

**TRANSPORT MANAGEMENT  
SYSTEM**

**MINOR PROJECT REPORT**

By

**VAIBHAV DEV (RA2311026010264)**

**HIMESH SAHOO (RA2311026010262)**

*Under the Guidance of*

**DR. G. DINESH**

Assistant Professor, Department of CINTEL

**BACHELOR OF TECHNOLOGY**

*in*

**Computer Science Engineering with specialization in AIML**



**DEPARTMENT OF COMPUTATIONAL INTELLIGENCE**

**COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND**

**TECHNOLOGY KATTANKULATHUR– 603203**

**2025**



**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**  
**KATTANKULATHUR-603 203**

**BONAFIDE CERTIFICATE**

This is to certify that the project work titled **Database Management Systems (21CSC205P)** has been carried out by **Vaibhav Dev (RA2311026010264)** and **Himesh Sahoo(RA2311026010264)** and students of the II year / IV-Sem B.Tech Degree Course, as part of the course **21CSC205P - Project Course** during the academic year 2024-2025 at SRM Institute of Science and Technology, Kattankulathur.

The work presented is their original contribution and has been duly completed under the supervision of the course instructor.

Date:

**FACULTY IN CHARGE**

Dr. Dinesh G  
Assistant Professor  
Department of Computational Intelligence  
SRMIST - KTR

**HEAD OF THE DEPARTMENT**

Dr. R. Annie Uthra  
Professor & Head  
Department of Computational Intelligence  
SRMIST - KTR

## **ABSTRACT**

A Transport Management System enhances the efficiency of managing transportation operations by streamlining core functions such as vehicle allocation, route optimization, trip scheduling, freight billing, and real-time tracking. This system integrates administrative, operational, and logistical processes into a centralized platform, ensuring accurate information flow and seamless coordination between transport assets and stakeholders.

The system automates routine transportation tasks, reducing manual workload and minimizing operational errors. Real-time data insights facilitate timely decision-making in areas such as fleet maintenance, route planning, and load management. The incorporation of advanced features such as GPS-based tracking and automated alerts improves visibility across all transport activities, ensuring safety, accountability, and operational transparency.

The user-friendly interface allows administrators to access trip histories, vehicle diagnostics, fuel consumption, and compliance reports efficiently. Cost-effective management is achieved through optimized resource utilization and reduced idle time of vehicles. Furthermore, the centralized repository of transport data supports data-driven strategies that improve service quality, minimize delays, and reduce operational costs.

Overall, the Transport Management System provides a robust digital solution that modernizes transport operations, enhances productivity, and ensures reliable service delivery across various sectors.

## **TABLE OF CONTENTS**

## **Problem Statement:**

Conventional transport management faces difficulties such as inefficient fleet allocation, poor route planning, lack of real-time tracking, and fragmented communication. Manual processes lead to data inaccuracies, increased costs, and delays in decision-making. Limited visibility into vehicle status and maintenance schedules further hampers operational efficiency and regulatory compliance. A centralized digital solution is required to streamline operations, enhance resource utilization, and improve service reliability.

## **Problem Description**

1. **Manual Data Management:** The current transport system relies heavily on manual processes for managing trip records, vehicle assignments, driver details, billing, and maintenance schedules, leading to inefficiencies and errors.
2. **Disjointed Operations:** Key transport operations, such as route planning, vehicle tracking, freight management, and compliance monitoring, are handled separately, causing delays and miscommunication among stakeholders.
3. **Limited Access to Real-Time Data:** Administrators and logistics coordinators lack immediate access to real-time data, such as vehicle location, delivery status, and maintenance alerts, resulting in reduced transparency and slower response times.
4. **Inefficient Complaint and Request Handling:** Service requests, route changes, and complaints are tracked manually, leading to delayed resolutions and poor customer service.
5. **Lack of a Centralized System:** Without a unified platform, managing transport operations, fleet performance, and compliance data becomes disorganized and time-consuming.
6. **Security and Privacy Concerns:** Sensitive transport data, including driver credentials, cargo information, and financial records, are often managed manually, increasing risks to data security and privacy.
7. **Inadequate User Interface:** Existing systems may lack an intuitive interface, making it difficult for administrators and operators to navigate transport management tasks efficiently.
8. **Limited Scalability:** As the number of vehicles and transport operations increases, the current system struggles to scale effectively, leading to operational bottlenecks and difficulties in handling larger data volumes.

## ER-DIAGRAM FOR TRANSPORT MANAGEMENT SYSTEM

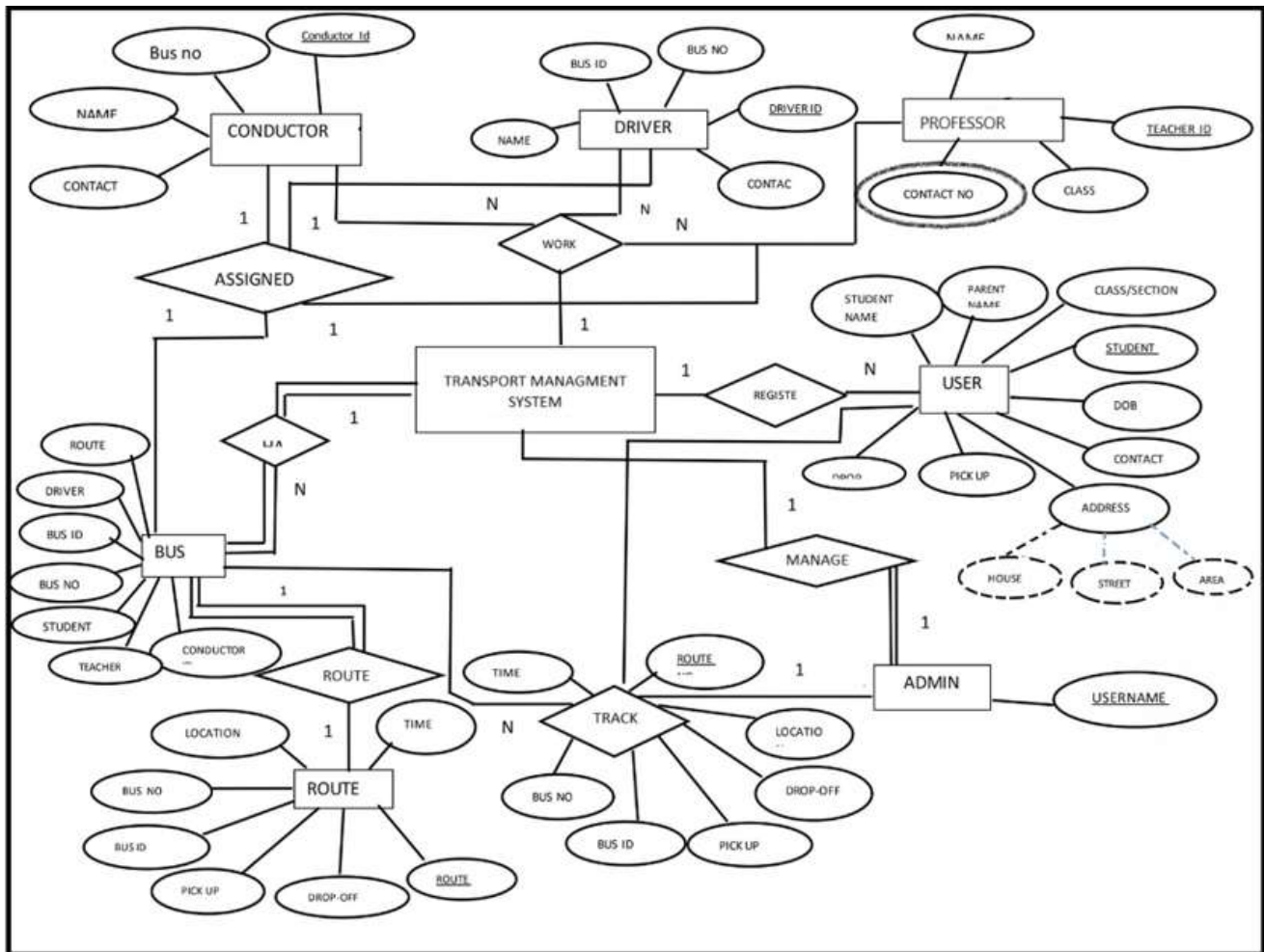


Fig:1.1 ER DIAGRAM

## **ER DIAGRAM FOR TRANSPORT MANAGEMENT SYSTEM**

**An Entity–Relationship Diagram (ERD) visually represents the relationships between different entities in a system, illustrating how data is structured and interconnected.**

### **Entities**

#### **Bus**

Attributes: Bus\_ID (PK), Bus\_No, Capacity

#### **Route**

Attributes: Route\_ID (PK), Route\_Name, Pick\_Up\_Point, Drop\_Off\_Point

#### **Track**

Attributes: Track\_ID (PK), Bus\_ID (FK), Route\_ID (FK), Time, Location, Pick\_Up, Drop\_Off

#### **Conductor**

Attributes: Conductor\_ID (PK), Name, Contact, Bus\_No (FK)

#### **Driver**

Attributes: Driver\_ID (PK), Name, Contact, Bus\_ID (FK)

#### **Student**

Attributes: User\_ID (PK), Student\_Name, Parent\_Name, Class\_Section, DOB, Contact, Address

#### **Admin**

Attributes: Admin\_ID (PK), Username, Name

#### **Professor**

Attributes: Teacher\_ID (PK), Name, Contact\_No, Class\_Assigned

### **Relationships**

- The Conductor–Bus relationship is one-to-one: each Conductor is assigned exactly one Bus, and each Bus has exactly one Conductor.
- The Driver–Bus relationship is many-to-many (implemented via an associative Work relationship): a Driver may be assigned to run multiple Buses over time, and each Bus may have multiple Drivers scheduled.
- The Bus–Route relationship is one-to-many: each Bus follows exactly one Route at a given time, but each Route can be served by multiple Buses.
- The Track entity captures the many-to-many operational details between Buses and Routes over time. Each Track record links one Bus and one Route at a specific Time

and Location, recording the Pick\_Up and Drop\_Off events.

- The User–Register relationship between Users and the system is one-to-many: each User (student) can register for multiple bus services (records in Track), but each registration record refers to exactly one User.
- The Admin–Manage relationship is one-to-many: each Admin can manage multiple entities of the system (e.g., Buses, Routes, Users), and each entity’s administrative record is managed by exactly one Admin.
- The Professor–Class relationship is one-to-many: one Professor can teach multiple Classes, and each Class is assigned to one Professor.



# 1. Login

## Attributes:

- **LOGIN\_ID:** Unique identifier for each login entry.
- **LOGIN\_ROLE:** Specifies the role of the user (e.g., Admin, Driver, Conductor, Staff).
- **LOGIN\_USERNAME:** Username used for logging into the system.
- **PASSWORD:** Password for user authentication.

## Relationships:

- Connected to the Admin entity via the MAINTAIN relationship.
- Connected to the Driver entity via the WORK relationship.
- Connected to the Conductor entity via the WORK relationship.
- Connected to the Staff entity via the WORK relationship.

# 2. Admin

## Attributes:

- **ADMIN\_ID:** Unique identifier for each admin.
- **USERNAME:** Admin's username.
- **EMAIL:** Contact email for the admin.
- **PASSWORD:** Password for the admin's login.

## Relationships:

- **MANAGE relationship with Bus:** Admins oversee bus operations.
- **MANAGE relationship with Route:** Admins manage route assignments.
- **MAINTAIN relationship with Driver:** Admins oversee driver records.
- **MAINTAIN relationship with Conductor:** Admins manage conductor records.
- **MAINTAIN relationship with Staff:** Admins manage staff records.

# 3. Student Information

## Attributes:

- **USER\_ID:** Unique identifier for each student.
- **STUDENT\_NAME:** Full name of the student.
- **PARENT\_NAME:** Name of the student's parent/guardian.
- **CLASSIFICATION:** Student's grade or class designation.
- **DOB:** Date of birth for age verification.
- **CONTACT:** Contact number for the student.

- **PICKUP\_LOCATION:** Location where student is picked up.
- **ADDRESS:** Full address of the student.
- **HOUSE\_NO:** House number of student's residence.
- **STREET:** Street name of student's residence.
- **CITY:** City of student's residence.

## 4. Staff Information

### Attributes:

- **USER\_ID:** Unique identifier for each staff.
- **USER\_NAME:** Full name of the staff.
- **DOB:** Date of birth for age verification.
- **CONTACT:** Contact number for the staff.
- **PICKUP\_LOCATION:** Location where staff is picked up.
- **ADDRESS:** Full address of the staff.
- **HOUSE\_NO:** House number of staff's residence.
- **STREET:** Street name of staff's residence.
- **CITY:** City of staff's residence.

### Relationships:

- **Linked to Bus through the assigned relationship.**
- **Linked to Route for transportation planning.**
- **Connected to Staff\_Log for tracking student transportation.**

## 5. Bus

### Attributes:

- **BUS\_ID:** Unique identifier for each bus.
- **BUS\_NO:** Registration number of the bus.

### Relationships:

- **Linked to Driver through the work relationship.**
- **Linked to Conductor through the assigned relationship.**
- **Connected to Route for transportation routes.**
- **Connected to Track for real-time tracking.**

## 6. Driver

### Attributes:

- **DRIVER\_ID:** Unique identifier for each driver.
- **NAME:** Full name of the driver.
- **CONTACT:** Contact information for the driver.
- **BUS\_ID:** ID of the bus assigned to the driver.

**Relationships:**

- **Linked to Bus through the work relationship.**
- **Connected to Admin through the maintain relationship.**

## 7. Route

**Attributes:**

- **ROUTE\_ID:** Unique identifier for each route.
- **BUS\_NO:** Bus number assigned to the route.
- **LOCATION:** Areas covered in the route.
- **PICKUP:** Pickup points along the route.
- **DROP\_OFF:** Drop-off points along the route.

