRN as input and returns their first name and last name as output parameters.

```
DELIMITER //
CREATE PROCEDURE get_student_name(
    IN p_prn INT,
    OUT p_first_name VARCHAR(50),
    OUT p_last_name VARCHAR(50))
BEGIN
    SELECT FirstName, LastName
    INTO p_first_name, p_last_name
    FROM Student
    WHERE PRN = p_prn;
END //
DELIMITER ;

CALL get_student_name(1002, @first_name, @last_name);
SELECT @first_name, @last_name;
```

| | @first_name | @last_name |
|---|-------------|------------|
| ▶ | Aditya | Raj |

Trigger

A trigger in a database is a set of predefined SQL statements that are automatically executed in response to specific events or actions within the database, such as insertions, deletions, updates, or other defined operations. Triggers are used to enforce data integrity, ensure business rules are followed, and automate certain database actions based on predefined conditions, thereby enhancing the database's functionality and maintaining data consistency.

1. Trigger to Update Last Modified Timestamp for Student Table.

```
CREATE TABLE StudentAudit (  
    AuditID INT AUTO_INCREMENT PRIMARY KEY,  
    PRN INT,  
    LastModified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
    CURRENT_TIMESTAMP,  
    FOREIGN KEY (PRN) REFERENCES Student(PRN)  
);
```

```
DELIMITER //  
CREATE TRIGGER student_update_trigger  
BEFORE UPDATE ON Student  
FOR EACH ROW  
BEGIN  
    INSERT INTO StudentAudit (PRN) VALUES (NEW.PRN);  
END //  
DELIMITER ;
```

```
UPDATE student SET firstname="Abhishek" WHERE prn=1001;  
SELECT * FROM StudentAudit;
```



| | AuditID | PRN | LastModified |
|---|---------|------|---------------------|
| ▶ | 1 | 1001 | 2024-04-08 18:03:17 |
| • | NULL | NULL | NULL |

2. Trigger to ensure that the email address entered for a student follows a valid format before insertion into the Student table

```
DELIMITER //  
CREATE TRIGGER validate_email_format_trigger  
BEFORE INSERT ON Admin  
FOR EACH ROW  
BEGIN  
    IF NEW.Email NOT REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'  
    THEN
```

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid email format.';
END IF;
END //
DELIMITER ;
```