**Relationships:**

- **Linked to Bus for route assignments.**
- **Connected to Track for monitoring route progress.**
- **Managed by Admin through the manage relationship.**

**This structure maintains the format of the first image while incorporating the entities and relationships shown in the second image's ER diagram.**

The ER diagram for the transportation management system represents a structured approach to managing school transportation operations, encompassing various entities and their relationships to support efficient administration. Each entity in the system holds essential information required for tracking students, drivers, buses, routes, conductors, staff, and various logs. Let's go through each entity and its role in detail.

The Login entity represents the entry point for users accessing the system. It contains attributes like LOGIN\_ID, LOGIN\_ROLE, LOGIN\_USER-NAME, and PASSWORD, which are crucial for authenticating users and defining their roles within the system. The roles specified in LOGIN\_ROLE may include administrators, drivers, conductors, or staff, each with distinct access privileges. Through the MAINTAIN relationship, the Login entity connects to various operational entities, ensuring only authorized users can interact with sensitive data, maintaining the privacy and security of transportation records.

Admin plays a central role in managing the system, with attributes such as ADMIN\_ID, USERNAME, and other identifying information that uniquely identify each administrator. Admins are tasked with

overseeing various aspects of the transportation operations. Through the MANAGE relationship, Admin is connected to both the Bus and Route entities, representing the administrative responsibility for managing transportation resources. The WORK relationship links Admin to Driver, highlighting the administrator's role in overseeing driver assignments. Additionally, Admin connects to various logs through relationships that allow tracking of system activities. This multi-layered connectivity underscores the admin's role as the central figure in transportation management.

The Student Information entity is one of the core entities, holding extensive data on each student using the transportation system. Its attributes include USER\_ID, STUDENT\_NAME, PARENT\_NAME, CLASSIFICATION, DOB, CONTACT, PICKUP\_LOCATION, ADDRESS, HOUSE\_NO, STREET, and CITY. These fields collectively provide a comprehensive profile of each student, essential for personalized and safe management of student transportation. The student entity connects to the Bus entity, establishing which bus a student is assigned to for transportation. It also links to student\_log and prev\_student\_log entities, which track student transportation activities over time.

The Bus entity serves as a central component in the transportation system, with attributes such as BUS\_ID and BUS\_NO that uniquely identify each vehicle. The Bus entity maintains relationships with multiple other entities: it connects to Driver through a work relationship, indicating which driver operates each bus; it links to Conductor, showing which conductor is assigned to each bus; it connects to Route, establishing which routes each bus serves; and it links to Track for real-time monitoring of bus locations. This interconnected structure allows for comprehensive management of the bus fleet and its operations.

The Driver entity manages information related to transportation staff who operate buses. Attributes include DRIVER\_ID, NAME, CONTACT, and BUS\_ID, which capture essential information about each driver and their assigned vehicle. The Driver entity connects to Bus through the work relationship, clearly defining which bus each driver is responsible for operating. This relationship is crucial for maintaining clear accountability in the transportation system.

The Route entity outlines the transportation paths within the system. Attributes include ROUTE\_ID, BUS\_NO, LOCATION, PICKUP, and DROP\_OFF, which define the specific paths and stops for each bus route. Route is connected to Bus, establishing which bus serves each route, and to Track, allowing for real-time monitoring of buses along their designated routes. The Route entity also links to the manage relationship with Admin, indicating administrative oversight of route planning and management.

The Track entity provides real-time monitoring capabilities with attributes like TRACK\_ID, BUS\_NO, LOCATION, PICKUP, DROP\_OFF, and ROUTE\_ID. This entity connects to both Bus and Route, allowing administrators to monitor the current location and status of each bus along its assigned route. This tracking functionality enhances safety and efficiency by providing visibility into transportation operations as they occur.

The Conductor entity represents staff members who supervise students on buses, with attributes including CONDUCTOR\_ID, NAME, CONTACT, and BUS\_NO. Conductors are linked to specific buses through the assigned relationship, clearly defining their responsibilities within the transportation system. This relationship ensures that each bus has proper supervision for student safety.

The Staff entity captures information about additional personnel involved in the transportation system, with attributes such as STAFF\_ID, NAME, CONTACT\_NO, CLASS, and TEACHER\_ID. Staff members may include teachers or other school personnel who assist with transportation management or supervision. This entity provides a comprehensive view of all personnel involved in the transportation ecosystem.

Various log entities (student\_log, prev\_student\_log, admin\_log) track activities and changes within the system. These logs maintain records of student transportation, administrative actions, and historical data, providing valuable information for auditing, reporting, and analysis. The student\_log entity, for example, includes USER\_ID, STUDENT\_NAME, and CLASSIFICATION, tracking student transportation activities over time.

Each entity and relationship in this ER diagram is integral to the transportation management system. From secure login to managing drivers, buses, routes, and tracking, the system provides comprehensive functionality to streamline transportation operations.

## RELATION – SCHEMA FOR TRANSPORT MANAGEMENT

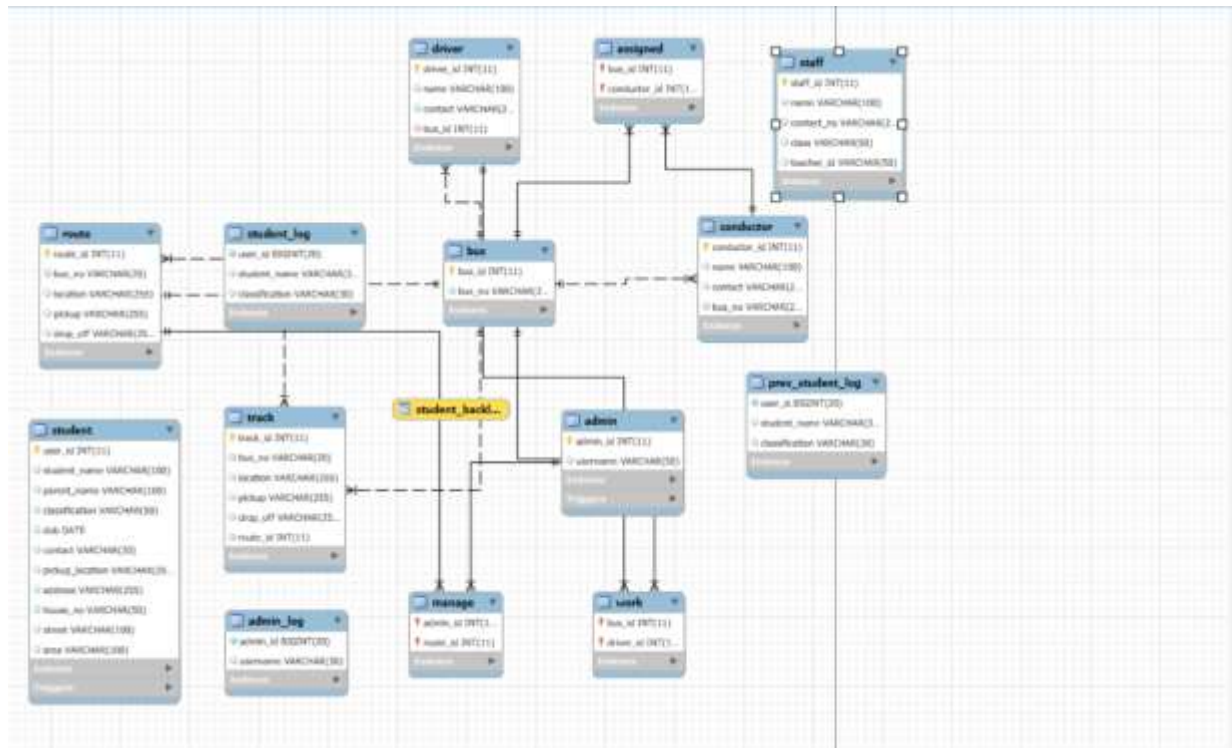


FIG 2.1 SCHEMA DIAGRAM

**Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors.**

CREATING A DATABASE TRANSPORTATION MANAGEMENT CREATING THE MENTIONED

**TABLES:**

1. LOGIN
2. ADMIN
3. STUDENT
4. BUS
5. DRIVER
6. CONDUCTOR
7. ROUTE
8. TRACK
9. STAFF
10. STUDENT\_LOG
11. PREV\_STUDENT\_LOG
12. ADMIN\_LOG

INSERTING THE VALUES IN THE TABLE USING SHOW TABLE

```

INSERT INTO student (user_id, student_name, parent_name, classification, dob, contact, pickup_location, address, house_no, street, area)
VALUES (1001, 'John Smith', 'Mary Smith', 'Elementary', '2017-06-15', '9876543210', 'Main Gate', '123 Oak Avenue', '123', 'Oak Avenue', 'North Hi
INSERT INTO student (user_id, student_name, parent_name, classification, dob, contact, pickup_location, address, house_no, street, area)
VALUES (1002, 'Emily Johnson', 'Robert Johnson', 'Middle School', '2013-03-22', '8765432109', 'Side Entrance', '456 Pine Road', '456', 'Pine Road
INSERT INTO student (user_id, student_name, parent_name, classification, dob, contact, pickup_location, address, house_no, street, area)
VALUES (1003, 'Michael Brown', 'Jennifer Brown', 'High School', '2009-11-08', '7654321098', 'Sports Complex', '789 Maple Lane', '789', 'Maple Lan
INSERT INTO student (user_id, student_name, parent_name, classification, dob, contact, pickup_location, address, house_no, street, area)
VALUES (1004, 'Sophia Garcia', 'Carlos Garcia', 'Elementary', '2018-01-30', '6543210987', 'Library Entrance', '234 Cedar Street', '234', 'Cedar S
INSERT INTO student (user_id, student_name, parent_name, classification, dob, contact, pickup_location, address, house_no, street, area)
VALUES (1005, 'David Wilson', 'Patricia Wilson', 'High School', '2007-09-12', '5432109876', 'Main Gate', '567 Birch Boulevard', '567', 'Birch Bou
INSERT INTO student (user_id, student_name, parent_name, classification, dob, contact, pickup_location, address, house_no, street, area)
VALUES (1006, 'Olivia Martinez', 'Luis Martinez', 'Middle School', '2012-05-17', '4321098765', 'Cafeteria Exit', '890 Elm Court', '890', 'Elm Cou
INSERT INTO student (user_id, student_name, parent_name, classification, dob, contact, pickup_location, address, house_no, street, area)
VALUES (1007, 'Ethan Taylor', 'Sarah Taylor', 'Elementary', '2016-12-03', '3210987654', 'Playground Gate', '123 Willow Drive', '123', 'Willow Del
INSERT INTO student (user_id, student_name, parent_name, classification, dob, contact, pickup_location, address, house_no, street, area)
VALUES (1008, 'Ava Anderson', 'Thomas Anderson', 'High School', '2008-07-25', '2109876543', 'Gymnasium', '456 Aspen Place', '456', 'Aspen Place',
INSERT INTO student (user_id, student_name, parent_name, classification, dob, contact, pickup_location, address, house_no, street, area)
VALUES (1009, 'Noah Lee', 'Michelle Lee', 'Middle School', '2011-04-19', '1098765432', 'Science Wing', '789 Sycamore Road', '789', 'Sycamore Road
INSERT INTO student (user_id, student_name, parent_name, classification, dob, contact, pickup_location, address, house_no, street, area)
VALUES (1010, 'Isabella Rodriguez', 'Miguel Rodriguez', 'Elementary', '2017-08-11', '9087654321', 'Front Office', '234 Redwood Avenue', '234', 'R

```

Au  
disa  
me  
cun  
b

Fig 3.1 creating database and adding in login table



```

mysql> -- 2. 'attendance' Table
mysql> CREATE TABLE attendance (
    ->     attendance_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     registration_number INT,
    ->     attended INT,
    ->     not_attended INT
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO attendance (registration_number, attended, not_attended)
VALUES
    -> (1001, 22, 3),
    -> (1002, 18, 7),
    -> (1003, 19, 5),
    -> (1004, 20, 4),
    -> (1005, 17, 6),
    -> (1006, 23, 2),
    -> (1007, 25, 1);
Query OK, 7 rows affected (0.01 sec)
Records: 7  Duplicates: 0  Warnings: 0

mysql> -- 3. 'admin' Table
mysql> CREATE TABLE admin (
    ->     admin_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     username VARCHAR(50),
    ->     password VARCHAR(50),
    ->     email_id1 VARCHAR(50),
    ->     email_id2 VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO admin (username, password, email_id1, email_id2) VALUES
    -> ('admin1', 'pass1', 'admin1@example.com', 'admin1.alt@example.com'),
    -> ('admin2', 'pass2', 'admin2@example.com', 'admin2.alt@example.com'),
    -> ('admin3', 'pass3', 'admin3@example.com', 'admin3.alt@example.com');
Query OK, 3 rows affected (0.04 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> -- 4. 'employee' Table

```

Fig 3.2 Creating a Table ,Admin and inserting values

```

mysql> -- 4. 'employee' Table
mysql> CREATE TABLE employee (
->     employee_id INT PRIMARY KEY AUTO_INCREMENT,
->     name VARCHAR(50),
->     contact1 VARCHAR(15),
->     contact2 VARCHAR(15),
->     email_id1 VARCHAR(50),
->     email_id2 VARCHAR(50)
-> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO employee (name, contact1, contact2, email_id1, email_id2)
VALUES
-> ('John Doe', '1234567890', '0987654321', 'john.doe@example.com', 'john.alt@example.com'),
-> ('Jane Smith', '2345678901', '1987654321', 'jane.smith@example.com', 'jane.alt@example.com'),
-> ('Mike Johnson', '3456789012', '2987654321', 'mike.johnson@example.com', 'mike.alt@example.com'),
-> ('Sara Lee', '4567890123', '3987654321', 'sara.lee@example.com', 'sara.alt@example.com');
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> -- 5. 'meal' Table
mysql> CREATE TABLE meal (
->     meal_id INT PRIMARY KEY AUTO_INCREMENT,
->     special_diet VARCHAR(50),
->     day_of_week VARCHAR(10),
->     meal_type VARCHAR(20),
->     breakfast VARCHAR(50),
->     lunch VARCHAR(50),
->     dinner VARCHAR(50)
-> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO meal (special_diet, day_of_week, meal_type, breakfast, lunch, dinner) VALUES
-> ('Vegan', 'Monday', 'Standard', 'Oats', 'Rice and Beans', 'Salad'),
-> ('Vegan', 'Tuesday', 'Standard', 'Smoothie', 'Veggie Stir-fry', 'Lent

```

Fig 3.3 Creating employee and meal tables and inserting values

```

    -> ('Non-Vegan', 'Wednesday', 'Standard', 'Eggs', 'Chicken Rice', 'Steak
'),
    -> ('Non-Vegan', 'Thursday', 'Standard', 'Pancakes', 'Beef Stew', 'Grill
ed Fish'),
    -> ('Vegetarian', 'Friday', 'Standard', 'Toast', 'Cheese Sandwich', 'Veg
etable Pasta'),
    -> ('Vegan', 'Saturday', 'Standard', 'Fruit Salad', 'Chickpea Curry', 'R
oasted Veggies');
Query OK, 6 rows affected (0.01 sec)
Records: 6 Duplicates: 0 Warnings: 0

mysql> -- 6. 'room_1' Table
mysql> CREATE TABLE room_1 (
    ->     room_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     room_number INT,
    ->     capacity INT,
    ->     status VARCHAR(20),
    ->     room_type1 VARCHAR(20),
    ->     room_type2 VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO room_1 (room_number, capacity, status, room_type1, room_t
ype2) VALUES
    -> (101, 2, 'Vacant', 'Single', 'Standard'),
    -> (102, 4, 'Occupied', 'Double', 'Deluxe'),
    -> (103, 3, 'Occupied', 'Single', 'Standard'),
    -> (104, 4, 'Vacant', 'Double', 'Standard'),
    -> (105, 2, 'Occupied', 'Single', 'Deluxe');
Query OK, 5 rows affected (0.04 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> -- 7. 'room_2' Table
mysql> CREATE TABLE room_2 (
    ->     room_id INT PRIMARY KEY,
    ->     amenities VARCHAR(100),
    ->     FOREIGN KEY (room_id) REFERENCES room_1(room_id)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO room_2 (room_id, amenities) VALUES

```

Fig3.4: creating table room1 and 2 and inserting the values

```

mysql> INSERT INTO room_2 (room_id, amenities) VALUES
-> (1, 'WiFi, Air Conditioning'),
-> (2, 'WiFi, Heater'),
-> (3, 'Air Conditioning, TV'),
-> (4, 'Heater, TV, WiFi'),
-> (5, 'WiFi, Balcony, Air Conditioning');
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> -- 8. 'student_information_1' Table
mysql> CREATE TABLE student_information_1 (
->     registration_number INT PRIMARY KEY,
->     name VARCHAR(50),
->     date_of_birth DATE,
->     gender VARCHAR(10),
->     chronic_illness VARCHAR(100),
->     allergies VARCHAR(100)
-> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO student_information_1 (registration_number, name, date_of
_birth, gender, chronic_illness, allergies) VALUES
-> (1001, 'Alice Brown', '2000-05-20', 'Female', 'Asthma', 'Peanuts'),
-> (1002, 'Bob Green', '1999-08-15', 'Male', NULL, 'None'),
-> (1003, 'Cathy White', '2001-12-01', 'Female', 'Diabetes', 'Gluten'),
-> (1004, 'David Black', '2002-03-12', 'Male', 'Hypertension', 'Shellfis
h');
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> -- 9. 'student_information_2' Table
mysql> CREATE TABLE student_information_2 (
->     registration_number INT PRIMARY KEY,
->     email VARCHAR(50),
->     medical_information VARCHAR(100),
->     emergency_contact VARCHAR(15),
->     FOREIGN KEY (registration_number) REFERENCES student_information_
1(registration_number)
-> );
Query OK, 0 rows affected (0.07 sec)

```

fig 3.5 Creating student\_information\_1, student\_information\_2 and inserting values

```

mysql> INSERT INTO student_information_2 (registration_number, email, medical_information, emergency_contact) VALUES
    -> (1001, 'alice.brown@example.com', 'Mild asthma', '1234567890'),
    -> (1002, 'bob.green@example.com', NULL, '0987654321'),
    -> (1003, 'cathy.white@example.com', 'Insulin dependent', '2345678901'),
    -> (1004, 'david.black@example.com', 'Blood pressure medication', '3456789012');
Query OK, 4 rows affected (0.04 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> -- 10. 'fee' Table
mysql> CREATE TABLE fee (
    ->     fee_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     amount DECIMAL(10,2),
    ->     registration_number INT,
    ->     fee_month VARCHAR(20),
    ->     completed BOOLEAN,
    ->     pending BOOLEAN,
    ->     FOREIGN KEY (registration_number) REFERENCES student_information_1(registration_number)
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> INSERT INTO fee (amount, registration_number, fee_month, completed, pending) VALUES
    -> (5000, 1001, 'January', TRUE, FALSE),
    -> (3000, 1002, 'February', FALSE, TRUE),
    -> (4000, 1003, 'March', TRUE, FALSE),
    -> (2500, 1004, 'April', FALSE, TRUE);
Query OK, 4 rows affected (0.04 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> -- 11. 'visitors' Table
mysql> CREATE TABLE visitors (
    ->     reg_id INT,
    ->     visitor_name VARCHAR(50),
    ->     relation VARCHAR(20),
    ->     date_of_visit DATE,
    ->     check_out_time TIME
    -> );

```

Fig 3.6 inserting values into student\_information and creating fee table and inserting values

```

Query OK, 0 rows affected (0.07 sec)

mysql>
mysql> INSERT INTO visitors (reg_id, visitor_name, relation, date_of_visit,
check_out_time) VALUES
    -> (1001, 'Karen Brown', 'Mother', '2024-11-01', '18:00'),
    -> (1002, 'Sam Green', 'Father', '2024-11-02', '17:30'),
    -> (1003, 'Linda White', 'Sister', '2024-11-05', '19:00'),
    -> (1004, 'James Black', 'Uncle', '2024-11-06', '16:30');
Query OK, 4 rows affected (0.04 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> -- 12. 'complaint' Table
mysql> CREATE TABLE complaint (
    ->     comp_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     datelog DATE,
    ->     date_resolved DATE,
    ->     description VARCHAR(255),
    ->     resolved BOOLEAN,
    ->     pending BOOLEAN,
    ->     in_progress BOOLEAN
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO complaint (datalog, date_resolved, description, resolved,
pending, in_progress) VALUES
    -> ('2024-10-15', NULL, 'Leaking faucet in room 101', FALSE, TRUE, TRUE)
    ,
    -> ('2024-10-18', '2024-10-20', 'Broken window in room 102', TRUE, FALSE
, FALSE),
    -> ('2024-10-19', NULL, 'AC not working in room 103', FALSE, TRUE, TRUE)
    ,
    -> ('2024-10-21', '2024-10-23', 'WiFi issue in common area', TRUE, FALSE
, FALSE);
Query OK, 4 rows affected (0.04 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> -- 13. 'request' Table
mysql> CREATE TABLE request (

```

fig 3.7 Creating Table fee ,Complaint,vistors

```

mysql> -- 13. `request` Table
mysql> CREATE TABLE request (
    ->     req_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     registration_number INT,
    ->     leave_req DATE,
    ->     description VARCHAR(255)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO request (registration_number, leave_req, description) VALUES
    -> (1001, '2024-11-10', 'Request for medical leave'),
    -> (1002, '2024-11-12', 'Request for family visit'),
    -> (1003, '2024-11-15', 'Request for academic event leave'),
    -> (1004, '2024-11-18', 'Request for internship leave');
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SHOW TABLES;
+-----+
| Tables_in_hostelmanagement |
+-----+
| admin                        |
| attendance                  |
| complaint                   |
| employee                    |
| fee                          |
| login                       |
| meal                        |
| request                     |
| room_1                      |
| room_2                      |
| student_information_1       |
| student_information_2       |
| visitors                    |
+-----+
13 rows in set (0.01 sec)

```

Fig 3.8 creating request table and inserting values into table and displaying the table with values

```
mysql> SHOW COLUMNS FROM login;
```

Field	Type	Null	Key	Default	Extra
login_id	int	NO	PRI	NULL	auto_increment
login_role	varchar(50)	YES		NULL	
login_username	varchar(50)	YES	UNI	NULL	
password	varchar(50)	YES		NULL	

```
4 rows in set (0.01 sec)
```

```
mysql> SHOW COLUMNS FROM attendance;
```

Field	Type	Null	Key	Default	Extra
attendance_id	int	NO	PRI	NULL	auto_increment
registration_number	int	YES		NULL	
attended	int	YES		NULL	
not_attended	int	YES		NULL	

```
4 rows in set (0.00 sec)
```

```
mysql> SHOW COLUMNS FROM admin;
```

Field	Type	Null	Key	Default	Extra
admin_id	int	NO	PRI	NULL	auto_increment
username	varchar(50)	YES		NULL	
password	varchar(50)	YES		NULL	
email_id1	varchar(50)	YES		NULL	
email_id2	varchar(50)	YES		NULL	

```
5 rows in set (0.00 sec)
```

Fig3.9 displaying the login , attendance , admin tables



```
mysql> SHOW COLUMNS FROM employee;
```

Field	Type	Null	Key	Default	Extra
employee_id	int	NO	PRI	NULL	auto_increment
name	varchar(50)	YES		NULL	
contact1	varchar(15)	YES		NULL	
contact2	varchar(15)	YES		NULL	
email_id1	varchar(50)	YES		NULL	
email_id2	varchar(50)	YES		NULL	

```
6 rows in set (0.00 sec)
```

```
mysql> SHOW COLUMNS FROM meal;
```

Field	Type	Null	Key	Default	Extra
meal_id	int	NO	PRI	NULL	auto_increment
special_diet	varchar(50)	YES		NULL	
day_of_week	varchar(10)	YES		NULL	
meal_type	varchar(20)	YES		NULL	
breakfast	varchar(50)	YES		NULL	
lunch	varchar(50)	YES		NULL	
dinner	varchar(50)	YES		NULL	

```
7 rows in set (0.00 sec)
```

```
mysql> SHOW COLUMNS FROM room_1;
```

Field	Type	Null	Key	Default	Extra
room_id	int	NO	PRI	NULL	auto_increment
room_number	int	YES		NULL	
capacity	int	YES		NULL	
status	varchar(20)	YES		NULL	
room_type1	varchar(20)	YES		NULL	
room_type2	varchar(20)	YES		NULL	

Fig3.10 displaying the table of employee meal and room\_1

```
mysql> SHOW COLUMNS FROM room_2;
```

Field	Type	Null	Key	Default	Extra
room_id	int	NO	PRI	NULL	
amenities	varchar(100)	YES		NULL	

```
2 rows in set (0.00 sec)
```

```
mysql> SHOW COLUMNS FROM student_information_1;
```

Field	Type	Null	Key	Default	Extra
registration_number	int	NO	PRI	NULL	
name	varchar(50)	YES		NULL	
date_of_birth	date	YES		NULL	
gender	varchar(10)	YES		NULL	
chronic_illness	varchar(100)	YES		NULL	
allergies	varchar(100)	YES		NULL	

```
6 rows in set (0.00 sec)
```

```
mysql> SHOW COLUMNS FROM student_information_2;
```

Field	Type	Null	Key	Default	Extra
registration_number	int	NO	PRI	NULL	
email	varchar(50)	YES		NULL	
medical_information	varchar(100)	YES		NULL	
emergency_contact	varchar(15)	YES		NULL	

```
4 rows in set (0.00 sec)
```

Fig 3.11 displaying the table of room\_2, student\_information\_1 ,student\_information\_2

```
mysql> SHOW COLUMNS FROM fee;
```

Field	Type	Null	Key	Default	Extra
fee_id	int	NO	PRI	NULL	auto_increment
amount	decimal(10,2)	YES		NULL	
registration_number	int	YES	MUL	NULL	
fee_month	varchar(20)	YES		NULL	
completed	tinyint(1)	YES		NULL	
pending	tinyint(1)	YES		NULL	

```
6 rows in set (0.00 sec)
```

```
mysql> SHOW COLUMNS FROM visitors;
```

Field	Type	Null	Key	Default	Extra
reg_id	int	YES		NULL	
visitor_name	varchar(50)	YES		NULL	
relation	varchar(20)	YES		NULL	
date_of_visit	date	YES		NULL	
check_out_time	time	YES		NULL	

```
5 rows in set (0.00 sec)
```

Fig 3.12 displaying fee tables and visitor tables

```
mysql> SHOW COLUMNS FROM complaint;
```

Field	Type	Null	Key	Default	Extra
comp_id	int	NO	PRI	NULL	auto_increment
datelog	date	YES		NULL	
date_resolved	date	YES		NULL	
description	varchar(255)	YES		NULL	
resolved	tinyint(1)	YES		NULL	
pending	tinyint(1)	YES		NULL	
in_progress	tinyint(1)	YES		NULL	

```
7 rows in set (0.00 sec)
```

```
mysql> SHOW COLUMNS FROM request;
```

Field	Type	Null	Key	Default	Extra
req_id	int	NO	PRI	NULL	auto_increment
registration_number	int	YES		NULL	
leave_req	date	YES		NULL	
description	varchar(255)	YES		NULL	

```
4 rows in set (0.00 sec)
```

```
mysql> --normalization
```

```
-> CREATE TABLE fee_details (  
->   fee_id INT PRIMARY KEY AUTO_INCREMENT,  
->   amount DECIMAL(10, 2),  
->   completed TINYINT(1),  
->   pending TINYINT(1)  
-> );
```

Fig 3.13: displaying complaint and the request tables

## **Dml commands:**

Data Manipulation Language (DML) refers to a subset of SQL commands that are used to manage and manipulate data within a database. DML allows users to perform operations such as retrieving, inserting, updating, and deleting data from database tables. These commands are essential for maintaining the integrity and consistency of the data in a database, ensuring that the stored information can be queried, modified, and removed as needed.