```
INSERT INTO Admin VALUES( 5, 'Raj', 'More', 'raj@gmail[]/.com');
```

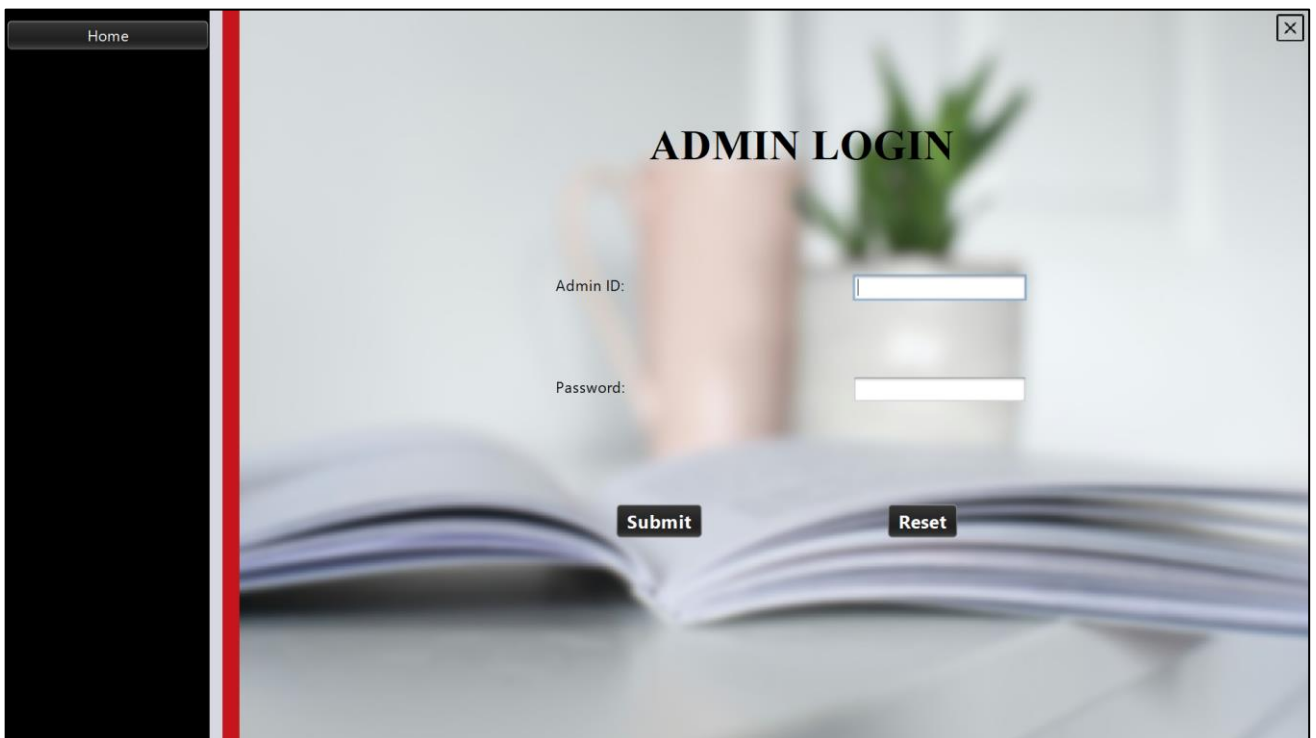
| | | | | |
|---|----|----------|---|---|
|  | 79 | 00:29:52 | CREATE TRIGGER validate_email_format_trigger BEFORE INSERT ON Admin FOR EACH ROW BEGIN IF ... | 0 row(s) affected |
|  | 80 | 00:29:56 | INSERT INTO Admin VALUES(5, 'Raj', 'More', 'raj@gmail[]/.com') | Error Code: 1644. Invalid email format. |

AWS

Amazon Web Services (AWS) is a comprehensive and leading cloud computing platform that offers an extensive array of services, including storage, compute, networking, machine learning, and database management. Known for its flexibility, scalability, and reliability, AWS provides businesses and developers with the tools and resources they need to efficiently manage applications and data in the cloud. Among its offerings is the Relational Database Management System (RDS) Service, which simplifies the deployment and management of relational databases. This service automates tasks such as backups, scaling, and software patching, allowing users to focus on their applications and data without the burden of database administration.

In our Integrated Student Management project, we have harnessed the power of AWS RDS to store data online, enabling a seamless and robust solution for managing student and faculty information. By integrating AWS RDS, the project benefits from high availability, data security, and automatic backups, ensuring reliable and uninterrupted access to critical information. This cloud-based approach enhances the overall efficiency and performance of the system, supporting the streamlined operation of administrative and academic processes within educational institutions.

Front End Sample Design



Logout

Home

Profile

Time-Table

Hostel

Course

Fees

Library

Attendance

Faculty

Grades

Branch

Student Details

Admin Details

Student Login Details

Admin Login Details

ADMIN PROFILE

Admin ID

4

First Name

Sneha

Last Name

Desai

Email

sneha@gmail.com

Logout

Home

Profile

Time-Table

Hostel

Course

Fees

Library

Attendance

Faculty

Grades

Branch

Student Details

Admin Details

Student Login Details

Admin Login Details

ADMIN LOGIN

| Admin ID | Admin Password |
|----------|----------------|
| 1 | password1 |
| 2 | password2 |
| 3 | password3 |
| 4 | password4 |

Add Admin

Update Details

Delete Specific Admin

Show

Hide

Find

Github Link

https://github.com/Abhishek-2502/Campus_Nexus.git

References

1. <https://www.geeksforgeeks.org/anomalies-in-relational-model/>
2. <https://www.scaler.com/topics/anomalies-in-dbms/>
3. <https://www.javatpoint.com/dbms-normalization>
4. <https://www.geeksforgeeks.org/normal-forms-in-dbms/>
5. <https://www.simplilearn.com/tutorials/sql-tutorial/what-is-normalization-in-sql>
6. <https://www.youtube.com/watch?v=5GDTIUVIHB8>
7. <https://dev.mysql.com/doc/refman/8.0/en/triggers.html>
8. <https://dev.mysql.com/doc/refman/8.0/en/functions.html>
9. <https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>
10. <https://www.geeksforgeeks.org/different-types-of-procedures-in-mysql/>
11. <https://www.geeksforgeeks.org/types-of-functional-dependencies-in-dbms/>
12. <https://manojahi.medium.com/flask-html-template-with-mysql-2f3b9405d0e2>
13. <https://www.geeksforgeeks.org/login-and-registration-project-using-flask-and- mysql/>