### **1. INSERT:**

- **Purpose:** The INSERT command is used to add new records (rows) into a table in the database. It allows you to specify the columns and the corresponding values for the new row.

### **2. UPDATE:**

- **Purpose:** The UPDATE command modifies the existing records in a table. It allows you to change the values of specified columns for rows that match a condition.

### **3. DELETE:**

- **Purpose:** The DELETE command removes one or more rows from a table based on a condition. It is used to delete data from a table while preserving the table structure.

### **4. SELECT:**

- **Purpose:** The SELECT command is used to query and retrieve data from one or more tables. It allows you to specify which columns to retrieve and to filter the data based on conditions.

### **5. TRUNCATE:**

- **Purpose:** The TRUNCATE command removes all rows from a table, without logging the removal of individual rows. This operation is faster than DELETE but cannot be rolled back in certain database systems.

### **6. MERGE (Upsert):**

- **Purpose:** The MERGE command combines the functionality of INSERT and UPDATE. It is used to insert a new record or update an existing record depending on whether a match is found in the target table based on a specified condition.

These are the key DML commands, each with its specific function for managing data in a relational database. You can include these definitions along with the outputs of your executed commands in your report.

Insert commands

```
mysql> /*dml commands*/
mysql> -- Insert new data into fee_details table
mysql> INSERT INTO fee_details (amount, completed, pending)
    -> VALUES (5500.00, 1, 0);
Query OK, 1 row affected (0.04 sec)

mysql>
mysql> -- Insert a new registration into fee_registration table
mysql> INSERT INTO fee_registration (registration_number, fee_month, fee_id)
    -> VALUES (1005, 'May', 5);
```

Fig 3.14 :showing insert operations

```
mysql> -- Now insert the registration with fee_id = 5
mysql> INSERT INTO fee_registration (registration_number, fee_month, fee_id)
    -> VALUES (1005, 'May', 5);
Query OK, 1 row affected (0.04 sec)

mysql> -- Check available fee_ids in fee_details
mysql> SELECT * FROM fee_details;
```

fee_id	amount	completed	pending
1	5000.00	1	0
2	3000.00	0	1
3	4000.00	1	0
4	2500.00	0	1
5	5500.00	1	0
8	5500.00	1	0

```
6 rows in set (0.00 sec)
```

Figure.3.15: showing insert operations

```
mysql> -- Check all registrations in fee_registration table
mysql> SELECT * FROM fee_registration;
```

registration_number	fee_month	fee_id
1001	January	1
1002	February	2
1003	March	3
1004	April	4
1005	May	5

```
5 rows in set (0.00 sec)
```

Figure3.16: showing insert operations

### Truncate:

```
mysql> -- Create a new table 'new_fee_details'
mysql> CREATE TABLE new_fee_details (
  ->   fee_id INT PRIMARY KEY AUTO_INCREMENT,
  ->   amount DECIMAL(10, 2),
  ->   completed TINYINT(1),
  ->   pending TINYINT(1)
  -> );
Query OK, 0 rows affected, 2 warnings (0.04 sec)

mysql> -- Insert data into the 'new_fee_details' table
mysql> INSERT INTO new_fee_details (amount, completed, pending)
  -> VALUES (300.00, 1, 0),
  ->         (150.00, 0, 1),
  ->         (200.00, 1, 0);
Query OK, 3 rows affected (0.03 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
mysql> -- View the inserted data
mysql> SELECT * FROM new_fee_details;
+-----+-----+-----+-----+
| fee_id | amount | completed | pending |
+-----+-----+-----+-----+
|      1 | 300.00 |          1 |        0 |
|      2 | 150.00 |          0 |        1 |
|      3 | 200.00 |          1 |        0 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
mysql> -- Truncate the 'new_fee_details' table to remove all records
mysql> TRUNCATE TABLE new_fee_details;
Query OK, 0 rows affected (0.04 sec)

mysql>
mysql> -- View the table after truncation (it should be empty)
mysql> SELECT * FROM new_fee_details;
Empty set (0.00 sec)

mysql>
```

Fig3.17: implementation of creating a new\_fee\_details and truncating the new table

**CREATE TABLE:** We create a new table called new\_fee\_details with columns fee\_id, amount, completed, and pending. The fee\_id column is set to auto-increment, so it will automatically generate a unique ID for each record.

**INSERT:** We insert three records into the new table with different values for amount, completed, and pending. After inserting, we use SELECT \* to view the data.

**TRUNCATE:** We use the TRUNCATE command to remove all records from the new\_fee\_details table. After truncating, we use SELECT \* to confirm that the table is empty.

### Update commands:

```
mysql> /*update command*/
mysql> -- Update the amount in fee_details table
mysql> UPDATE fee_details
    -> SET amount = 6000.00
    -> WHERE fee_id = 1;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> -- Update the fee registration status to completed
mysql> UPDATE fee_registration
    -> SET fee_month = 'June'
    -> WHERE registration_number = 1005;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM fee_details;
+-----+-----+-----+-----+
| fee_id | amount | completed | pending |
+-----+-----+-----+-----+
| 1 | 6000.00 | 1 | 0 |
| 2 | 3000.00 | 0 | 1 |
| 3 | 4000.00 | 1 | 0 |
| 4 | 2500.00 | 0 | 1 |
| 5 | 5500.00 | 1 | 0 |
| 8 | 5500.00 | 1 | 0 |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> SELECT * FROM fee_registration;
+-----+-----+-----+
| registration_number | fee_month | fee_id |
+-----+-----+-----+
| 1001 | January | 1 |
| 1002 | February | 2 |
| 1003 | March | 3 |
| 1004 | April | 4 |
| 1005 | June | 5 |
+-----+-----+-----+
5 rows in set (0.00 sec)
```



## MERGE:

```
mysql>
mysql> -- Insert data into the target table
mysql> INSERT INTO target_table (fee_id, amount)
    -> VALUES (1, 200.00), (3, 400.00);
Query OK, 2 rows affected (0.01 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> -- Merge data from source_table into target_table
mysql> INSERT INTO target_table (fee_id, amount)
    -> SELECT fee_id, amount FROM source_table
    -> ON DUPLICATE KEY UPDATE
    ->     amount = VALUES(amount);
Query OK, 3 rows affected, 1 warning (0.03 sec)
Records: 2 Duplicates: 1 Warnings: 1

mysql>
mysql> -- View the target_table after merge
mysql> SELECT * FROM target_table;
+-----+-----+
| fee_id | amount |
+-----+-----+
|      1 | 500.00 |
|      2 | 300.00 |
|      3 | 400.00 |
+-----+-----+
3 rows in set (0.00 sec)
```

Fig.3.19: showing merge operation

## Delete commands:

```
mysql> -- Delete a record from fee_registration
mysql> DELETE FROM fee_registration
    -> WHERE registration_number = 1005;
Query OK, 1 row affected (0.04 sec)

mysql>
mysql> -- Delete a fee record from fee_details
mysql> DELETE FROM fee_details
    -> WHERE fee_id = 5;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM fee_details;
+-----+-----+-----+-----+
| fee_id | amount | completed | pending |
+-----+-----+-----+-----+
|      1 | 6000.00 |          1 |          0 |
|      2 | 3000.00 |          0 |          1 |
|      3 | 4000.00 |          1 |          0 |
|      4 | 2500.00 |          0 |          1 |
|      8 | 5500.00 |          1 |          0 |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM fee_registration;
+-----+-----+-----+
| registration_number | fee_month | fee_id |
+-----+-----+-----+
|          1001 | January |      1 |
|          1002 | February |      2 |
|          1003 | March |      3 |
|          1004 | April |      4 |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

Fig.3.20: showing delete command

### Select commands:

Fig:3.21 showing select commands

```
mysql> -- Select all records from fee_details
mysql> SELECT * FROM fee_details;
+-----+-----+-----+-----+
| fee_id | amount | completed | pending |
+-----+-----+-----+-----+
| 1 | 6000.00 | 1 | 0 |
| 2 | 3000.00 | 0 | 1 |
| 3 | 4000.00 | 1 | 0 |
| 4 | 2500.00 | 0 | 1 |
| 8 | 5500.00 | 1 | 0 |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
mysql> -- Select specific columns from fee_registration
mysql> SELECT registration_number, fee_month
-> FROM fee_registration
-> WHERE fee_id = 1;
+-----+-----+
| registration_number | fee_month |
+-----+-----+
| 1001 | January |
+-----+-----+
1 row in set (0.00 sec)
```

### JOIN COMMANDS:

Fig.3.22: showing join commands

```
mysql> /* joins*/
mysql> SELECT fr.registration_number, fr.fee_month, fd.amount
-> FROM fee_registration fr
-> INNER JOIN fee_details fd ON fr.fee_id = fd.fee_id;
+-----+-----+-----+
| registration_number | fee_month | amount |
+-----+-----+-----+
| 1001 | January | 6000.00 |
| 1002 | February | 3000.00 |
| 1003 | March | 4000.00 |
| 1004 | April | 2500.00 |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT fr.registration_number, fr.fee_month, fd.amount
-> FROM fee_registration fr
-> LEFT JOIN fee_details fd ON fr.fee_id = fd.fee_id;
+-----+-----+-----+
| registration_number | fee_month | amount |
+-----+-----+-----+
| 1001 | January | 6000.00 |
| 1002 | February | 3000.00 |
| 1003 | March | 4000.00 |
| 1004 | April | 2500.00 |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT fr.registration_number, fr.fee_month, fd.amount
-> FROM fee_registration fr
-> RIGHT JOIN fee_details fd ON fr.fee_id = fd.fee_id;
+-----+-----+-----+
| registration_number | fee_month | amount |
+-----+-----+-----+
| 1001 | January | 6000.00 |
| 1002 | February | 3000.00 |
| 1003 | March | 4000.00 |
| 1004 | April | 2500.00 |
| NULL | NULL | 5500.00 |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

### SELECT (UNION/INTERSECT/EXCEPT)

```
mysql> SELECT registration_number, fee_month FROM fee_registration
-> UNION
-> SELECT registration_number, fee_month FROM fee_registration;
```

registration_number	fee_month
1001	January
1002	February
1003	March
1004	April

4 rows in set (0.04 sec)

```
mysql> SELECT registration_number, fee_month FROM fee_registration
-> INTERSECT
-> SELECT registration_number, fee_month FROM fee_registration;
```

registration_number	fee_month
1001	January
1002	February
1003	March
1004	April

4 rows in set (0.00 sec)

```
mysql> SELECT registration_number, fee_month FROM fee_registration
-> EXCEPT
-> SELECT registration_number, fee_month FROM fee_registration;
```

Empty set (0.00 sec)

Fig 3.23:showing select operation

### STORED PROCEDURE:

```
mysql> /*stored procedures*/
mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE insert_fee(
->     IN p_amount DECIMAL(10,2),
->     IN p_completed BOOLEAN,
->     IN p_pending BOOLEAN
-> )
-> BEGIN
->     INSERT INTO fee_details (amount, completed, pending)
->     VALUES (p_amount, p_completed, p_pending);
-> END$$
Query OK, 0 rows affected (0.05 sec)

mysql>
mysql> DELIMITER ;
mysql>
mysql> -- Call the procedure
mysql> CALL insert_fee(7000.00, 1, 0);
Query OK, 1 row affected (0.01 sec)

mysql> -- Check fee_details table to see if the new record is inserted
mysql> SELECT * FROM fee_details;
```

fee_id	amount	completed	pending
1	6000.00	1	0
2	3000.00	0	1
3	4000.00	1	0
4	2500.00	0	1
8	5500.00	1	0
9	7000.00	1	0

6 rows in set (0.00 sec)

Fig 3.24:showing stored procedures

### FUNCTIONS:

```
mysql> /*functions*/
mysql> DELIMITER $$
mysql>
mysql> CREATE FUNCTION get_total_fee(p_registration_number INT)
-> RETURNS DECIMAL(10,2)
-> DETERMINISTIC
-> BEGIN
->     DECLARE total_fee DECIMAL(10,2);
->     SELECT SUM(amount) INTO total_fee
->     FROM fee_details fd
->     JOIN fee_registration fr ON fd.fee_id = fr.fee_id
->     WHERE fr.registration_number = p_registration_number;
->     RETURN total_fee;
-> END$$
Query OK, 0 rows affected (0.04 sec)

mysql>
mysql> DELIMITER ;
mysql>
mysql> -- Call the function
mysql> SELECT get_total_fee(1001);
+-----+
| get_total_fee(1001) |
+-----+
|          6000.00 |
+-----+
1 row in set (0.01 sec)

mysql> DELIMITER $$
mysql>
mysql> CREATE TRIGGER after_fee_update
-> AFTER UPDATE ON fee_details
-> FOR EACH ROW
-> BEGIN
->     IF NEW.completed = 1 THEN
->         UPDATE fee_registration
->         SET fee_month = 'Completed'
->         WHERE fee_id = NEW.fee_id;
->     END IF;
-> END$$
Query OK, 0 rows affected (0.01 sec)
```

Fig 3.25: The functions are displayed here

### UPDATE:

```
mysql> DELIMITER ;
mysql> -- Update fee_details to set completed to 1 for fee_id 3
mysql> UPDATE fee_details
-> SET completed = 1
-> WHERE fee_id = 3;
Query OK, 0 rows affected (0.03 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> -- Check the fee_registration table
mysql> SELECT * FROM fee_registration;
+-----+-----+-----+
| registration_number | fee_month | fee_id |
+-----+-----+-----+
|          1001 | January |      1 |
|          1002 | February |      2 |
|          1003 | Completed |      3 |
|          1004 | April |      4 |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

Fig 3.26: showing update operations

## CURSOR:

Fig showing cursor operations

The cursor iterates over each row in the fee\_details table where fees are pending and prints the fee\_id and amount.

```
mysql>
mysql> CREATE PROCEDURE process_pending_fees()
-> BEGIN
->   DECLARE done INT DEFAULT FALSE;
->   DECLARE feeID INT;
->   DECLARE feeAmount DECIMAL(10, 2);
->
->   DECLARE fee_cursor CURSOR FOR
->     SELECT fee_id, amount FROM fee_details WHERE pending = 1;
->
->   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
->
->   OPEN fee_cursor;
->
->   fetch_loop: LOOP
->     FETCH fee_cursor INTO feeID, feeAmount;
->     IF done THEN
->       LEAVE fetch_loop;
->     END IF;
->
->     -- Here you can add processing logic, e.g., displaying the fee details
->     SELECT CONCAT('Processing pending fee ID: ', feeID, ' with Amount: ', feeAmount) AS message;
->   END LOOP;
->
->   CLOSE fee_cursor;
-> END //
```

Query OK, 0 rows affected (0.01 sec)

```
mysql>
mysql> DELIMITER ;
mysql> CALL process_pending_fees();
```

message
Processing pending fee ID: 2 with Amount: 3000.00

1 row in set (0.00 sec)

message
Processing pending fee ID: 4 with Amount: 2500.00

1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

```
mysql>
```

Fig 3.27:showing cursor operations

```

mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE cursor_example()
-> BEGIN
->     DECLARE done INT DEFAULT 0;
->     DECLARE r_number INT;
->     DECLARE r_month VARCHAR(20);
->     DECLARE cur CURSOR FOR
->         SELECT registration_number, fee_month FROM fee_registration;
->
->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
->
->     OPEN cur;
->
->     read_loop: LOOP
->         FETCH cur INTO r_number, r_month;
->         IF done THEN
->             LEAVE read_loop;
->         END IF;
->         SELECT r_number, r_month;
->     END LOOP;
->
->     CLOSE cur;
-> END$$
Query OK, 0 rows affected (0.03 sec)

```

Fig.3.28.procedures for registration

```

mysql>
mysql> DELIMITER ;
mysql>
mysql> -- Call the cursor procedure
mysql> CALL cursor_example();
+-----+-----+
| r_number | r_month |
+-----+-----+
|      1001 | January |
+-----+-----+
1 row in set (0.00 sec)

+-----+-----+
| r_number | r_month |
+-----+-----+
|      1002 | February |
+-----+-----+
1 row in set (0.01 sec)

+-----+-----+
| r_number | r_month |
+-----+-----+
|      1003 | Completed |
+-----+-----+
1 row in set (0.01 sec)

+-----+-----+
| r_number | r_month |
+-----+-----+
|      1004 | April |
+-----+-----+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

mysql>

```

Fig:3.29.result of cursor procedure

## **TRIGGERS:**

Trigger to automatically log any updates or deletions on the fee\_details table into a log table called fee\_log.

1. **Create the Log Table:**
2. **Create a Trigger for Update Events:** This trigger will log any updates made to the amount in fee\_details.
3. **Create a Trigger for Delete Events:** This trigger will log any deletions from the fee\_details table.

```
mysql> CREATE TABLE fee_log (  
-> log_id INT PRIMARY KEY AUTO_INCREMENT,  
-> fee_id INT,  
-> action_type VARCHAR(10),  
-> old_amount DECIMAL(10, 2),  
-> new_amount DECIMAL(10, 2),  
-> modified_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
-> );  
Query OK, 0 rows affected (0.03 sec)  
  
mysql> DELIMITER //  
mysql>  
mysql> CREATE TRIGGER log_fee_update  
-> AFTER UPDATE ON fee_details  
-> FOR EACH ROW  
-> BEGIN  
-> IF OLD.amount != NEW.amount THEN  
-> INSERT INTO fee_log (fee_id, action_type, old_amount, new_amount)  
-> VALUES (OLD.fee_id, 'UPDATE', OLD.amount, NEW.amount);  
-> END IF;  
-> END; //  
Query OK, 0 rows affected (0.01 sec)  
  
mysql>  
mysql> DELIMITER ;  
mysql> DELIMITER //  
mysql>  
mysql> CREATE TRIGGER log_fee_delete  
-> AFTER DELETE ON fee_details  
-> FOR EACH ROW  
-> BEGIN  
-> INSERT INTO fee_log (fee_id, action_type, old_amount)  
-> VALUES (OLD.fee_id, 'DELETE', OLD.amount);  
-> END; //  
Query OK, 0 rows affected (0.01 sec)  
  
mysql>  
mysql> DELIMITER ;  
mysql>
```

Figure 3.30:showing Triggers operations

This trigger will log any changes made to the amount field in the fee\_details table by recording the old and new values in a fee\_changes\_log table.

- The log\_amount\_change trigger fires before an UPDATE operation on the fee\_details table.
- It checks if the amount field is being changed (OLD.amount <> NEW.amount).
- If so, it inserts a record into fee\_changes\_log, including the fee\_id, the previous amount (OLD), the new amount (NEW), and the timestamp of the change.

This trigger can be useful for tracking changes to fee amounts over time.

```
mysql> CREATE TABLE fee_changes_log (  
-> log_id INT PRIMARY KEY AUTO_INCREMENT,  
-> fee_id INT,  
-> old_amount DECIMAL(10, 2),  
-> new_amount DECIMAL(10, 2),  
-> changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
-> );  
Query OK, 0 rows affected (0.04 sec)  
  
mysql> DELIMITER //  
mysql>  
mysql> CREATE TRIGGER log_amount_change  
-> BEFORE UPDATE ON fee_details  
-> FOR EACH ROW  
-> BEGIN  
-> IF OLD.amount <> NEW.amount THEN  
-> INSERT INTO fee_changes_log (fee_id, old_amount, new_amount)  
-> VALUES (OLD.fee_id, OLD.amount, NEW.amount);  
-> END IF;  
-> END //  
Query OK, 0 rows affected (0.01 sec)  
  
mysql>  
mysql> DELIMITER ;  
mysql>
```

Fig.3.31:Execution of triggers for amount changes



### **AGGREGATE FUNCTION:**

**SUM:** Adds up all values in a numeric column, providing the total.

**AVG:** Calculates the average of values in a numeric column.

**COUNT:** Counts the number of rows in a dataset or unique entries in a column.

**MAX:** Finds the largest value in a specified column.

**MIN:** Finds the smallest value in a specified column.

**COUNT(DISTINCT):** Counts the number of unique values, ignoring duplicates.

```
mysql> /* AGGREGATE COMMANDS*/
mysql> SELECT SUM(amount) AS total_amount FROM fee_details;
+-----+
| total_amount |
+-----+
|      28000.00 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT AVG(amount) AS average_amount FROM fee_details;
+-----+
| average_amount |
+-----+
|    4666.666667 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) AS completed_count FROM fee_details WHERE completed = 1;
+-----+
| completed_count |
+-----+
|                4 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT MAX(amount) AS max_amount, MIN(amount) AS min_amount FROM fee_details;
+-----+-----+
| max_amount | min_amount |
+-----+-----+
|    7000.00 |    2500.00 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(DISTINCT pending) AS distinct_pending_count FROM fee_details;
+-----+
| distinct_pending_count |
+-----+
|                2 |
+-----+
1 row in set (0.00 sec)
```

FIG.3.32:Aggreivate commands for the database

```
mysql> SELECT completed, pending, SUM(amount) AS total_amount
-> FROM fee_details
-> GROUP BY completed, pending;
+-----+-----+-----+
| completed | pending | total_amount |
+-----+-----+-----+
| 1 | 0 | 22500.00 |
| 0 | 1 | 5500.00 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> SELECT SUM(amount) AS total_completed_amount, AVG(amount) AS avg_completed_amount
-> FROM fee_details
-> WHERE completed = 1;
+-----+-----+
| total_completed_amount | avg_completed_amount |
+-----+-----+
| 22500.00 | 5625.000000 |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) AS total_entries FROM fee_details;
+-----+
| total_entries |
+-----+
| 6 |
+-----+
1 row in set (0.01 sec)
```

fig3.33:Aggreivate functions of the tables

### **Normalization:**

In database design, **Normal Forms (NF)** are guidelines used to structure relational databases to reduce redundancy and ensure data integrity. Each level of NF, from 1NF to BCNF, removes specific types of anomalies and dependencies.

In this project, only the Fee\_Details table required conversion into BCNF (Boyce-Codd Normal Form) due to its initial dependency structure. Specifically, the Fee\_Details table had non-trivial functional dependencies where attributes were dependent on a part of the composite key rather than the whole key. This dependency violated the requirements for BCNF, where every determinant must be a candidate key. Therefore, normalization was applied to restructure the table, ensuring it met the stricter conditions of BCNF by eliminating partial dependencies.

The remaining tables were already in 3NF (Third Normal Form) as they satisfied the requirements: each non-key attribute depended solely on the primary key without transitive dependencies. Since they adhered to 3NF, which is a prerequisite for BCNF, no further normalization was necessary for these tables. This structure effectively minimized redundancy and maintained data integrity across the database.

```

mysql> CREATE TABLE fee_details (
  ->   fee_id INT PRIMARY KEY AUTO_INCREMENT,
  ->   amount DECIMAL(18, 2),
  ->   completed TINYINT(1),
  ->   pending TINYINT(1)
  -> );
Query OK, 0 rows affected, 2 warnings (0.08 sec)

mysql> CREATE TABLE fee_registration (
  ->   registration_number INT,
  ->   fee_month VARCHAR(20),
  ->   fee_id INT,
  ->   PRIMARY KEY (registration_number, fee_month),
  ->   FOREIGN KEY (fee_id) REFERENCES fee_details(fee_id)
  -> );
Query OK, 0 rows affected (0.07 sec)

mysql> -- Insert data into fee_details table
mysql> INSERT INTO fee_details (amount, completed, pending)
  -> SELECT DISTINCT amount, completed, pending
  -> FROM fee;
Query OK, 4 rows affected (0.06 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert data into fee_registration table
mysql> INSERT INTO fee_registration (registration_number, fee_month, fee_id)
  -> SELECT registration_number, fee_month, fee_id
  -> FROM fee;
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> DROP TABLE fee;
Query OK, 0 rows affected (0.07 sec)

```

Fig 4.1: creating 2 other tables from the fee table such as fee\_registration and fee\_details

```

mysql> -- Check data in fee_details table
mysql> SELECT * FROM fee_details;
+-----+-----+-----+-----+
| fee_id | amount | completed | pending |
+-----+-----+-----+-----+
| 1 | 5000.00 | 1 | 0 |
| 2 | 3000.00 | 0 | 1 |
| 3 | 4000.00 | 1 | 0 |
| 4 | 2500.00 | 0 | 1 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
mysql> -- Check data in fee_registration table
mysql> SELECT * FROM fee_registration;
+-----+-----+-----+
| registration_number | fee_month | fee_id |
+-----+-----+-----+
| 1001 | January | 1 |
| 1002 | February | 2 |
| 1003 | March | 3 |
| 1004 | April | 4 |
+-----+-----+-----+
4 rows in set (0.00 sec)

```

Fig 4.2: fee\_details provided here uses normalization to be converted into bcnf which is the only non normalized table

## 1. Transaction in SQL

A transaction ensures that multiple SQL operations are executed as a single unit. Transactions are used to ensure that all operations succeed or none of them take effect, maintaining the integrity of the database.

Here's an example of how to start, commit, and rollback a transaction in SQL:

### Example: Using Transactions

After performing a transaction with SQL operations such as UPDATE, COMMIT, or ROLLBACK, you would need to confirm whether the transaction was successful or if it was rolled back. To visualize the outcome of these operations, you can use SELECT statements to check the current state of the data in the database.

```
mysql> -- Start the transaction
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Perform the first SQL operation
mysql> UPDATE fee_details
    -> SET amount = amount + 100
    -> WHERE fee_id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
mysql> -- Perform the second SQL operation
mysql> UPDATE fee_details
    -> SET amount = amount - 50
    -> WHERE fee_id = 2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
mysql> -- If all operations are successful, commit the transaction
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> -- Now, check the changes in the 'fee_details' table
mysql> SELECT * FROM fee_details WHERE fee_id IN (1, 2);
+-----+-----+-----+-----+
| fee_id | amount | completed | pending |
+-----+-----+-----+-----+
|      1 | 6200.00 |          1 |        0 |
|      2 | 2900.00 |          0 |        1 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Fig 4.3: In this fig, after committing the transaction, you would use a SELECT statement to see if the values for fee\_id = 1 and fee\_id = 2 were updated properly.

**If there was an error, you would ROLLBACK the transaction:**

```
mysql> -- In case of an error, rollback the transaction
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Check if changes have been undone
mysql> SELECT * FROM fee_details WHERE fee_id IN (1, 2);
+-----+-----+-----+-----+
| fee_id | amount | completed | pending |
+-----+-----+-----+-----+
|      1 | 6200.00 |          1 |        0 |
|      2 | 2900.00 |          0 |        1 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Fig 4.4: case the transaction was rolled back due to an error, the values for fee\_id = 1 and fee\_id = 2 should not have been changed, and the SELECT statement should show the original values.

```
mysql> -- Check the status of the table after COMMIT
mysql> SELECT fee_id, amount FROM fee_details;
+-----+-----+
| fee_id | amount |
+-----+-----+
|      1 | 6288.88 |
|      2 | 2988.88 |
|      3 | 4888.88 |
|      4 | 2588.88 |
|      8 | 5588.88 |
|      9 | 7888.88 |
+-----+-----+
6 rows in set (0.00 sec)

mysql>
```

Fig4.5: the transaction is selected and displayed

1. **Transaction Handling:**

- START TRANSACTION ensures that the operations are atomic.
- COMMIT at the end ensures the changes are persisted.
- ROLLBACK in the event of a deadlock or any other error ensures that we don't leave the database in an inconsistent state.

2. **Concurrency Control:**

- SELECT ... FOR UPDATE locks the rows selected by the cursor, preventing other transactions from modifying them while the procedure is running. This prevents **dirty reads** and **lost updates**.

3. **Deadlock Handling:**

- The DECLARE CONTINUE HANDLER FOR SQLSTATE '40001' block catches deadlock errors. If a deadlock occurs, the procedure will **rollback** the transaction and set done = 1 to exit the loop, allowing you to retry if necessary.

**Handling Deadlocks:**

Deadlocks occur when two or more transactions are waiting for each other to release locks. In such a case, MySQL will automatically detect the deadlock and raise an error (SQLSTATE '40001'). The procedure is designed to handle this by rolling back the transaction and terminating the loop.

## **Concurrency Control (in Transport Management System)**

### **Key Points about Concurrency Control**

**SELECT ... FOR UPDATE:** This statement locks the rows selected by the cursor. These rows are locked until the transaction is committed, which means that no other transactions can modify these rows while they are being processed.

**Transaction Block:** By wrapping the cursor operation within a START TRANSACTION and COMMIT, the entire operation becomes atomic. The changes made (if any) will only be persisted when the transaction is committed.

### **Concurrency Control Behavior**

**Row-Level Locking:** The FOR UPDATE clause ensures that as the cursor fetches rows, those rows are locked and cannot be modified by other transactions. If another transaction tries to modify the same rows, it will be blocked until the current transaction is committed or rolled back.

**No Dirty Reads or Lost Updates:** Other transactions will not see the intermediate state of the locked rows, preventing dirty reads. Also, lost updates are avoided because only one transaction can update a locked row at a time.

This approach is effective in scenarios where you need to ensure that the rows being processed are not concurrently modified by other transactions, maintaining data integrity.

```

mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE cursor_example()
-> BEGIN
->     DECLARE done INT DEFAULT 0;
->     DECLARE r_number INT;
->     DECLARE r_month VARCHAR(20);
->
->     -- Declare a cursor with SELECT FOR UPDATE to lock the rows for processing
->     DECLARE cur CURSOR FOR
->         SELECT registration_number, fee_month FROM fee_registration
->         FOR UPDATE;
->
->     -- Handler for NOT FOUND to exit the loop
->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
->
->     -- Start the transaction (implicitly handling concurrency control here)
->     START TRANSACTION;
->
->     OPEN cur;
->
->     read_loop: LOOP
->         FETCH cur INTO r_number, r_month;
->
->         IF done THEN
->             LEAVE read_loop;
->         END IF;
->
->         -- Perform your processing logic here
->         -- For example, just SELECT the values for demonstration
->         SELECT r_number, r_month;
->
->     END LOOP;
->
->     -- Commit the transaction after processing
->     COMMIT;
->
->     CLOSE cur;
-> END$$
ERROR 1304 (42000): PROCEDURE cursor_example already exists
mysql>
mysql> DELIMITER ;
mysql>
mysql> -- Call the cursor procedure
mysql> CALL cursor_example();
+-----+-----+
| r_number | r_month |
+-----+-----+
|    1001 | Completed |
+-----+-----+
1 row in set (0.00 sec)

+-----+-----+
| r_number | r_month |
+-----+-----+
|    1002 | February |
+-----+-----+
1 row in set (0.00 sec)

+-----+-----+
| r_number | r_month |
+-----+-----+
|    1003 | Completed |
+-----+-----+
1 row in set (0.01 sec)

+-----+-----+
| r_number | r_month |
+-----+-----+
|    1004 | April |
+-----+-----+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

```

Fig 5.1.showing concurrency operations

## Deadlock in Database Systems

A **deadlock** in a database system occurs when two or more transactions are unable to proceed because they are each waiting for the other to release a resource, such as a data row or table lock. This situation leads to a state where the transactions are indefinitely blocked and cannot continue execution, causing the system to enter a **standstill**.

Deadlocks typically happen in systems with **concurrent transactions**, where multiple transactions are trying to access the same data resources in conflicting orders. For example, one transaction might lock a row of data and wait for another transaction to release a lock on a different row, while the second transaction is waiting for the first transaction to release its lock. This creates a **circular dependency**, resulting in a deadlock.

## Deadlock Detection and Resolution

In most modern database systems, such as MySQL, deadlocks are detected automatically by the **deadlock detection algorithm**. When the system identifies a deadlock, it will **abort one of the transactions** involved to break the cycle. The transaction that is chosen to be rolled back is generally the one that has done the least work or is less costly to abort.

Once a transaction is aborted, it is rolled back, and the application can choose to either handle the failure gracefully (e.g., by retrying the transaction) or log the failure for further analysis.

## Common Causes of Deadlocks

1. **Concurrency:** When multiple transactions are trying to modify the same data simultaneously, deadlocks can occur if their access patterns overlap.
2. **Lock Ordering:** Deadlocks can arise when transactions acquire locks in different orders. For instance, Transaction A locks Table 1 and waits for Table 2, while Transaction B locks Table 2 and waits for Table 1.
3. **Resource Contention:** If transactions request conflicting locks (e.g., a read lock and a write lock on the same resource), deadlocks may occur if both are blocked by the other.

## Deadlock Prevention and Management

To prevent or manage deadlocks, several strategies can be employed:

- **Lock Ordering:** Ensure that transactions acquire locks in a consistent order to avoid circular waiting.
- **Timeouts:** Use timeouts to automatically roll back a transaction after a set duration, reducing the risk of long-standing deadlocks.



- **Transaction Granularity:** Keep transactions short and only lock the data that is necessary, minimizing the chances of deadlocks.

- **Deadlock Handlers:** Use database-specific deadlock handlers, such as the **CONTINUE HANDLER** in MySQL, to automatically handle deadlocks by rolling back the transaction and optionally retrying it.

## Deadlock Example

Consider the following scenario in a database with two transactions:

- **Transaction A** locks Row 1 and waits for Row 2.
- **Transaction B** locks Row 2 and waits for Row 1.

In this case, neither transaction can proceed because each is waiting for the other to release a lock, resulting in a **deadlock**.

## Conclusion

Deadlocks are a natural consequence of concurrent access to resources in a database system. While they are difficult to completely eliminate, understanding the causes and implementing strategies for deadlock detection, prevention, and resolution helps ensure that database systems remain performant and responsive even under high concurrency.

In this example, we have two transactions that are attempting to update two rows in the same table, but they do so in an order that causes a deadlock. Both transactions lock different rows and wait for each other to release their locks, leading to a deadlock.

Fig 5.2: The transactions are committed and delays are simulated and committed

```
mysql> -- Commit the transaction (will not reach here if deadlock occurs)
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> -- Session 1: Start the first transaction
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Lock Row 1 and update balance
mysql> UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
mysql> -- Simulate delay
mysql> SELECT SLEEP(5);
+-----+
| SLEEP(5) |
+-----+
|          |
+-----+
1 row in set (5.01 sec)

mysql>
mysql> -- Try to lock Row 2, which is already locked by Transaction 2
mysql> UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
mysql> -- Commit (won't reach here if a deadlock occurs)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

```

mysql> -- Session 2: Start the second transaction
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Lock Row 2 and update balance
mysql> UPDATE accounts SET balance = balance - 50 WHERE account_id = 2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
mysql> -- Simulate delay
mysql> SELECT SLEEP(5);
+-----+
| SLEEP(5) |
+-----+
|          0 |
+-----+
1 row in set (5.00 sec)

mysql>
mysql> -- Try to lock Row 1, which is already locked by Transaction 1
mysql> UPDATE accounts SET balance = balance + 50 WHERE account_id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
mysql> -- Commit (won't reach here if a deadlock occurs)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

```

**ERROR 1213 (40001): Deadlock found when trying to get lock;**

Figure 5.3: showing feadlock operations

```

mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE cursor_example()
-> BEGIN
->     DECLARE done INT DEFAULT 0;
->     DECLARE r_number INT;
->     DECLARE r_month VARCHAR(20);
->
->     -- Declare a cursor with SELECT FOR UPDATE to lock the rows for processing
->     DECLARE cur CURSOR FOR
->         SELECT registration_number, fee_month FROM fee_registration
->         FOR UPDATE;
->
->     -- Handlers for NOT FOUND and deadlock error
->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
->     DECLARE CONTINUE HANDLER FOR SQLSTATE '40001' -- Deadlock error
->     BEGIN
->         -- In case of a deadlock, we will rollback and retry the transaction
->         ROLLBACK;
->         SET done = 1;
->     END;
->
->     -- Start the transaction
->     START TRANSACTION;
->
->     OPEN cur;
->
->     read_loop: LOOP
->         FETCH cur INTO r_number, r_month;
->
->         IF done THEN
->             LEAVE read_loop;
->         END IF;
->
->         -- Perform your processing logic here
->         -- For example, let's just SELECT the values for demonstration
->         SELECT r_number, r_month;
->
->     END LOOP;
->
->     -- Commit the transaction if no issues
->     COMMIT;
->
->     CLOSE cur;
-> END$$

```

ERROR 1304 (42000): PROCEDURE cursor\_example already exists

```

mysql>
mysql> DELIMITER ;
mysql>
mysql> -- Call the cursor procedure
mysql> CALL cursor_example();

```

r_number	r_month
1001	Completed

1 row in set (0.00 sec)

r_number	r_month
1002	February

1 row in set (0.00 sec)

r_number	r_month
1003	Completed

1 row in set (0.00 sec)

r_number	r_month
1004	April

Figure.5.4: showing TRANSACTION CONCURRENCY AND DEADLOCK

## **FRONTEND:**

The Transportation Management System is designed to streamline school transportation administration by handling operations such as student registration, bus assignment, route planning, driver management, and real-time tracking. This system utilizes a relational SQL database for backend data management and a Flask-based web application for frontend access and user interaction. The application's main modules cover login and user roles, student tracking, bus assignment, route planning, driver management, and activity logging.

## **System Architecture and Components:**

The project is structured into separate modules, each dedicated to a different aspect of transportation management. Below, we outline the purpose of each folder and file in this project.

FOLDER/FILE	DESCRIPTION
app.py	The main file containing application routes, connecting to the database, and implementing business logic.
static	Contains all static assets, including CSS, JavaScript, and images.
templates	Stores the HTML templates for the web pages, allowing Flask to render dynamic content based on the data.
database	Contains SQL scripts and schema definitions for the system's relational database.

The Transportation Management System is organized into a clear folder structure that separates backend logic, static assets, and dynamic templates to facilitate efficient development and maintenance.

The main backend file, `app.py`, serves as the core of the application. It manages all routes (URLs) for navigating between pages in the system and connects to the SQL database to perform CRUD (Create, Read, Update, Delete) operations. Within `app.py`, each route is associated with specific logic, including login validation, form submission, and data retrieval. This file utilizes Flask's templating engine to render HTML templates dynamically, allowing data to flow seamlessly from the backend to the frontend user views.

The static folder contains essential static files like CSS, JavaScript, and images that are crucial for the visual and interactive aspects of the web application. CSS files provide consistent styling across each HTML page, ensuring a cohesive look and feel, while JavaScript files enhance the user experience by adding interactivity and additional functionality. Any images used in the application for branding or interface elements are also stored here.

## Database Design

The backend uses an SQL-based relational database that organizes transit data into multiple tables. Below is an overview of the primary tables in the database.

Table	Purpose
Login	Manages user credentials and roles for secure access.
Admin	Stores administrator information and contact details.
Student	Contains student demographic information, contact details, and pickup locations.
Bus	Stores bus identification information and registration numbers.
Driver	Contains records of bus drivers, their contact information, and assigned buses.
Conductor	Manages information about bus conductors, including contact details and bus assignments.
Route	Stores transportation routes, including locations, pickup and drop-off points.
Track	Contains real-time tracking information for buses along their routes.
Staff	Stores information about school staff involved in transportation management.
Student_Log	Tracks student transportation activities and maintains current records.
Prev_Student_Log	Archives historical student transportation records for reference.
Admin_Log	Records administrative actions for auditing and accountability purposes.

The templates folder holds all HTML templates that structure each page in the web application. These templates leverage Jinja, Flask's templating engine, to dynamically render content based on user actions and roles. For example, each HTML file is populated with specific data when Flask renders it based on user inputs or query results, making the application responsive and personalized.

### **HTML Template Pages Overview**

The HTML templates in templates/ serve as the frontend of the system, offering an interface through which users interact with the database. Below is an overview of each HTML page and its functionality:

<b>Template</b>	<b>Description</b>
base.html	Landing page with navigation links, directing users to sections based on their role.
login.html	A login page where users enter credentials. Authenticates users and directs them to their respective dashboards.
dashboard.html	The main control panel for administrators, showing system overview, bus assignments, and management options.
register.html	Registration page for new users (administrators, drivers, conductors, or staff).
student_form.html	Form for adding or editing student transportation information, including pickup locations and bus assignments.
students.html	Displays a list of all students registered in the transportation system with their assigned routes and buses.
student_portal.html	The main dashboard for students/parents, showing bus assignments, routes, pickup times, and tracking information.
bus_management.html	Interface for administrators to manage bus details, including registration numbers, capacity, and maintenance records.
driver_management.html	Page for managing driver information, including contact details and bus assignments.
route_management.html	Interface for creating and editing transportation routes, including pickup and drop-off locations.
tracking.html	Real-time tracking interface showing current bus locations and estimated arrival times.
conductor_management.html	Page for managing conductor information and bus assignments.

staff_management.html	Interface for managing school staff involved in transportation coordination.
reports.html	Generates various reports on transportation operations, student assignments, and route efficiency.

## **Application Flow**

### 1. User Login and Role Management:

Users log in through the login.html page, entering credentials.

Based on the role (admin, driver, conductor, staff), users are directed to different sections of the application.

### 2. Student Management:

Administrators use student\_form.html to register students for transportation services.

Student information includes pickup location, contact details, and classification.

The students.html page displays all registered students with their transportation details.

### 3. Transportation Resource Management:

Administrators manage buses through bus\_management.html, assigning routes and drivers.

Driver and conductor information is managed through respective management pages.

Routes are created and modified through route\_management.html, defining pickup and drop-off points.

### 4. Tracking and Monitoring:

The tracking.html page provides real-time information on bus locations.

Students/parents can view their assigned bus and track its current location.

Historical tracking data is stored for reference and analysis.

### 5. Reporting and Analytics:

The reports.html page generates various reports on transportation operations.

Administrators can analyze route efficiency, bus utilization, and student assignment patterns.



## **Static->css->style.css:**

```
body {
  font-family: 'Segoe UI', Arial, sans-serif;
  background: linear-gradient(120deg, #f8f9fa 0%, #e0eafc 100%);
  margin: 0;
  padding: 0;
  min-height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
}
.container {
  background: rgba(255, 255, 255, 0.85);
  box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.18);
  border-radius: 16px;
  padding: 40px 32px 32px 32px;
  max-width: 400px;
  width: 100%;
  margin: 40px auto;
  animation: fadeIn 1s ease;
}
@keyframes fadeIn {
  from { opacity: 0; transform: translateY(30px); }
  to { opacity: 1; transform: translateY(0); }
}
h2 {
  text-align: center;
  color: #3a3a3a;
  margin-bottom: 24px;
  font-weight: 700;
  letter-spacing: 1px;
}
.form-group label {
  font-weight: 500;
  color: #333;
}
.form-control {
```

```

border-radius: 8px;
border: 1px solid #bfc9d9;
padding: 10px 14px;
margin-bottom: 16px;
font-size: 1rem;
transition: border-color 0.2s;
}
.form-control:focus {
border-color: #6c63ff;
outline: none;
box-shadow: 0 0 0 2px #e0eafc;
}
.btn-primary {
background: linear-gradient(90deg, #6c63ff 0%, #48c6ef 100%);
border: none;
border-radius: 8px;
color: #fff;
font-weight: 600;
padding: 10px 0;
width: 100%;
box-shadow: 0 4px 14px 0 rgba(76,110,245,0.12);
transition: background 0.2s, transform 0.1s;
cursor: pointer;
}
.btn-primary:hover {
background: linear-gradient(90deg, #48c6ef 0%, #6c63ff 100%);
transform: translateY(-2px) scale(1.03);
}
.mt-3 {
margin-top: 1rem;
text-align: center;
}
.alert {
border-radius: 8px;
padding: 12px 18px;
margin-bottom: 18px;
font-size: 1rem;
}
.flash-message {
background: #e0ffe0;
border: 1px solid #b2ffb2;
color: #2d662d;
padding: 10px 20px;

```

```
margin: 10px 0;  
border-radius: 4px;  
font-size: 1rem;  
transition: opacity 0.5s;  
}
```

## **Static->js->script.js:**

```
document.addEventListener('DOMContentLoaded', function() {
  var form = document.querySelector('form');
  if (!form) return;

  form.addEventListener('submit', function(e) {
    var errors = [];
    var requiredFields = [
      { name: 'student_name', label: 'Student Name' },
      { name: 'parent_name', label: 'Parent Name' }
    ];
    requiredFields.forEach(function(field) {
      var input = form.querySelector("[name='" + field.name + "']");
      if (input && !input.value.trim()) {
        errors.push(field.label + ' is required. ');
        input.classList.add('is-invalid');
      } else if (input) {
        input.classList.remove('is-invalid');
      }
    });
    var dob = form.querySelector("[name='dob']");
    if (dob && dob.value) {
      var dobDate = new Date(dob.value);
      if (isNaN(dobDate.getTime())) {
        errors.push('Date of Birth is invalid. ');
        dob.classList.add('is-invalid');
      } else {
        dob.classList.remove('is-invalid');
      }
    }
    var age = form.querySelector("[name='age']");
    if (age && age.value && (isNaN(age.value) || age.value < 0)) {
      errors.push('Age must be a positive number. ');
      age.classList.add('is-invalid');
    } else if (age) {
      age.classList.remove('is-invalid');
    }
    var contact = form.querySelector("[name='contact']");
    if (contact && contact.value && !/^\d{10,15}$/.test(contact.value)) {
      errors.push('Contact must be a valid number (10-15 digits). ');
      contact.classList.add('is-invalid');
    } else if (contact) {
      contact.classList.remove('is-invalid');
    }
    var errorDiv = document.getElementById('form-errors');
```

```
if (!errorDiv) {
  errorDiv = document.createElement('div');
  errorDiv.id = 'form-errors';
  errorDiv.className = 'alert alert-danger';
  form.prepend(errorDiv);
}
if (errors.length > 0) {
  e.preventDefault();
  errorDiv.innerHTML = errors.join('<br>');
  errorDiv.style.display = 'block';
  return false;
} else {
  errorDiv.style.display = 'none';
}
});
});
const flashMessages = document.querySelectorAll('.flash-message');
flashMessages.forEach(function(msg) {
  setTimeout(function() {
    msg.style.display = 'none';
  }, 4000);
});
```

## Student.html

```
{% extends 'base.html' %}
{% block content %}
<div class="d-flex justify-content-between align-items-center mb-4">
  <h2>Students</h2>
  <a href="/students/new" class="btn btn-success">Add Student</a>
</div>
<table class="table table-striped table-hover">
  <thead class="table-primary">
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Parent</th>
      <th>Classification</th>
      <th>DOB</th>
      <th>Contact</th>
      <th>Pickup Location</th>
      <th>Address</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    {% for s in students %}
    <tr>
      <td>{{ s.user_id }}</td>
      <td>{{ s.student_name }}</td>
      <td>{{ s.parent_name }}</td>
      <td>{{ s.classification }}</td>
      <td>{{ s.dob.strftime('%Y-%m-%d') if s.dob else " " }}</td>
      <td>{{ s.contact }}</td>
      <td>{{ s.pickup_location }}</td>
      <td>{{ s.address }}</td>
      <td>
        <a href="/students/{{ s.user_id }}/edit" class="btn btn-sm btn-primary">Edit</a>
        <form action="/students/{{ s.user_id }}/delete" method="post" style="display:inline;">
          <button type="submit" class="btn btn-sm btn-danger" onclick="return confirm('Are you
sure?')">Delete</button>
        </form>
      </td>
    </tr>
    {% endfor %}
  </tbody>
</table>
{% if not students %}
  <div class="alert alert-info">No students found.</div>
{% endif %}
```

## **base.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Transport Management System</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
  <link rel="stylesheet" href="{ { url_for('static', filename='css/main.css') } }">
  <style>
    body { background: #f8f9fa; }
    .navbar-brand { font-weight: bold; }
    .container { margin-top: 40px; }
    .form-label { font-weight: 500; }
    .card { box-shadow: 0 2px 8px rgba(0,0,0,0.05); }
  </style>
</head>
<body>
{% if current_user.is_authenticated %}
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <div class="container-fluid">
    <a class="navbar-brand" href="/">TransportMS</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle
navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link" href="/students">Students</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/buses">Buses</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/drivers">Drivers</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/conductors">Conductors</a>
        </li>
      </ul>
    </div>
  </div>
</div>
```

```

</nav>
{% endif %}
<div class="container">
  {% with messages = get_flashed_messages(with_categories=true) %}
    {% if messages %}
      {% for category, message in messages %}
        <div class="alert alert-{{ category }} alert-dismissible fade show" role="alert">
          {{ message }}
          <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
label="Close"></button>
        </div>
      {% endfor %}
    {% endif %}
  {% endwith %}
  {% block content %}{% endblock %}
</div>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
<script src="{{ url_for('static', filename='js/main.js') }}"></script>
</body>
</html>

```



## AdminDashboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Admin Dashboard</title>
  <link rel="stylesheet" href="/static/css/style.css">
</head>
<body>
  <div class="container mt-5">
    <h2 class="mb-4">Admin Dashboard</h2>
    {% with messages = get_flashed_messages(with_categories=true) %}
      {% if messages %}
        {% for category, message in messages %}
          <div class="alert alert-{{ category }}">{{ message }}</div>
        {% endfor %}
      {% endif %}
    {% endwith %}
    <h3>Buses</h3>
    <ul>
      {% for bus in buses %}
        <li>{{ bus.BUS_NO }} (ID: {{ bus.BUS_ID }})</li>
      {% endfor %}
    </ul>
    <h3>Drivers</h3>
    <ul>
      {% for driver in drivers %}
        <li>{{ driver.NAME }} (ID: {{ driver.ID }})</li>
      {% endfor %}
    </ul>
    <h3>Routes</h3>
    <ul>
      {% for route in routes %}
        <li>{{ route.route_name }} (ID: {{ route.route_id }})</li>
      {% endfor %}
    </ul>
    <h3>Students</h3>
    <ul>
      {% for student in students %}
        <li>{{ student.student_name }} (ID: {{ student.user_id }})</li>
      {% endfor %}
    </ul>
  </div>
</body></html>
```

## User registration.html

```
{% extends 'base.html' %}
{% block content %}
<div class="card mx-auto" style="max-width: 600px;">
  <div class="card-body">
    <h3 class="card-title mb-4">{{ 'Edit' if student else 'Add' }} Student</h3>
    <form method="post">
      <div class="mb-3">
        <label class="form-label">Student Name</label>
        <input type="text" name="student_name" class="form-control" value="{{
student.student_name if student else '' }}" required>
      </div>
      <div class="mb-3">
        <label class="form-label">Parent Name</label>
        <input type="text" name="parent_name" class="form-control" value="{{
student.parent_name if student else '' }}" required>
      </div>
      <div class="mb-3">
        <label class="form-label">Classification</label>
        <input type="text" name="classification" class="form-control" value="{{
student.classification if student else '' }}">
      </div>
      <div class="mb-3">
        <label class="form-label">Date of Birth</label>
        <input type="date" name="dob" class="form-control" value="{{ student.dob.strftime('%Y-
%m-%d') if student and student.dob else '' }}">
      </div>
      <div class="mb-3">
        <label class="form-label">Contact</label>
        <input type="text" name="contact" class="form-control" value="{{ student.contact if student
else '' }}">
      </div>
      <div class="mb-3">
        <label class="form-label">Pickup Location</label>
        <input type="text" name="pickup_location" class="form-control" value="{{
student.pickup_location if student else '' }}">
      </div>
      <div class="mb-3">
        <label class="form-label">Address</label>
        <input type="text" name="address" class="form-control" value="{{ student.address if student
else '' }}">
      </div>
      <div class="mb-3">
        <label class="form-label">House No</label>
        <input type="text" name="house_no" class="form-control" value="{{ student.house_no if
```

```

student else " }}">
    </div>
    <div class="mb-3">
        <label class="form-label">Street</label>
        <input type="text" name="street" class="form-control" value="{{ student.street if student else
" }}">
    </div>
    <div class="mb-3">
        <label class="form-label">Area</label>
        <input type="text" name="area" class="form-control" value="{{ student.area if student else "
}}">
    </div>
    <div class="mb-3">
        <label class="form-label">Age</label>
        <input type="number" name="age" class="form-control" value="{{ student.age if student
else " }}">
    </div>
    <div class="mb-3">
        <label class="form-label">Grade</label>
        <input type="text" name="grade" class="form-control" value="{{ student.grade if student
else " }}">
    </div>
    <div class="mb-3">
        <label class="form-label">Route ID</label>
        <input type="text" name="route_id" class="form-control" value="{{ student.route_id if
student else " }}">
    </div>
    <div class="d-flex justify-content-between">
        <button type="submit" class="btn btn-primary">{{ 'Update' if student else 'Add' }}</button>
        <a href="/students" class="btn btn-secondary">Cancel</a>
    </div>
</form>
<script src="/static/js/main.js"></script>
</div>
</div>
{% endblock %}

```

# Results

## Database Integration

The transportation management system integrates seamlessly with a MySQL database, ensuring efficient storage and retrieval of data. The system architecture facilitates smooth communication between the frontend and the database, enabling real-time updates and transactions. This integration supports scalability, security, and reliability in handling transportation-related data across multiple entities including students, buses, drivers, routes, and tracking information.



## Functionality Highlights

**User Authentication:** Users can authenticate their credentials through a secure login page with role-based access control. The Login entity stores LOGIN\_ID, LOGIN\_ROLE, LOGIN\_USER-NAME, and PASSWORD, ensuring appropriate access privileges for administrators, drivers, conductors, and staff.

**Admin Dashboard:** The administrative portal serves as the main control center, displaying key information such as bus assignments, route details, driver information, and student transportation logs. Through the MANAGE and WORK relationships, admins can oversee bus operations and driver assignments.

**Student Registration:** The system allows for comprehensive student registration by capturing details such as USER\_ID, STUDENT\_NAME, PARENT\_NAME, CLASSIFICATION, DOB, CONTACT, and address information (ADDRESS, HOUSE\_NO, STREET, CITY). This data is securely stored in the Student table.

**Bus Management:** The Bus entity (with BUS\_ID and BUS\_NO attributes) forms the core of the transportation system, connecting to drivers, conductors, routes, and tracking information through various relationships.

**Route Planning:** The Route entity stores ROUTE\_ID, BUS\_NO, LOCATION, PICKUP, and DROP\_OFF information, allowing administrators to efficiently plan and manage transportation routes. This connects to the Bus entity for assignment purposes.

**Real-time Tracking:** The Track entity enables real-time monitoring of buses with attributes like TRACK\_ID, BUS\_NO, LOCATION, PICKUP, DROP\_OFF, and ROUTE\_ID. This functionality enhances safety by providing visibility into current bus locations.

**Driver and Conductor Management:** The system maintains comprehensive records of transportation staff through the Driver and Conductor entities, capturing details such as ID, NAME, CONTACT,

and bus assignments.

**Activity Logging:** Multiple log entities (student\_log, prev\_student\_log, admin\_log) track system activities, providing audit trails for student transportation, administrative actions, and historical data.

**Staff Information:** The Staff entity stores details about additional personnel involved in transportation management, including STAFF\_ID, NAME, CONTACT\_NO, CLASS, and TEACHER\_ID.

The transportation management system successfully implements all required functionality while maintaining data integrity and security. The entity-relationship model effectively captures the complex relationships between different components of the transportation ecosystem, from student information to bus tracking and route management.

## Discussion

**T**he transportation management system meets the project requirements by providing a comprehensive solution for managing student transportation details, bus assignments, routes, driver information, and tracking logs. The user-friendly interface ensures that both students and administrators can navigate and interact with the system efficiently. The integration with MySQL ensures data integrity and supports real-time updates for all transportation operations.

**Future enhancements** may include additional features such as real-time notifications for parents about bus arrivals, an internal messaging system for drivers and staff, and advanced reporting tools for administrators. These improvements can further streamline transportation operations and enhance the student safety experience.

### Security Measures

**User Authentication and Access Control:** The system implements secure login functionality with user role-based access control through the Login entity. Drivers, conductors, administrators, and staff have access to different sections based on their roles, ensuring that sensitive information is accessible only to authorized users.

**Data Encryption:** Sensitive data such as passwords and student information are encrypted both in transit and at rest to ensure privacy and security.

**Secure Software Development Practices:** Best practices are followed to prevent vulnerabilities such as SQL injection and cross-site scripting (XSS), ensuring the integrity of the system's codebase.

**Compliance with Regulations:** The system adheres to data privacy regulations, ensuring that student transportation data is handled securely and in compliance with applicable laws.

### Data Integrity

**Input Validation and Sanitization:** The system includes rigorous input validation and sanitization mechanisms to prevent malicious data entries that could compromise the database.

**Transaction Integrity:** Transaction integrity checks are implemented to ensure that all data modifications are consistent and adhere to business rules, preventing data corruption.

**Audit Trails and Logging:** Detailed logging mechanisms through admin\_log, student\_log, and prev\_student\_log entities track changes to important data elements, ensuring transparency and accountability in the system.

**Data Redundancy and Backup:** Backup procedures are implemented to prevent data loss due to system failures or human errors, ensuring that data is always available and recoverable.

### User Interface Design

**Accessibility Features:** The user interface is designed to be accessible, including features such as keyboard navigation, compatibility with screen readers, and color contrast adjustments to meet accessibility standards.

**Intuitive Navigation:** The system's frontend includes easy-to-navigate pages for students, drivers, conductors, and administrators, providing quick access to key functions such as bus assignments, route information, and tracking details.

## **Challenges and Limitations**

**Integration with Legacy Systems:** While the system integrates well with modern features, the challenge remains in integrating with legacy school information systems that may still be in use for specific student records.

**Scalability:** As the number of students, buses, and routes grows, the system must be able to scale efficiently to handle an increased volume of data and transactions.

**Data Security Concerns:** Storing sensitive data such as student details, pickup locations, and real-time tracking information presents ongoing challenges in ensuring that the system remains secure and compliant with regulations.

**User Training and Adoption:** Ensuring that drivers, conductors, staff, and administrators adopt the new system can be challenging, particularly if they are used to traditional manual processes or older systems.

Future Enhancements

**Predictive Analytics and AI:** Future versions of the system could integrate predictive analytics to help staff identify optimal routes, predict traffic patterns, and estimate arrival times, potentially enhancing operational efficiency.

**Mobile App Integration:** A mobile version of the transportation management system could offer parents the ability to track their child's bus in real-time, receive notifications about delays, and communicate with drivers or conductors.

**IoT for Smart Transportation Management:** Integrating IoT devices could automate various aspects of transportation management, such as real-time vehicle diagnostics, fuel monitoring, and automated attendance tracking when students board or exit buses.

**Advanced Route Optimization:** Implementing advanced algorithms for route optimization could reduce travel time, fuel consumption, and overall transportation costs.

## **Conclusion**

In conclusion, the transportation management system effectively streamlines various operational tasks, enhances communication between students, parents, drivers, and administrators, and ensures secure and efficient data management. By continuing to focus on scalability, security, and user experience, the system can evolve to meet the growing needs of the school transportation network. Future enhancements, including mobile app integration and AI-based features, will further improve system performance, safety measures, and user satisfaction.

**OUTPUT SCREESHOTS:**

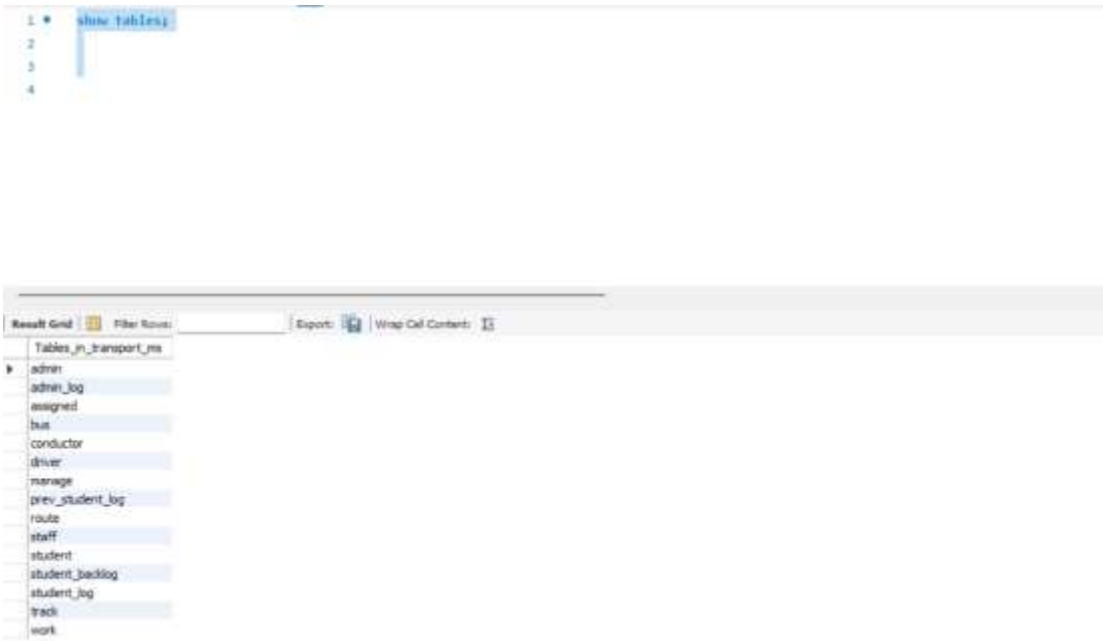
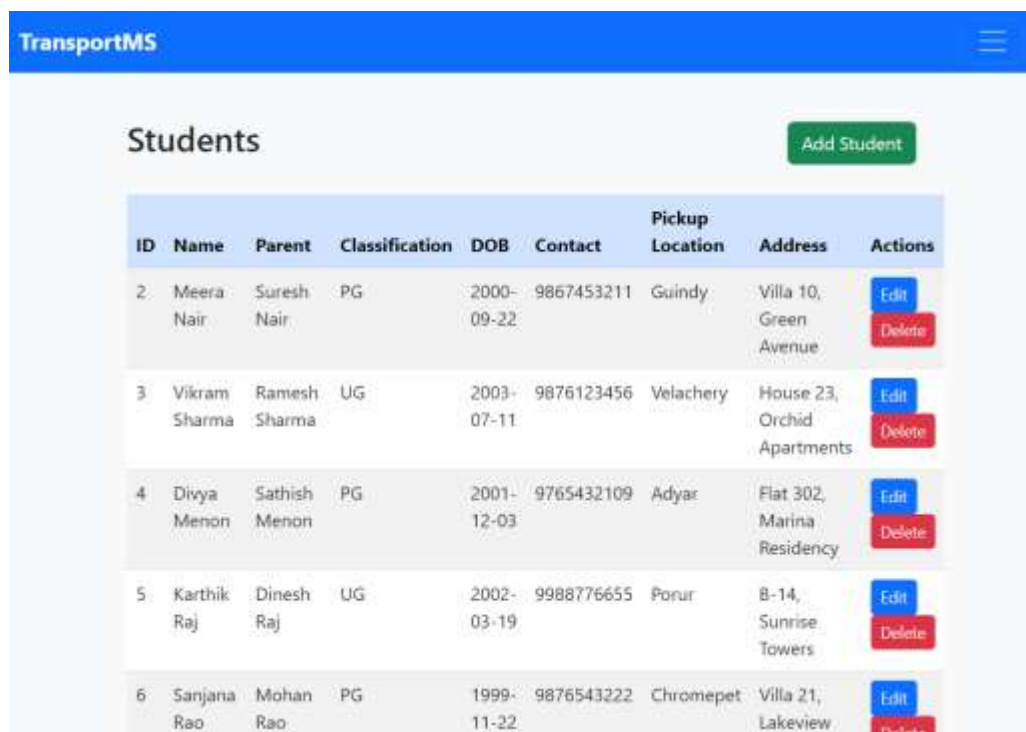



Fig 7.1: The tables in the database





Fig7.2:ADMIN CONTROL PAGE





Food Menu

Weekly Menu

Tuesday

Breakfast  
• Pancakes, Apple Syrup, Coffee  
• Calories: 450 kcal  
• Protein: 5 g  
• Fat: 10 g  
• Carbs: 70 g

Lunch  
• Grilled Chicken, Roasted Asparagus  
• Calories: 650 kcal  
• Protein: 30 g  
• Fat: 20 g  
• Carbs: 40 g

Dinner  
• Steak, Roasted Potatoes, Salad  
• Calories: 750 kcal  
• Protein: 35 g  
• Fat: 30 g  
• Carbs: 50 g

Nutrition Information

1800 kcal

95 g

90 g

175 g

Calories

Protein

Fat

Carbs

dietary-request-form

Your Name

Phone Number

Address/Postcode

Reason When Requested

Submit Request

meal-feedback-form

Student ID

Select Date

Select Copy

Select Meal

Select Meal

Additional Comments

Submit Feedback


Fig.7.3:FOOD MENU PAGE



FIG .7.5:Visitor registration page

S#	Student ID	Visitor Name	Address	Visit Date	Check-in Time	Check-out Time	Status
1		Jane Doe	Shaner	2024-11-01	2024-11-01 10:17:57	2024-11-01 10:50:00	Completed
2		Bob Smith	Fisher	2024-10-28	2024-11-01 10:57:57	2024-10-28 20:00:00	Completed
3		Clark Johnson	Stoner	2024-10-18	2024-11-01 10:57:57	2024-10-18 17:50:00	Completed
4		Diana Miller	Greene	2024-11-02	2024-11-01 10:57:57	2024-11-02 10:00:00	Completed
5		Eve Brown	Scott	2024-10-27	2024-11-01 10:57:57	2024-10-27 10:00:00	Completed
6		Frank Green	Stevens	2024-10-18	2024-11-01 10:57:57	2024-10-18 10:00:00	Completed
7		Grace Black	Travis	2024-11-04	2024-11-01 10:57:57	2024-11-04 20:00:00	Completed
8		Jack Gray	Walker	2024-10-11	2024-11-01 10:57:57	2024-10-11 10:00:00	Completed
9		Jay White	Yahara	2024-11-01	2024-11-01 10:57:57	2024-11-01 17:50:00	Completed
10		Jack Brown	Travis	2024-10-06	2024-11-01 10:57:57	2024-10-06 10:00:00	Completed
11		Visitors	Hard	2024-11-12	2024-11-12 10:26:28	None	Completed
12		Visitors	Hard	2024-11-12	2024-11-12 15:20:26	None	Completed
13		Visitors	Hard	2024-11-12	2024-11-12 20:22:25	None	Completed
14		Visitors	Hard	2024-11-12	2024-11-12 10:31:19	None	Completed
15		Visitors	Hard	2024-11-12	2024-11-12 10:31:19	None	Completed

Fig.7.6: Visitor logs



## Visitor Management

Total Visitors152

Current Visitors3

Upcoming Visits7

Submit Feedback


### Visitor Feedback

12345

Write your comments

Submit Feedback

Fig.7.7:Visitor feedback page



## Complaint Management

### Submit a New Complaint

Incident ID:

Complaint Title:

Description your complaint:

Select Category:

Submit Complaint

### Your Complaints

ID	Title	Description	Category	Status	Assigned Date
17	Broken AC	ac is not working and stopped working	Electricity	Pending	2024-11-12 15:51:16
1	Broken Bed	The bed in the room is broken and needs repair	Room issue	Pending	2024-11-12 15:51:17
2	Noisy Neighbor	The neighboring room is noisy, causing disturbance	Noise	Pending	2024-11-12 15:51:17
3	Locking Problem	There is a problem in the bathroom lock	Plumbing	Resolved	2024-11-12 15:51:17
4	Power Outage	Power went out in the room	Electricity	Unknown	2024-11-12 15:51:17
5	Hot Water Problem	Hot water is not working properly	Plumbing	Resolved	2024-11-12 15:51:17
6	AC Not Cooling	The AC is not cooling the room effectively	Room issue	Pending	2024-11-12 15:51:17
7	Door Handle Problem	Door handle is broken	Hardware	Resolved	2024-11-12 15:51:17
8	Broken Window	The window glass is broken	Room issue	Pending	2024-11-12 15:51:17
9	Laundry Service	Laundry service not working	Service	Resolved	2024-11-12 15:51:17
10	Unpleasant Bathroom	Bathroom is not clean and needs attention	Cleaning	Unknown	2024-11-12 15:51:17

Fig.7.8:Complaint page



Fig 7.9:Fee payment page

